C++

programiranje

za apsolutne početnike

PRO - MIL

Naslov knjige:	**C++ programiranje za apsolutne početnike**
Autor:	Jakopec Ratko, ing
Naklada:	**PRO-MIL d.o.o.** za nakladu, informatiku i edukaciju, Varaždin, R. Boškovića 20, 42000 Varaždin, tel: 042 / 203 981, 233 971, fax: 042 / 203 991, www.pro-mil.hr
Urednik:	**Nenad Milijaš, dipl. inf.**
Lektura:	**Ružica Gregurić, dipl. učitelj**
Korektura:	**PRO-MIL d.o.o.**
Recenzija:	**Igor Kos, dipl. inf.**
Naslovnica:	Ratko Jakopec, ing., **Nenad Milijaš, dipl. inf.**
Tiskara:	**Tiskara Varteks**, Varaždin
ISBN:	**953-7156-19-2**

Sadržaj

Uvod	**7**
Što je to program, a što programiranje?	8
Je li teško naučiti programirati?	8
Zašto baš C++?	9
Zašto baš Dev-C++?	10
Kako se uči programiranje?	10
Kako dalje nakon ove knjige?	10
Instaliranje Dev-C++ programskog okruženja	12
Formiranje radne mape	20
Instalacija grafike	22
Naš prvi program	**25**
Formiranje projekta	26
Mogući problem	32
Pokretanje programa	34
Greška u programu	38
Analiza programa	40
Gruba skica programa	42
Varijacije programa	43
Distribucija programa	47
Učitavanje spremljenog projekta	49
Varijable	**53**
Osnovne računske operacije	54
Varijable	58

Analiza programa	60
Varijacije programa	63
Problem dijeljenja	67
Uljepšavanje programa	69
Nekoliko primjera programa	71
Grafika	**75**
Podešavanje programskog okruženja	76
Naš prvi grafički program	77
Analiza programa	78
Varijacije programa	83
Crtanje pravokutnika	86
Crtanje crte	88
Crtanje točke	95
Ispis teksta	97
Donošenje odluke	**103**
Funkcije	104
Donošenje odluke	108
Switch naredba	126
Petlje	**129**
For petlja	130
Do while petlja	168
Generator slučajnih brojeva	170
Korištenje slučajnih brojeva	179

Polja	**183**
Jednodimenzionalna polja	184
Spremanje brojeva u datoteku	204
Dvodimenzionalna polja	212
Obrada teksta	**215**
String objekti	216
Polje char tipa	223
Obrada string objekata	234
Premještanje teksta iz char polja u string	246
Premještanje teksta iz stringa u char polje	247
Spremanje teksta u datoteku	248
Objekti	**251**
Jednostavan primjer klase i objekata	252
Složeniji primjeri klasa i objekata	262
Komunikacija s metodama	265
Razdvajanje deklaracije i definicije	276
Uporaba konstruktora	278
Nasljeđivanje	280
Veliki program	**283**
Jednostavna igra u jednoj datoteci	284
Jednostavna igra u više datoteka	288
Sažimanje koda	**301**
Primjeri sažetog pisanja koda	302

Uvod

Što je to program, a što programiranje?

Program je niz uputa računalu o tome kako da napravi određeni posao. Posao može bit vođenje skladišta, izračun plaće radnicima ili pretvaranje slike u boji u crno bijelu sliku. Programiranje je pisanje tih uputa. Za pisanje uputa odnosno programa koriste se određene naredbe. Skup naredbi koje čine cjelinu naziva se programski jezik.

Budući da su naredbe pisane našim jezikom, a računalo “razumije” samo nule i jedinice, program se nakon pisanja mora prevesti u oblik razumljiv računalu. Taj postupak se naziva prevođenje. Na engleskom jeziku naziva se Compile. Hrvatski programeri taj postupak obično nazivaju kompajliranje. U programu koji ćemo mi koristiti, postupak prevođenja naredbi u oblik razumljiv računalu naziva se kompajlaj, a ovoj knjizi koristit ćemo izraz prevođenje.

Da bismo uspješno programirali, očigledno je da nam treba program u kojem ćemo program pisati i program koji će napisani program prevesti u oblik razumljiv računalu. U današnje vrijeme sve te funkcije, a i mnoge dodatne objedinjene su u jedan program koji ćemo u ovoj knjizi nazivati programska okolina. On će nam omogućiti pisanje programa, prevođenje programa u oblik razumljiv računalu, traženje grešaka i pokretanje programa. Postoji cijeli niz takvih programa, a mi ćemo koristiti Dev-C++.

Je li teško naučiti programirati?

Da, teško je. Ako mislimo da ćemo uzeti nekakav programski jezik i da ćemo za par dana pisati programe, kao što možemo uzeti FrontPage pa za par dana napraviti WEB stranicu, od toga neće biti ništa.

Neki, ako ne i većina profesionalnih učitelja programiranja, započet će poduku nekakvim algoritmima i time kako je to sve jednostavno jer mi ionako mnoge stvari radimo po algoritmu. Npr. ako kuhamo kavu, prvo stavljamo vodu u posudu, zatim čekamo da zavri, pa onda stavljamo ... da sad ne kompliciramo, mi smo i do sada stvari radili po nekakvom algoritmu, a programiranje je nešto slično.

To su gluposti. Možda i jesmo do sada stvari radili po određenom algoritmu, ali nismo o tome na taj način razmišljali, nemamo um posložen na taj način. Osim toga u programiranju se upotrebljavaju malo drugačiji algoritmi nego što su algoritmi za kuhanje kave.

Programiranje je zapravo potpuno drugačiji način razmišljanja od onog na koji smo do sada navikli i trebat će godine rada da počnemo razmišljati na taj način. Cijeli problem dodatno komplicira činjenica da su današnja računala vrlo složeni sustavi i treba nam puno vremena da barem približno počnemo shvaćati kako računala funkcioniraju.

Zbog svega toga trebat će nam dvije do tri godine svakodnevnog rada da bismo koliko - toliko naučili programirati. Pod pojmom naučiti programirati ne mislim na dobivanje dvojke u školi ili pisanje programa za zbrajanje dva broja. Pod pojmom naučiti programirati mislim na to da smo u stanju napisati program koji zadovoljava nečije potrebe i taj netko nam je spreman za to platiti. Nema puno smisla reći da znam programirati, ali ne znam napraviti ništa za što bi netko bio spreman platiti.

Zbog svega toga važno je da smo radišni i strpljivi. Nemojmo iz činjenice da prva tri mjeseca nećemo puno toga razumjeti, zaključiti da nismo nadareni za programiranje i odustati. Nakon tri mjeseca nećemo puno toga razumjeti, ali nakon tri godine sasvim sigurno hoćemo.

Ovim uvodom vas nisam htio preplašiti, već sam vas htio ohrabriti da ustrajete u učenju programiranja, iako će vam se na početku možda činiti da ništa ne razumijete i da sporo napredujete.

Zašto baš C++?

Zato što je to trenutno najrašireniji jezik, osobito u profesionalnoj uporabi. U računalstvu se klonite ekskluzivnosti. Ako vas veseli da budete posebni, obrijte glavu ili obojite kosu na zeleno. Imate bezbroj mogućnosti, ali kad su računala u pitanju, najbolje je koristiti računalo koje koristi većina i programski jezik koji koristi većina.

U tom slučaju lako dolazimo do programa, lako dolazimo do literature, puno ljudi se razumije u to pa možemo dobiti savjet. Konačno, ako tražimo posao u smislu da programerska ekipa treba još jednoga (jednu), veća je vjerojatnost da oni već koriste sustav s kojim znamo raditi.

Bježite od genijalnih računala, revolucionarnih sustava i drugačijih razmišljanja. Zamislite koliko je ugodno živjeti u malom gradu i biti jedini vlasnik takvog genijalnog sustava. Nema literature, nema programa, nema savjeta i kad konačno savladamo rad u takvom sustavu nitko nas ne treba jer svi rade na nekim drugim sustavima.

Iako po mom sudu C++ ima niz prednosti u odnosu na druge jezike, nemojmo se suviše opterećivati time koji ćemo jezik započeti učiti i koju inačicu tog jezika. Glavni problem je naučiti jedan jezik, a kad nam to uspije, ako se ukaže potreba, lako ćemo prijeći na drugi. Da bi netko tko nikad nije programirao naučio jedan jezik, treba mu do tri godine, ali kad već jedan jezik naučimo, da bismo naučili drugi, dovoljno nam je do tri mjeseca. Naučivši jedan jezik, počinjemo razumijevati kako se programira, a to je najteže i najdulje traje. Najlakše je naučiti nove naredbe ili bolje rečeno novi način pisanja sličnih naredbi u drugom jeziku.

Zašto baš Dev-C++?

Programsko okruženje koje ćemo koristiti za pisanje naših programa naziva se Dev-C++. Zašto sam odabrao upravo ovo okruženje? Zato što je besplatno, ugodnog izgleda, na hrvatskom jeziku i malih dimenzija. Zbog toga što je malih dimenzija, zadovoljavajuće će raditi i na starijim računalima, a nove inačice moguće je u prihvatljivom vremenu učitati s interneta i pomoću spore internetske veze.

Iako je taj program relativno malih dimenzija, posjeduje sve osobine koje su potrebne za učenje programiranja, a mogao bi poslužiti i za manje složeno profesionalno programiranje.

Kako se uči programiranje?

Programiranje se ne uči tako da se nauči sadržaj neke knjige, a zatim se eventualno nešto radi na računalu. Programiranje se uči tako da se uključi računalo, otvori knjiga, čita sadržaj knjige i istovremeno prikazano u knjizi nastojimo napraviti na računalu.

Nakon što smo određenu naredbu shvatili, nismo završili s učenjem. Nakon toga treba napraviti nekoliko desetaka manjih programa u kojima ćemo tu naredbu koristiti da bismo stekli rutinu u njenom korištenju.

Kako dalje nakon ove knjige?

Ova knjiga nije zamišljena kao knjiga iz koje ćemo saznati sve tajne programiranja, nego kao knjiga koja bi ljudima koji nikad nisu programirali i ništa o tome na znaju trebala pomoći da započnu učenje programiranja.

Postoje velike i debele knjige s puno sitnog teksta koje na 1000 stranica objašnjavaju sve tajne pojedinih programskih jezika. Takve knjige su korisne i nabavite ih ako vam se ukaže prilika, ali takve knjige nisu pogodne za početnike. Zbog mnoštva detalja i šturih opisa, početnik će se u njima jako teško snaći.

Nakon što proučite ovu knjigu, imat ćete dovoljno predznanja da možete pratiti knjige koje detaljno govore o programskom jeziku C++. (Ili nekom drugom programskom jeziku.)

Kakve knjige biste trebali nabaviti i s čime biste se trebali baviti nakon što proučite ovu knjigu i želite dalje napredovati?

- Svakako neku knjigu koja na barem 800 stanica detaljno opisuje programski jezik koji želite naučiti.

- Budete li odlučili koristiti neko složenije programsko okruženje za pisanje progra-

ma, proučite upute za njegovo korištenje. Naravno, te upute ne moraju biti u obliku tiskane knjige. Najčešće će biti u obliku teksta na računalu.

- Programi rade unutar nekog operacijskog sustava, najčešće Windowsa ili Linuxa. Da bi mogli pisati programe za pojedine operacijske sustave, moramo razumjeti kako ti sustavi funkcioniraju pa ćemo morati nabaviti knjigu koja govori o strukturi operacijskog sustava za koji želimo pisati programe.

- Dobro je nabaviti i knjige koje govore općenito o programiranju i o načinima rješavanja pojedinih programskih problema. Želimo li napisati program koji sortira određenu grupu podataka na određeni način, ne moramo ga izmišljati. Drugi ljudi prije nas susreli su se s problemom sortiranja i o tom problemu napisali knjige u kojima iznose optimalna rješenja za pojedine probleme.

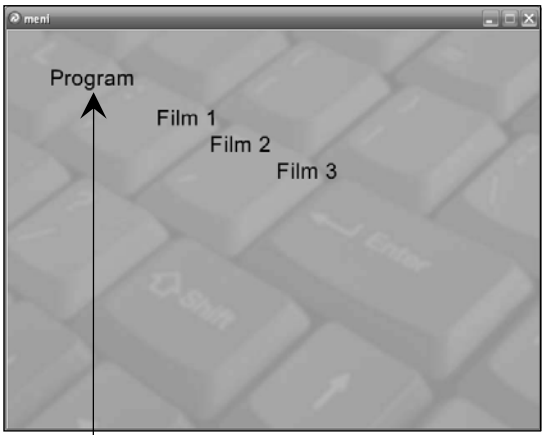
- S vremenom ćemo se specijalizirati za pojedine teme, npr. za pisanje igara, za grafiku, za obradu zvuka ili nešto slično, pa je dobro nabaviti knjigu koja govori o toj problematici.

Dakle, kad se detaljno upoznamo s nekim programskim jezikom, kad upoznamo određeni operacijski sustav, kad se upoznamo s programerskim tehnikama i određenim programerskim područjem, tek ćemo onda biti sposobni pisati kvalitetne komercijalne programe, odnosno programirati za novce.

Zbog toga sam na početku rekao da nam treba dvije do tri godine da naučimo programirati.

U ovom trenutku to vam možda zvuči obeshrabrujuće, ali ako sve to savladate, steći ćete vještinu kojom ne vlada baš svatko i koja ima svoju tržišnu cijenu.

Instaliranje Dev-C++ programskog okruženja

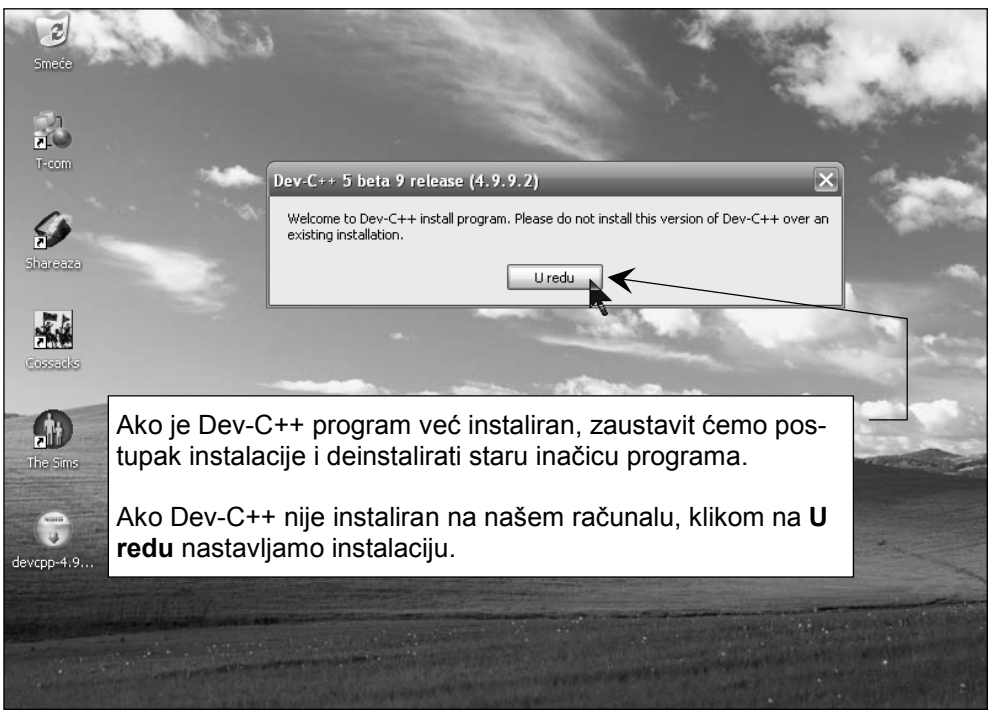


Na CD-u priloženom uz knjigu nalazi se program Dev-C++.

Ovaj izbornik trebao bi se pokrenuti nakon umetanja CD-a u računalo. Ako se to ne dogodi, pokrenite ga sami dvostrukim klikom na program meni.exe koji se nalazi na CD-ju.

Instalaciju pokrećemo klikom na **Program**.

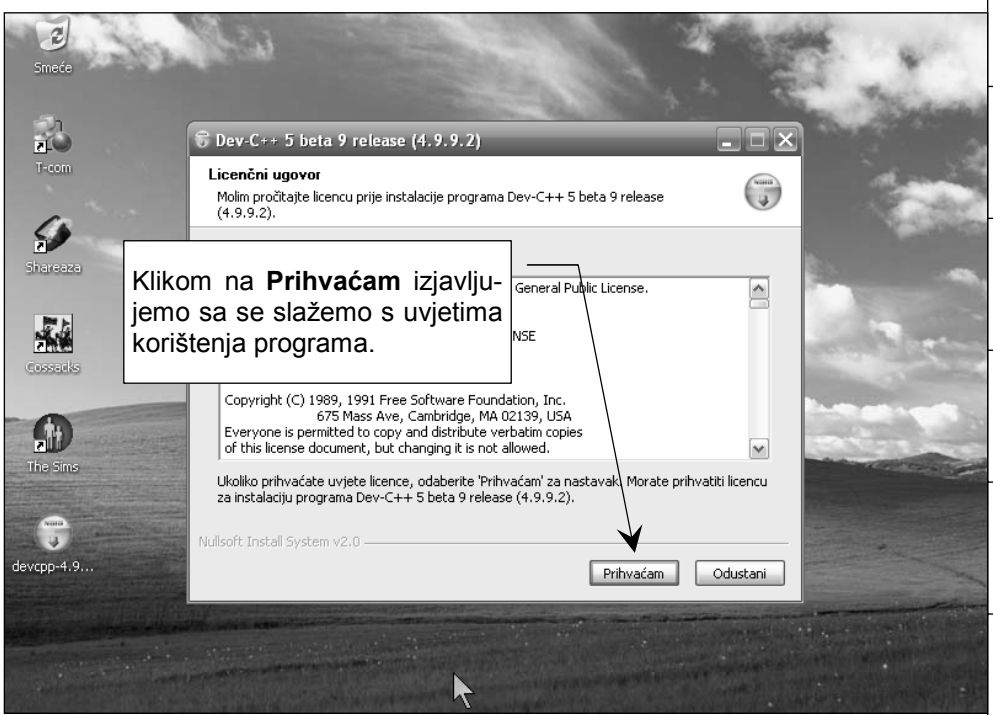
Nakon klika na **Program** započet će instaliranje Dev-C++ programa. Program će nas prvo upozoriti da ne smijemo instalirati program, ako je program već instaliran. Ako zbog nekog razloga instalaciju želimo ponoviti, moramo deinstalirati stari program, a tek nakon toga možemo ga ponovo instalirati.

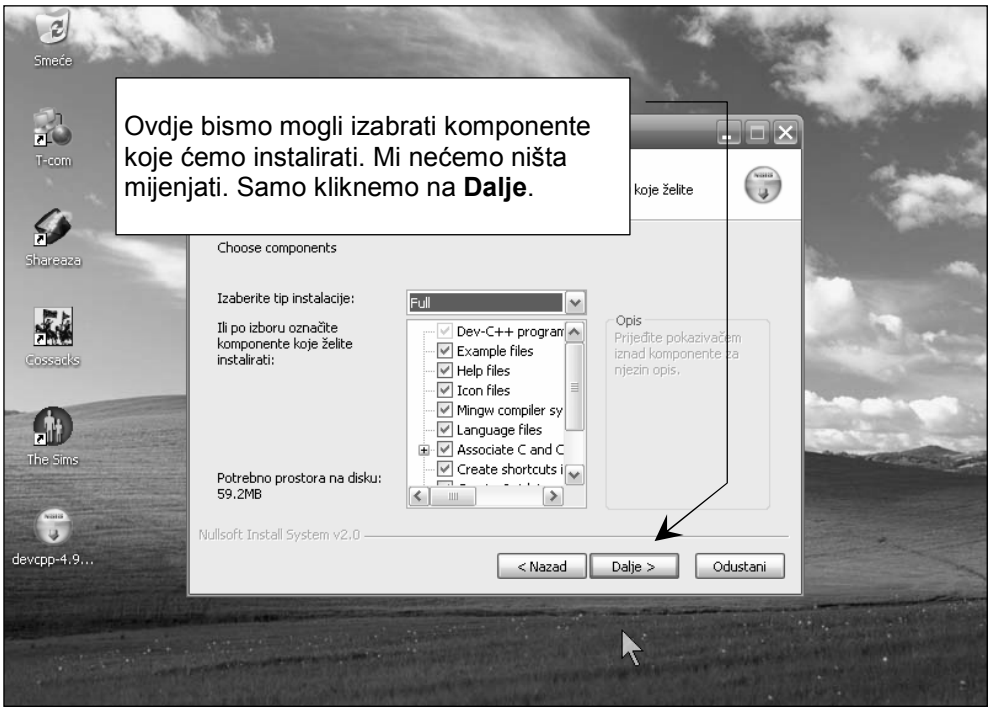


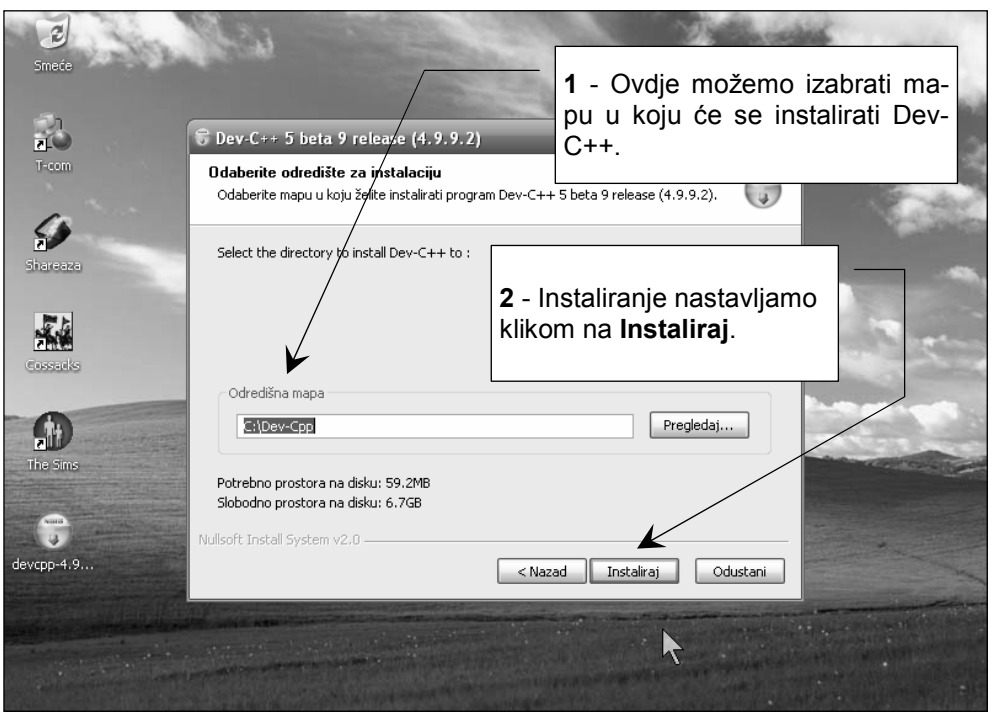
Ako je Dev-C++ program već instaliran, zaustavit ćemo postupak instalacije i deinstalirati staru inačicu programa.

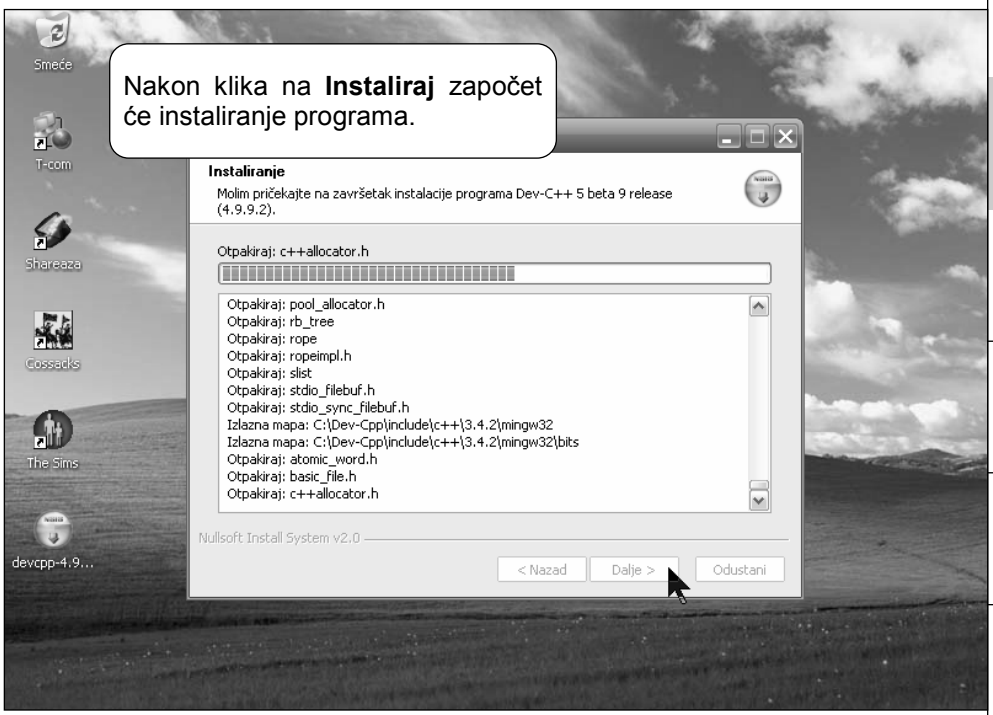
Ako Dev-C++ nije instaliran na našem računalu, klikom na **U redu** nastavljamo instalaciju.

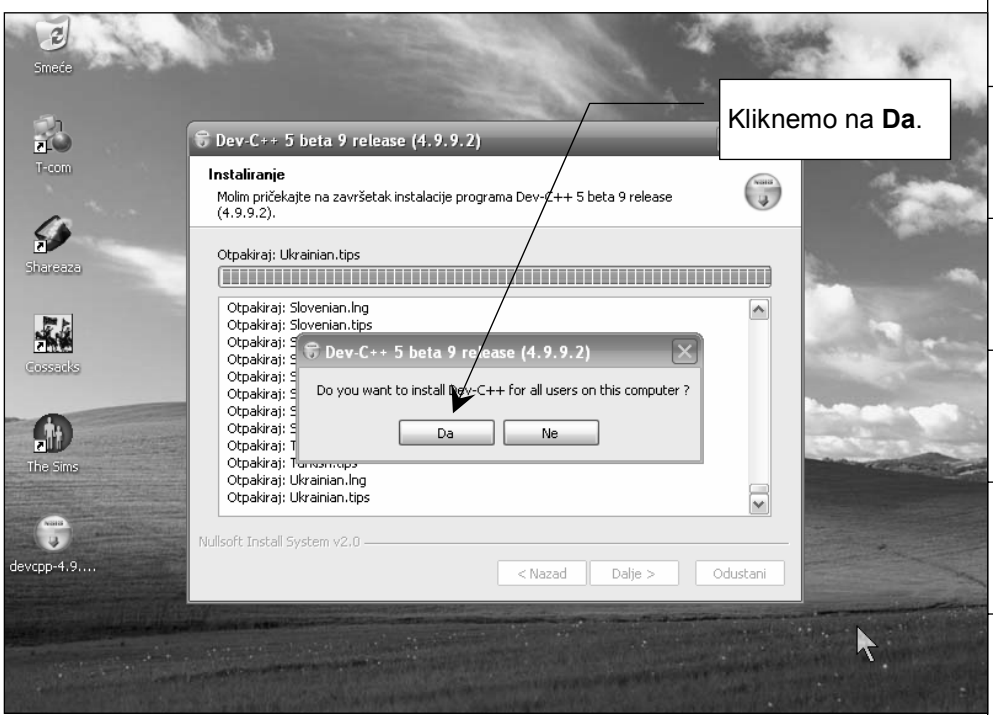

Smeće
T-com
Shareaza
Cossacks
The Sims
devcpp-4.9...
1 - Izaberemo jezik.
Installer Language
Please select a language.
Hrvatski
OK
Cancel
2 - Klikom na OK nastavljamo instalaciju.

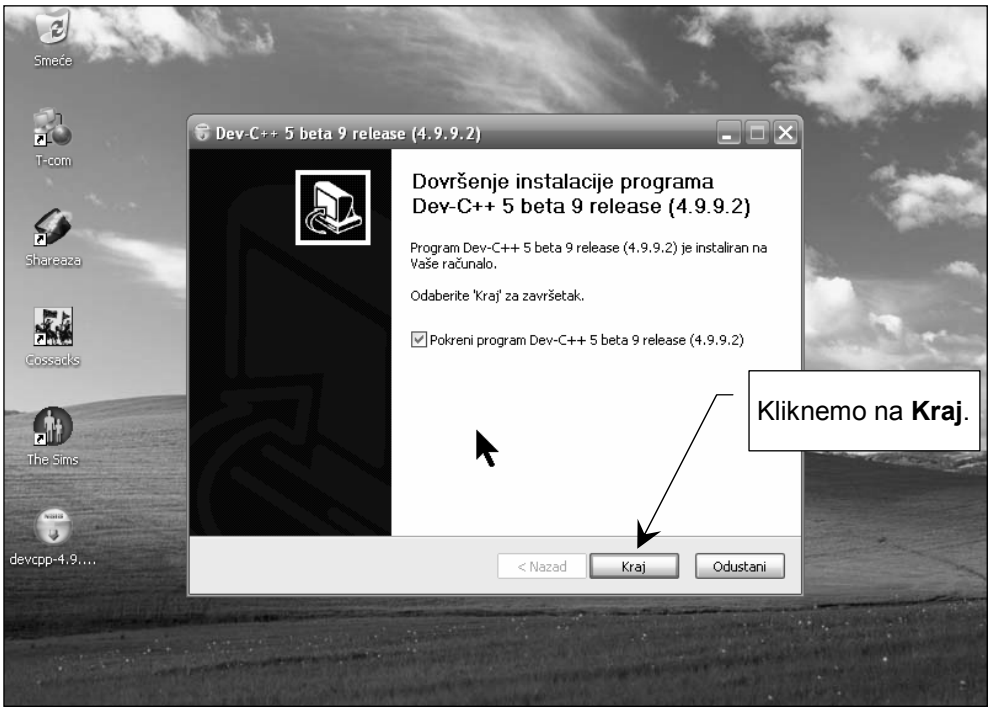

Smeće
T-com
Shareaza
Cossacks
The Sims
devcpp-4.9...
Dev-C++ 5 beta 9 release (4.9.9.2)
Licenčni ugovor
Molim pročitajte licencu prije instalacije programa Dev-C++ 5 beta 9 release (4.9.9.2).
Klikom na Prihvaćam izjavljujemo sa se slažemo s uvjetima korištenja programa.
General Public License.
NSE
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
675 Mass Ave, Cambridge, MA 02139, USA
Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.
Ukoliko prihvaćate uvjete licence, odaberite 'Prihvaćam' za nastavak. Morate prihvatiti licencu za instalaciju programa Dev-C++ 5 beta 9 release (4.9.9.2).
Nullsoft Install System v2.0
Prihvaćam
Odustani

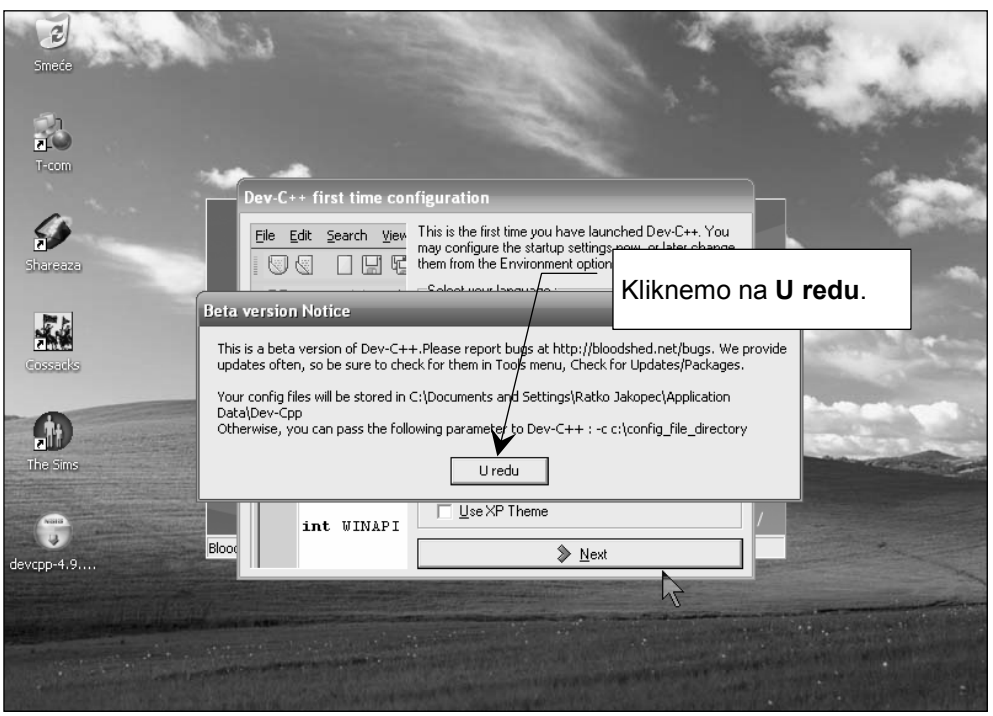

Ovdje bismo mogli izabrati komponente koje ćemo instalirati. Mi nećemo ništa mijenjati. Samo kliknemo na **Dalje**.
Choose components
Izaberite tip instalacije:
Full
Ili po izboru označite komponente koje želite instalirati:
Dev-C++ program
Example files
Help files
Icon files
Mingw compiler sy
Language files
Associate C and C
Create shortcuts
Opis
Prijeđite pokazivačem iznad komponente za njezin opis.
Potrebno prostora na disku: 59.2MB
Nullsoft Install System v2.0
< Nazad
Dalje >
Odustani

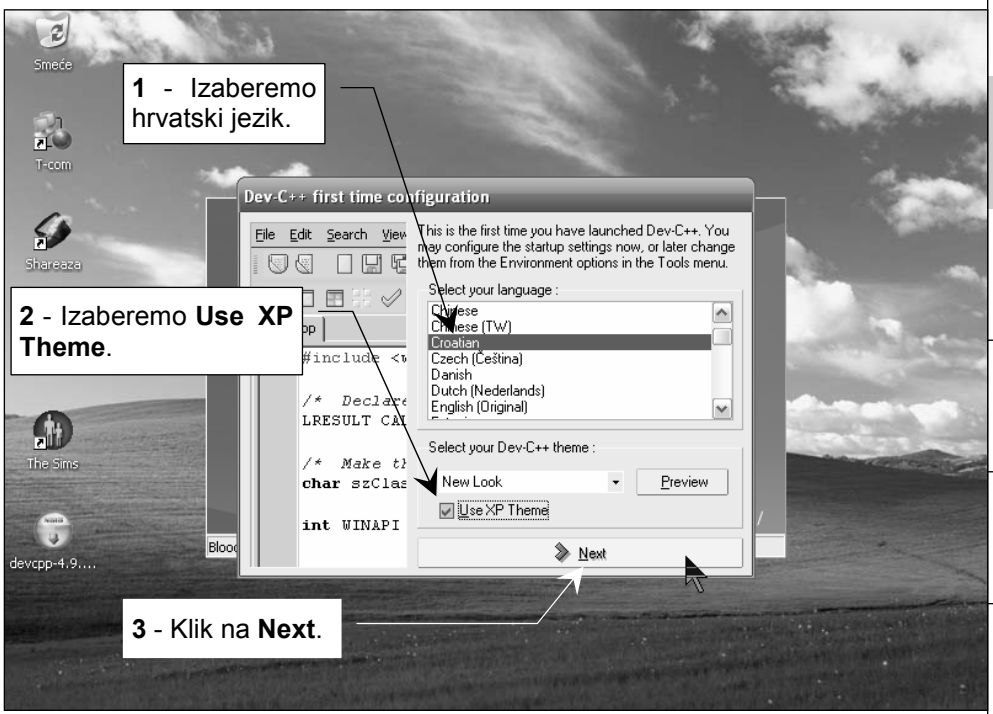

1 - Ovdje možemo izabrati mapu u koju će se instalirati Dev-C++.
Dev-C++ 5 beta 9 release (4.9.9.2)
Odaberite odredište za instalaciju
Odaberite mapu u koju želite instalirati program Dev-C++ 5 beta 9 release (4.9.9.2).
Select the directory to install Dev-C++ to :
2 - Instaliranje nastavljamo klikom na **Instaliraj**.
Odredišna mapa
C:\Dev-Cpp
Pregledaj...
Potrebno prostora na disku: 59.2MB
Slobodno prostora na disku: 6.7GB
Nullsoft Install System v2.0
< Nazad
Instaliraj
Odustani

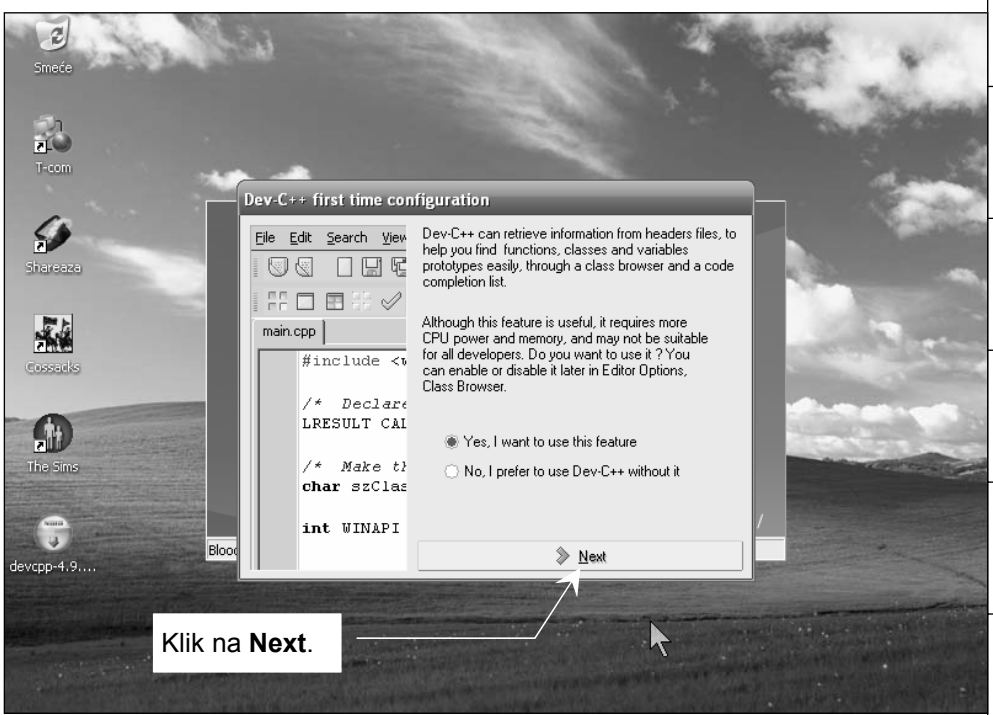

Nakon klika na **Instaliraj** započet će instaliranje programa.
Instaliranje
Molim pričekajte na završetak instalacije programa Dev-C++ 5 beta 9 release (4.9.9.2).
Otpakiraj: c++allocator.h
Otpakiraj: pool_allocator.h
Otpakiraj: rb_tree
Otpakiraj: rope
Otpakiraj: ropeimpl.h
Otpakiraj: slist
Otpakiraj: stdio_filebuf.h
Otpakiraj: stdio_sync_filebuf.h
Izlazna mapa: C:\Dev-Cpp\include\c++\3.4.2\mingw32
Izlazna mapa: C:\Dev-Cpp\include\c++\3.4.2\mingw32\bits
Otpakiraj: atomic_word.h
Otpakiraj: basic_file.h
Otpakiraj: c++allocator.h
Nullsoft Install System v2.0
< Nazad
Dalje >
Odustani

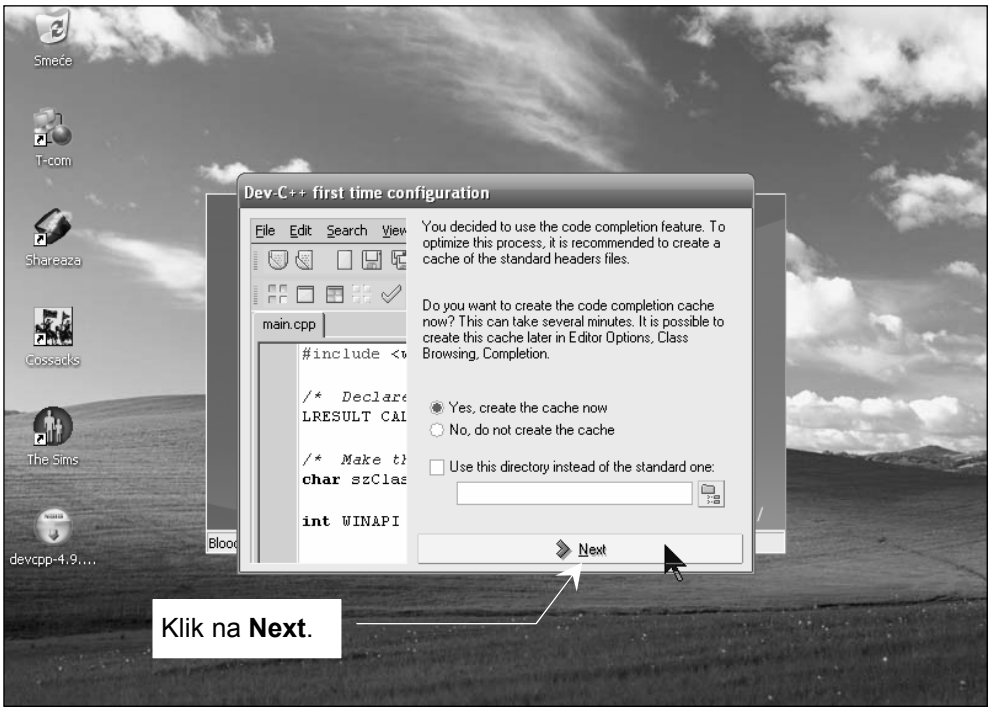

Kliknemo na **Da**.
Dev-C++ 5 beta 9 release (4.9.9.2)
Instaliranje
Molim pričekajte na završetak instalacije programa Dev-C++ 5 beta 9 release (4.9.9.2).
Otpakiraj: Ukrainian.tips
Otpakiraj: Slovenian.lng
Otpakiraj: Slovenian.tips
Dev-C++ 5 beta 9 release (4.9.9.2)
Do you want to install Dev-C++ for all users on this computer ?
Da
Ne
Otpakiraj: Ukrainian.lng
Otpakiraj: Ukrainian.tips
Nullsoft Install System v2.0
< Nazad
Dalje >
Odustani

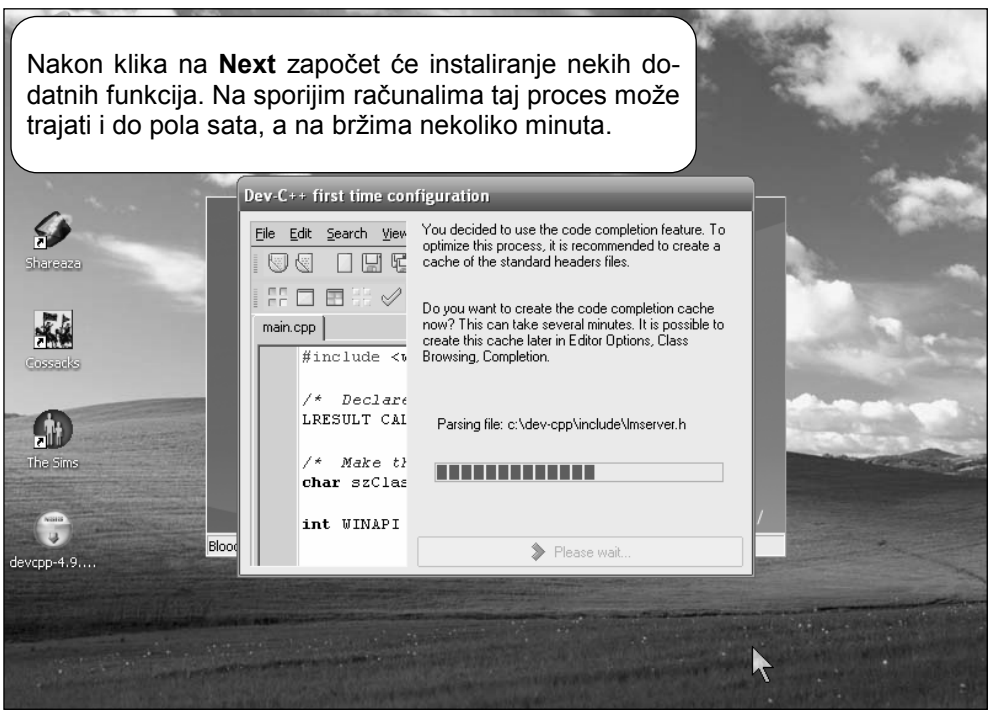

Smeće
T-com
Shareaza
Cossacks
The Sims
devcpp-4.9....
Dev-C++ 5 beta 9 release (4.9.9.2)
Dovršenje instalacije programa
Dev-C++ 5 beta 9 release (4.9.9.2)
Program Dev-C++ 5 beta 9 release (4.9.9.2) je instaliran na Vaše računalo.
Odaberite 'Kraj' za završetak.
Pokreni program Dev-C++ 5 beta 9 release (4.9.9.2)
< Nazad
Kraj
Odustani
Kliknemo na **Kraj**.

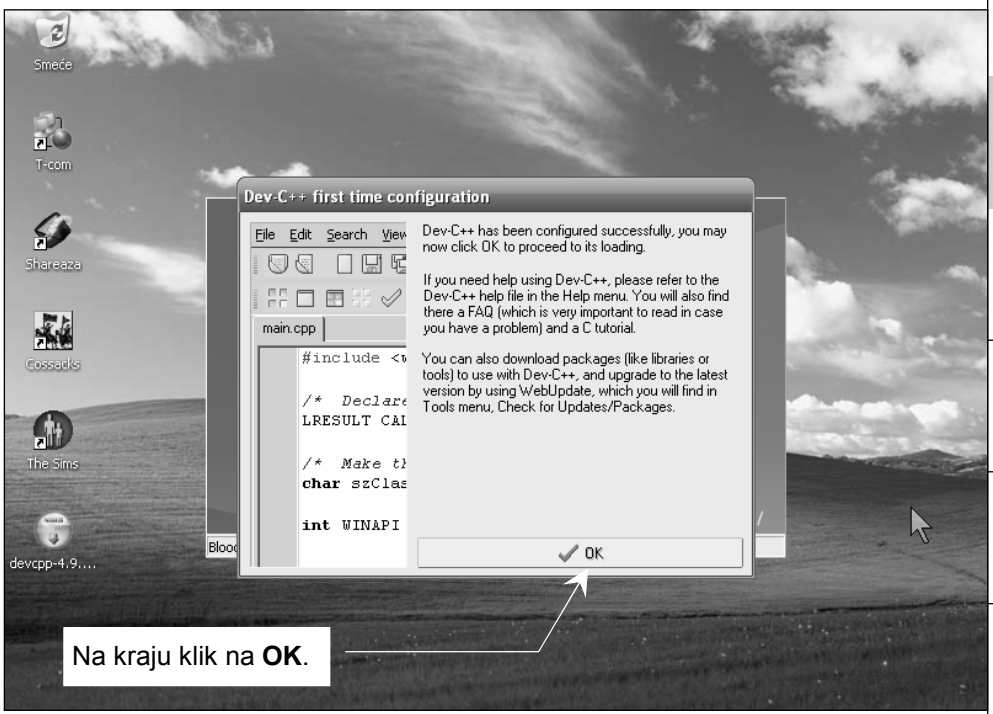

Smeće
T-com
Shareaza
Cossacks
The Sims
devcpp-4.9....
Dev-C++ first time configuration
File Edit Search View
This is the first time you have launched Dev-C++. You may configure the startup settings now, or later change them from the Environment option
Beta version Notice
This is a beta version of Dev-C++.Please report bugs at http://bloodshed.net/bugs. We provide updates often, so be sure to check for them in Tools menu, Check for Updates/Packages.
Your config files will be stored in C:\Documents and Settings\Ratko Jakopec\Application Data\Dev-Cpp
Otherwise, you can pass the following parameter to Dev-C++ : -c c:\config_file_directory
U redu
Use XP Theme
int WINAPI
Next
Kliknemo na **U redu**.

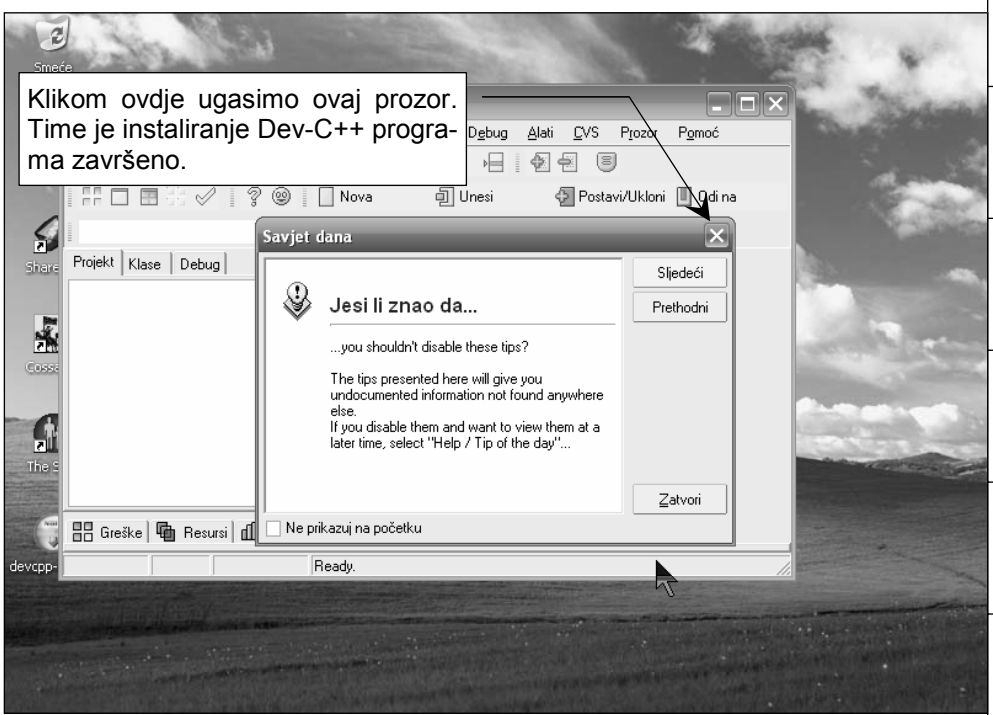

1 - Izaberemo hrvatski jezik.
2 - Izaberemo Use XP Theme.
3 - Klik na Next.
Dev-C++ first time configuration
File Edit Search View
This is the first time you have launched Dev-C++. You may configure the startup settings now, or later change them from the Environment options in the Tools menu.
Select your language :
Chinese
Chinese (TW)
Croatian
Czech (Čeština)
Danish
Dutch (Nederlands)
English (Original)
Select your Dev-C++ theme :
New Look
Preview
Use XP Theme
Next
Smeće
T-com
Shareaza
The Sims
devcpp-4.9....


Klik na Next.
Dev-C++ first time configuration
File Edit Search View
Dev-C++ can retrieve information from headers files, to help you find functions, classes and variables prototypes easily, through a class browser and a code completion list.
Although this feature is useful, it requires more CPU power and memory, and may not be suitable for all developers. Do you want to use it ? You can enable or disable it later in Editor Options, Class Browser.
Yes, I want to use this feature
No, I prefer to use Dev-C++ without it
Next
Smeće
T-com
Shareaza
Cossacks
The Sims
devcpp-4.9....

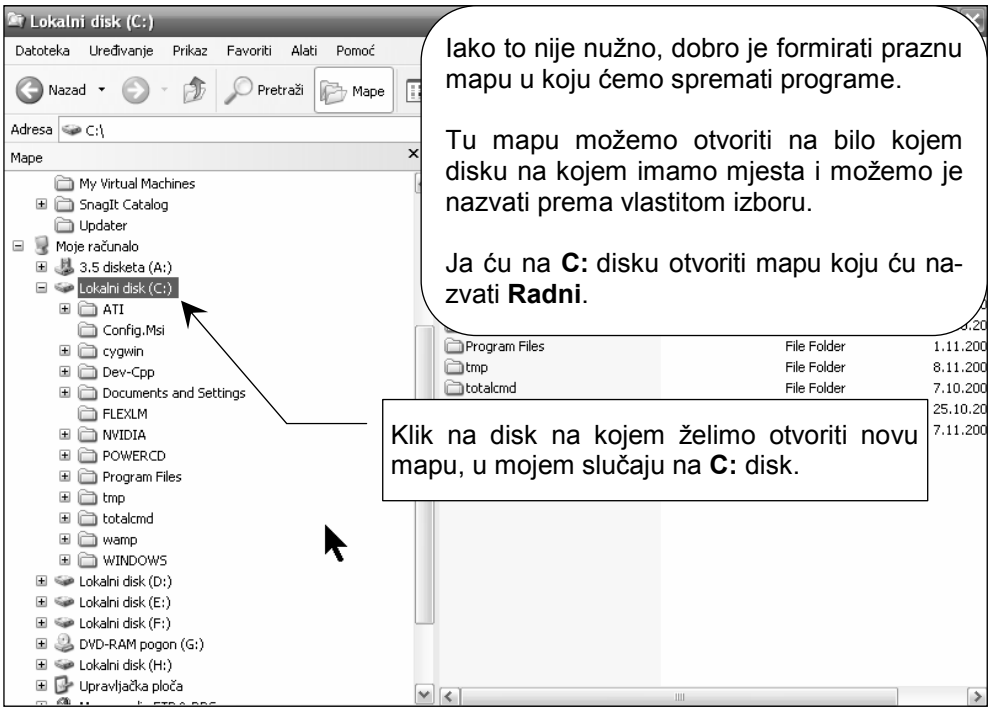

Smeće
T-com
Shareaza
Cossacks
The Sims
devcpp-4.9....
Dev-C++ first time configuration
File Edit Search View
main.cpp
#include <w
/* Declare
LRESULT CAL
/* Make th
char szClas
int WINAPI
You decided to use the code completion feature. To optimize this process, it is recommended to create a cache of the standard headers files.
Do you want to create the code completion cache now? This can take several minutes. It is possible to create this cache later in Editor Options, Class Browsing, Completion.
Yes, create the cache now
No, do not create the cache
Use this directory instead of the standard one:
Next
Klik na **Next**.

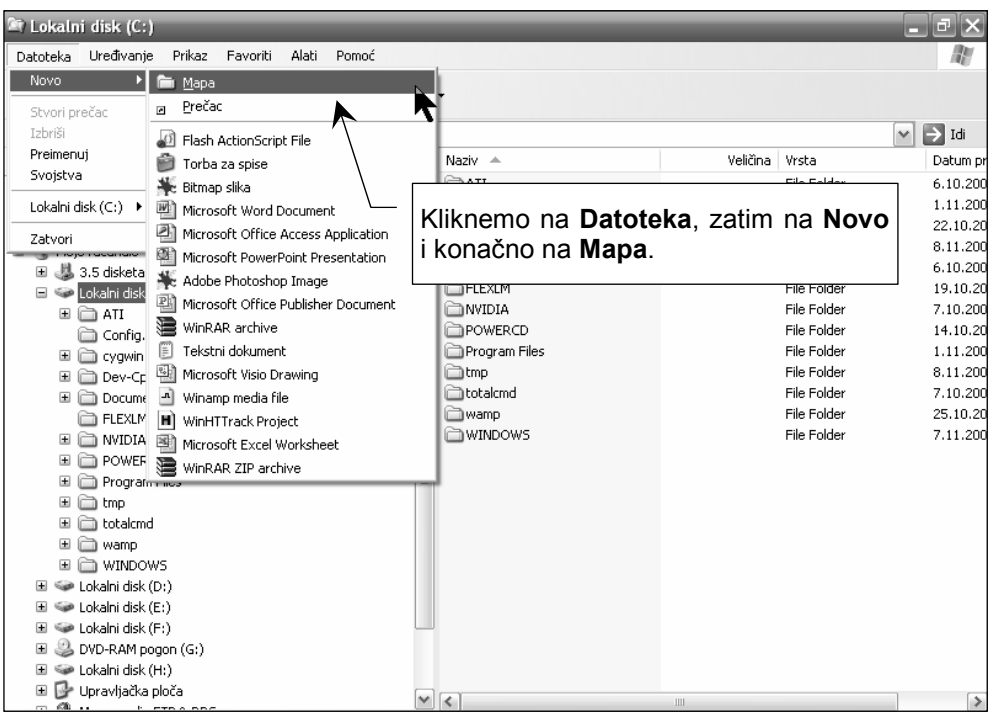

Nakon klika na **Next** započet će instaliranje nekih dodatnih funkcija. Na sporijim računalima taj proces može trajati i do pola sata, a na bržima nekoliko minuta.
Shareaza
Cossacks
The Sims
devcpp-4.9....
Dev-C++ first time configuration
File Edit Search View
main.cpp
#include <w
/* Declare
LRESULT CAL
/* Make th
char szClas
int WINAPI
You decided to use the code completion feature. To optimize this process, it is recommended to create a cache of the standard headers files.
Do you want to create the code completion cache now? This can take several minutes. It is possible to create this cache later in Editor Options, Class Browsing, Completion.
Parsing file: c:\dev-cpp\include\lmserver.h
Please wait...

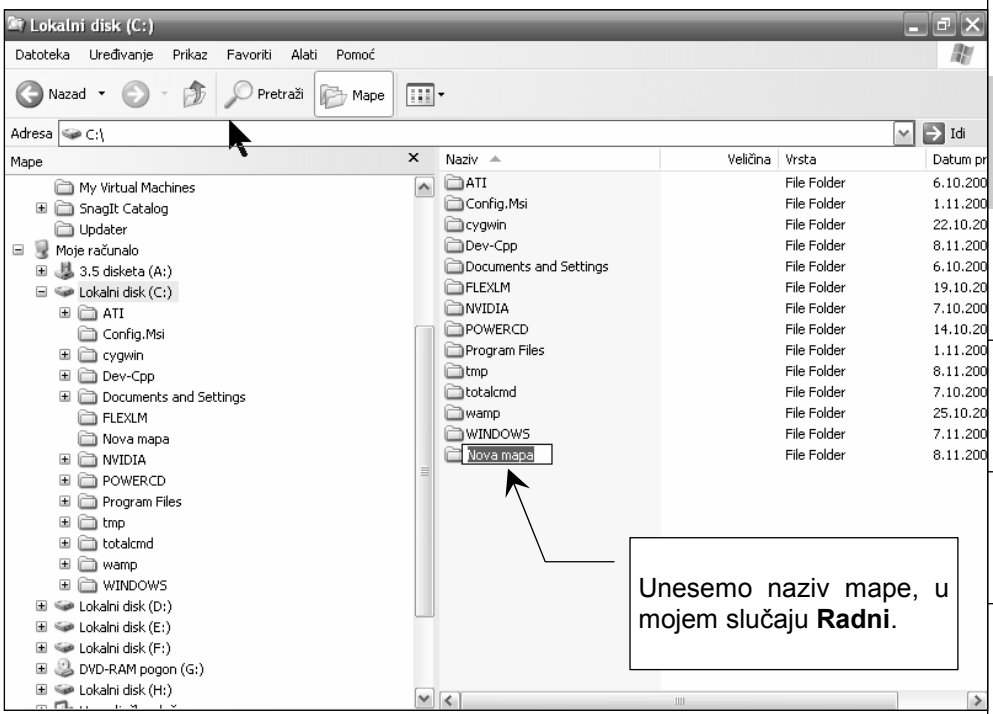

Dev-C++ first time configuration
File Edit Search View
Dev-C++ has been configured successfully, you may now click OK to proceed to its loading.
If you need help using Dev-C++, please refer to the Dev-C++ help file in the Help menu. You will also find there a FAQ (which is very important to read in case you have a problem) and a C tutorial.
You can also download packages (like libraries or tools) to use with Dev-C++, and upgrade to the latest version by using WebUpdate, which you will find in Tools menu, Check for Updates/Packages.
main.cpp
OK
Na kraju klik na **OK**.

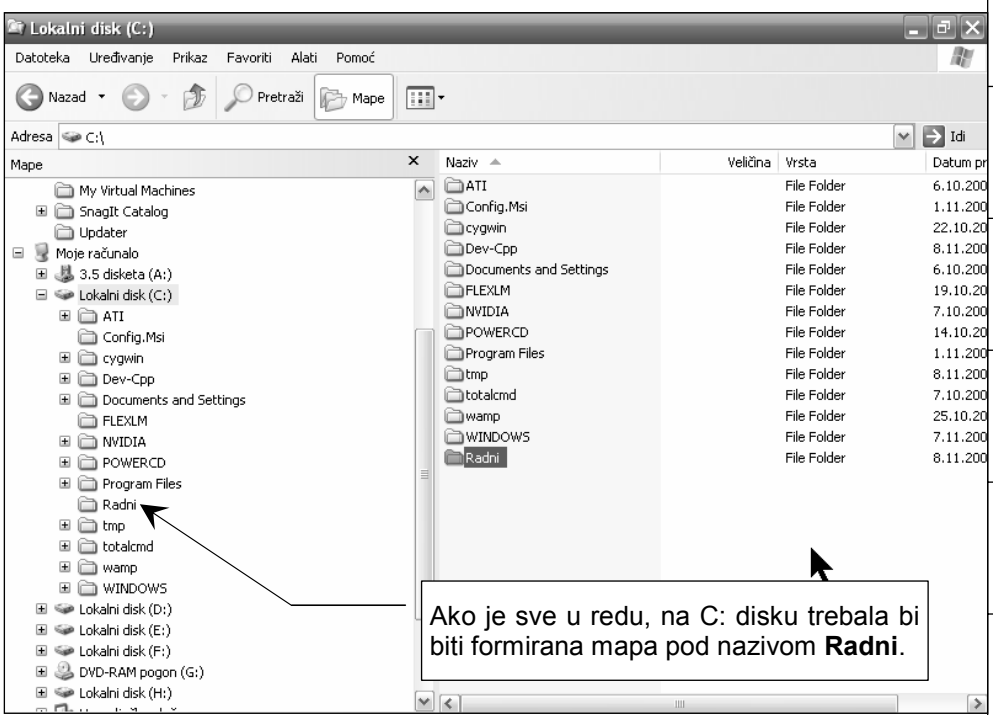

Klikom ovdje ugasimo ovaj prozor. Time je instaliranje Dev-C++ programa završeno.
Savjet dana
Jesi li znao da...
...you shouldn't disable these tips?
The tips presented here will give you undocumented information not found anywhere else.
If you disable them and want to view them at a later time, select "Help / Tip of the day"...
Sljedeći
Prethodni
Zatvori
Ne prikazuj na početku
Projekt
Klase
Debug
Greške
Resursi
Ready.

Formiranje radne mape

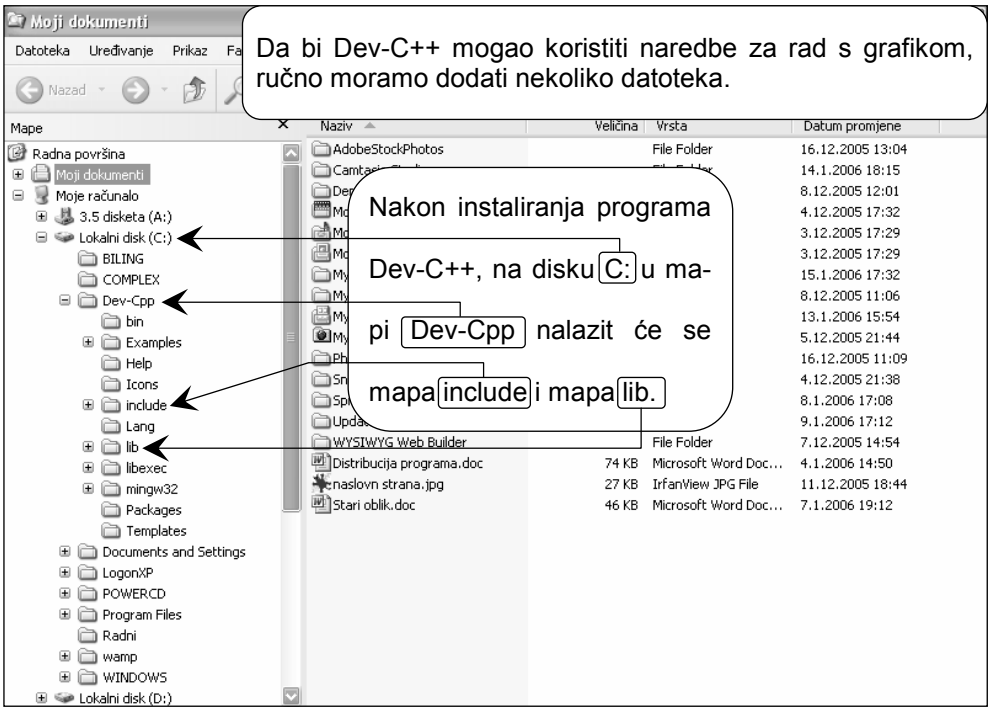


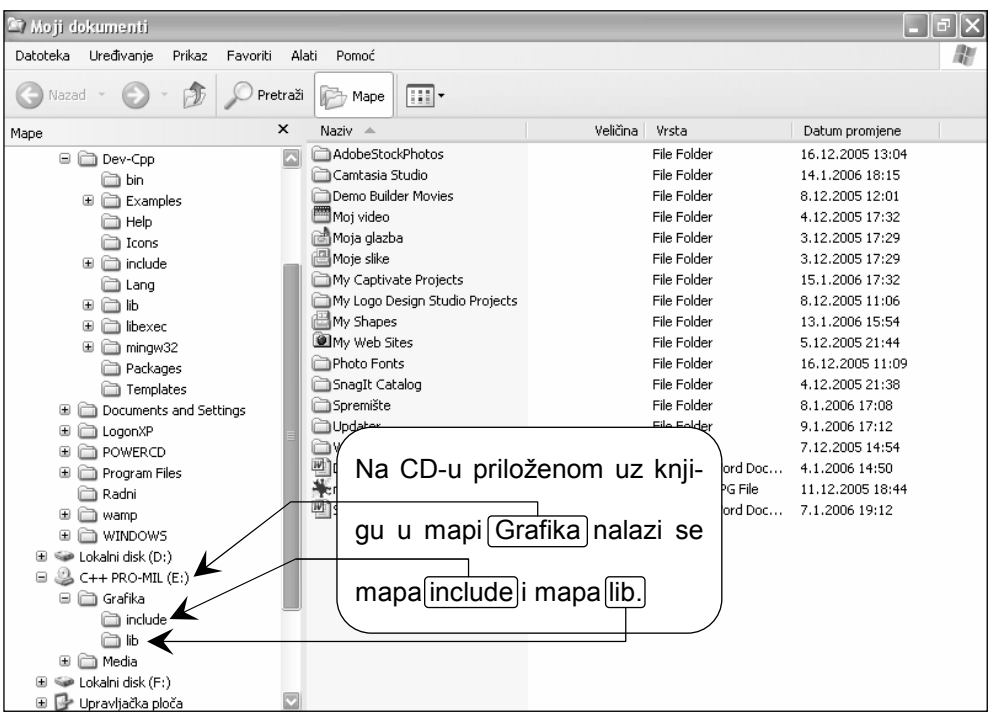


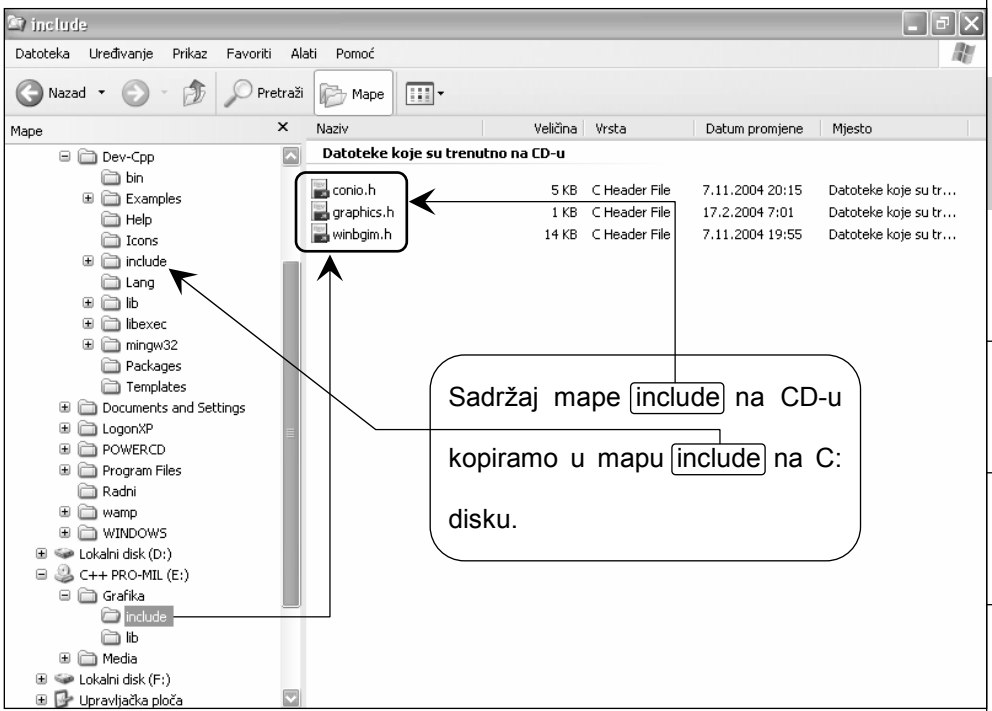

Lokalni disk (C:)
Datoteka Uređivanje Prikaz Favoriti Alati Pomoć
Nazad Pretraži Mape
Adresa C:\
Idi
Mape
My Virtual Machines
SnagIt Catalog
Updater
Moje računalo
3.5 disketa (A:)
Lokalni disk (C:)
ATI
Config.Msi
cygwin
Dev-Cpp
Documents and Settings
FLEXLM
Nova mapa
NVIDIA
POWERCD
Program Files
tmp
totalcmd
wamp
WINDOWS
Lokalni disk (D:)
Lokalni disk (E:)
Lokalni disk (F:)
DVD-RAM pogon (G:)
Lokalni disk (H:)
Naziv
Veličina
Vrsta
Datum pr
File Folder
Unesemo naziv mape, u mojem slučaju **Radni**.

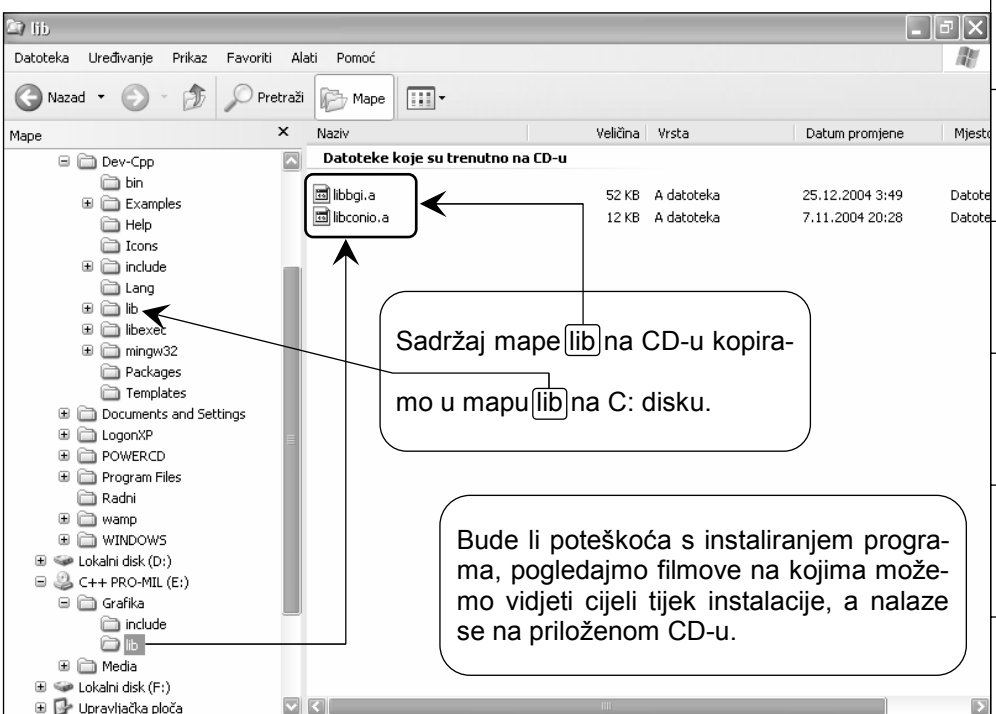

Lokalni disk (C:)
Datoteka Uređivanje Prikaz Favoriti Alati Pomoć
Nazad Pretraži Mape
Adresa C:\
Idi
Mape
My Virtual Machines
SnagIt Catalog
Updater
Moje računalo
3.5 disketa (A:)
Lokalni disk (C:)
ATI
Config.Msi
cygwin
Dev-Cpp
Documents and Settings
FLEXLM
NVIDIA
POWERCD
Program Files
Radni
tmp
totalcmd
wamp
WINDOWS
Lokalni disk (D:)
Lokalni disk (E:)
Lokalni disk (F:)
DVD-RAM pogon (G:)
Lokalni disk (H:)
Naziv
Veličina
Vrsta
Datum pr
File Folder
Ako je sve u redu, na C: disku trebala bi biti formirana mapa pod nazivom **Radni**.

Instalacija grafike



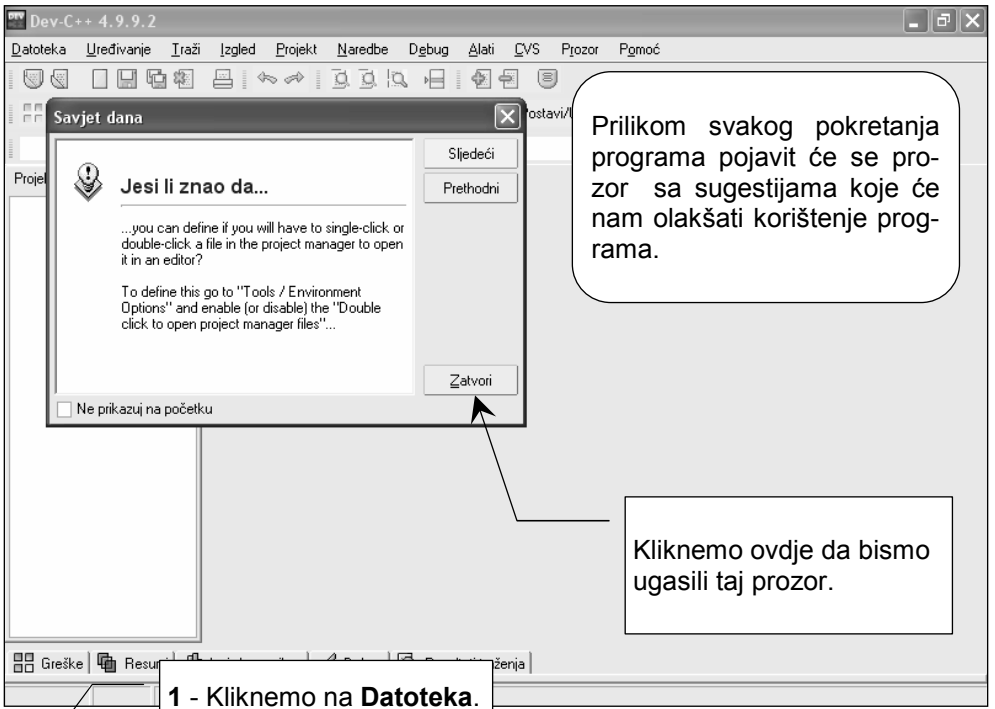


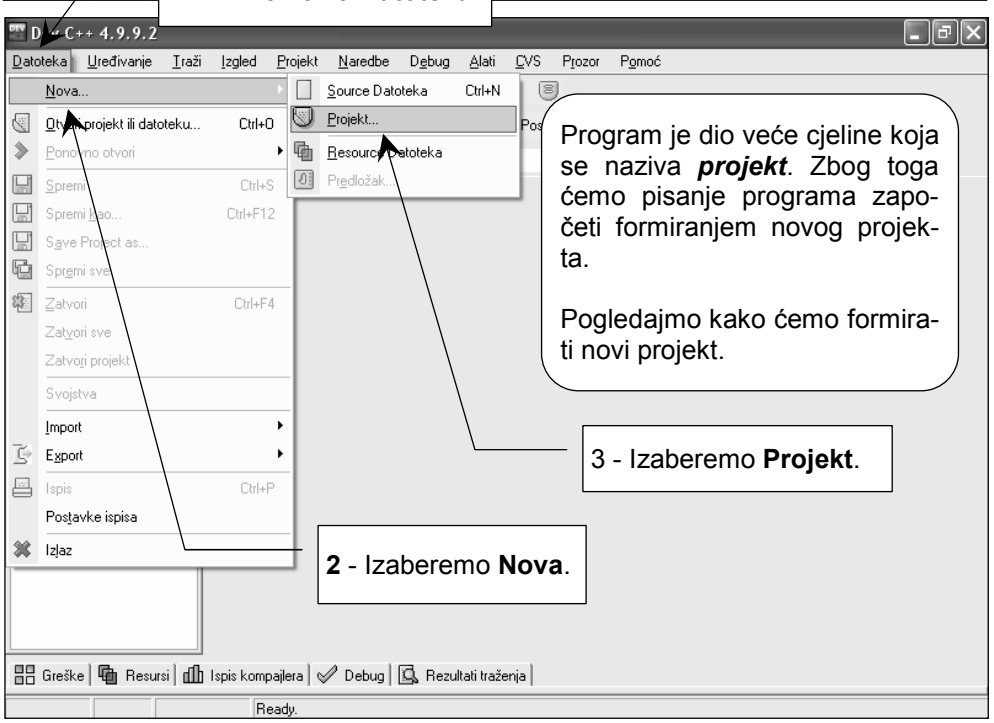

include
Datoteka Uređivanje Prikaz Favoriti Alati Pomoć
Nazad Pretraži Mape
Mape
Naziv Veličina Vrsta Datum promjene Mjesto
Datoteke koje su trenutno na CD-u
conio.h 5 KB C Header File 7.11.2004 20:15 Datoteke koje su tr...
graphics.h 1 KB C Header File 17.2.2004 7:01 Datoteke koje su tr...
winbgim.h 14 KB C Header File 7.11.2004 19:55 Datoteke koje su tr...
Dev-Cpp
bin
Examples
Help
Icons
include
Lang
lib
libexec
mingw32
Packages
Templates
Documents and Settings
LogonXP
POWERCD
Program Files
Radni
wamp
WINDOWS
Lokalni disk (D:)
C++ PRO-MIL (E:)
Grafika
include
lib
Media
Lokalni disk (F:)
Upravljačka ploča
Sadržaj mape include na CD-u kopiramo u mapu include na C: disku.


lib
Datoteka Uređivanje Prikaz Favoriti Alati Pomoć
Nazad Pretraži Mape
Mape
Naziv Veličina Vrsta Datum promjene
Datoteke koje su trenutno na CD-u
libbgi.a 52 KB A datoteka 25.12.2004 3:49
libconio.a 12 KB A datoteka 7.11.2004 20:28
Dev-Cpp
bin
Examples
Help
Icons
include
Lang
lib
libexec
mingw32
Packages
Templates
Documents and Settings
LogonXP
POWERCD
Program Files
Radni
wamp
WINDOWS
Lokalni disk (D:)
C++ PRO-MIL (E:)
Grafika
include
lib
Media
Lokalni disk (F:)
Upravljačka ploča
Sadržaj mape lib na CD-u kopiramo u mapu lib na C: disku.
Bude li poteškoća s instaliranjem programa, pogledajmo filmove na kojima možemo vidjeti cijeli tijek instalacije, a nalaze se na priloženom CD-u.

Naš prvi program

Formiranje projekta



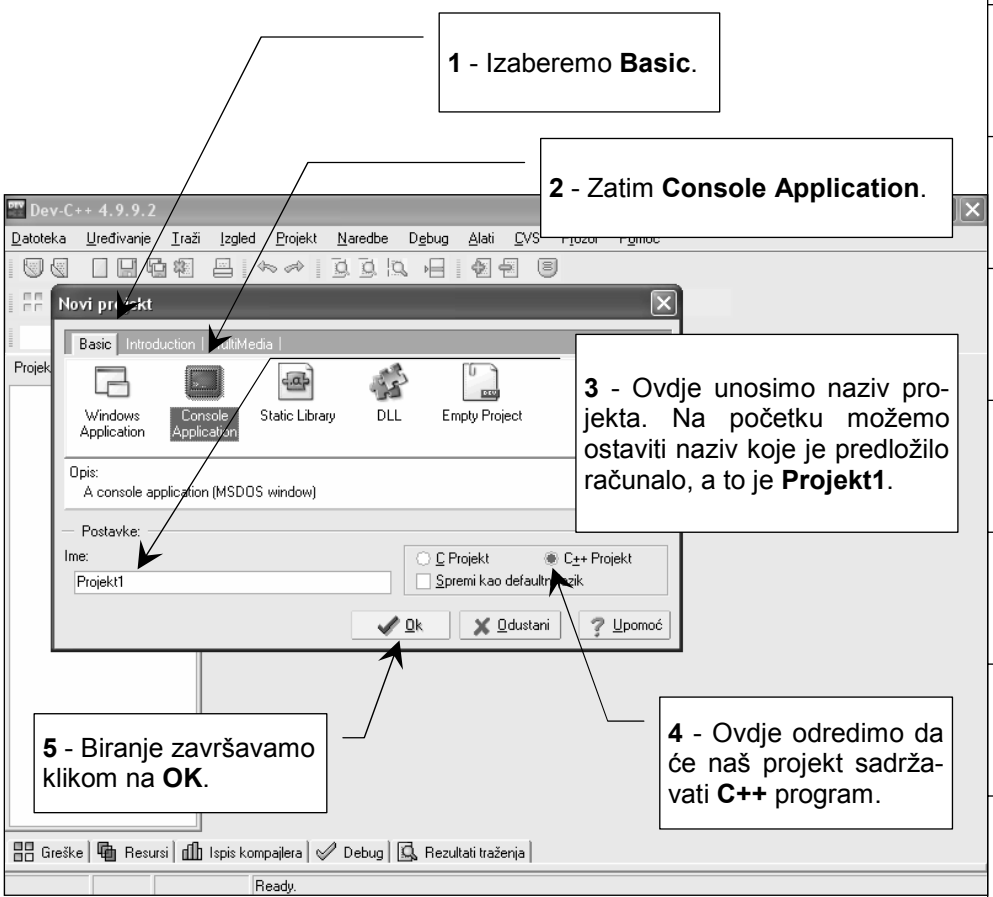


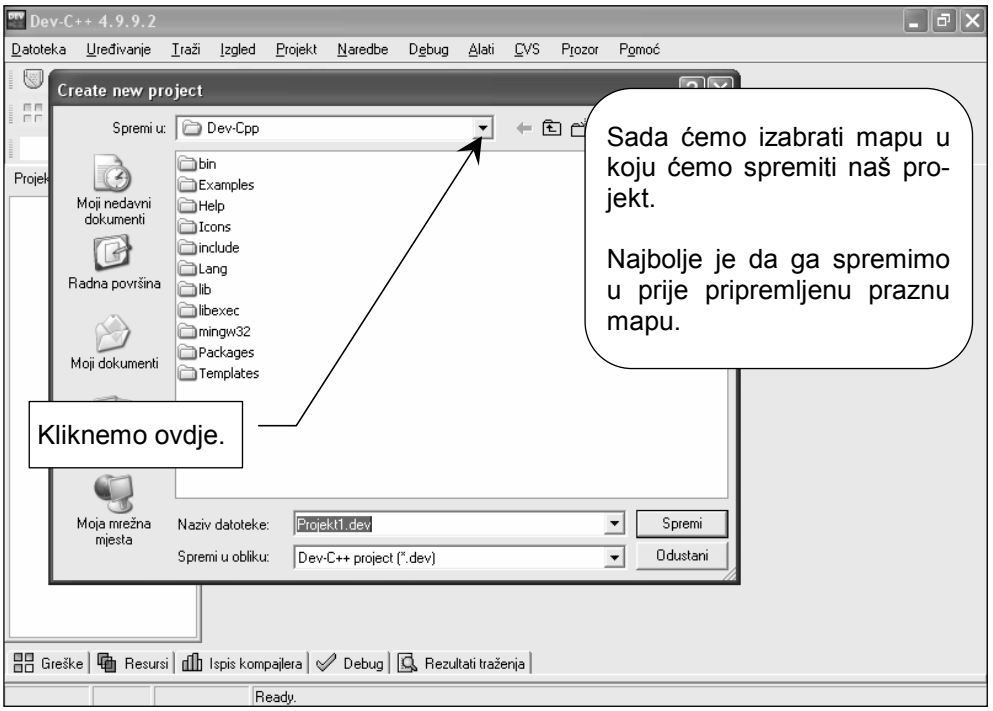
Otvorit će se novi prozor koji se naziva **Novi projekt** i u kojem ćemo izabrati tip projekta. Najvažniji tipovi su ***Windows Application*** i ***Console Application***.

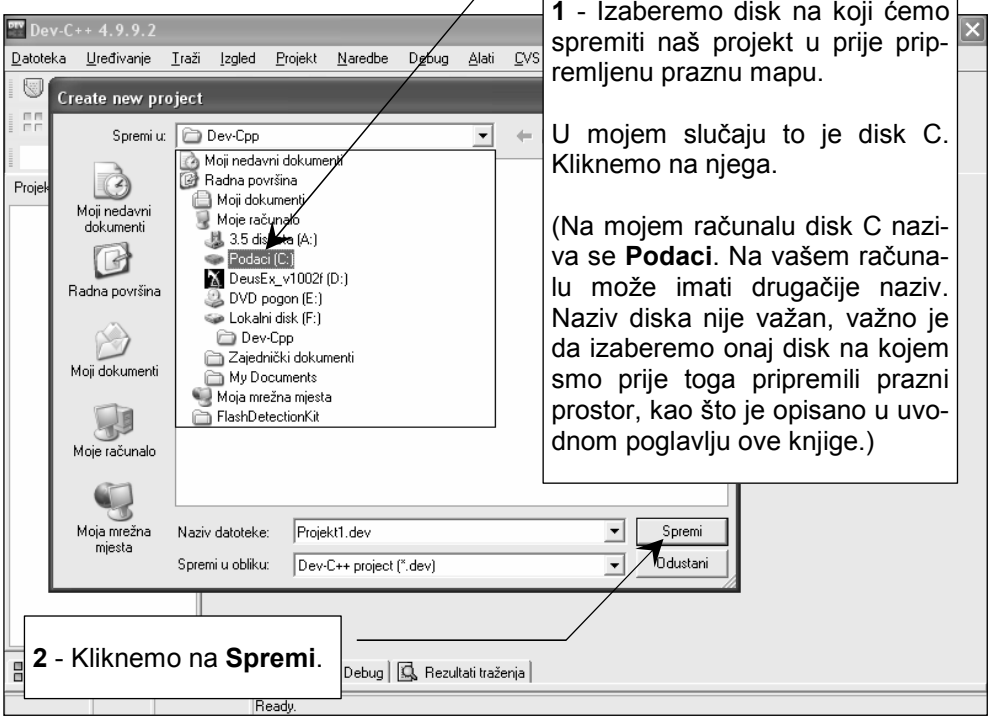
U čemu je razlika?

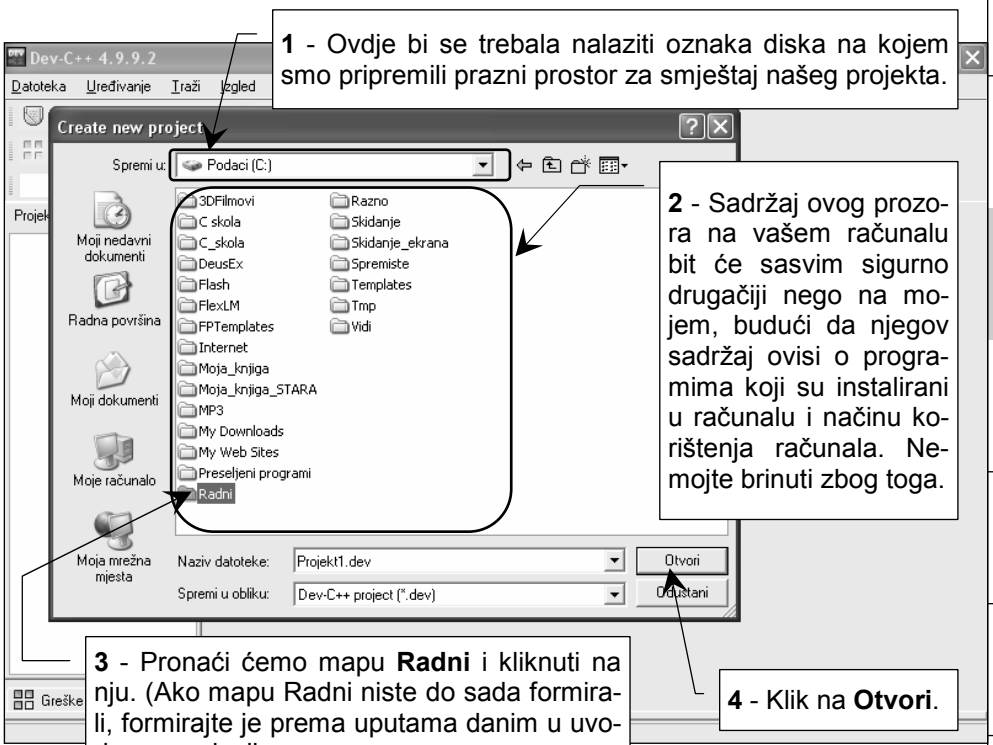
Windows Application – izvodi se unutar klasičnog windows prozora. Prednost ovog načina pisanja jest što ćemo dobiti program koji će izgledati kao svaki drugi windows program. Nedostatak mu je što je ovaj način pisanja programa nešto složeniji, pa je manje prikladan za početnike.

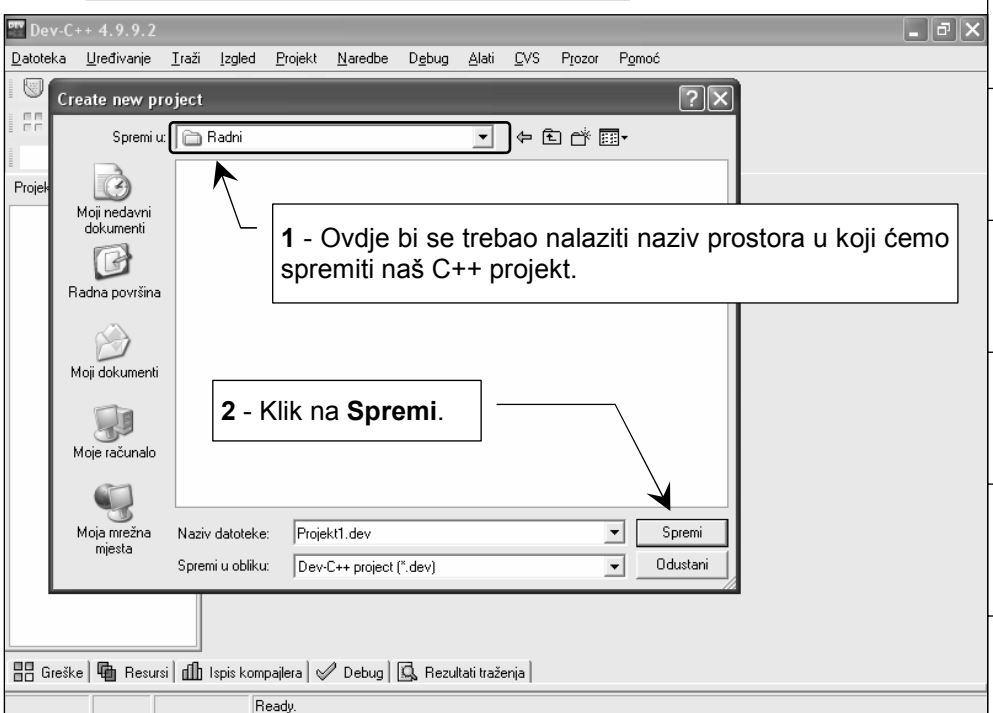
Console Application – ne izvodi se unutar klasičnog windows prozora, već unutar pojednostavljene verzije s crnom pozadinom. Nedostatak ovog načina pisanja programa jest što je program veoma ružan i što taj oblik prozora ima manje mogućnosti od klasičnog prozora. Jedina prednost mu je što je jednostavan, pa je prikladniji za početnike.

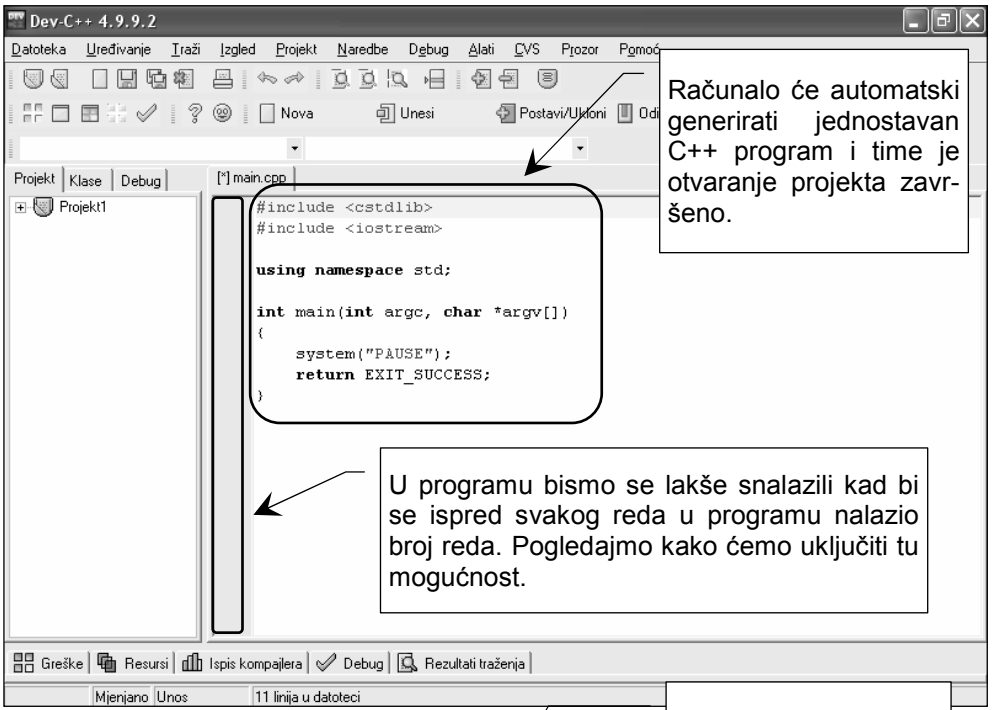



Dev-C++ 4.9.9.2
Datoteka Uređivanje Traži Izgled Projekt Naredbe Debug Alati CVS Prozor Pomoć
Create new project
Spremi u: Dev-Cpp
bin
Examples
Help
Icons
include
Lang
lib
libexec
mingw32
Packages
Templates
Moji nedavni dokumenti
Radna površina
Moji dokumenti
Moja mrežna mjesta
Naziv datoteke: Projekt1.dev
Spremi u obliku: Dev-C++ project (*.dev)
Spremi
Odustani
Sada ćemo izabrati mapu u koju ćemo spremiti naš projekt.
Najbolje je da ga spremimo u prije pripremljenu praznu mapu.
Kliknemo ovdje.
Greške Resursi Ispis kompajlera Debug Rezultati traženja
Ready.

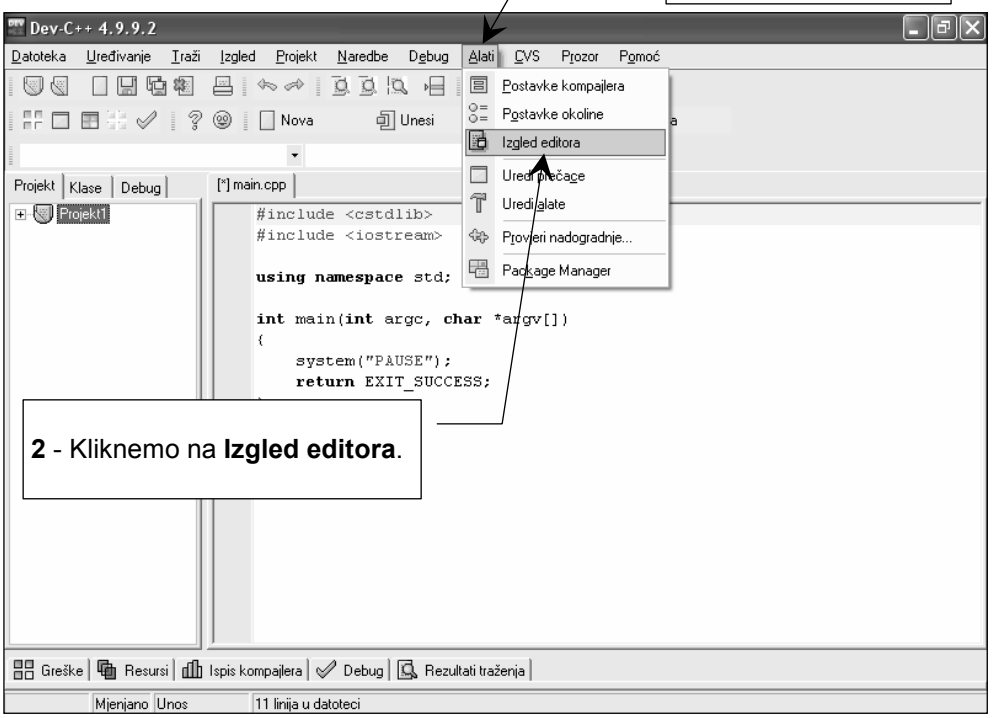

Dev-C++ 4.9.9.2
Create new project
Spremi u: Dev-Cpp
Moji nedavni dokumenti
Radna površina
Moji dokumenti
Moje računalo
3.5 disketa (A:)
Podaci (C:)
DeusEx_v1002f (D:)
DVD pogon (E:)
Lokalni disk (F:)
Dev-Cpp
Zajednički dokumenti
My Documents
Moja mrežna mjesta
FlashDetectionKit
Naziv datoteke: Projekt1.dev
Spremi u obliku: Dev-C++ project (*.dev)
Spremi
Odustani
1 - Izaberemo disk na koji ćemo spremiti naš projekt u prije pripremljenu praznu mapu.
U mojem slučaju to je disk C. Kliknemo na njega.
(Na mojem računalu disk C naziva se Podaci. Na vašem računalu može imati drugačije naziv. Naziv diska nije važan, važno je da izaberemo onaj disk na kojem smo prije toga pripremili prazni prostor, kao što je opisano u uvodnom poglavlju ove knjige.)
2 - Kliknemo na Spremi.
Debug Rezultati traženja
Ready.

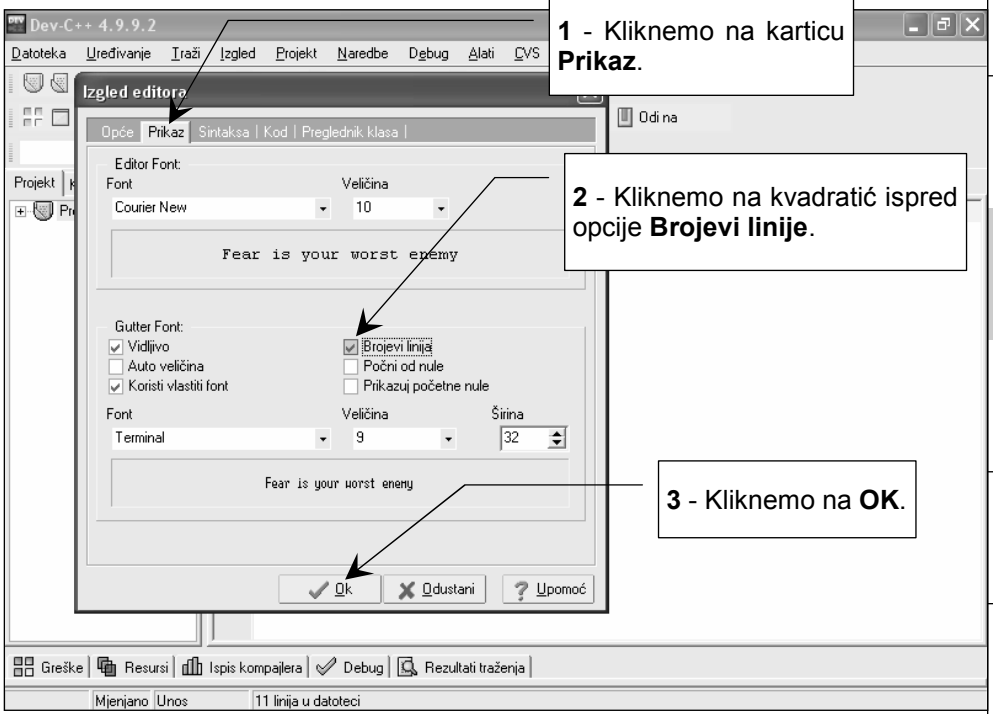

1 - Ovdje bi se trebala nalaziti oznaka diska na kojem smo pripremili prazni prostor za smještaj našeg projekta.
Dev-C++ 4.9.9.2
Create new project
Spremi u: Podaci (C:)
2 - Sadržaj ovog prozora na vašem računalu bit će sasvim sigurno drugačiji nego na mojem, budući da njegov sadržaj ovisi o programima koji su instalirani u računalu i načinu korištenja računala. Nemojte brinuti zbog toga.
Naziv datoteke: Projekt1.dev
Spremi u obliku: Dev-C++ project (*.dev)
Otvori
Odustani
3 - Pronaći ćemo mapu Radni i kliknuti na nju. (Ako mapu Radni niste do sada formirali, formirajte je prema uputama danim u uvodnom poglavlju.
4 - Klik na Otvori.

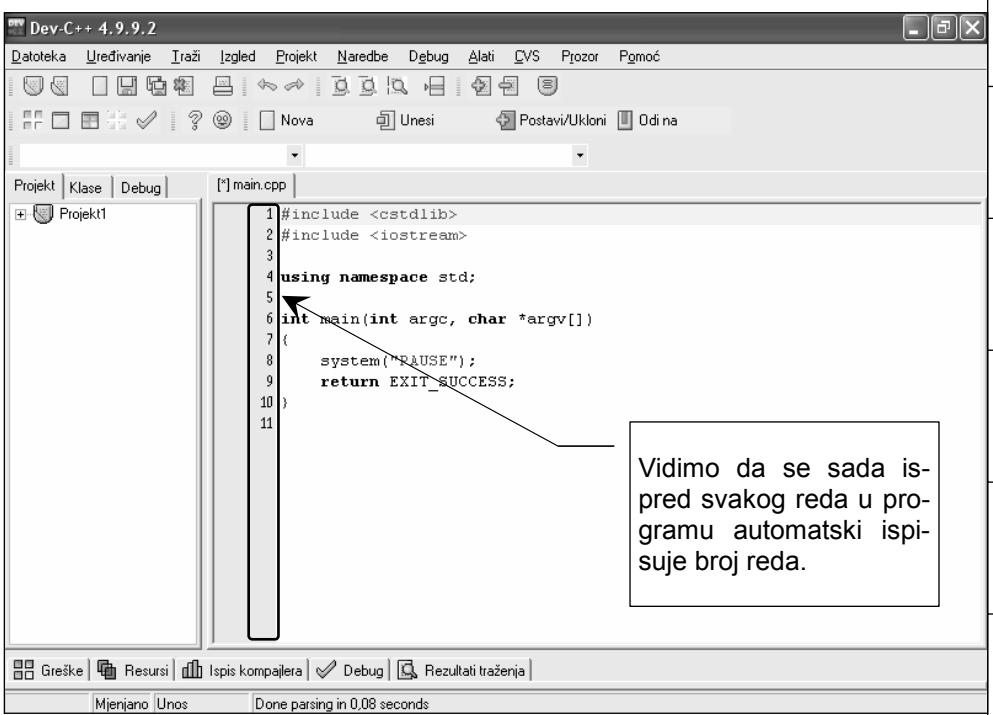

Dev-C++ 4.9.9.2
Create new project
Spremi u: Radni
1 - Ovdje bi se trebao nalaziti naziv prostora u koji ćemo spremiti naš C++ projekt.
2 - Klik na Spremi.
Naziv datoteke: Projekt1.dev
Spremi u obliku: Dev-C++ project (*.dev)
Spremi
Odustani
Ready.



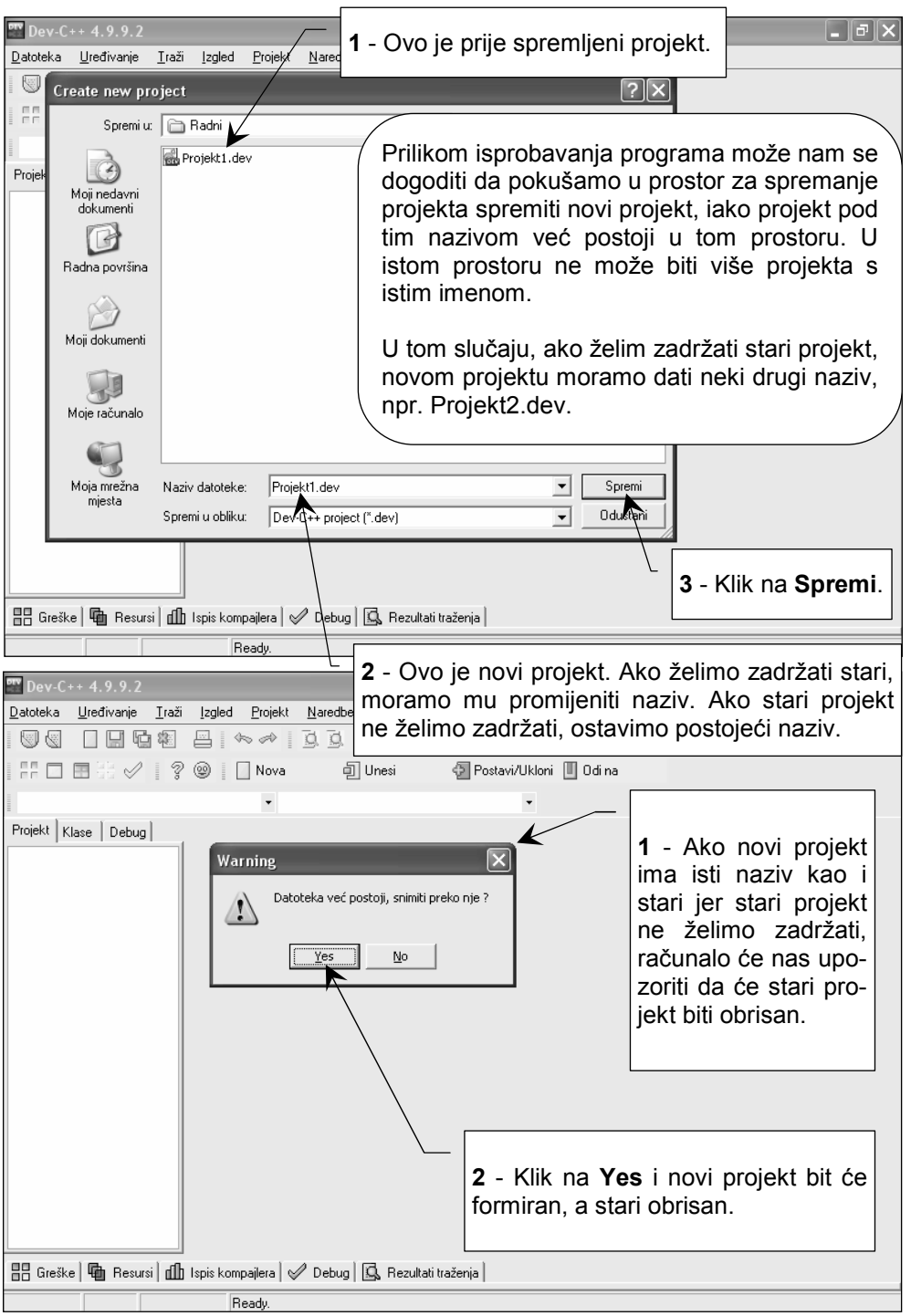
1 - Kliknemo na **Alati**.



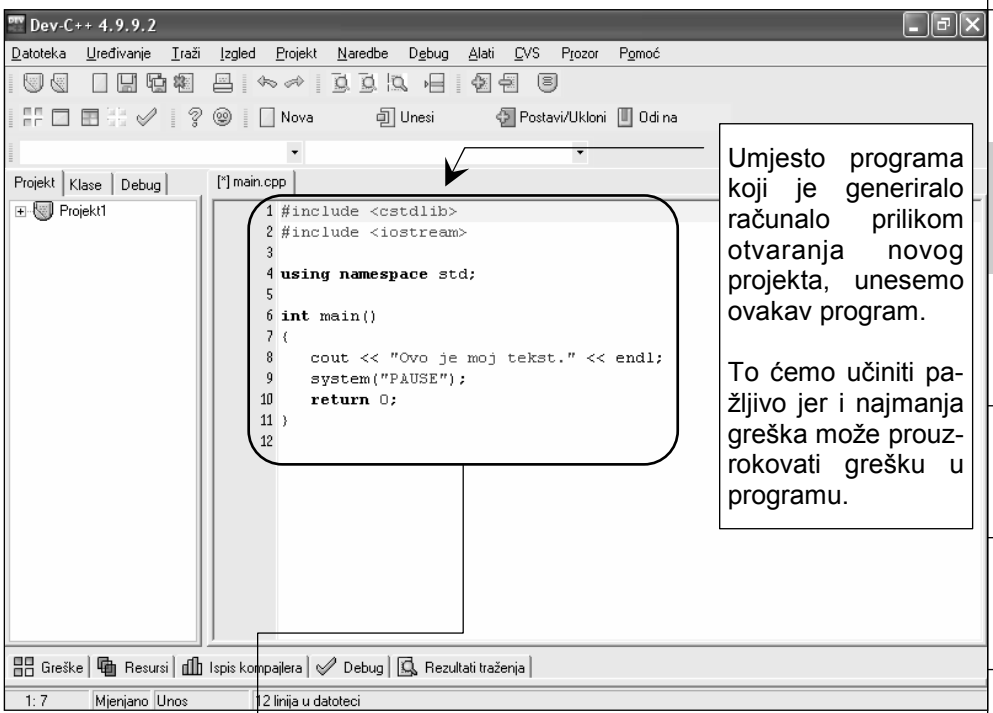

1 - Kliknemo na karticu **Prikaz**.
2 - Kliknemo na kvadratić ispred opcije **Brojevi linije**.
3 - Kliknemo na **OK**.
Izgled editora
Opće
Prikaz
Sintaksa | Kod | Pregledni
Editor Font:
Font
Veličina
Courier New
10
Fear is your worst enemy
Gutter Font:
Vidljivo
Brojevi linija
Auto veličina
Počni od nule
Koristi vlastiti font
Prikazuj početne nule
Font
Veličina
Širina
Terminal
9
32
Fear is your worst enemy
Ok
Odustani
Upomoć


Dev-C++ 4.9.9.2
Datoteka Uređivanje Traži Izgled Projekt Naredbe Debug Alati CVS Prozor Pomoć
Nova
Unesi
Postavi/Ukloni
Odi na
Projekt Klase Debug
[*] main.cpp
Projekt1
#include <cstdlib>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
system("PAUSE");
return EXIT_SUCCESS;
}
Greške Resursi Ispis kompajlera Debug Rezultati traženja
Mjenjano Unos
Done parsing in 0,08 seconds
Vidimo da se sada ispred svakog reda u programu automatski ispisuje broj reda.

Mogući problem



Unos prvog programa



Umjesto programa koji je generiralo računalo prilikom otvaranja novog projekta, unesemo ovakav program.

To ćemo učiniti pažljivo jer i najmanja greška može prouzrokovati grešku u programu.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << "Ovo je moj tekst." << endl;
   system("PAUSE");
   return 0;
}
```

Na početku će nam se često događati da umjesto točke sa zarezom ovdje unesemo dvotočje što će uzrokovati grešku u programu.

Lijevu vitičastu zagradu dobit ćemo tako da pritisnemo tipku AltGr i ne puštajući tu tipku, pritisnemo tipku b.

Desnu ćemo dobiti tako da pritisnemo AltGr i, ne puštajući tu tipku, pritisnemo tipku n.

Ovdje se nalazi broj nula, a ne veliko slovo o.

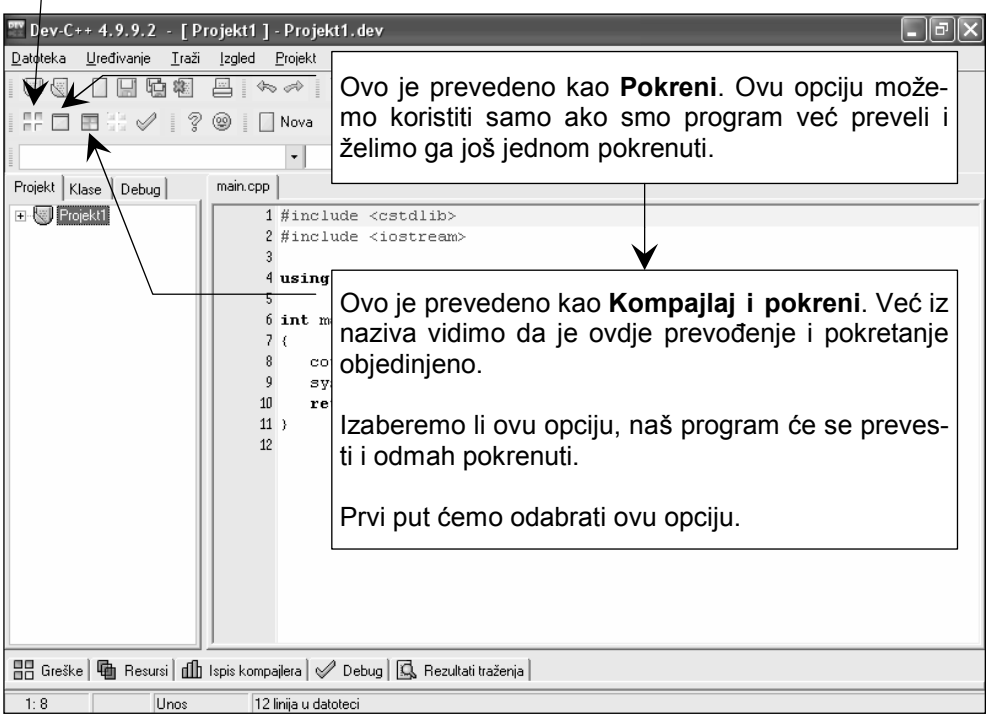
Pokretanje programa

Već smo rekli da računalo razumije samo nule i jedinice. Svi sadržaji, muzika, filmovi, slike itd. moraju biti prevedeni u nule i jedinice da bi ih računalo razumjelo. Tako je i s programom.

Da bi računalo razumjelo naš program, najprije ga iz oblika koji smo upravo napisali moramo prevesti u nule i jedinice, a tek nakon toga možemo narediti računalu da ga pokrene. Taj postupak prevođenja na engleskom se jeziku naziva ***Compile***, a prevoditelj našeg C++ programskog okruženja preveo je to sa ***Kompajlaj***. Mi ćemo u našoj knjizi za taj postupak koristiti izraz ***prevođenje***.

U našem programskom okruženju imamo tri važne naredbe vezane uz prevođenje i pokretanje programa.

Ovo je prevedeno kao **Kompajlaj**. Izaberemo li ovu opciju, naš program će se prevesti u oblik razumljiv računalu, ali se neće pokrenuti. Želite li tako prevedeni program pokrenuti, morate izabrati opciju **Pokreni**.



Ovo je prevedeno kao **Pokreni**. Ovu opciju možemo koristiti samo ako smo program već preveli i želimo ga još jednom pokrenuti.

Ovo je prevedeno kao **Kompajlaj i pokreni**. Već iz naziva vidimo da je ovdje prevođenje i pokretanje objedinjeno.

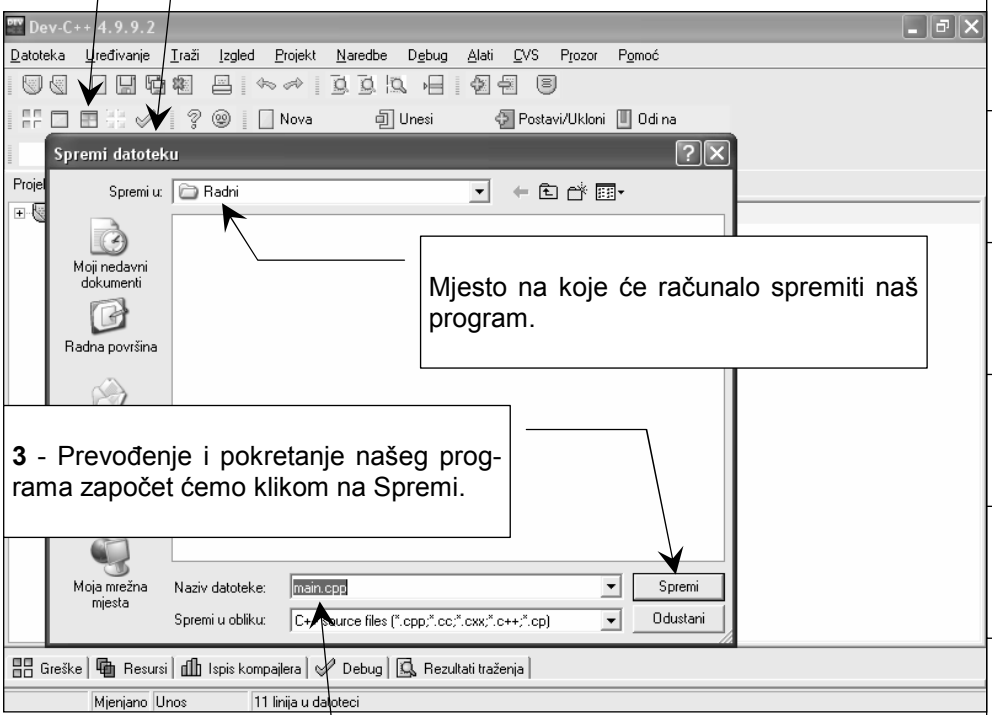
Izaberemo li ovu opciju, naš program će se prevesti i odmah pokrenuti.

Prvi put ćemo odabrati ovu opciju.

1 - Klikom na **Kompajlaj i pokreni** započet ćemo prevođenje i pokretanje našeg programa.

2 - Otvara se prozor u kojem nas računalo pita kamo i pod kojim nazivom ćemo spremiti naš program.

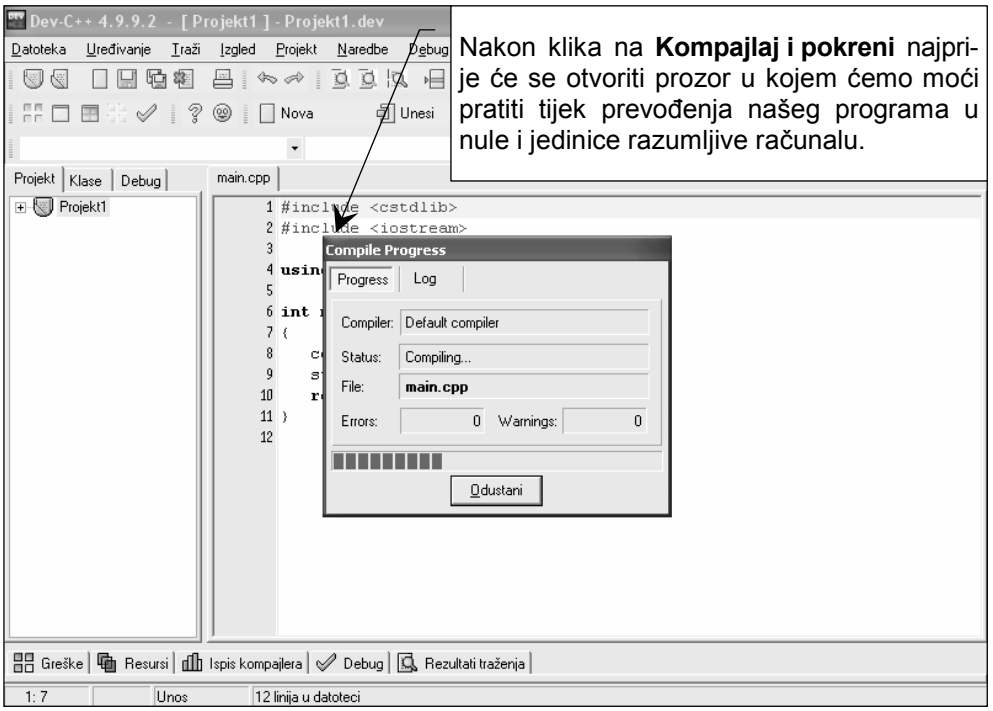
Računalo će predložiti da to bude ono isto mjesto u koje smo spremili projekt što ćemo mi prihvatiti.



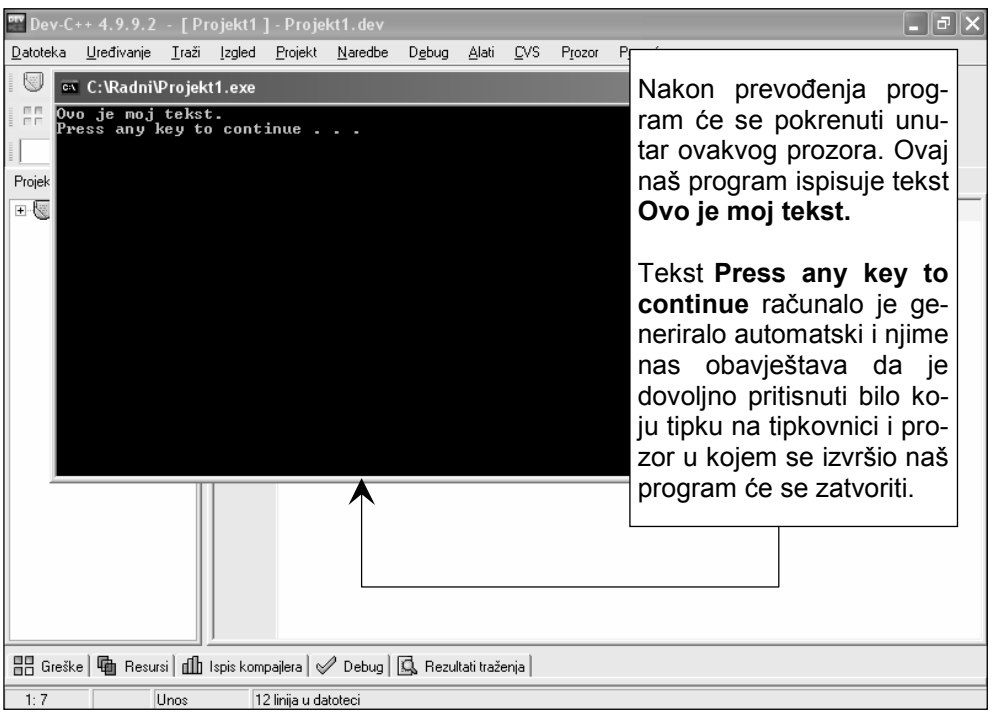
Mjesto na koje će računalo spremiti naš program.

3 - Prevođenje i pokretanje našeg programa započet ćemo klikom na Spremi.

Ovdje se nalazi naziv našeg programa, **main.cpp**. Iako bismo taj naziv mogli promijeniti, na početku je najbolje da zadržimo naziv koji je predložilo računalo.



Nakon klika na **Kompajlaj i pokreni** najprije će se otvoriti prozor u kojem ćemo moći pratiti tijek prevođenja našeg programa u nule i jedinice razumljive računalu.



Nakon prevođenja program će se pokrenuti unutar ovakvog prozora. Ovaj naš program ispisuje tekst **Ovo je moj tekst.**

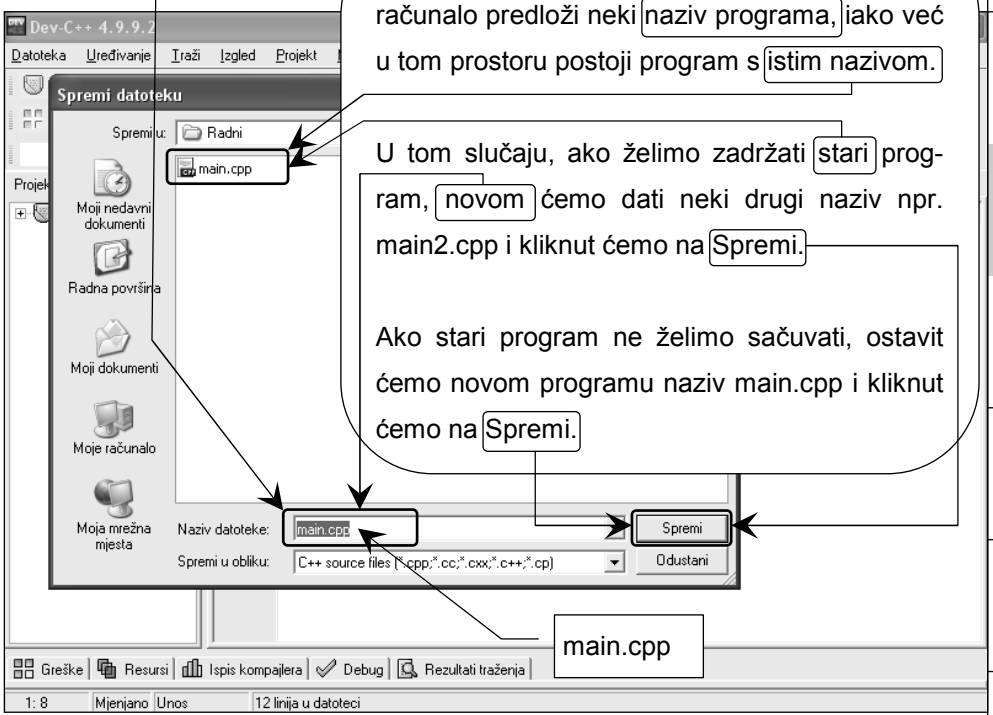
Tekst **Press any key to continue** računalo je generiralo automatski i njime nas obavještava da je dovoljno pritisnuti bilo koju tipku na tipkovnici i prozor u kojem se izvršio naš program će se zatvoriti.

Mogući problem

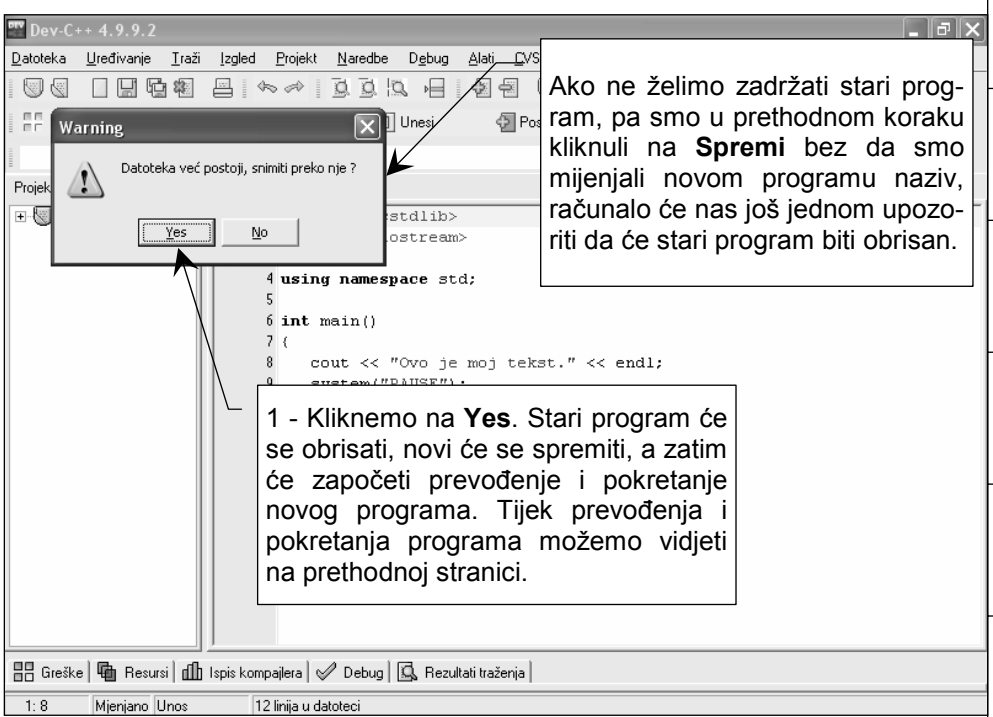
Tijekom rada može nam se dogoditi da nam računalo predloži neki naziv programa, iako već u tom prostoru postoji program s istim nazivom.

U tom slučaju, ako želimo zadržati stari program, novom ćemo dati neki drugi naziv npr. main2.cpp i kliknut ćemo na Spremi.

Ako stari program ne želimo sačuvati, ostavit ćemo novom programu naziv main.cpp i kliknut ćemo na Spremi.



main.cpp



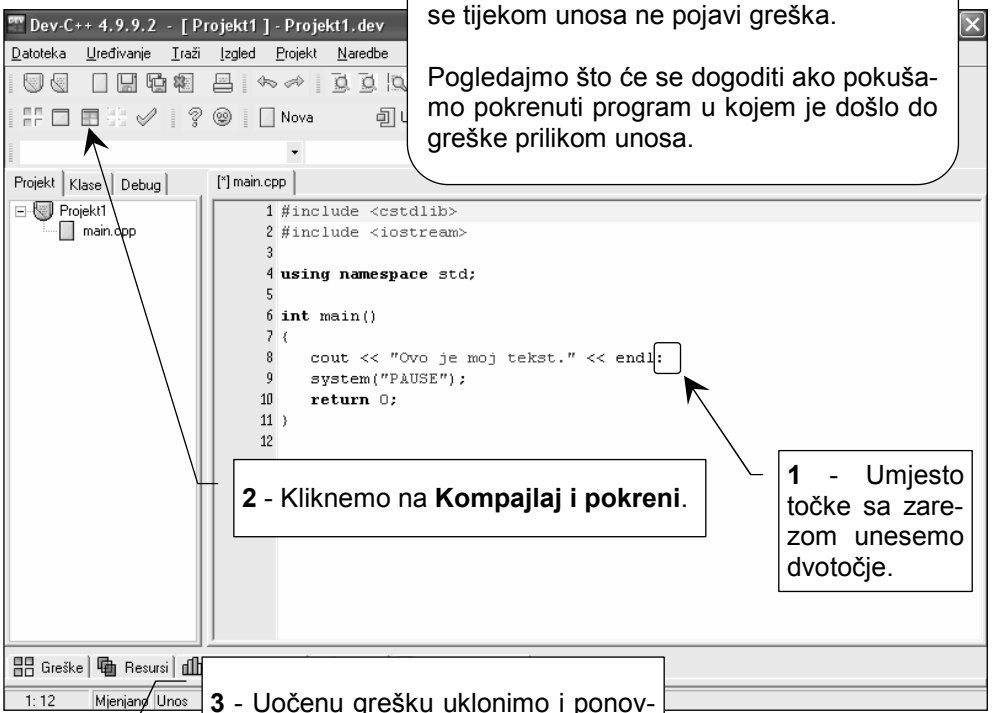
Ako ne želimo zadržati stari program, pa smo u prethodnom koraku kliknuli na **Spremi** bez da smo mijenjali novom programu naziv, računalo će nas još jednom upozoriti da će stari program biti obrisan.

1 - Kliknemo na **Yes**. Stari program će se obrisati, novi će se spremiti, a zatim će započeti prevođenje i pokretanje novog programa. Tijek prevođenja i pokretanja programa možemo vidjeti na prethodnoj stranici.

Greška u programu

Gotovo je nemoguće unijeti program, a da se tijekom unosa ne pojavi greška.

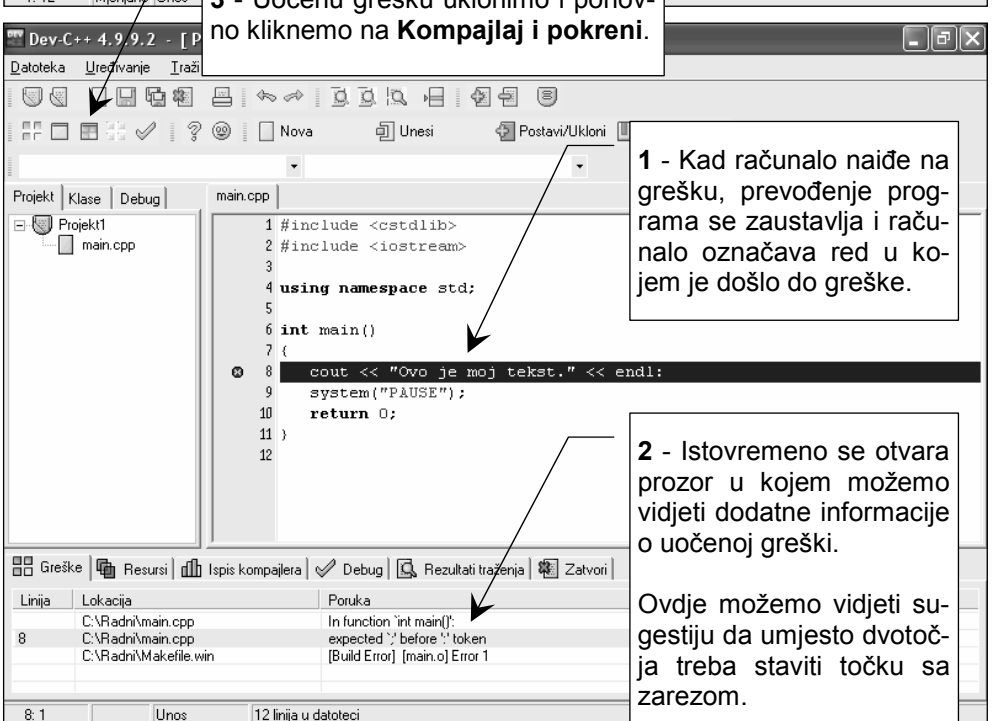
Pogledajmo što će se dogoditi ako pokušamo pokrenuti program u kojem je došlo do greške prilikom unosa.



2 - Kliknemo na **Kompajlaj i pokreni**.

1 - Umjesto točke sa zarezom unesemo dvotočje.

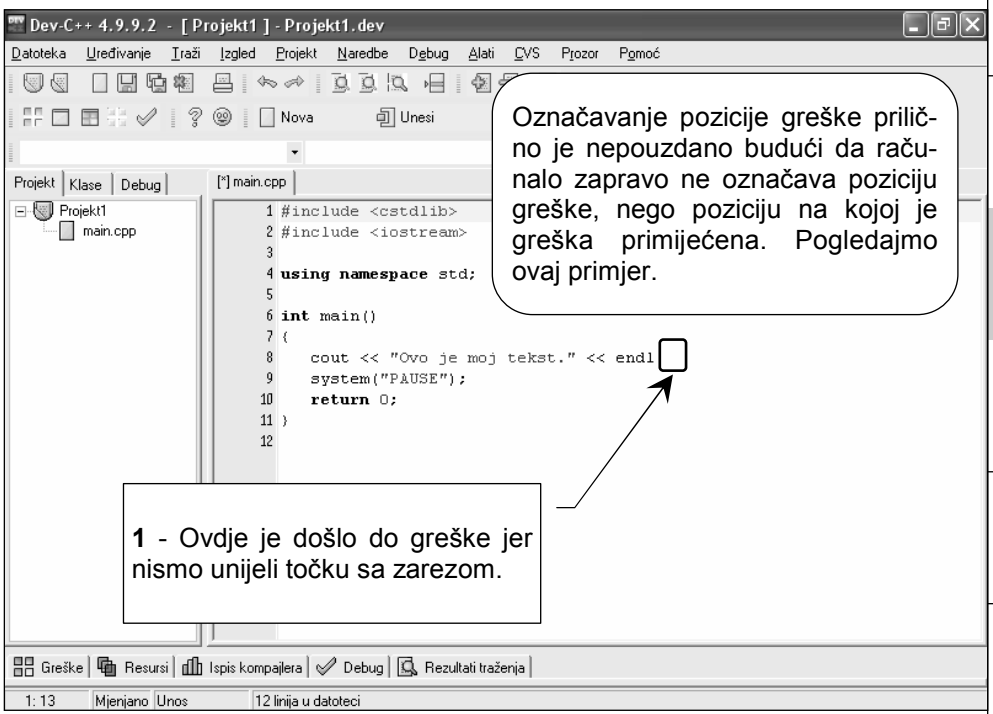
3 - Uočenu grešku uklonimo i ponovno kliknemo na **Kompajlaj i pokreni**.

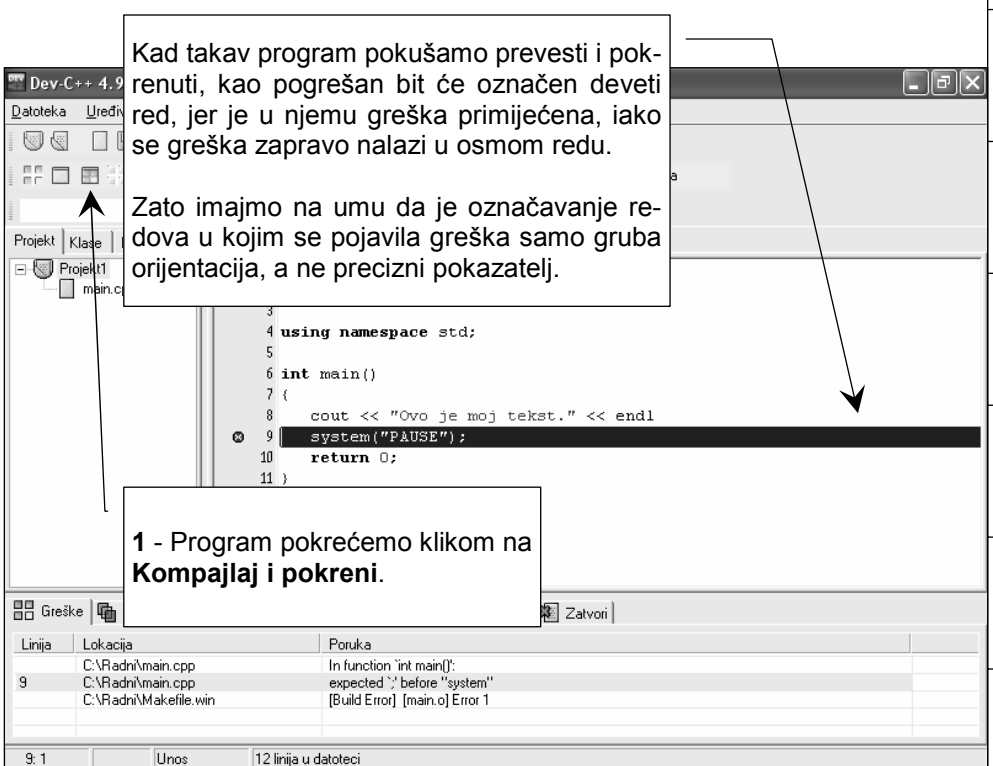


1 - Kad računalo naiđe na grešku, prevođenje programa se zaustavlja i računalo označava red u kojem je došlo do greške.

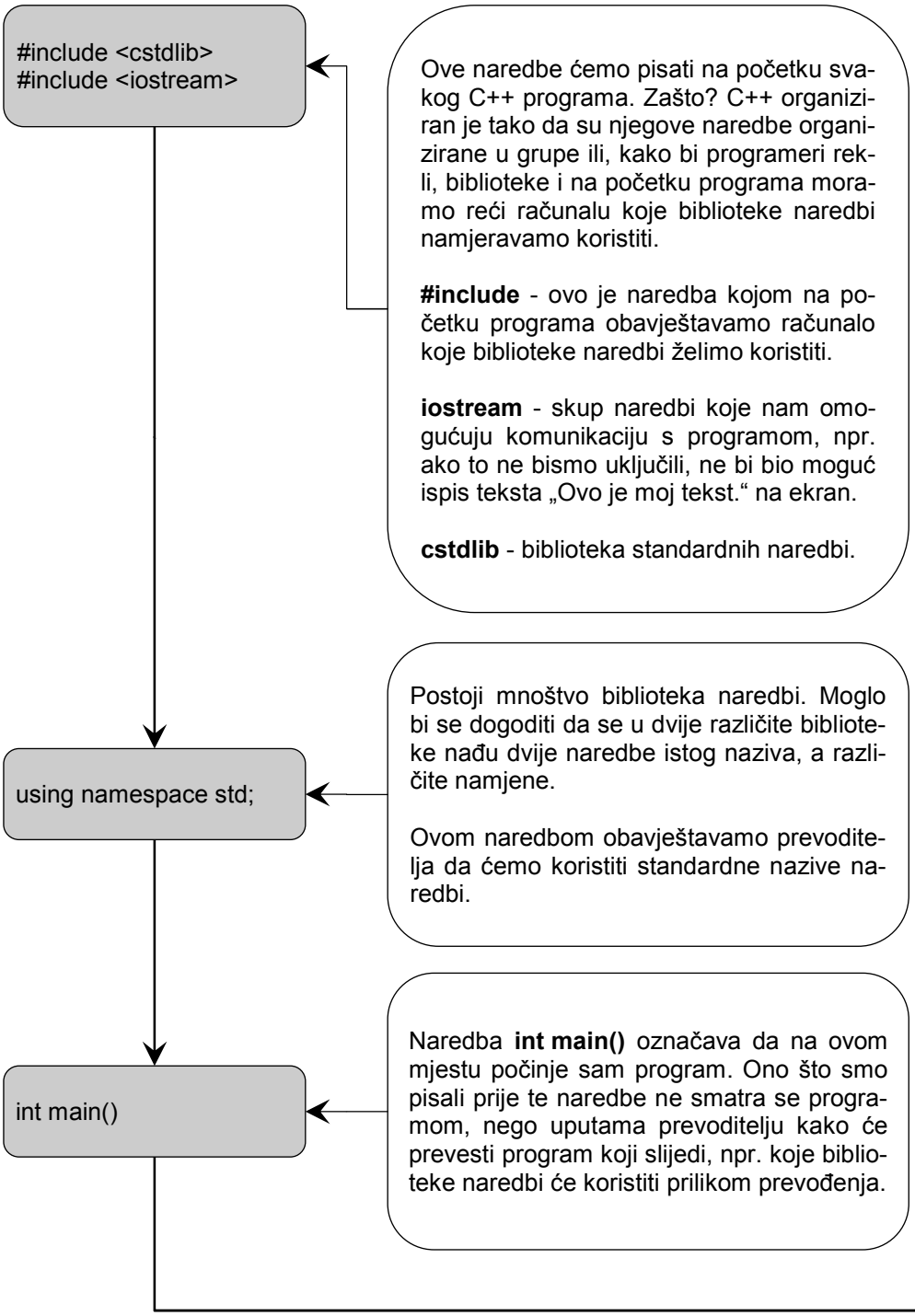
2 - Istovremeno se otvara prozor u kojem možemo vidjeti dodatne informacije o uočenoj greški.

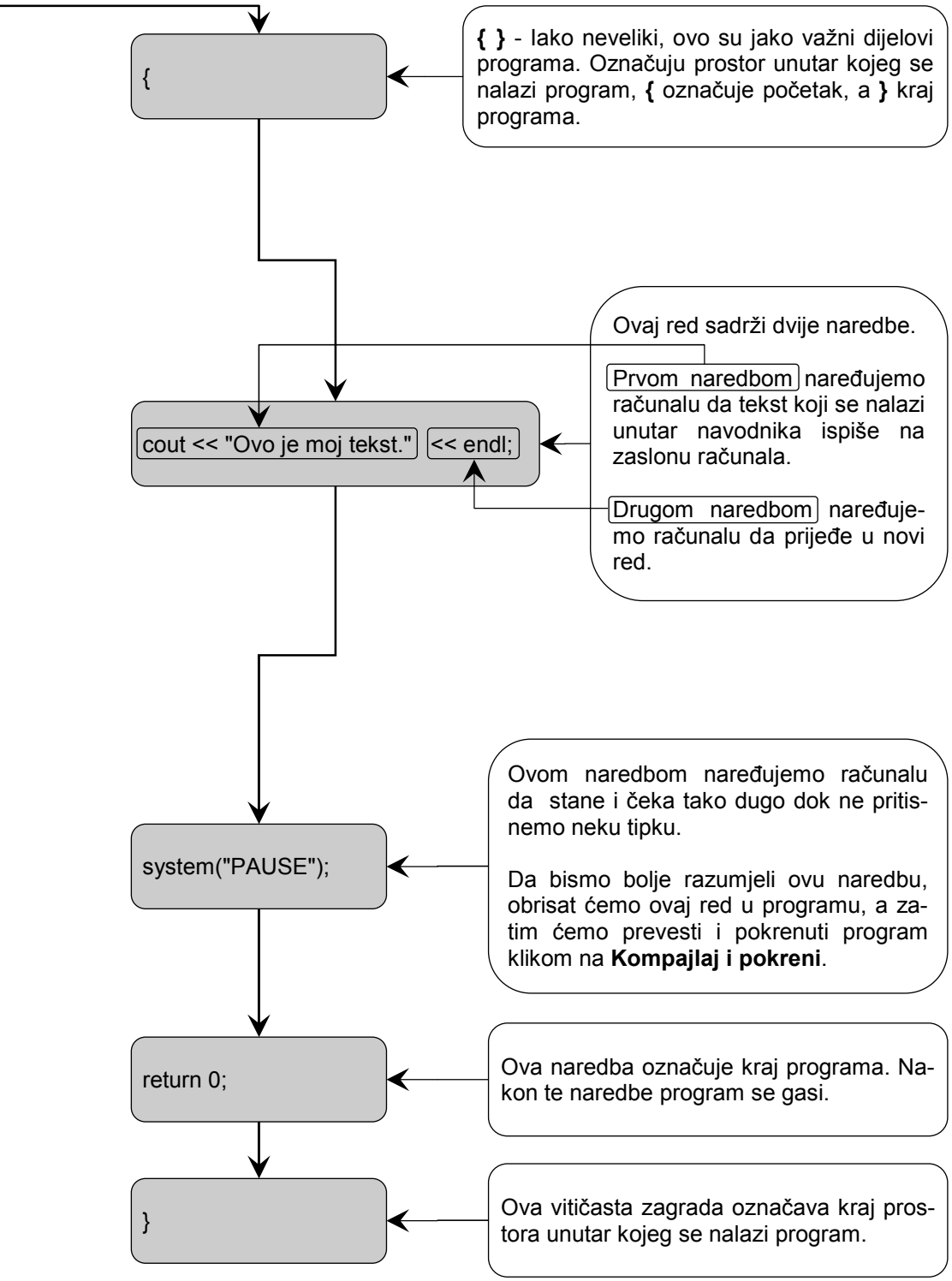
Ovdje možemo vidjeti sugestiju da umjesto dvotočja treba staviti točku sa zarezom.


Dev-C++ 4.9.9.2 - [Projekt1] - Projekt1.dev
Datoteka Uređivanje Traži Izgled Projekt Naredbe Debug Alati CVS Prozor Pomoć
Nova
Unesi
Projekt Klase Debug
[*] main.cpp
Projekt1
main.cpp
1 #include <cstdlib>
2 #include <iostream>
3
4 using namespace std;
5
6 int main()
7 {
8 cout << "Ovo je moj tekst." << endl
9 system("PAUSE");
10 return 0;
11 }
12
Označavanje pozicije greške prilično je nepouzdano budući da računalo zapravo ne označava poziciju greške, nego poziciju na kojoj je greška primijećena. Pogledajmo ovaj primjer.
1 - Ovdje je došlo do greške jer nismo unijeli točku sa zarezom.
Greške Resursi Ispis kompajlera Debug Rezultati traženja
1: 13 Mjenjano Unos 12 linija u datoteci


Kad takav program pokušamo prevesti i pokrenuti, kao pogrešan bit će označen deveti red, jer je u njemu greška primijećena, iako se greška zapravo nalazi u osmom redu.
Zato imajmo na umu da je označavanje redova u kojim se pojavila greška samo gruba orijentacija, a ne precizni pokazatelj.
using namespace std;
int main()
{
cout << "Ovo je moj tekst." << endl
system("PAUSE");
return 0;
}
1 - Program pokrećemo klikom na Kompajlaj i pokreni.
Greške
Zatvori
Linija Lokacija Poruka
C:\Radni\main.cpp In function `int main()':
9 C:\Radni\main.cpp expected `;' before "system"
C:\Radni\Makefile.win [Build Error] [main.o] Error 1
9: 1 Unos 12 linija u datoteci

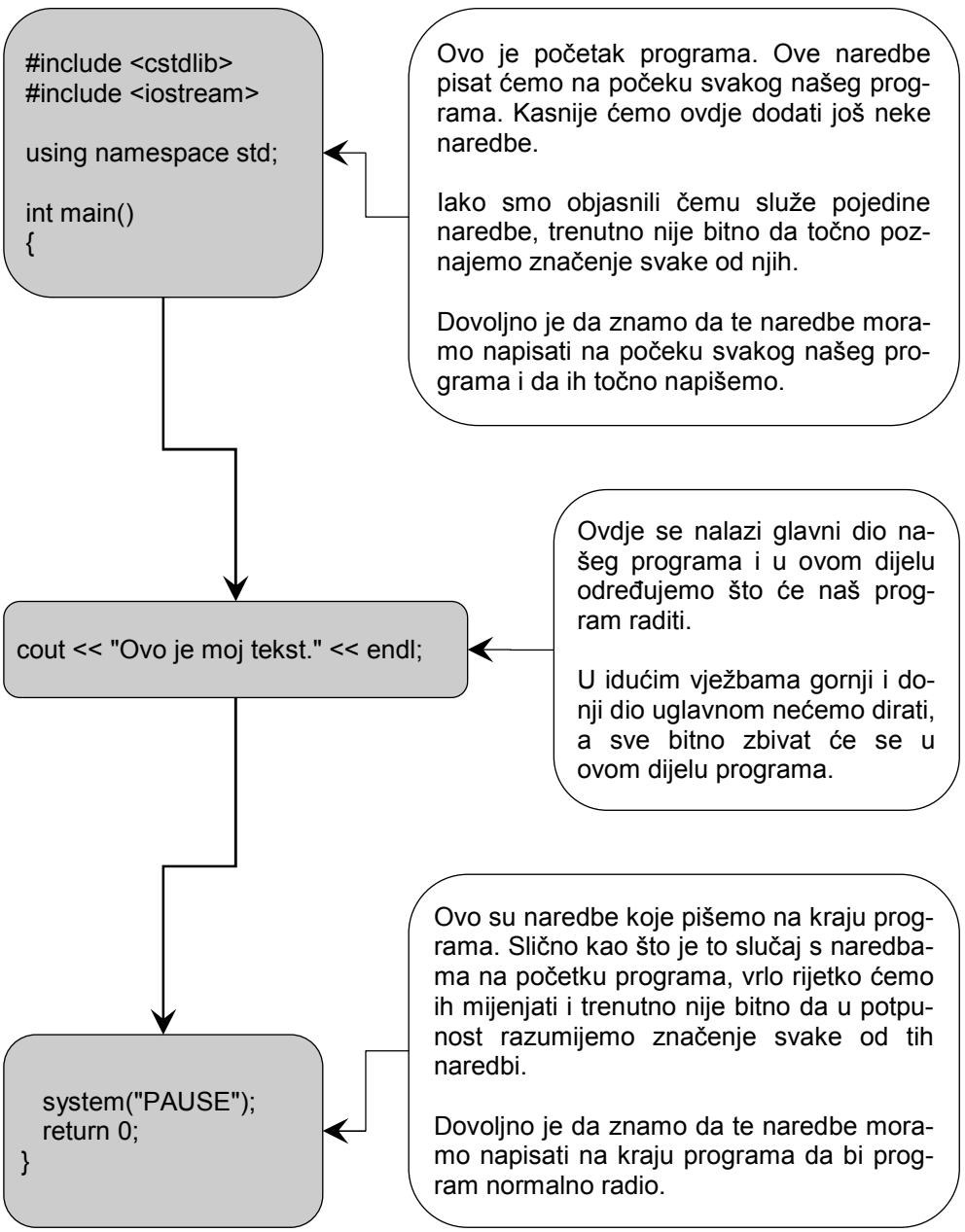
Analiza programa




{
{ } - lako neveliki, ovo su jako važni dijelovi programa. Označuju prostor unutar kojeg se nalazi program, **{** označuje početak, a **}** kraj programa.
cout << "Ovo je moj tekst." << endl;
Ovaj red sadrži dvije naredbe.
Prvom naredbom naređujemo računalu da tekst koji se nalazi unutar navodnika ispiše na zaslonu računala.
Drugom naredbom naređujemo računalu da prijeđe u novi red.
system("PAUSE");
Ovom naredbom naređujemo računalu da stane i čeka tako dugo dok ne pritisnemo neku tipku.
Da bismo bolje razumjeli ovu naredbu, obrisat ćemo ovaj red u programu, a zatim ćemo prevesti i pokrenuti program klikom na **Kompajlaj i pokreni**.
return 0;
Ova naredba označuje kraj programa. Nakon te naredbe program se gasi.
}
Ova vitičasta zagrada označava kraj prostora unutar kojeg se nalazi program.

Gruba skica programa

Da bismo se lakše snašli u našem programu, podijelit ćemo ga na nekoliko većih cjelina.



Varijacije programa

Jedna od najboljih metoda koje nam pomažu da bolje razumijemo funkcioniranje programa i pojedinih naredbi jest da u program unosimo manje izmjene, a zatim opažamo kakve posljedice će takve izmjene izazvati.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << "Ovo je moj tekst." << endl;
   system("PAUSE");
   return 0;
}
```

Kao što smo već vidjeli, pokrenemo li ovaj program, na zaslonu računala dobit ćemo ovakav rezultat.

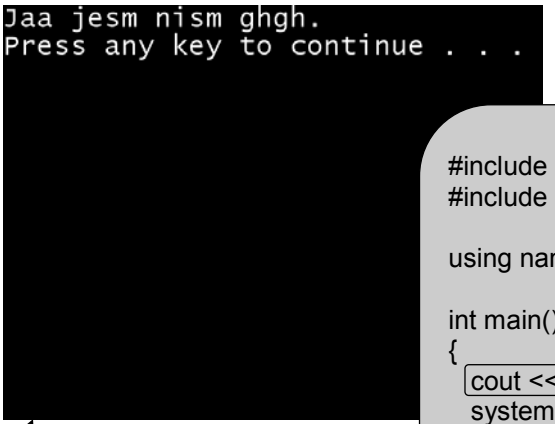
```
Ovo je moj tekst.
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << "OVDJE STAVLJAMO TEKST." << endl;
   system("PAUSE");
   return 0;
}
```

```
OVDJE STAVLJAMO TEKST.
Press any key to continue . . .
```

Promijenimo li ovaj red dobit ćemo ovakav rezultat.

Očigledno je da je funkcija ovog reda u programu ispis teksta koji se nalazi unutar navodnika i da taj tekst možemo po volji mijenjati.




```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  cout << "Jaa jesm nism ghgh." << endl;
  system("PAUSE");
  return 0;
}
```

Promijenimo ponovno ovaj red i pokrenimo program.

Dobit ćemo ovakav rezultat.

Očigledno je da ovim redom u programa naređujemo računalu da ispiše tekst unutar navodnika na zaslon računala, ali da računalo ne ulazi u to ima li taj tekst smisla ili nema.

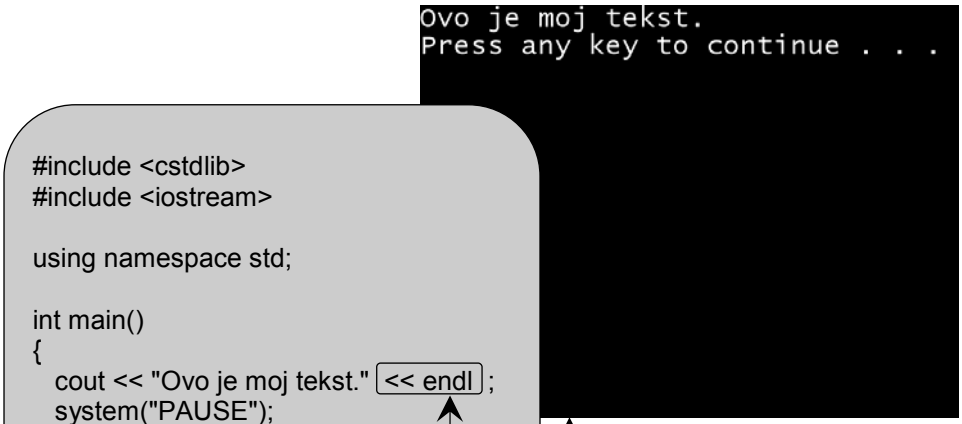
Iako računalo ne ulazi u smisao teksta koji smo napisali unutar navodnika, ovdje ipak postoji jedno ograničenje, a to je da unutar navodnika ne smijemo koristiti navodnik.

Navodnici označuju početak i kraj teksta, pa bi računalo navodnik unutar teksta smatralo krajem teksta, a sadržaj iza tog navodnika greškom.

Preuredimo ovaj red i pokušajmo program pokrenuti. Vidjet ćemo da će računalo dojaviti grešku.

Navodnik koji smo dodali.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  cout << "Jaa jesm " nism ghgh." << endl;
  system("PAUSE");
  return 0;
}
```



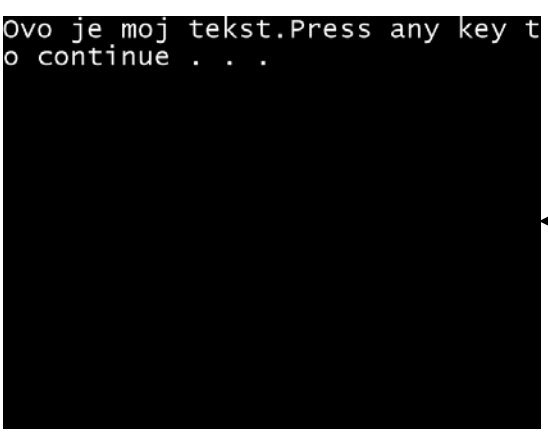

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  cout << "Ovo je moj tekst." << endl ;
  system("PAUSE");
  return 0;
}
```

Vratimo naš program u prvobitno stanje, a zatim ga pokrenimo klikom na **Kompajlaj i pokreni**. Ovako će izgledati rezultat rada našeg programa.

Da bismo bolje razumjeli njegovu funkciju, obrišemo ovaj dio programa. Nova inačica programa trebala bi ovako izgledati.

Klikom na **Kompajlaj i pokreni** pokrenemo program.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  cout << "Ovo je moj tekst.";
  system("PAUSE");
  return 0;
}
```



Ovako bi trebao izgledati rezultat izvođenja ove inačice programa.

Program radi i bez obrisanog dijela, ali nakon ispisa našeg teksta računalo nije skočilo u novi red, nego je sljedeći tekst nastavilo pisati u istom redu. Očigledno da je **<< endl** naredba za skok u novi red.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << "Ovo je moj tekst." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Vratimo obrisanu **<< endl** naredbu.

Postavlja se pitanje ako s **<< endl** naređujemo računalu da skoči u novi red, bi li s više takvih naredbi u nizu više puta skočilo u novi red.

Pogledajmo što će se dogoditi ako u naš program dodamo još jednu **<< endl** naredbu.

```
Ovo je moj tekst.

Press any key to continue . . .
```

Vidimo da se je dogodilo ono što smo očekivali.

Između prvog i drugog reda teksta pojavio se je jedan prazni red.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << "Ovo je moj tekst." << endl;
   cout << endl;
   cout << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Dodajmo u naš program još nekoliko **<< endl** naredbi.

```
Ovo je moj tekst.


Press any key to continue . . .
```

Pokrenemo li takav program vidimo da se je razmak između dva reda u programu povećao.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << "Ovo je moj tekst." << endl << endl << endl << endl;
   system("PAUSE");
   return 0;
}
```

Više naredbi << endl možemo napisati i unutar jednog reda.

Pokrenemo li ovu inačicu programa, uvjerit ćemo se da daje isti rezultat kao i prethodna.

Pogledajmo sada podrobnije red u kojem se nalaze naredbe za ispis teksta i skok u novi red.

Propisana riječ kojom obavještavamo računalo da ćemo sadržaj koji slijedi poslati na neku izlaznu jedinicu, u našem slučaju na zaslon računala.

Uočimo da naredbu **<< endl** ne možemo koristiti ako prije nismo stavili naredbu **cout**.

Ove oznake obvezno se stavljaju ispred sadržaja koji će iz računala ići na neku izlaznu jedinicu, npr. zaslon računala.

```
cout << "Ovo je moj tekst." << endl ;
```

Iako nevelik, ovo je JAKO važan dio programa.

Označava kraj naredbe ili grupe naredbi koje čine cjelinu.

Tekst unutar navodnika ispisat će se na zaslon računala.

Uočimo da ovaj dio čini cjelinu sa **<<** oznakama i da bez tih oznaka neće funkcionirati.

Ovom naredbom naređujemo računalu da prilikom ispisa skoči u novi red.

Uočimo da ova naredba čini cjelinu s << oznakama.

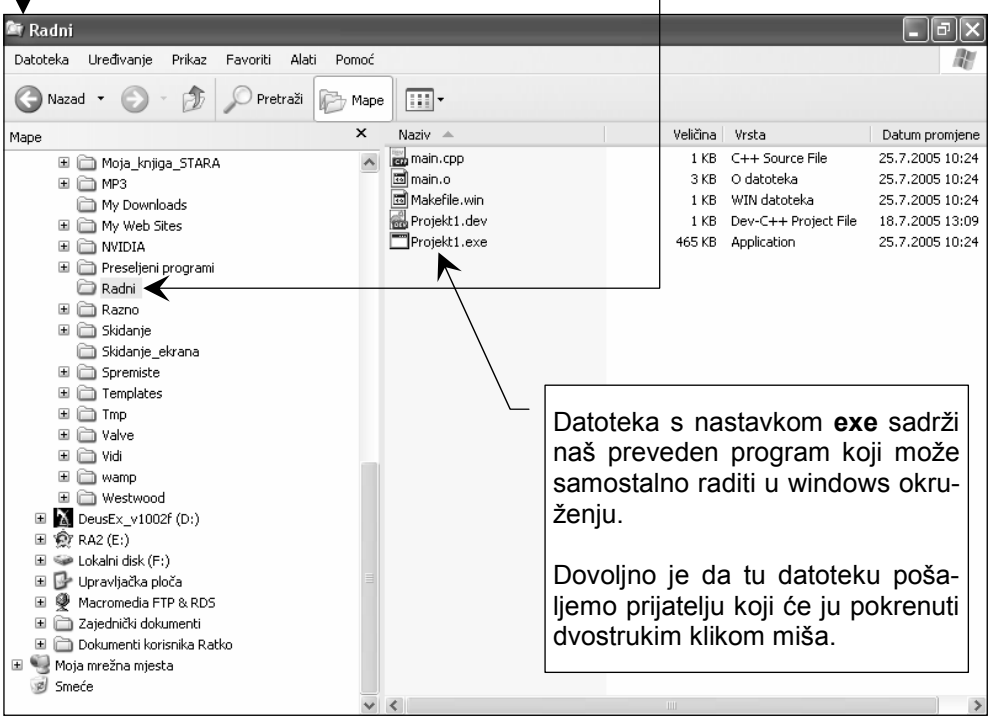
Distribucija programa

Nakon što naše programersko remek djelo proradi, osjetit ćemo potrebu naš program podijeliti s prijateljima ili ga staviti na internet. Postavlja se pitanje što moramo učiniti da bi naš program proradio i na prijateljevom računalu.

Prilikom prevođenja programa programsko okruženje iz našeg programa generira prevedeni program koji sadrži sve elemente koji su mu potrebni da bi se samostalno pokrenuo u windows programskom okruženju kao i svaki drugi windows program.

A gdje se nalazi prevedena inačica našeg programa?

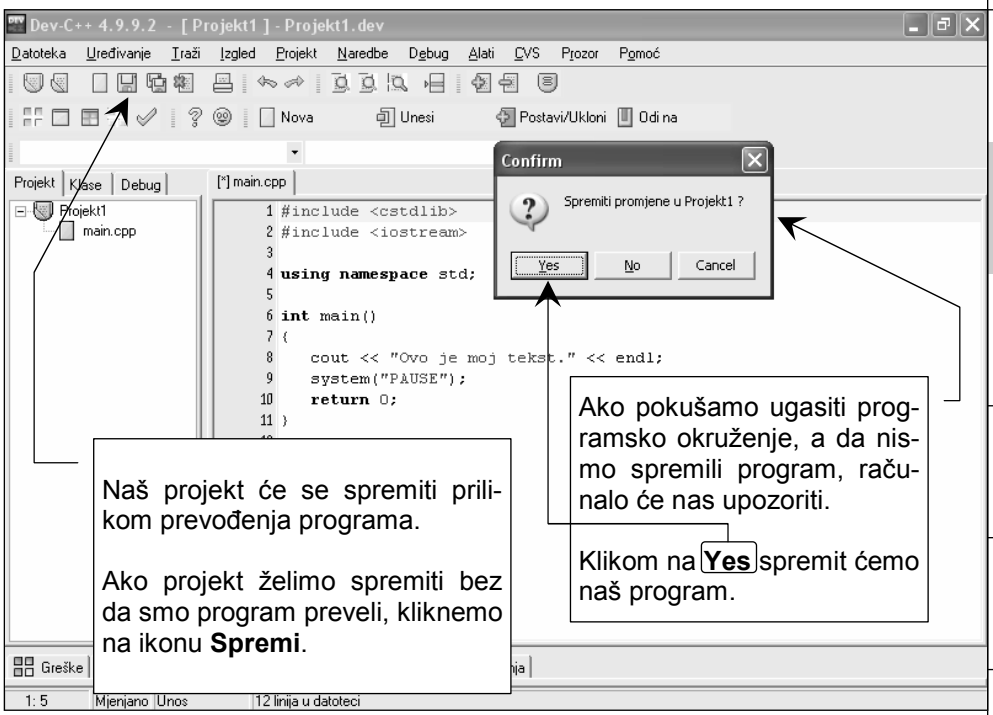
Otvorimo Windows Explorer i u njemu otvorimo mapu u koju smo koristili tijekom prevođenja programa. U našem slučaju to je **Radni** mapa na **C** disku.

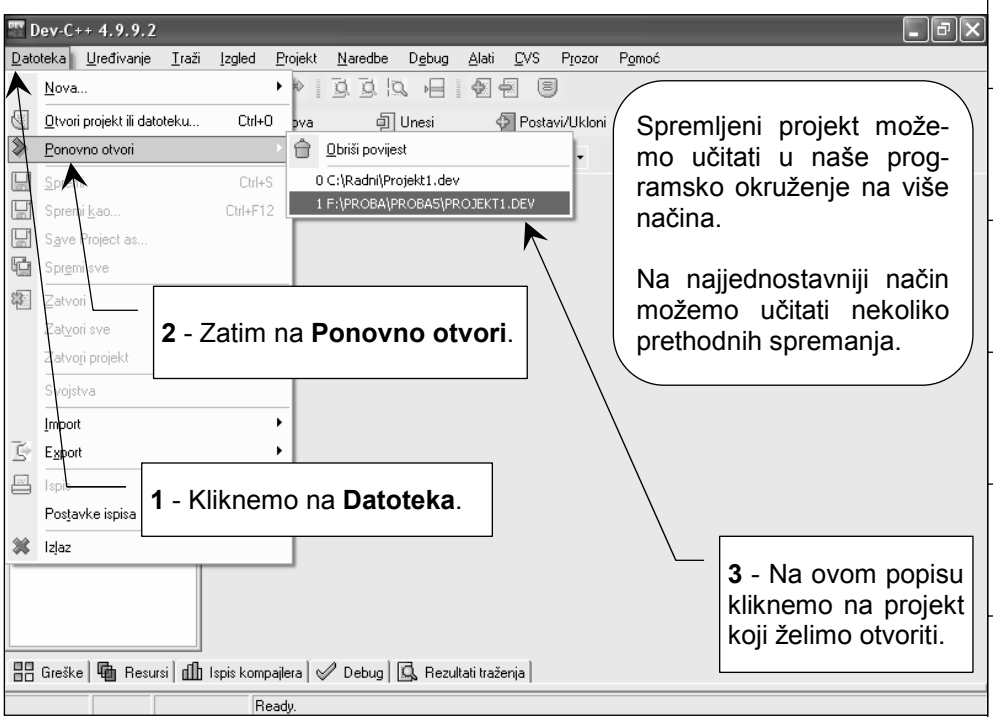


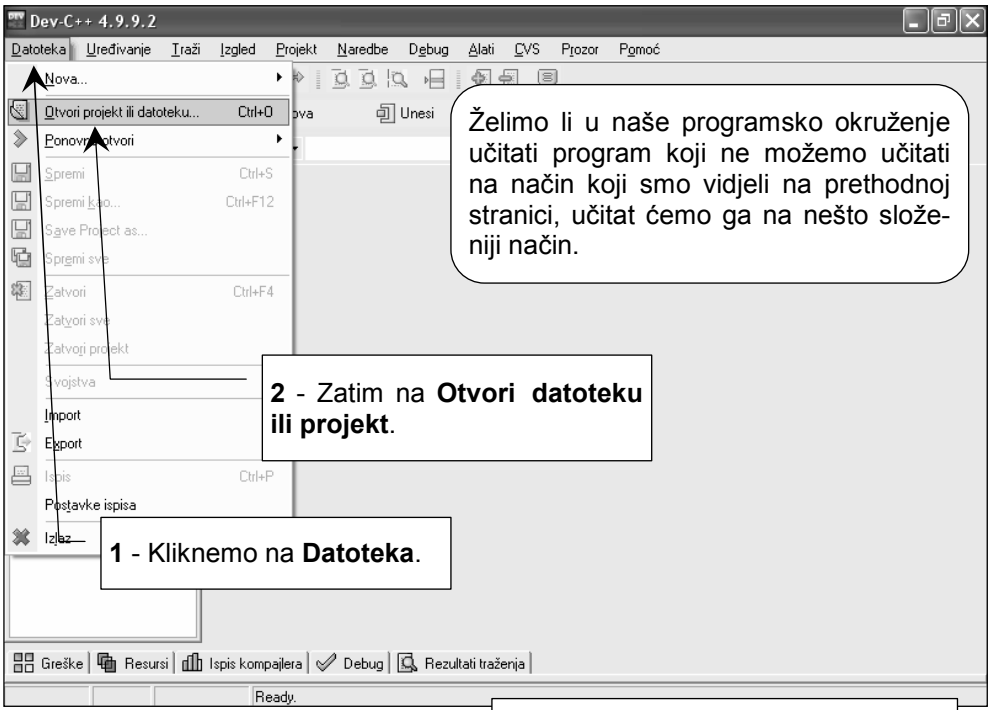
Datoteka s nastavkom **exe** sadrži naš preveden program koji može samostalno raditi u windows okruženju.

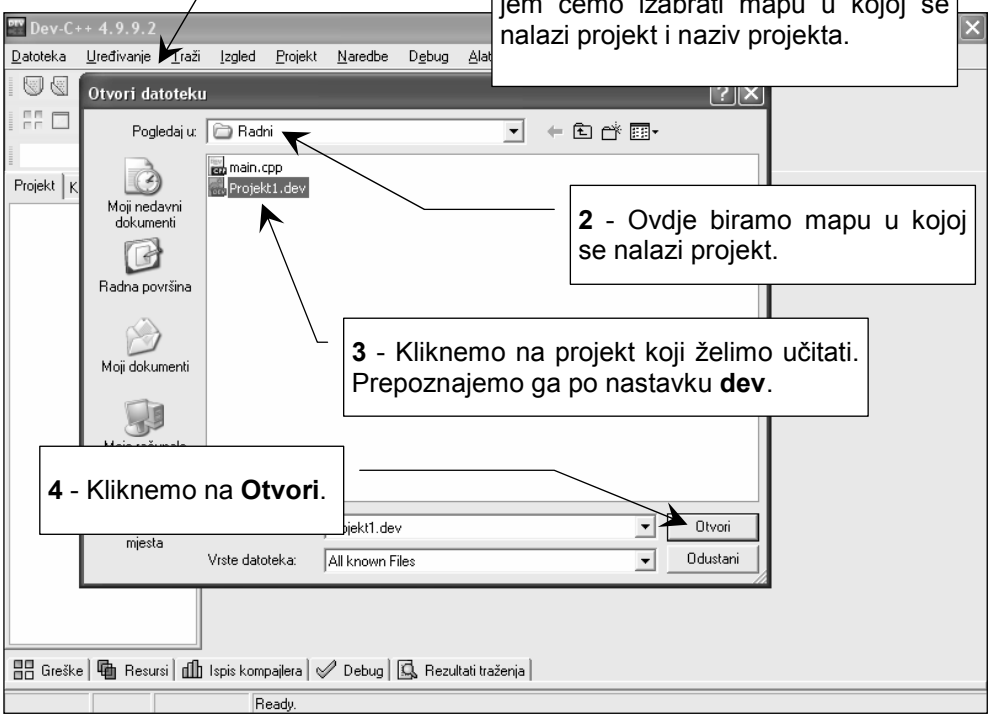
Dovoljno je da tu datoteku pošaljemo prijatelju koji će ju pokrenuti dvostrukim klikom miša.

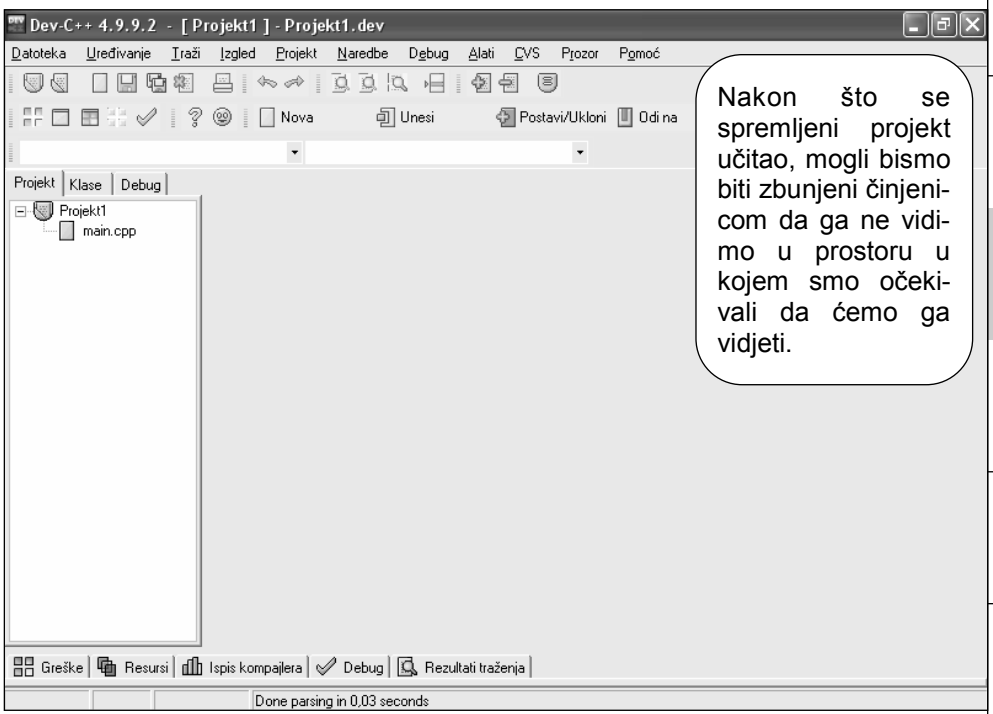
Učitavanje spremljenog projekta



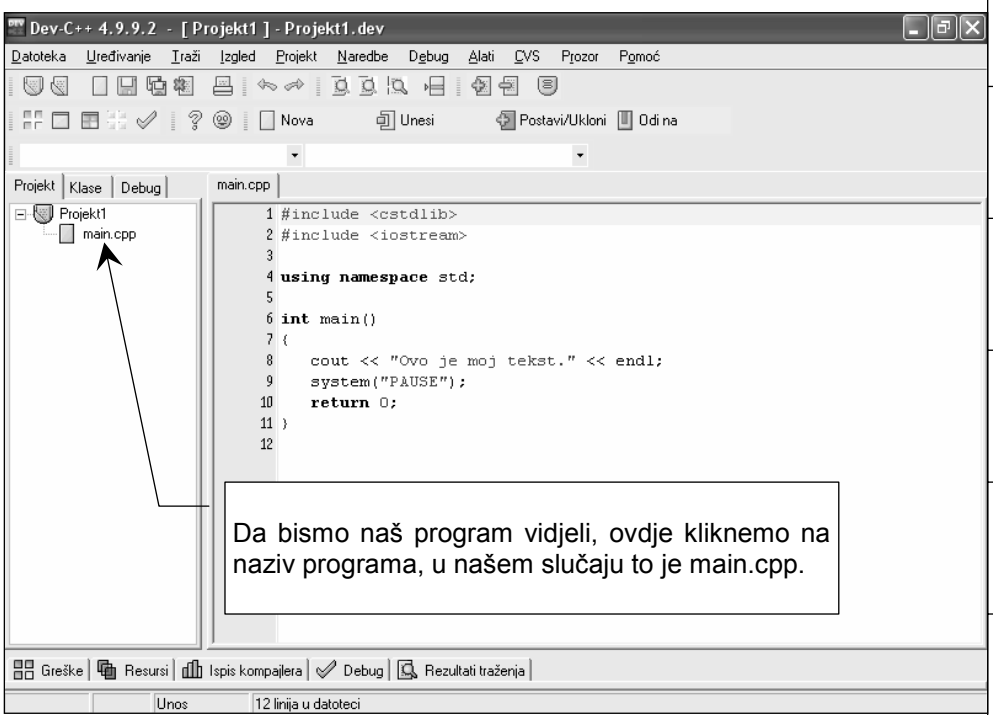



Dev-C++ 4.9.9.2
Datoteka Uređivanje Traži Izgled Projekt Naredbe Debug Alati CVS Prozor Pomoć
Nova...
Otvori projekt ili datoteku... Ctrl+O
Ponovno otvori
Spremi Ctrl+S
Spremi kao... Ctrl+F12
Save Project as...
Spremi sve
Zatvori Ctrl+F4
Zatvori sve
Zatvori projekt
Svojstva
Import
Export
Ispis Ctrl+P
Postavke ispisa
Izlaz
Unesi
Želimo li u naše programsko okruženje učitati program koji ne možemo učitati na način koji smo vidjeli na prethodnoj stranici, učitat ćemo ga na nešto složeniji način.
2 - Zatim na Otvori datoteku ili projekt.
1 - Kliknemo na Datoteka.
Greške Resursi Ispis kompajlera Debug Rezultati traženja
Ready.


1 - Otvorit će se dodatni prozor u kojem ćemo izabrati mapu u kojoj se nalazi projekt i naziv projekta.
Otvori datoteku
Pogledaj u: Radni
main.cpp
Projekt1.dev
Moji nedavni dokumenti
Radna površina
Moji dokumenti
2 - Ovdje biramo mapu u kojoj se nalazi projekt.
3 - Kliknemo na projekt koji želimo učitati. Prepoznajemo ga po nastavku dev.
4 - Kliknemo na Otvori.
Otvori
Odustani
All known Files
Vrste datoteka:
Greške Resursi Ispis kompajlera Debug Rezultati traženja
Ready.



Nakon što se spremljeni projekt učitao, mogli bismo biti zbunjeni činjenicom da ga ne vidimo u prostoru u kojem smo očekivali da ćemo ga vidjeti.



Da bismo naš program vidjeli, ovdje kliknemo na naziv programa, u našem slučaju to je main.cpp.

Varijable

Osnovne računske operacije

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << 2+3;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Sada ćemo računalo iskoristiti za računanje.

Unesimo ovaj program u C++ programsko okruženje.

Klikom na **Kompajlaj i pokreni** ga pokrenemo. Dobit ćemo ovakav rezultat.

```
5
Press any key to continue . . .
```

Ovaj program zbraja broj 2 i broj 3, a rezultat koji je u ovom slučaju broj 5 ispisuje na zaslon.

Pogledajmo sada strukturu našeg novog programa.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << 2+3;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

U ovom redu naređujemo računalu da zbroj broja 2 i broja 3, a to je broj 5, ispiše na zaslon računala.

U ovom redu naređujemo računalo da skoči u novi red.

Početak i kraj programa isti su kao i do sada, dok je srednji dio promijenjen.

Pokušajmo zbrojiti neka druga dva broja, npr. 8 i 4.

Pokrenimo program i vidjet ćemo da će računalo zbrojiti ta dva broja.

```
12
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << 8+4;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Pokušajmo neku drugu računsku operaciju, npr. oduzimanje.

U ovom programu oduzet ćemo od broja 12 broj 8.

Pokrenimo program i pogledajmo naš rezultat.

```
4
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << 12-8;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Dok smo za zbrajanje i oduzimanje koristili oznake uobičajene u matematici, za množenje ćemo koristiti zvjezdicu. Nalazi se iznad broja devet s desne strane, na brojčanoj tipkovnici.

Množenje broja 2 i broja 3 zapisat ćemo kao 2*3.

Pogledamo rezultat.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << 2*3;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

```
6
Press any key to continue . . .
```

Oznaka za dijeljenje također ima drugačiji oblik od onog na koji smo navikli u matematici. Kao oznaka dijeljenje koristi se kosa crta koja se nalazi iznad broja 8 s desne strane, na brojčanom dijelu tipkovnice.

12 podijeljeno sa 4 napisat ćemo kao 12/4.

Pokrenimo program i pogledamo rezultat.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << 12/4;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

```
3
Press any key to continue . . .
```

Na prvi pogled učinit će nam se da je program za dijeljenje ispravan.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << 13/4;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Pokušajmo ovdje umjesto 12/4 staviti 13/4.

Pokrenimo program i pogledajmo rezultat.

Rezultat je očigledno pogrešan.

```
3
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << 15/3;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Pokušajmo ovdje staviti 15/3 i pokrenimo program.

Rezultat je ispravan.

```
5
Press any key to continue . . .
```

Umjesto 15/3 stavimo 17/3 i pokrenimo program.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << 17/3;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

```
5
Press any key to continue . . .
```

Rezultat je pogrešan. Mogli bismo zaključiti da će dijeljenje biti ispravno, ako je rezultat cjelobrojni broj.

Ako je rezultat decimalni broj, dolazi do greške.

Kako riješiti taj problem, vidjet ćemo nešto kasnije u ovom poglavlju.

Varijable

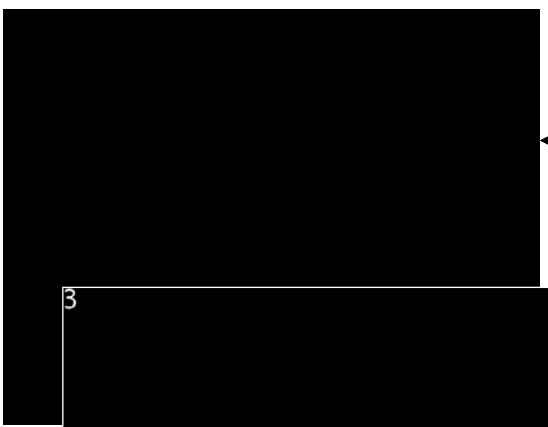
Vidjeli smo da osnovne računske operacije, osim dijeljenja, funkcioniraju, ali ne baš na način kako smo mi zamišljali da bi program trebao raditi.

Problem naših dosadašnjih programa jest u tome što svaki program vrši računsku operaciju između dva konkretna broja i ako želimo izvršiti računsku operaciju između druga dva broja, program moramo iznova pisati i prevoditi.

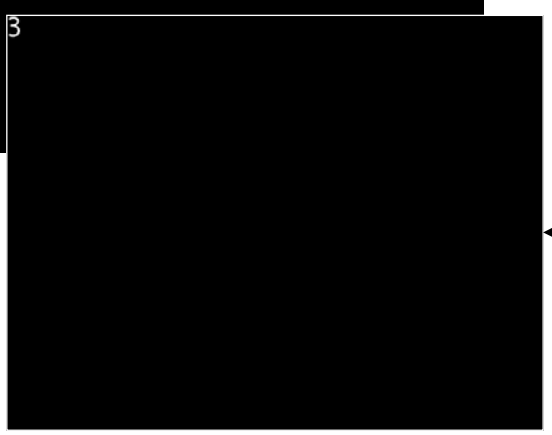
Postavlja se pitanje bismo li mogli napisati npr. program za zbrajanje koji ne bi zbrajao dva konkretna broja, nego bi mogao zbrojiti bilo koja dva broja.

To je, naravno, moguće. Unesimo, prevedimo i pokrenimo ovaj program.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int a;
   int b;
   int c;
   cin >> a;
   cin >> b;
   c = a + b;
   cout << c;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

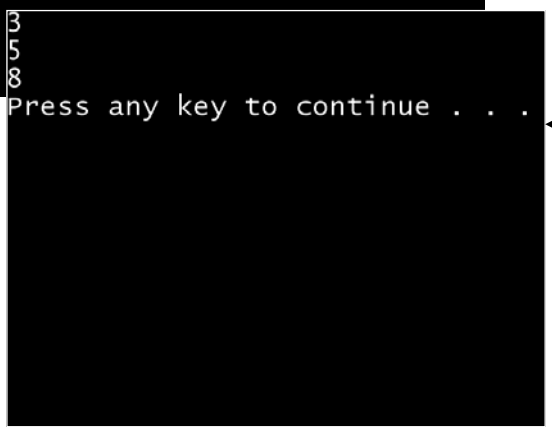


Nakon pokretanja programa ugledat ćemo zbunjujući crni ekran bez sadržaja.



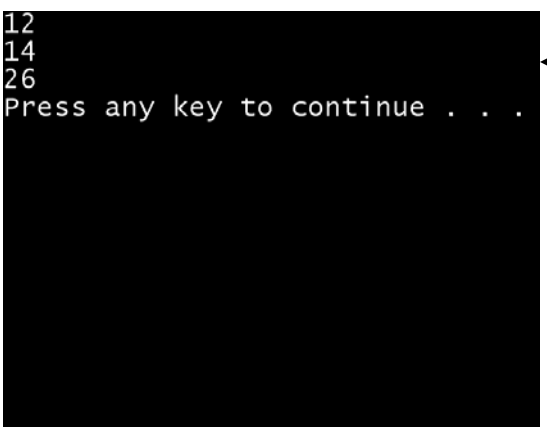
Na tipkovnici izaberemo prvi broj koji želimo zbrajati, npr. broj 3 i zatim pritisnemo **Enter** tipku na tipkovnici.

Na tipkovnici izaberemo drugi broj koji želimo zbrajati, npr. broj 5 i zatim pritisnemo Enter tipku na tipkovnici.



Nakon pritiska na **Enter** tipku na tipkovnici, a zaslonu računala ugledat ćemo rezultat zbrajanja broja 3 i broja 5, a to je broj 8.

Pokrenimo bez ikakvih izmjena isti program još jednom, ali sada unesimo neke druge brojeve, npr. broj 12 i broj 14.

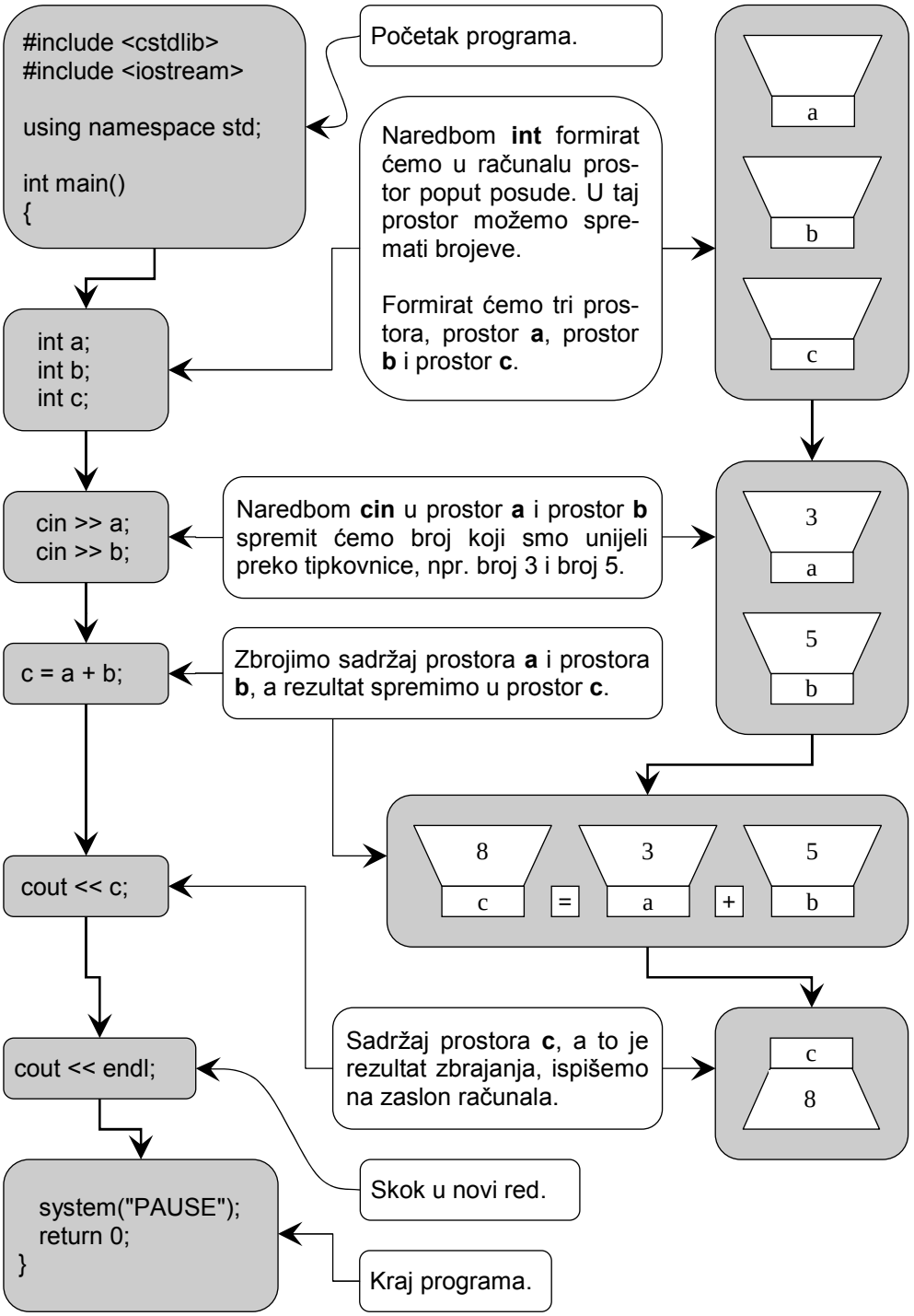


Na sličan način kao i u prethodnom primjeru, dakle tako da nakon svakog unosa broja pritisnemo tipku **Enter** na tipkovnici, unesimo brojeve 12 i 14.

Kao rezultat dobit ćemo njihov zbroj, a to je broj 26.

Vidimo da smo uspjeli postići ono što smo željeli. Pomoću jednog programa možemo zbrajati različite brojeve, bez da program moramo mijenjati.

Analiza programa



Ovom naredbom formiramo prostor u računalu u koji ćemo spremati brojeve.

Na početku taj prostor možemo zamisliti kao posudice u koje možemo staviti brojeve, iako su to zapravo rezervirana mjesta u memoriji našeg računala.

Naredba za formiranje prostora.

```
int a;
```

Oznaka **;** obvezno dolazi na kraju naredbe.

```
int b;
int c;
```

Ovim naredbama se formiraju još dva prostora, prostor **b** i prostor **c**.

Ovdje se nalazi naziv prostora.

Zašto svaki prostor ima naziv???

Zato što u jednom programu možemo imati više takvih prostora, pa nam naziv prostora omogućuje da ih razlikujemo.

Nakon formiranja prostor za spremanje brojeva je prazan i bio bi nam sasvim beskoristan kad u njega ne bismo mogli staviti neki broj.

U taj prostor broj možemo staviti na više načina, a jedan od njih je da ga unesemo pomoću tipkovnice. Za to ćemo koristiti naredbu **cin**.

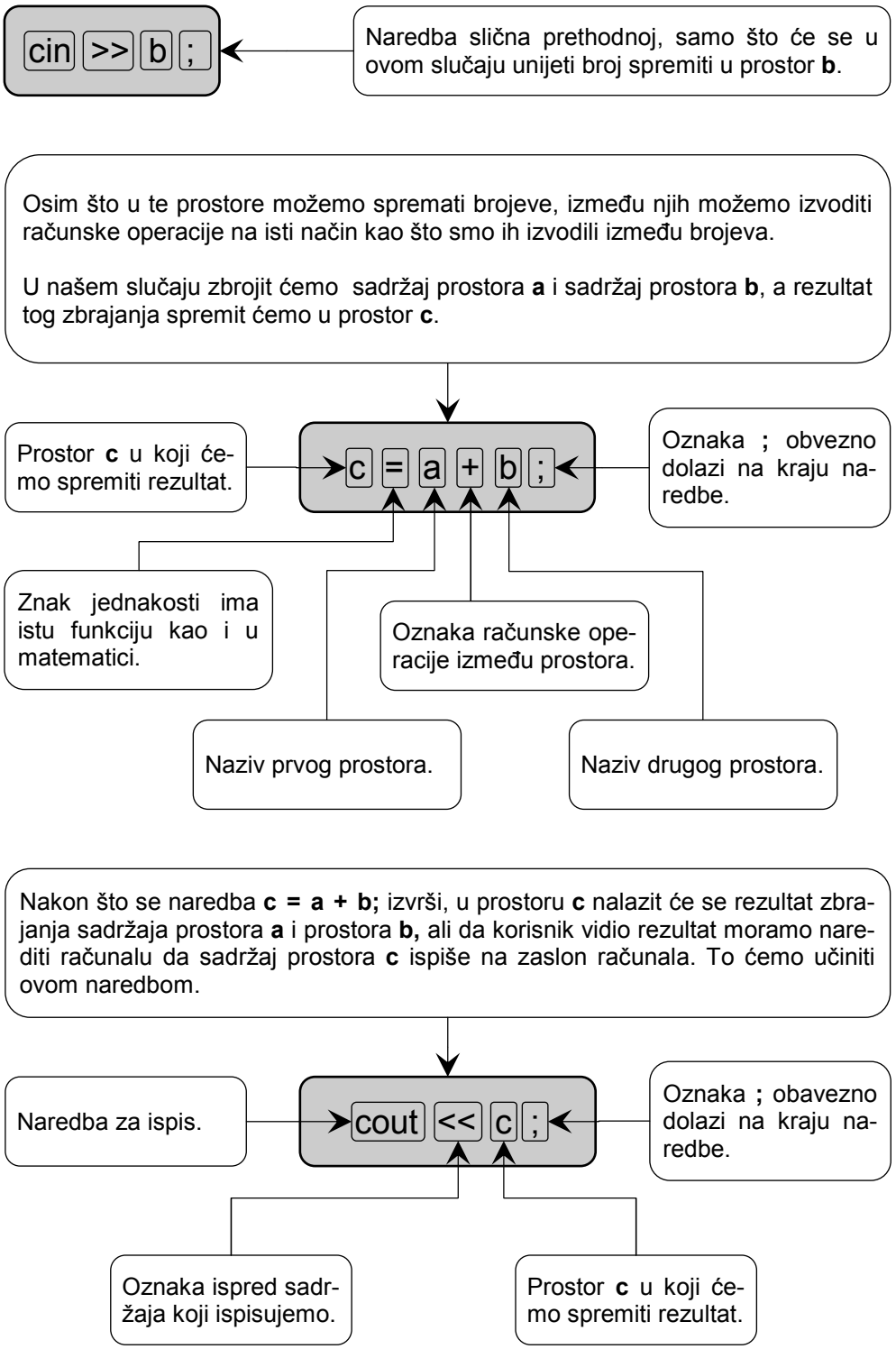
Naredba za unos.

```
cin >> a ;
```

Oznaka **;** obvezno dolazi na kraju naredbe.

Ova oznaka stavlja se ispred naziva prostora u koji ćemo unijeti broj.

Ovdje stavljamo oznaku prostora u koji ćemo unijeti broj. Dakle, broj ćemo unijeti u prostor **a** i ne u neki drugi prostor, npr. **b** ili **c**.


cin >> b ;
Naredba slična prethodnoj, samo što će se u ovom slučaju unijeti broj spremiti u prostor **b**.
Osim što u te prostore možemo spremati brojeve, između njih možemo izvoditi računske operacije na isti način kao što smo ih izvodili između brojeva.
U našem slučaju zbrojit ćemo sadržaj prostora **a** i sadržaj prostora **b**, a rezultat tog zbrajanja spremit ćemo u prostor **c**.
Prostor **c** u koji ćemo spremiti rezultat.
c = a + b ;
Oznaka **;** obvezno dolazi na kraju naredbe.
Znak jednakosti ima istu funkciju kao i u matematici.
Oznaka računske operacije između prostora.
Naziv prvog prostora.
Naziv drugog prostora.
Nakon što se naredba **c = a + b;** izvrši, u prostoru **c** nalazit će se rezultat zbrajanja sadržaja prostora **a** i prostora **b,** ali da korisnik vidio rezultat moramo narediti računalu da sadržaj prostora **c** ispiše na zaslon računala. To ćemo učiniti ovom naredbom.
Naredba za ispis.
cout << c ;
Oznaka **;** obavezno dolazi na kraju naredbe.
Oznaka ispred sadržaja koji ispisujemo.
Prostor **c** u koji ćemo spremiti rezultat.

Varijacije programa

Da bismo bolje razumjeli izradu programa koji koriste prostor za spremanje brojeva, napravit ćemo niz izmjena na našem programu i opažat ćemo kako te izmijene utječu na rad program.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int prvi;
   int drugi;
   int rezultat;
   cin >> prvi;
   cin >> drugi;
   rezultat = prvi + drugi;
   cout << rezultat;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Ovaj program ima potpuno istu strukturu kao prethodni, samo što ima drugačije nazive prostora za spremanje brojeva.

Prvi prostor više se ne naziva **a**, nego **prvi**. Drugi prostor više se ne naziva **b**, nego **drugi**. Treći prostor više se ne naziva **c**, nego **rezultat**.

Prevedimo i pokrenimo ovaj program i vidjet ćemo da radi jednako kao i prethodni program.

```
8
4
12
Press any key to continue . . .
```

Sad se postavlja pitanje po kojoj logici se biraju nazivi prostora za spremanje brojeva. Pogledajmo neka pravila i sugestije.

- Ne smiju se koristi naredbe; npr. **int aaa;** je dozvoljeno, a **int cout;** nije.

- Koriste se samo brojevi i slova engleske abecede, dakle bez čžš.

- Naziv ne smije sadržavati razmak; **PrviBroj** i **prvi_broj** je dozvoljeno, a **prvi broj** nije.

- Ako naziv sadrži brojeve, prvi znak mora biti slovo, dakle **a1** je dozvoljeno, a **1a** nije.

- Pametno je da naziv prostora govori što se u prostoru nalazi. Bolje je da prostor u koji ćemo spremati površinu nazovemo **povrsina** nego **qqy1**.

- Dva različita prostora ne mogu imati isto ime.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int A;
   int B;
   int c;
   cin >> a;
   cin >> b;
   c = a + b;
   cout << c;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

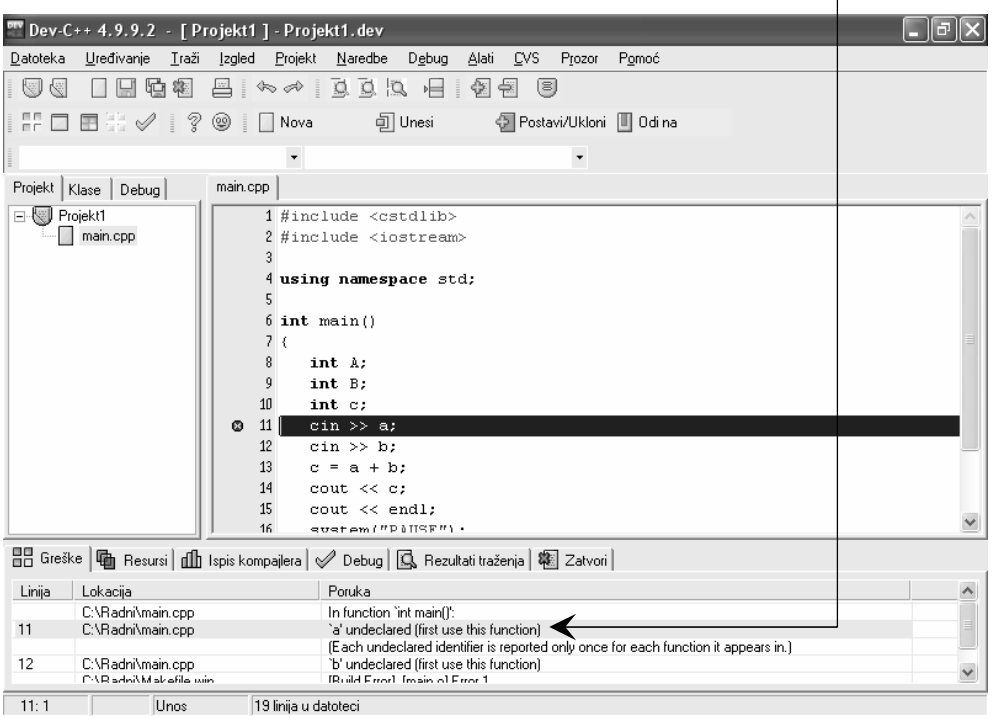
Pokušamo li pokrenuti ovaj program, računalo će dojaviti grešku.

Do greške je došlo zato što smo ovdje formirali prostor **A**, a ovdje smo pokušali unijeti broj u prostor **a**.

Računalo razlikuje velika i mala slova pa njemu prostor **a** i prostor **A** nisu isto.

S njegove točke gledišta mi ovdje pokušavamo unijeti broj u prostor koji ne postoji.

Zato će nas računalo u komentaru greške upozoriti da ne zna što je to **a**.



```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int a;
   int b;
   int c;
   cin >> a;
   cin >> b;
   c = a + b;
   cout << "c";
   cout << endl;
   system("PAUSE");
   return 0;
}
```

U ovom programu ovu naredbu nismo napisali u ispravnom obliku kao **cout << c;** nego u obliku **cout << "c";** dakle sadržaj koji se ispisuje stavili smo u navodnike kao što smo to naviknuli prilikom ispisa teksta korištenjem **cout** naredbe.

Pokrenemo li ovaj program, on će se izvršiti i računalo nam neće dojaviti nikakvu grešku, ali kao rezultat nećemo dobiti ono što smo očekivali.

Vidimo da smo u prostor **a** unijeli 8, u prostor **b** unijeli smo 4, ali kao rezultat nismo dobili broj 12, nego slovo c.

Zašto?

cout << c; - ispisuje sadržaj prostora **c**.

cout << "c"; - ispisuje slovo c.

```
8
4
c
Press any key to continue . . .
```

Ovaj tip greške veoma je teško pronaći u složenim programima, budući da računalo neće dojaviti grešku i poziciju greške prilikom prevođenja.

To se događa zato što je program formalno ispravno napisan; **cout << "c";** je ispravna naredba, ali ne radi ono što smo htjeli.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int broj1;
   int broj2;
   int rezultat;
   cin >> broj1;
   cin >> broj2;
   rezultat = broj1 - broj2;
   cout << rezultat;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Prostore za spremanje brojeva možemo koristiti i za druge matematičke operacije.

Ovdje možemo vidjeti program za oduzimanje.

Od sadržaja prostora **broj1** oduzet će se sadržaj prostora **broj2**, a rezultat će se spremiti u prostor **rezultat**.

```
9
6
3
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int broj1;
   int broj2;
   int rezultat;
   cin >> broj1;
   cin >> broj2;
   rezultat = broj1 * broj2;
   cout << rezultat;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Na sličan način možemo napraviti program za množenje sadržaja dvaju prostora.

```
3
4
12
Press any key to continue . . .
```

Problem dijeljenja

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int broj1;
   int broj2;
   int rezultat;
   cin >> broj1;
   cin >> broj2;
   rezultat = broj1 / broj2;
   cout << rezultat;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Ostao nam je neriješen problem dijeljenja.

Sadržaje prostora za spremanje brojeva možemo dijeliti, ali će kao i prošli put kad smo pokušali dijeliti, rezultat biti ispravan samo za brojeve čiji rezultat dijeljenja nije decimalni broj.

Ako je rezultat dijeljenja decimalni broj, rezultat će biti pogrešan.

```
14
3
4
Press any key to continue . . .
```

Problem ćemo riješiti ovako napisanim programom. Umjesto naredbom **int**, prostor za spremanje brojeva formirat ćemo naredbom **float;**

Pokrenemo li taj program, dobit ćemo ispravan rezultat.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   float broj1;
   float broj2;
   float rezultat;
   cin >> broj1;
   cin >> broj2;
   rezultat = broj1 / broj2;
   cout << rezultat;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

```
14
3
4.66667
Press any key to continue . . .
```

Ovo je još jedan način na koji možemo formirati prostor za spremanje brojeva.

U čemu je razlika između prostora formiranog **int** naredbom i prostora formiranog **float** naredbom???

U prostor formiran **int** naredbom možemo spremati samo brojeve bez decimalnog zareza, a u prostore formirane **float** naredbom možemo spremati brojeve s decimalnim zarezom.

Naredba za formiranje prostora za spremanje brojeva s decimalnim zarezom.

```
float broj1 ;
```

Oznaka **;** obvezno dolazi na kraju naredbe.

Naziv prostora. Uočimo da između broj i 1 nema razmaka.

Postavlja se pitanje bismo li mogli u naš program unijeti broj s decimalnim zarezom. Pokrenimo još jednom program iz prethodnog primjera i pokušajmo.

```
16.25
2.5
6.5
Press any key to continue . . .
```

Vidimo da je moguće unijeti decimalne brojeve u naš program.

Broj 16,25 podijelili smo brojem 2,5 i kao rezultat ispravno dobili 6,5.

Pri tome moramo imati u vidu da C++ kao decimalni zarez zapravo ne koristi zarez, nego točku.

Dakle, ako želimo unijeti npr. 3,14 unijet ćemo 3.14 korištenjem tipkovnice.

Uljepšavanje programa

Naši programi doduše rade, ali su tako ružni da bi teško bili ružniji. Korisnika dočeka crni ekran bez ikakvih uputa. Nakon što unesemo dva broja, dobit ćemo treći bez ikakvog komentara što taj treći broj predstavlja.

Budući da smo mi napisali program, mi znamo da treba unijeti dva broja, da nakon svakog unosa treba pritisnuti tipku Enter i mi znamo što predstavlja treći broj koji smo dobili kao rezultat, ali netko drugi veoma bi teško koristio naš program. Unesite i pokrenite ovaj program.

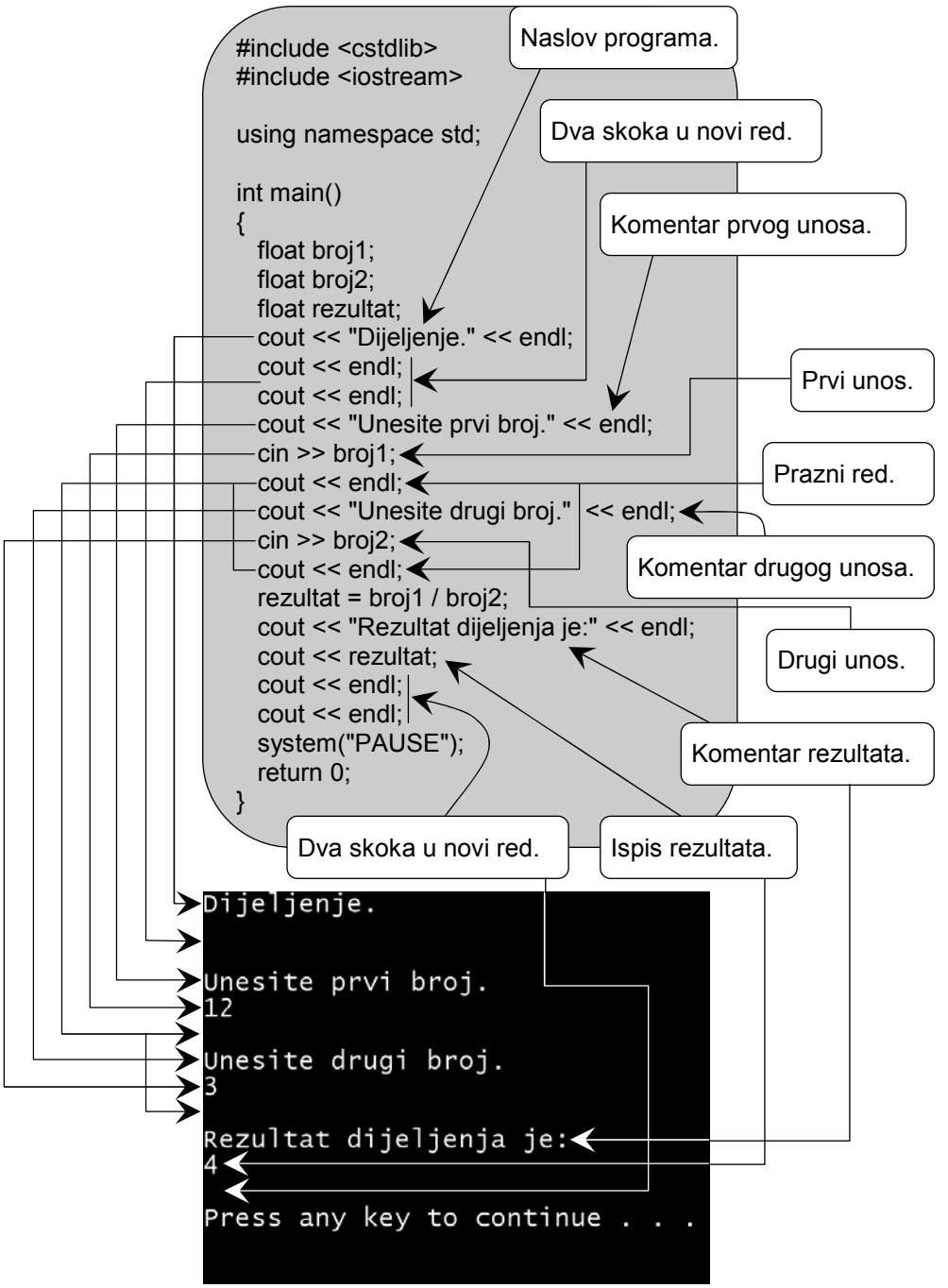
```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   float broj1;
   float broj2;
   float rezultat;
   cout << "Dijeljenje." << endl;
   cout << "Unesite prvi broj." << endl;
   cin >> broj1;
   cout << "Unesite drugi broj." << endl;
   cin >> broj2;
   rezultat = broj1 / broj2;
   cout << "Rezultat dijeljenja je:" << endl;
   cout << rezultat;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

U ovom programu na početku programa, te prije svakog unosa i ispisa rezultata naredbom **cout** ispisali smo adekvatan komentar koji pomaže korisniku da lakše shvati čemu program služi, što mora unijeti i što je dobio kao rezultat.

Ovakav program puno je ugodniji za upotrebu od programa bez komentara.

```
Dijeljenje.
Unesite prvi broj.
12
Unesite drugi broj.
3
Rezultat dijeljenja je:
4
Press any key to continue . . .
```

Prethodni program mogli bismo još malo uljepšati dodavanjem praznih redova. Unesimo i pokrenimo ovaj program.



Nekoliko primjera programa

```
// **********************************
// Program za izračun površine
// **********************************

#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   float stranica_A;
   float stranica_B;
   float povrsina;
   cout << "Povrsina." << endl;
   cout << endl;
   cout << endl;
   cout << "Unesite prvu stranicu." << endl;
   cin >> stranica_A;
   cout << endl;
   cout << "Unesite drugu stranicu." << endl;
   cin >> stranica_B;
   cout << endl;
   povrsina = stranica_A * stranica_B;
   cout << "Povrsina je:" << endl;
   cout << povrsina;
   cout << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Ovo je program za izračun površine pravokutnika. Unijet ćemo ga u računalo i pokrenuti.

Struktura je ista kao i struktura prethodnog programa, ali su prilagođeni komentari i nazivi prostora za spremanje brojeva.

Jedina novost je komentar na početku programa koji se nalazi iza *//* oznaka i opisuje namjenu programa.

```
Povrsina.


Unesite prvu stranicu.
3

Unesite drugu stranicu.
4

Povrsina je:
12

Press any key to continue . . .
```

Komentar započinje *//* oznakom i pomaže nam da se lakše snađemo ako program želimo kasnije koristiti.

Komentirati možemo namjenu programa ili pojedine segmente programa.

Prilikom prevođenja programa, kad prevoditelj naiđe na *//* oznaku, skače u novi red ne prevodeći ostatak reda.

```
// ******************************
// Program za izračun brzine
// ******************************

#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  float put;        // prostor za put
  float vrijeme;  // prostor za vrijeme
  float brzina;    // prostor za brzinu
  cout << "Brzina." << endl;
  cout << endl;
  cout << endl;
  cout << "Unesite prevaljeni put." << endl;
  cin >> put;
  cout << endl;
  cout << "Unesite vrijeme." << endl;
  cin >> vrijeme;
  cout << endl;
  brzina = put / vrijeme;
  cout << "Brzina je:" << endl;
  cout << brzina;
  cout << endl;
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Osim što komentare možemo pisati u zasebnom redu, možemo ih pisati i na kraju naredbi.

Pri tome imajmo na umu da je

float put; // komentar

ispravno, ali

// komentar float put;

ili

float // komentar put;

nije. Dakle, ako komentar unosimo u red s naredbom, komentar mora biti na kraju reda.

```
Brzina.


Unesite prevaljeni put.
200

Unesite vrijeme.
2

Brzina je:
100

Press any key to continue . . .
```

Postavlja se pitanje čemu takvi komentari služe. Ako smo upravo napisali program, a program ima deset redova, onda nam komentari ne trebaju.

Ako napišemo veći program, a želimo ga korigirati nakon dvije godine, onda će nam komentari pojedinih dijelova programa uvelike pomoći pri snalaženju u programu.

Komentari su korisni i ako moramo korigirati program koji je netko drugi pisao.

```
// ***********************************************
// Program za izračun volumena kvadrata
// ***********************************************

#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   float stranica_A;
   float stranica_B;
   float stranica_C;
   float volumen;
   cout << "Izracun volumena." << endl;
   cout << endl;
   cout << "Unesite stranicu a." << endl;
   cin >> stranica_A;
   cout << "Unesite stranicu b." << endl;
   cin >> stranica_B;
   cout << "Unesite stranicu c." << endl;
   cin >> stranica_C;
   cout << endl;
   volumen = stranica_A * stranica_B * stranica_C;
   cout << "Volumen je:" << endl;
   cout << volumen;
   cout << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

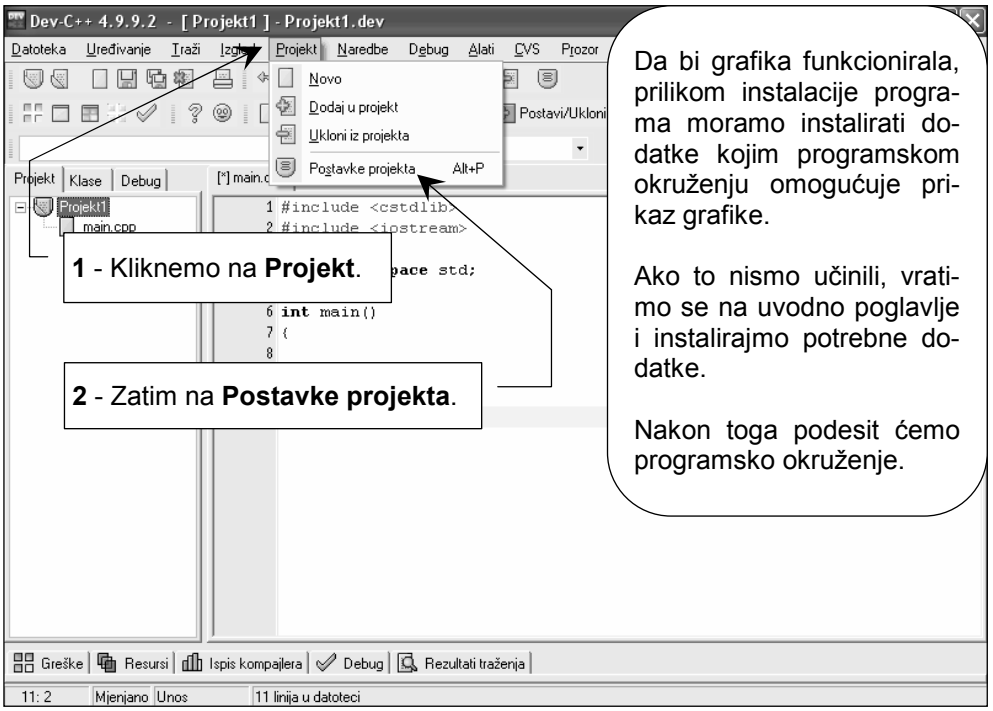
Ovo je primjer programa za izračun volumena kvadrata.

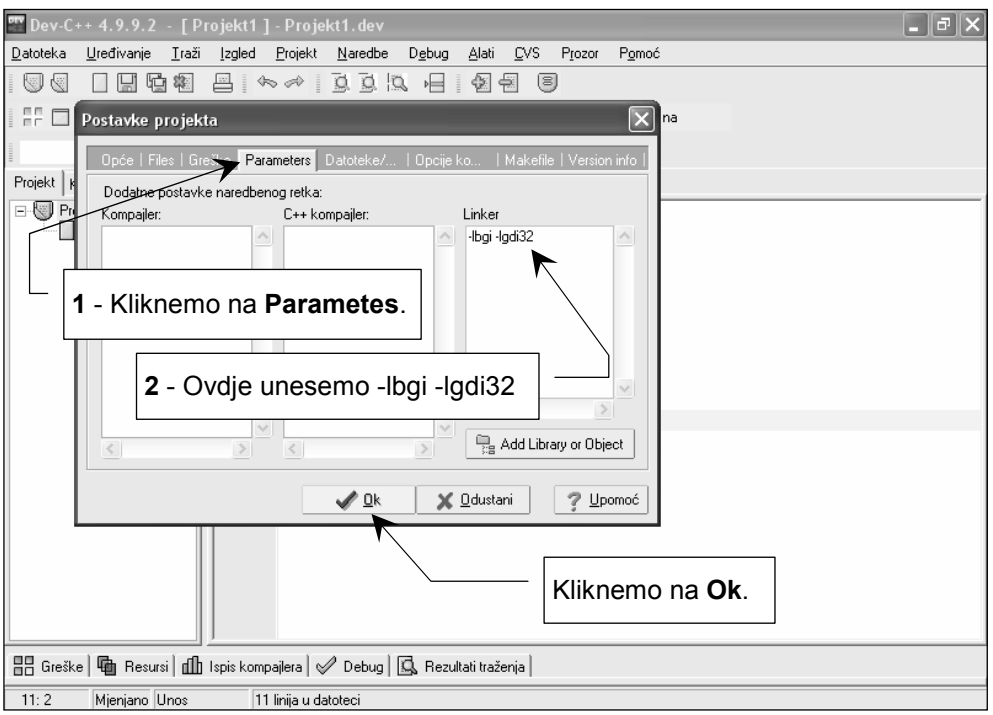
Pokušajmo za vježbu sami smisliti nekoliko sličnih programa.

```
Izracun volumena.

Unesite stranicu a.
2
Unesite stranicu b.
3
Unesite stranicu c.
4

Volumen je:
24

Press any key to continue . . .
```

Grafika

Podešavanje programskog okruženja





Naš prvi grafički program

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   circle(320,240,180);
   getch();
   closegraph();
   return 0;
}
```

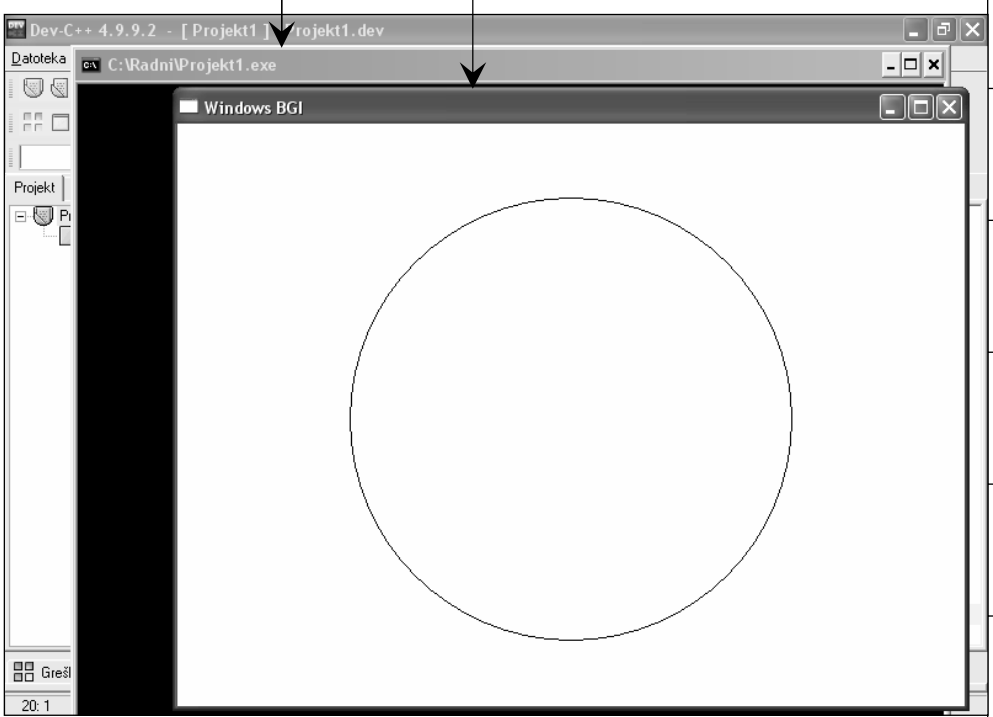
Unesimo i pokrenimo ovaj program. Uočimo da su početak i kraj programa izmijenjeni.

Na početku dodan je ovaj red, a na kraju je izbačena naredba: system("PAUSE");

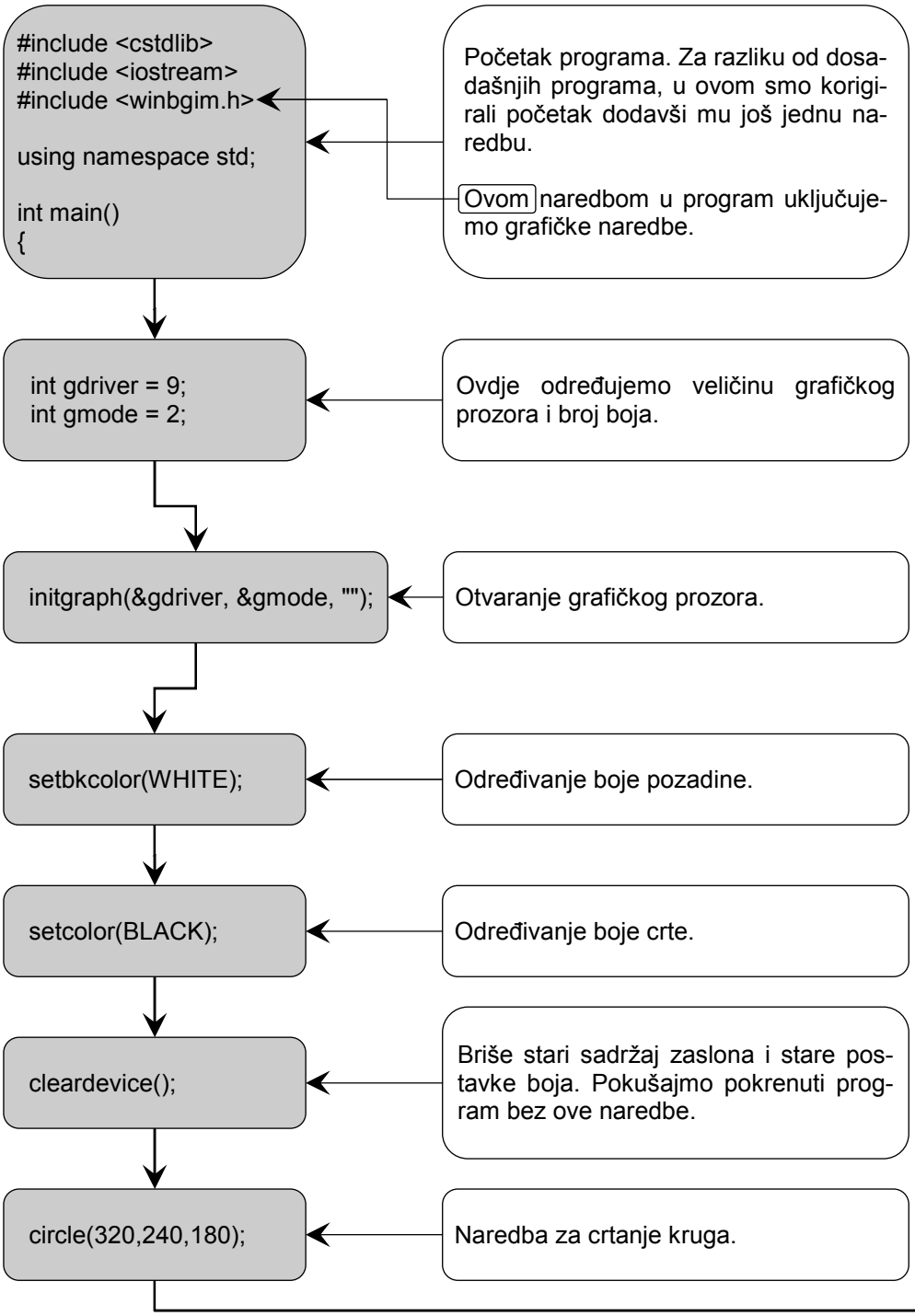
Klikom na **Kompajlaj i pokreni** pokrenemo program.

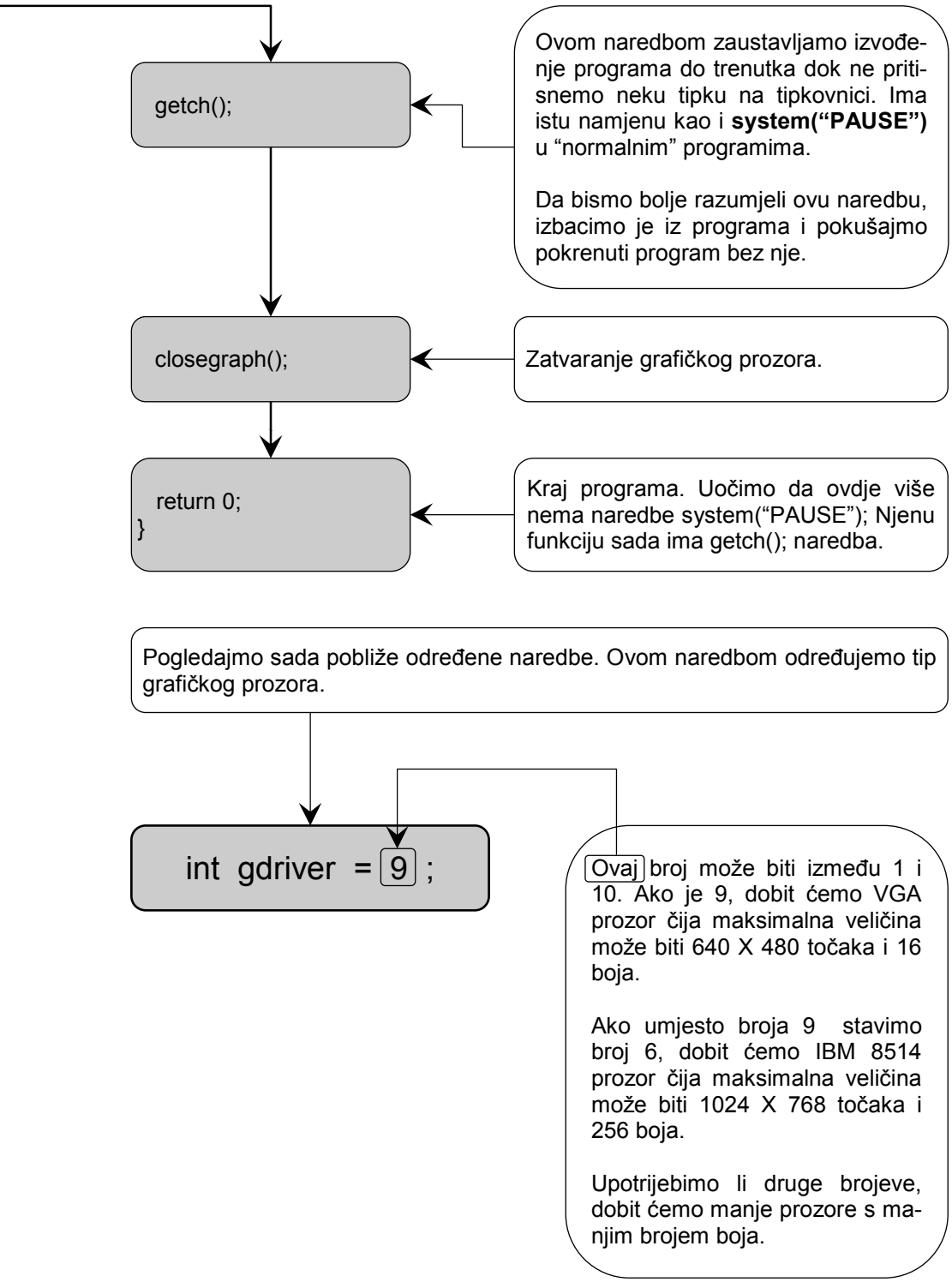
Otvorit će se dva prozora:

- tekstualni koji smo do sada koristili
- grafički u kojem će računalo nacrtati krug.



Analiza programa




getch();
Ovom naredbom zaustavljamo izvođenje programa do trenutka dok ne pritisnemo neku tipku na tipkovnici. Ima istu namjenu kao i **system("PAUSE")** u "normalnim" programima.
Da bismo bolje razumjeli ovu naredbu, izbacimo je iz programa i pokušajmo pokrenuti program bez nje.
closegraph();
Zatvaranje grafičkog prozora.
return 0;
}
Kraj programa. Uočimo da ovdje više nema naredbe system("PAUSE"); Njenu funkciju sada ima getch(); naredba.
Pogledajmo sada pobliže određene naredbe. Ovom naredbom određujemo tip grafičkog prozora.
int gdriver = 9 ;
Ovaj broj može biti između 1 i 10. Ako je 9, dobit ćemo VGA prozor čija maksimalna veličina može biti 640 X 480 točaka i 16 boja.
Ako umjesto broja 9 stavimo broj 6, dobit ćemo IBM 8514 prozor čija maksimalna veličina može biti 1024 X 768 točaka i 256 boja.
Upotrijebimo li druge brojeve, dobit ćemo manje prozore s manjim brojem boja.

Nakon što smo naredbom **int gdriver = 9;** odabrali tip grafičkog prozora, ovom naredbom biramo rezoluciju grafičkog prozora.

```
int gmode = 2 ;
```

Značenje ovog broja ovisi o tipu grafičkog prozora koji smo odabrali.

Ako smo naredbom int gdriver = 9; odabrali VGA prozor, stavimo li ovdje broj 0, dobit ćemo prozor veličine 640 X 200 točaka. Stavimo li 1, dobit ćemo prozor veličine 640 X 350 točaka, a stavimo li broj 2 koji smo mi odabrali, dobit ćemo prozor veličine 640 X 480 točaka.

Ako smo naredbom int gdriver = 6 ; odabrali IBM 8514 prozor, stavimo li ovdje broj 0, dobit ćemo prozor veličine 640 X 480 točaka, a stavimo li broj 1, dobit ćemo prozor veličine 1024 X 768 točaka.

Ovom naredbom određuje se boja pozadine. U našem slučaju bit će bijela.

```
setbkcolor( WHITE );
```

Ovdje unosimo naziv boje. Naziv boje može biti:

BLACK,
BLUE,
GREEN,
CYAN,
RED,
MAGENTA,
BROWN,
LIGHTGRAY,
DARKGRAY,
LIGHTBLUE,
LIGHTGREEN,
LIGHTCYAN,
LIGHTRED,
LIGHTMAGENTA,
YELLOW,
WHITE.

Osim navođenja imena boja, boje možemo odrediti i navođenjem njihove brojčane vrijednosti.

```
setbkcolor(COLOR( 255, 255, 255 ));
```

Plava.

Crvena.

Zelena.

Dodat ćemo naredbu COLOR iza koje se nalaze tri broja čija vrijednost može biti između 0 i 255.

Boje u računalu sastoje se od tri boje; crvene, zelene i plave. Sve ostale boje nastaju miješanjem tih boja.

Prvim brojem određujemo intenzitet crvene boje, drugim intenzitet zelene i trećim intenzitet plave boje.

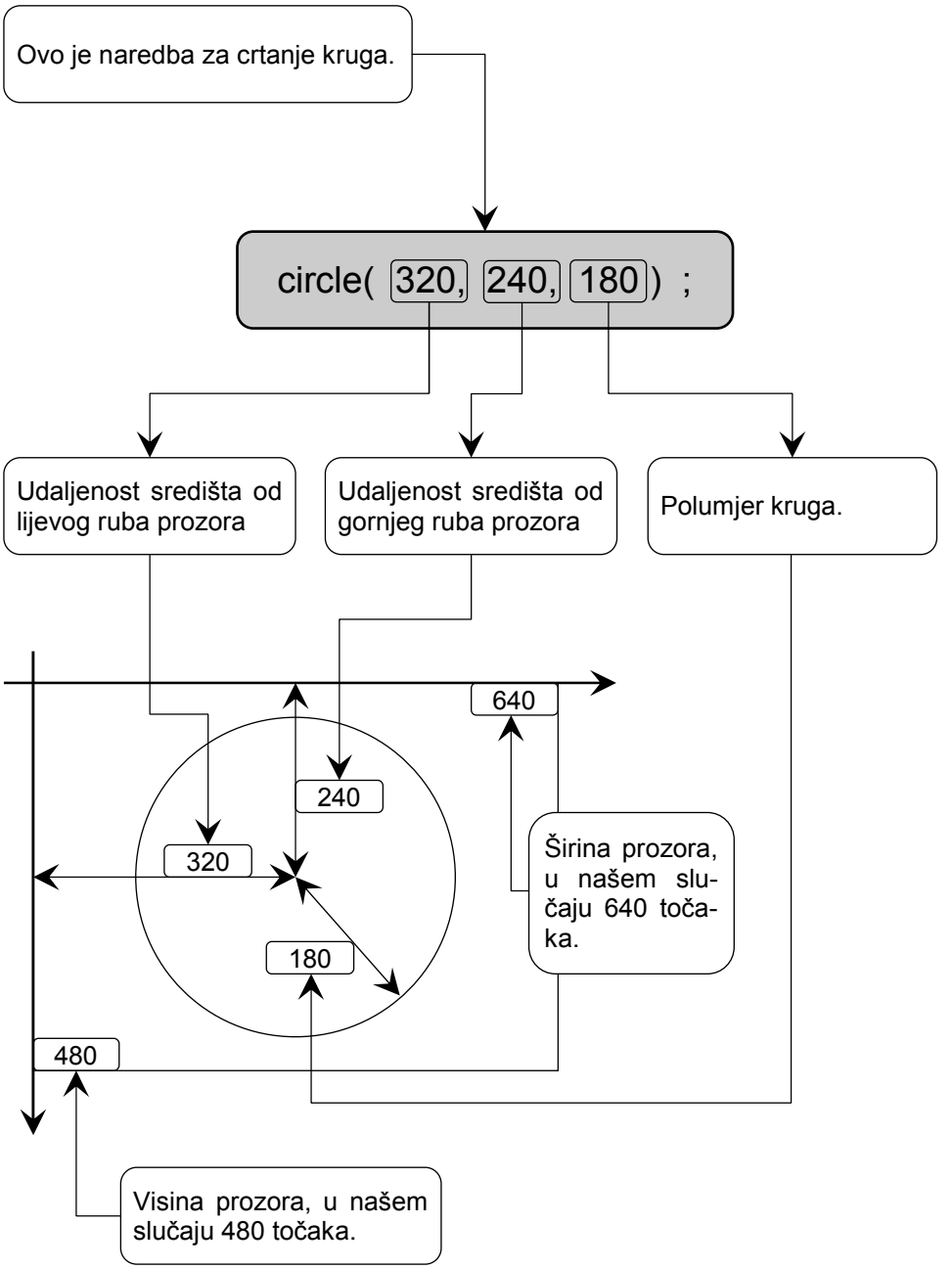
Naredbom setbkcolor(COLOR(255,255,255)); dobit ćemo bijelu boju, a naredbom setbkcolor(COLOR(0,0,0)); dobit ćemo crnu boju.

Ovom naredbom određuje se boja kojom crtamo. Koristimo iste nazive boja kao i u prethodnoj naredbi.

```
setcolor(BLACK);
```

```
setcolor(COLOR (0,0,0));
```

Kao i u prethodnoj naredbi, boje možemo odrediti navođenjem njihovog naziva, a možemo i pomoću brojeva upotrebom naredbe COLOR.


Ovo je naredba za crtanje kruga.
circle(320, 240, 180) ;
Udaljenost središta od lijevog ruba prozora
Udaljenost središta od gornjeg ruba prozora
Polumjer kruga.
640
240
320
180
480
Širina prozora, u našem slučaju 640 točaka.
Visina prozora, u našem slučaju 480 točaka.

Varijacije programa

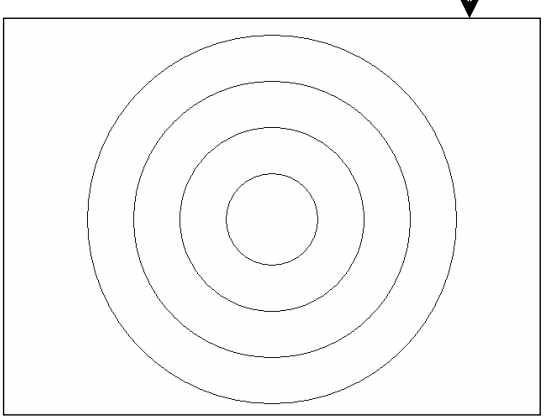
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   circle(320,240,220);
   circle(320,240,165);
   circle(320,240,110);
   circle(320,240,55);
   getch();
   closegraph();
   return 0;
}
```

Napravit ćemo nekoliko primjera da bismo lakše razumjeli kako radi naredba za crtanje kruga.

Ovaj program sličan je prethodnom. Razlika je u tome što je prethodni crtao samo jedan krug, a ovaj će nacrtati nekoliko krugova različitog polumjera, ali sa središtem u istoj točki.

Ovdje se nalaze četiri naredbe za crtanje četiri kruga.

Pokrenemo li ovaj program, trebali bismo dobiti ovakav rezultat.

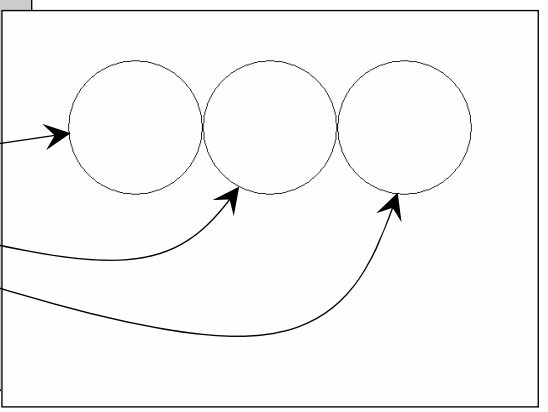


Pokušajmo program promijeniti tako da povećamo ili smanjimo broj krugova.

Pokušajmo promijeniti brojeve koji određuju polumjer kruga i položaj središta kruga.

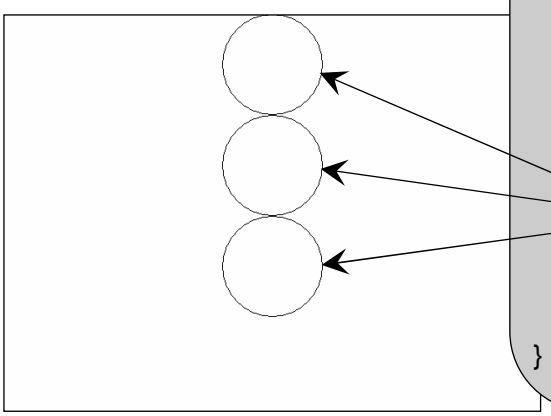
U ovom primjeru napravit ćemo tri jednaka kruga, jedan pored drugoga, što ćemo postići promjenom udaljenosti središta krugova od lijevog ruba prozora.

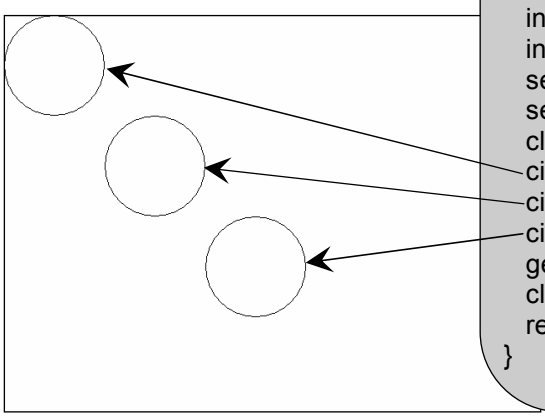
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   circle(160,140,80);
   circle(320,140,80);
   circle(480,140,80);
   getch();
   closegraph();
   return 0;
}
```



Pogledajmo još nekoliko mogućnosti. U ovom slučaju promjenom udaljenosti od gornjeg ruba prozora dobili smo tri kruga, jedan iznad drugoga.

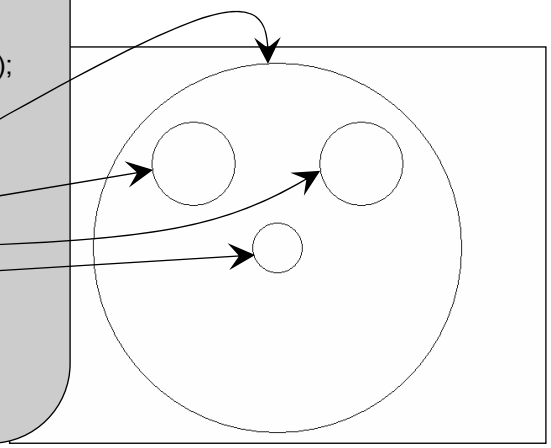
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   circle (320, 60, 60);
   circle (320, 180, 60);
   circle (320, 300, 60);
   getch();
   closegraph();
   return 0;
}
```





```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   circle (60, 60, 60);
   circle (180, 180, 60);
   circle (300, 300, 60);
   getch();
   closegraph();
   return 0;
}
```

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   circle(320,240,220);
   circle(220,140,50);
   circle(420,140,50);
   circle(320,240,30);
   getch();
   closegraph();
   return 0;
}
```

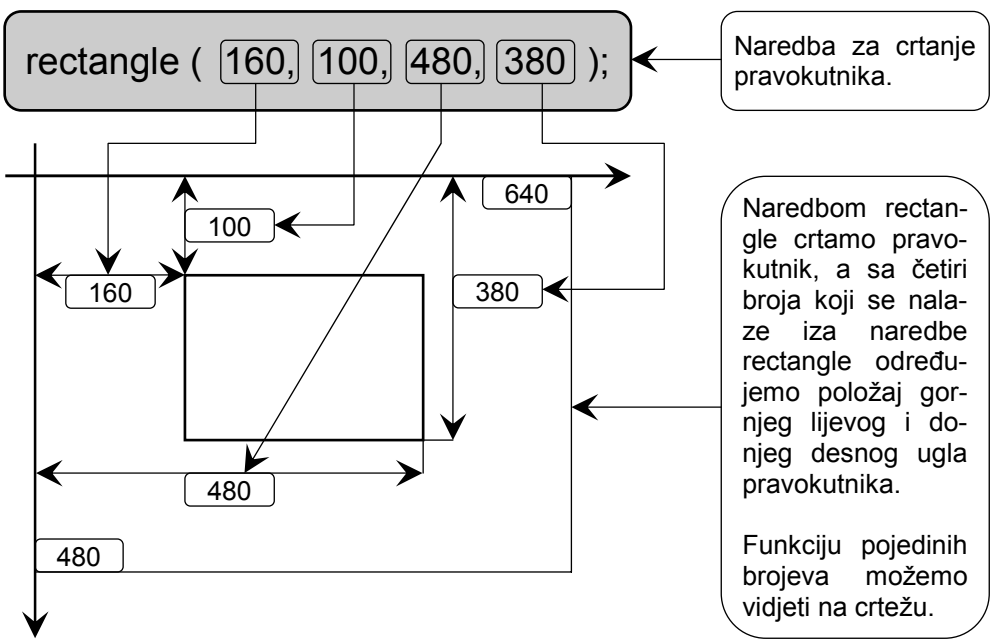


Crtanje pravokutnika

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   rectangle (160, 100, 480, 380);
   getch();
   closegraph();
   return 0;
}
```

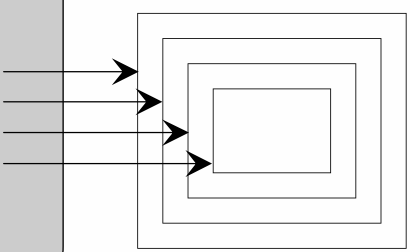
Osim kružnice možemo nacrtati i druge oblike, npr. pravokutnik.

Ovdje u program unesemo naredbu za crtanje pravokutnika, a na zaslonu računala dobit ćemo ovakav rezultat.



```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   rectangle (160, 100, 480, 380);
   rectangle (190, 130, 450, 350);
   rectangle (220, 160, 420, 320);
   rectangle (250, 190, 390, 290);
   getch();
   closegraph();
   return 0;
}
```

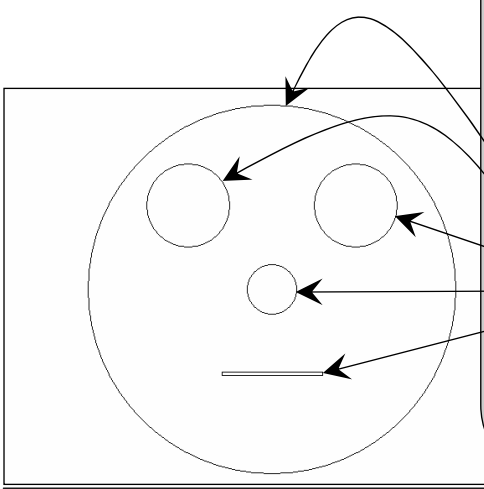
Međusobno možemo kombinirati više pravokutnika.



Pravokutnik možemo kombinirati s drugim oblicima.

Pokušajmo za vježbu sami izmisliti nekoliko primjera, npr. deblo stabla nacrtamo kao pravokutnik, a krošnju kao krug.

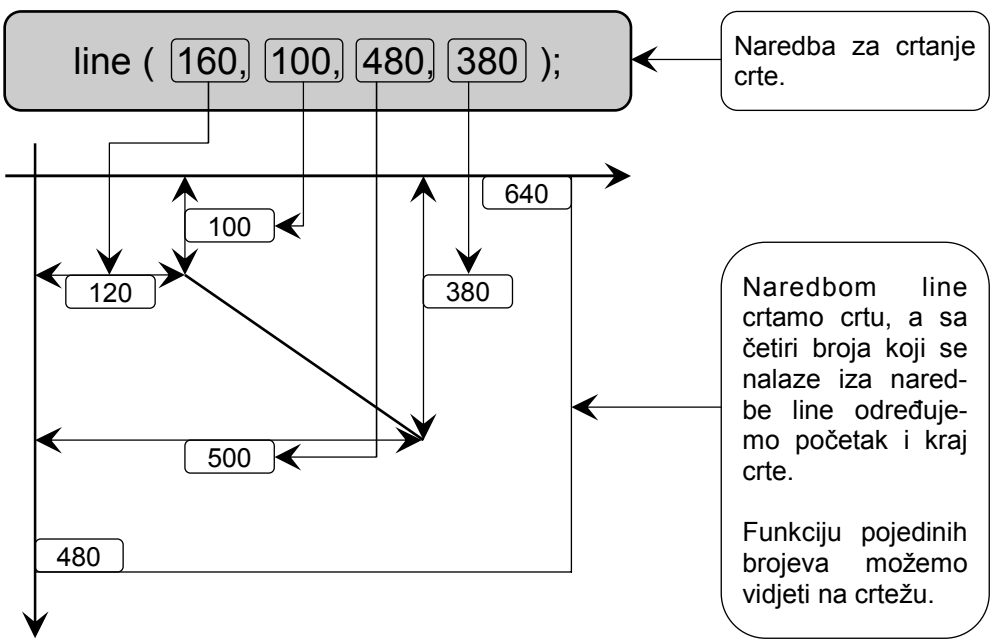
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   circle (320, 240, 220);
   circle (220, 140, 50);
   circle (420, 140, 50);
   circle (320, 240, 30);
   rectangle (260, 342, 380, 338);
   getch();
   closegraph();
   return 0;
}
```



Crtanje crte

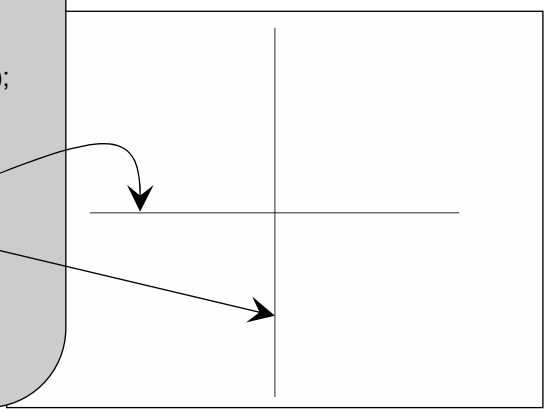
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   line (120, 100, 500, 380);
   getch();
   closegraph();
   return 0;
}
```

Crtu možemo nacrtati na više načina, a uporaba naredbe line jedan je od njih.

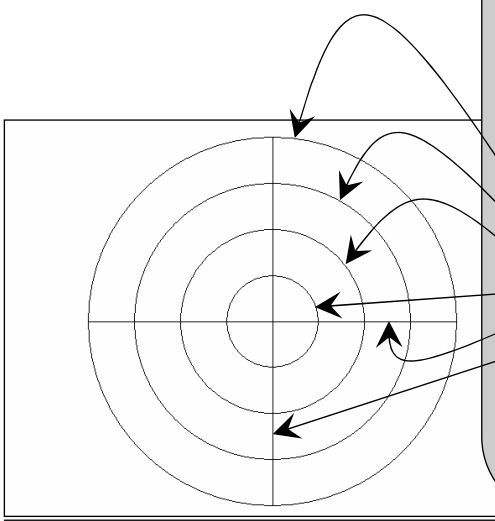


Napravimo nekoliko crteža uporabom naredbe za crtanje crte.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   line (100, 240, 540, 240);
   line (320, 20, 320, 460);
   getch();
   closegraph();
   return 0;
}
```



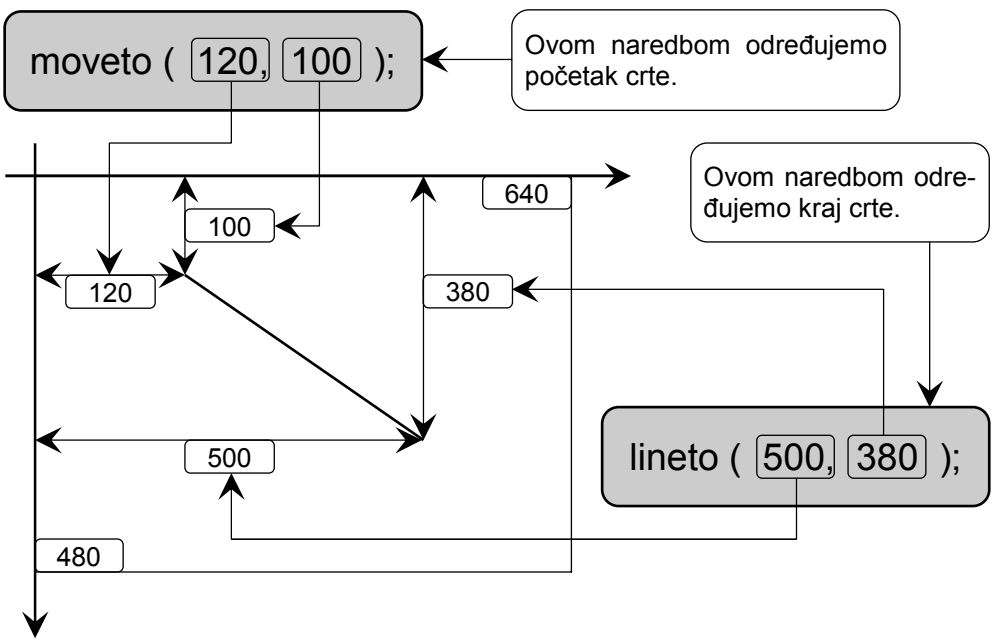
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   circle (320, 240, 220);
   circle (320, 240, 165);
   circle (320, 240, 110);
   circle (320, 240, 55);
   line (100, 240, 540, 240);
   line (320, 20, 320, 460);
   getch();
   closegraph();
   return 0;
}
```



```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   moveto (120, 100);
   lineto (500, 380);
   getch();
   closegraph();
   return 0;
}
```

Osim uporabom naredbe line, crtu možemo nacrtati i uporabom naredbi **moveto** i **lineto**.

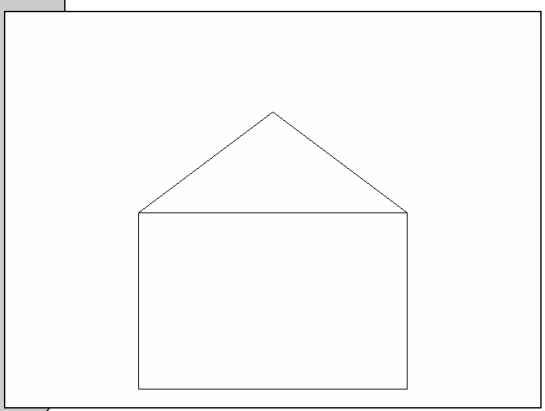
Naredbom **moveto** određujemo početak crte, a naredbom **lineto** određujemo kraj crte.



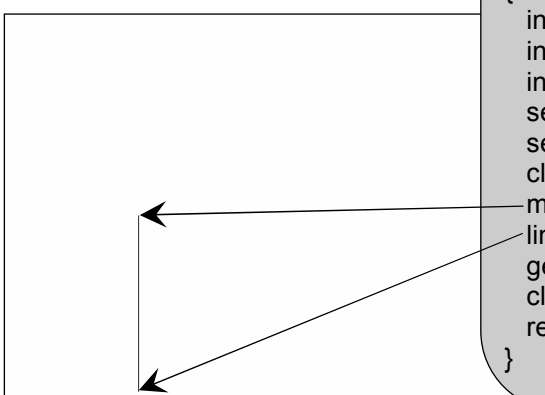
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   moveto(160, 240);
   lineto(160, 450);
   lineto(480, 450);
   lineto(480, 240);
   lineto(160, 240);
   lineto(320, 120);
   lineto(480, 240);
   getch();
   closegraph();
   return 0;
}
```

Ovaj način crtanja pogodan je za crtanje više crta u neprekinutom nizu.

Pogledajmo ovaj primjer.

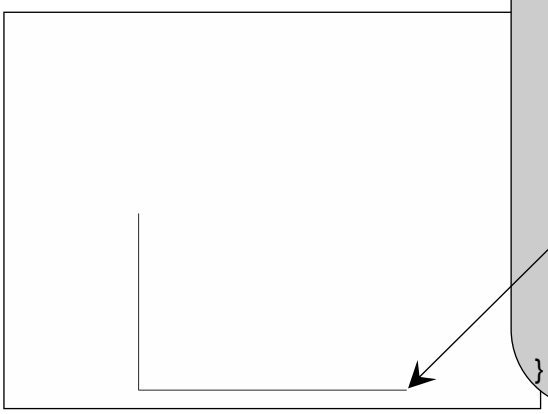


Pogledajmo, korak po korak, kako je nastao ovaj crtež.

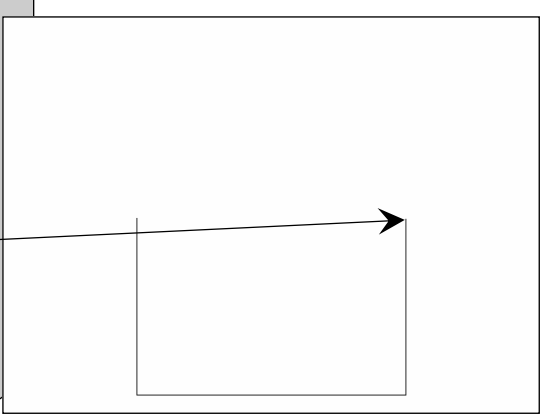


```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   moveto(160, 240);
   lineto(160, 450);
   getch();
   closegraph();
   return 0;
}
```

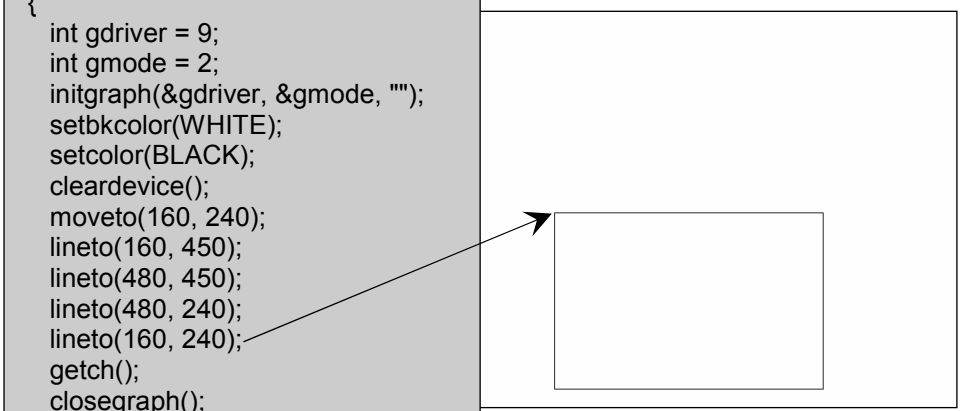
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   moveto(160, 240);
   lineto(160, 450);
   lineto(480, 450);
   getch();
   closegraph();
   return 0;
}
```



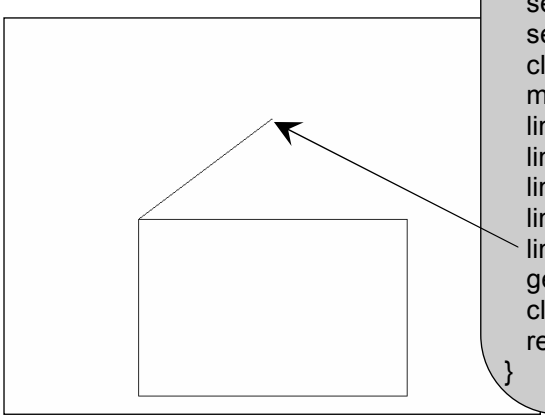
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   moveto(160, 240);
   lineto(160, 450);
   lineto(480, 450);
   lineto(480, 240);
   getch();
   closegraph();
   return 0;
}
```

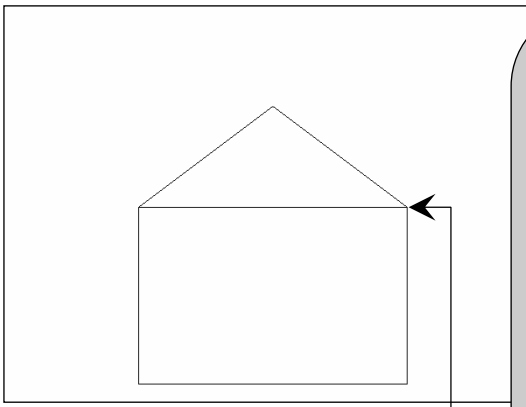


```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   moveto(160, 240);
   lineto(160, 450);
   lineto(480, 450);
   lineto(480, 240);
   lineto(160, 240);
   getch();
   closegraph();
   return 0;
}
```



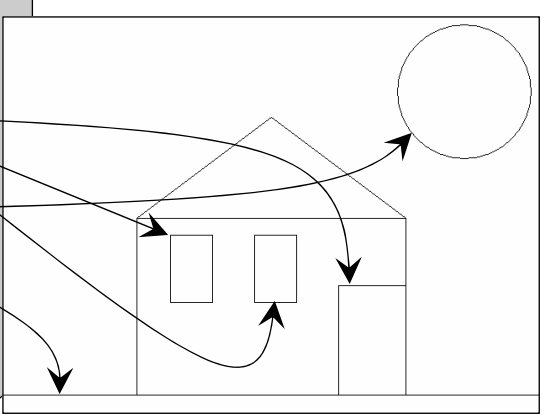
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   moveto(160, 240);
   lineto(160, 450);
   lineto(480, 450);
   lineto(480, 240);
   lineto(160, 240);
   lineto(320, 120);
   getch();
   closegraph();
   return 0;
}
```





```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   moveto(160, 240);
   lineto(160, 450);
   lineto(480, 450);
   lineto(480, 240);
   lineto(160, 240);
   lineto(320, 120);
   lineto(480, 240);
   getch();
   closegraph();
   return 0;
}
```

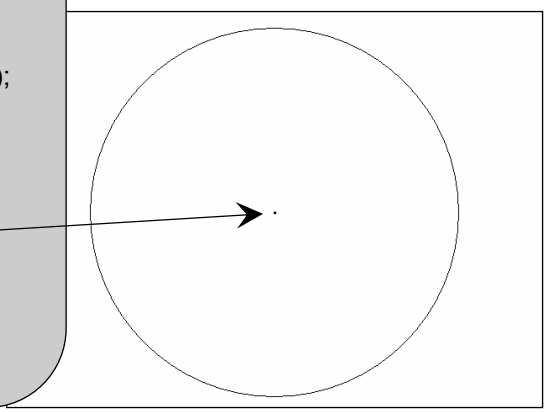
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   moveto(160, 240);
   lineto(160, 450);
   lineto(480, 450);
   lineto(480, 240);
   lineto(160, 240);
   lineto(320, 120);
   lineto(480, 240);
   rectangle(400, 320, 480, 450);
   rectangle(200, 260, 250, 340);
   rectangle(300, 260, 350, 340);
   circle (550, 90, 80);
   line (0, 450, 640, 450);
   getch();
   closegraph();
   return 0;
}
```



Crtanje točke

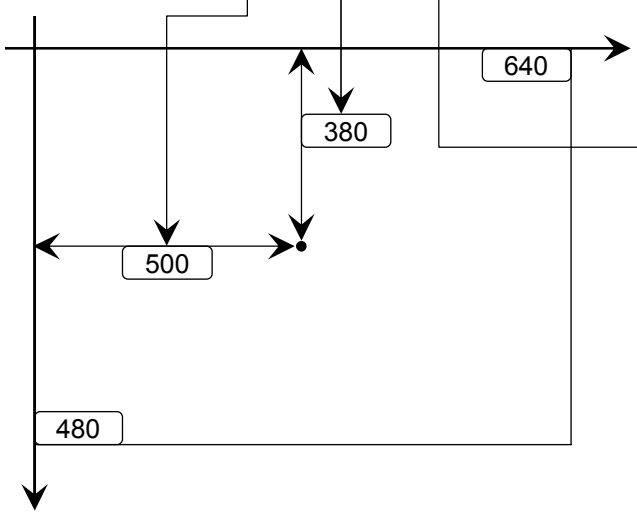
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   circle (320, 240, 220);
   putpixel (320, 240, BLACK);
   getch();
   closegraph();
   return 0;
}
```

Ponekad nam je potrebna samo jedna točka, kao u ovom primjeru gdje smo točkom označili središte kruga.



putpixel (320, 240, BLACK);

Naredba za crtanje točke.



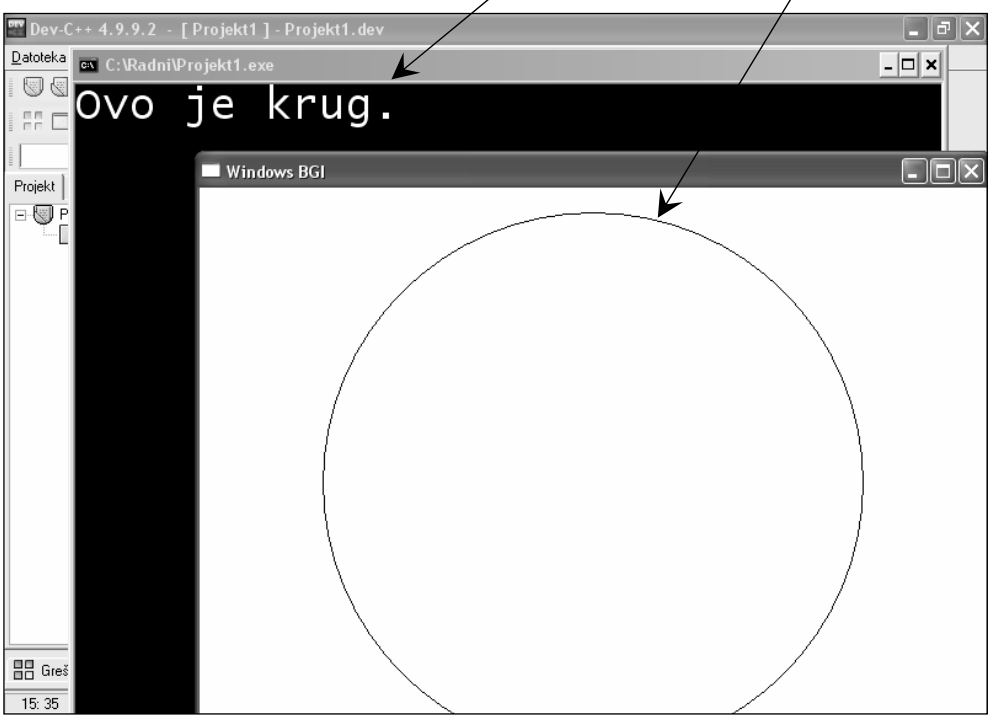
Ovdje unosimo boju točke koju određujemo na isti način kao i boju pozadine ili boju crte.

Ispis teksta

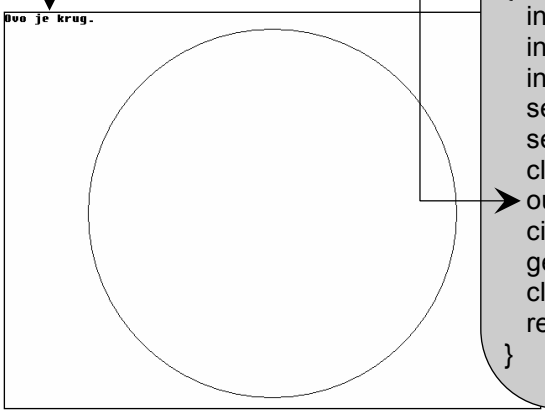
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   cout << "Ovo je krug." << endl;
   circle (320, 240, 220);
   getch();
   closegraph();
   return 0;
}
```

Možemo li unutar grafičkog prozora staviti tekst? Možemo, ali nažalost, ne na način na koji smo to do sada navikli raditi.

Ako prije naredbe za crtanje kruga stavimo naredbu i za ispis teksta, tekst će se ispisati, ali ne unutar grafičkog prozora, već unutar tekstualnog prozora.



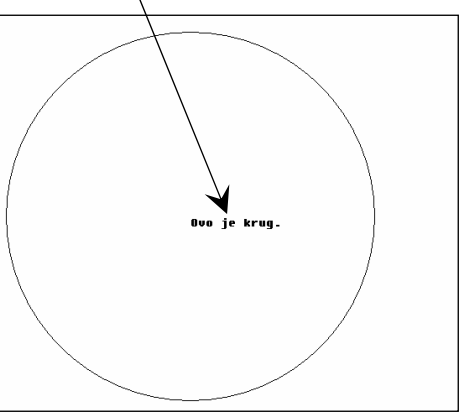
Upotrijebimo li ovu naredbu, tekst će se ispisati unutar grafičkog prozora.




```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   outtext ("Ovo je krug.");
   circle (320, 240, 220);
   getch();
   closegraph();
   return 0;
}
```

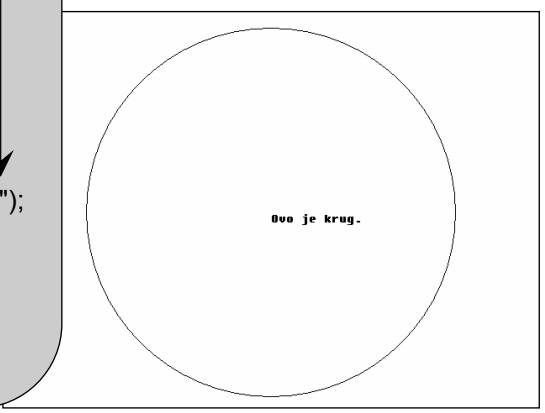
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   moveto (320, 240);
   outtext ("Ovo je krug.");
   circle (320, 240, 220);
   getch();
   closegraph();
   return 0;
}
```

Naredbu za određivanje pozicije početka crte možemo iskoristiti za određivanje početka teksta.



```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   outtextxy (320, 240, "Ovo je krug.");
   circle (320, 240, 220);
   getch();
   closegraph();
   return 0;
}
```

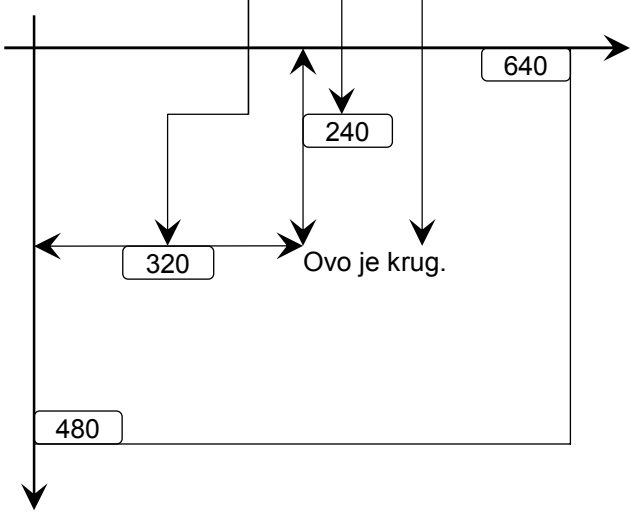
Naredba **outtextxy** objedinjuje naredbu za određivanje pozicije teksta i naredbu za ispis teksta.



Brojevima određujemo poziciju ispisa.

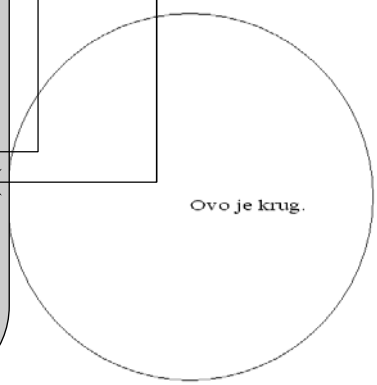
Unutar navodnika je tekst koji ispisujemo.

```
outtextxy ( 320, 240, "Ovo je krug." );
```



```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   settextstyle (9, HORIZ_DIR, 1);
   outtextxy (320, 240, "Ovo je krug.");
   circle (320, 240, 220);
   getch();
   closegraph();
   return 0;
}
```

Naredbom **settextstyle** možemo mijenjati oblik i veličinu slova te smjer ispisa teksta. Stavljamo je prije naredbe za ispis teksta.



Naredba za određivanje veličine i oblika slova te smjera ispisa.

```
settextstyle ( 9, HORIZ_DIR, 1 );
```

Ovaj broj određuje oblik slova. Njegova vrijednost može biti između 0 i 10.

Veličina slova.

Smjer teksta:

HORIZ_DIR - vodoravni tekst.

VERT_DIR - uspravni tekst od dolje prema gore.

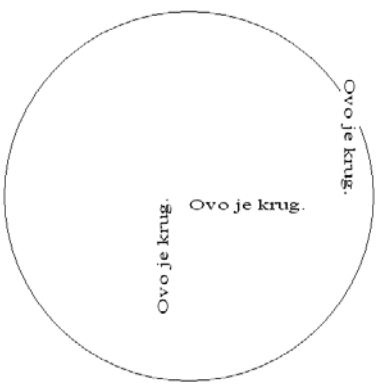
-VERT_DIR - uspravni tekst od gore prema dolje.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   circle (320, 240, 220);
   settextstyle(9, HORIZ_DIR, 1);
   outtextxy(320, 200, "Ovo je krug.");
   settextstyle(9, HORIZ_DIR, 2);
   outtextxy(320, 240, "Ovo je krug.");
   settextstyle(9, HORIZ_DIR, 3);
   outtextxy(320, 280, "Ovo je krug.");
   settextstyle(9, HORIZ_DIR, 4);
   outtextxy(320, 320, "Ovo je krug.");
   getch();
   closegraph();
   return 0;
}
```

Pogledajmo primjer promjene veličine slova.

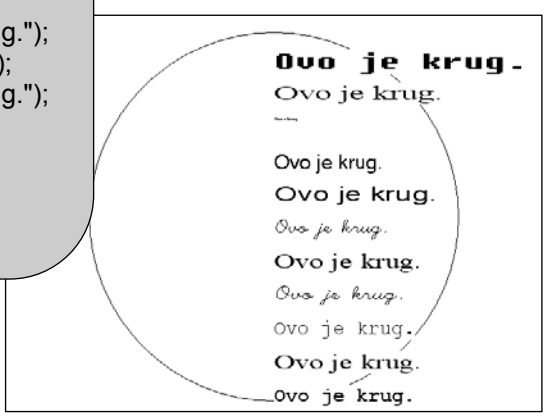
Pogledajmo primjer promjene smjera ispisa teksta.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   circle (320, 240, 220);
   settextstyle(9, HORIZ_DIR, 1);
   outtextxy(320, 240, "Ovo je krug.");
   settextstyle(9, VERT_DIR, 1);
   outtextxy(300, 240, "Ovo je krug.");
   settextstyle(9, -VERT_DIR, 1);
   outtextxy(500, 240, "Ovo je krug.");
   getch();
   closegraph();
   return 0;
}
```



Pogledajmo primjer promjene oblika slova.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor(WHITE);
   setcolor(BLACK);
   cleardevice();
   circle (320, 240, 220);
   settextstyle(0, HORIZ_DIR, 3);
   outtextxy(320, 40, "Ovo je krug.");
   settextstyle(1, HORIZ_DIR, 3);
   outtextxy(320, 80, "Ovo je krug.");
   settextstyle(2, HORIZ_DIR, 3);
   outtextxy(320, 120, "Ovo je krug.");
   settextstyle(3, HORIZ_DIR, 3);
   outtextxy(320, 160, "Ovo je krug.");
   settextstyle(4, HORIZ_DIR, 3);
   outtextxy(320, 200, "Ovo je krug.");
   settextstyle(5, HORIZ_DIR, 3);
   outtextxy(320, 240, "Ovo je krug.");
   settextstyle(6, HORIZ_DIR, 3);
   outtextxy(320, 280, "Ovo je krug.");
   settextstyle(7, HORIZ_DIR, 3);
   outtextxy(320, 320, "Ovo je krug.");
   settextstyle(8, HORIZ_DIR, 3);
   outtextxy(320, 360, "Ovo je krug.");
   settextstyle(9, HORIZ_DIR, 3);
   outtextxy(320, 400, "Ovo je krug.");
   settextstyle(10, HORIZ_DIR, 3);
   outtextxy(320, 440, "Ovo je krug.");
   getch();
   closegraph();
   return 0;
}
```



Donošenje odluke

Funkcije

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
   float broj;
   float potencija;
   float rezultat;
   cout << "Unesite broj:" << endl;
   cin >> broj;
   cout << endl;
   cout << "Unesite potenciju:" << endl;
   cin >> potencija;
   cout << endl;
   rezultat = pow(broj,potencija);
   cout << "Rezultat je:" << endl;
   cout << rezultat << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Pojam funkcije zvuči jako pametno, ali iza tog naziva kriju se uglavnom jednostavne stvari, matematičke operacije poput potenciranja ili vađenja kvadratnog korijena.

Pogledajmo program za potenciranje.

Da bismo u programu mogli koristiti matematičke funkcije, moramo u program uključiti biblioteku matematičkih funkcija.

```
Unesite broj:
4

Unesite potenciju:
2

Rezultat je:
16

Press any key to continue . . .
```

Varijabla u koju spremamo rezultat potenciranja.

```
rezultat = pow ( broj, potencija );
```

Potencija na koju potenciramo.

Naredba za potenciranje.

Broj koji potenciramo.

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
   float kut;
   float rezultat;
   cout << "Kut u radijanima:" << endl;
   cin >> kut;
   cout << endl;
   rezultat = sin(kut);
   cout << "Sinus kuta je:" << endl;
   cout << rezultat << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Ovo je program za izračun sinusa.

Ako ne znate što je to sinus, nemojte se time opterećivati. To nije bitno za razumijevanje C++ programskog jezika.

Važno je uočiti da ova funkcija veličinu kuta ne izražava stupnjevima nego radijanima.

```
Kut u radijanima:
1.57

Sinus kuta je:
1

Press any key to continue . . .
```

Varijabla u koju spremamo rezultat.

```
rezultat = sin ( kut );
```

Naredba za izračun sinusa kuta.

Varijabla u koju spremamo veličinu kuta kojem ćemo računati sinus. Kut mora biti izražen u radijanima, a ne u stupnjevima.

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
   float broj;
   float rezultat;
   cout << "Unesite negativni broj:" << endl;
   cin >> broj;
   cout << endl;
   rezultat = abs(broj);
   cout << "Broj bez minusa:" << endl;
   cout << rezultat << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Ponekad je u programu potrebno negativan broj pretvoriti u pozitivan.

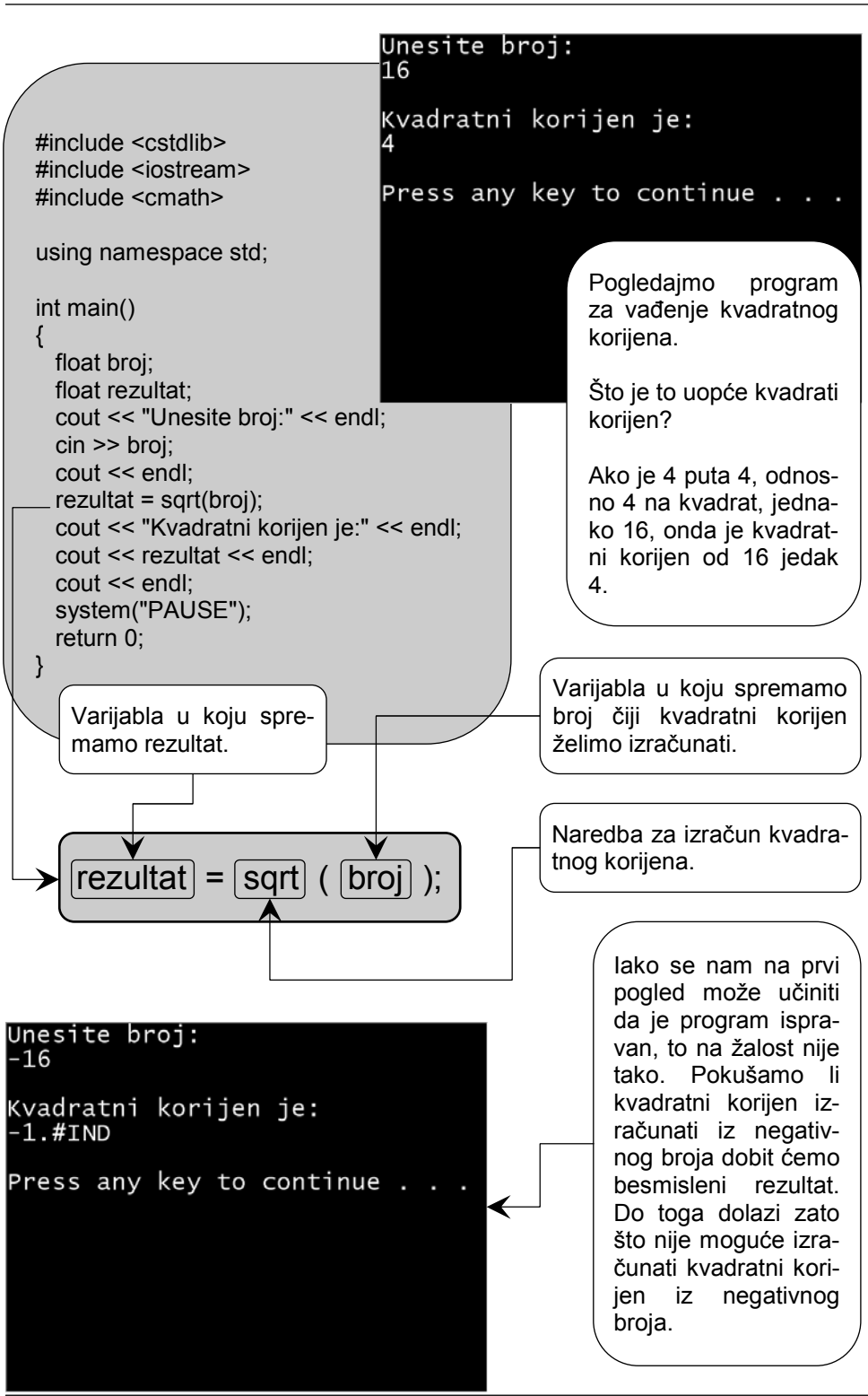
```
Unesite negativni broj:
-12

Broj bez minusa:
12

Press any key to continue . . .
```

Varijabla u koju spremamo rezultat.

```
rezultat = abs ( broj );
```

Naredba za pretvaranje negativnog broja u pozitivan broj.

Varijabla u koju spremamo broj koji ćemo iz negativnog pretvoriti u pozitivan.


Unesite broj:
16
Kvadratni korijen je:
4
Press any key to continue . . .
#include <cstdlib>
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
float broj;
float rezultat;
cout << "Unesite broj:" << endl;
cin >> broj;
cout << endl;
rezultat = sqrt(broj);
cout << "Kvadratni korijen je:" << endl;
cout << rezultat << endl;
cout << endl;
system("PAUSE");
return 0;
}
Pogledajmo program za vađenje kvadratnog korijena.
Što je to uopće kvadrati korijen?
Ako je 4 puta 4, odnosno 4 na kvadrat, jednako 16, onda je kvadratni korijen od 16 jedak 4.
Varijabla u koju spremamo rezultat.
Varijabla u koju spremamo broj čiji kvadratni korijen želimo izračunati.
rezultat = sqrt (broj);
Naredba za izračun kvadratnog korijena.
Unesite broj:
-16
Kvadratni korijen je:
-1.#IND
Press any key to continue . . .
Iako se nam na prvi pogled može učiniti da je program ispravan, to na žalost nije tako. Pokušamo li kvadratni korijen izračunati iz negativnog broja dobit ćemo besmisleni rezultat. Do toga dolazi zato što nije moguće izračunati kvadratni korijen iz negativnog broja.

Donošenje odluke

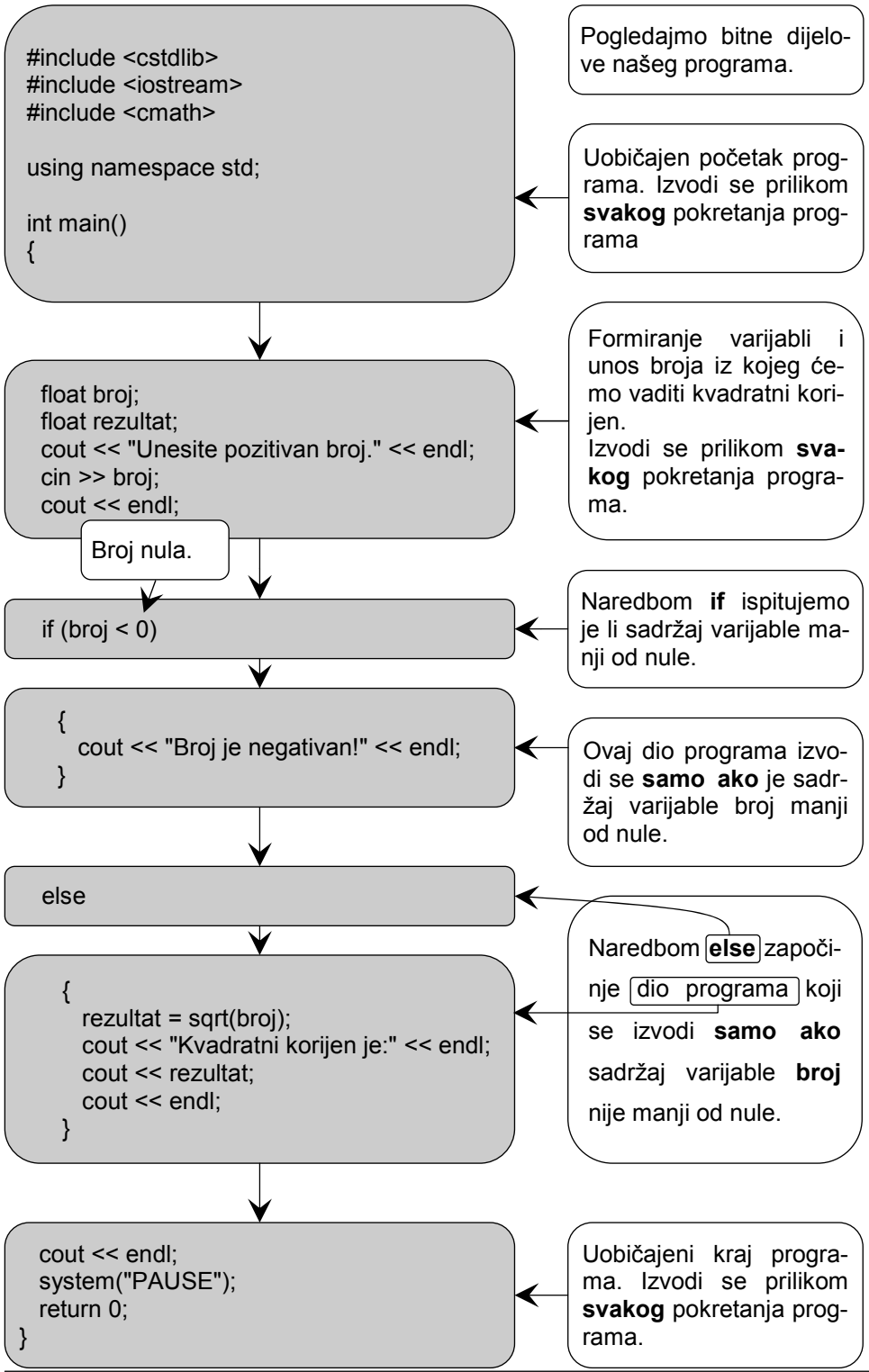
Pokušajmo kvadrati korijen izračunati pomoću ovakvog programa.

```
Unesite pozitivan broj.
16

Kvadratni korijen je:
4

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  float broj;
  float rezultat;
  cout << "Unesite pozitivan broj." << endl;
  cin >> broj;
  cout << endl;
  if (broj < 0)
    {
      cout << "Broj je negativan!" << endl;
    }
  else
    {
      rezultat = sqrt(broj);
      cout << "Kvadratni korijen je:" << endl;
      cout << rezultat;
      cout << endl;
    }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Ova inačica programa izračunava kvadratni korijen ako smo unijeli pozitivan broj ili nas obavještava da je unesen negativan broj ako unesemo negativan broj.

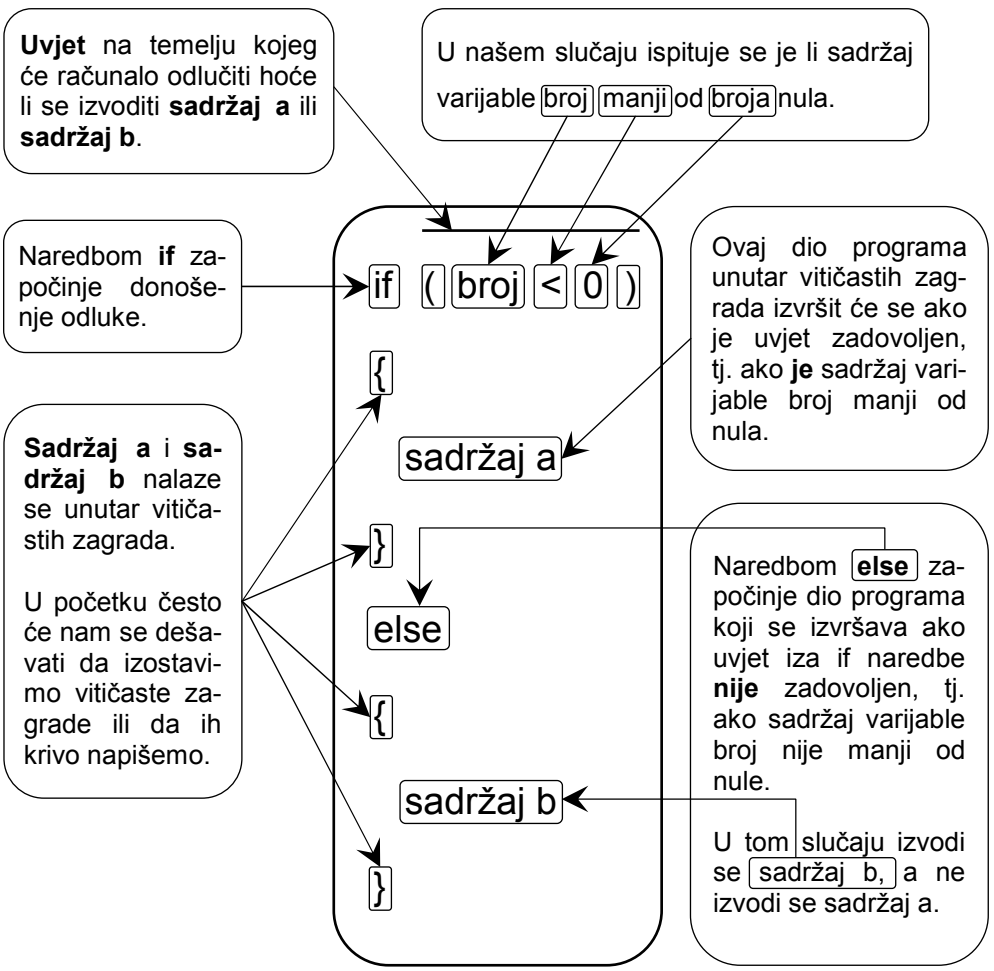
```
Unesite pozitivan broj.
-16

Broj je negativan!

Press any key to continue . . .
```


#include <cstdlib>
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
Pogledajmo bitne dijelove našeg programa.
Uobičajen početak programa. Izvodi se prilikom **svakog** pokretanja programa
float broj;
float rezultat;
cout << "Unesite pozitivan broj." << endl;
cin >> broj;
cout << endl;
Formiranje varijabli i unos broja iz kojeg ćemo vaditi kvadratni korijen.
Izvodi se prilikom **svakog** pokretanja programa.
Broj nula.
if (broj < 0)
Naredbom **if** ispitujemo je li sadržaj varijable manji od nule.
{
cout << "Broj je negativan!" << endl;
}
Ovaj dio programa izvodi se **samo ako** je sadržaj varijable broj manji od nule.
else
Naredbom **else** započinje dio programa koji se izvodi **samo ako** sadržaj varijable **broj** nije manji od nule.
{
rezultat = sqrt(broj);
cout << "Kvadratni korijen je:" << endl;
cout << rezultat;
cout << endl;
}
cout << endl;
system("PAUSE");
return 0;
}
Uobičajeni kraj programa. Izvodi se prilikom **svakog** pokretanja programa.

Svi programi koje smo do sada vidjeli izvodili su se redom od prve do posljednje naredbe. U slučaju programa za vađenje kvadratnog korijena to nam ne odgovara jer želimo da program na jedan način reagira ako unesemo pozitivan broj, a na drugi način ako unesemo negativan.

Ako unesemo pozitivan broj, izračunat ćemo kvadratni korijen i nećemo ispisivati obavijest da je broj negativan, a ako unesemo negativan broj, ispisat ćemo obavijest da je broj negativan i nećemo računati kvadratni korijen, budući da iz negativnog broja nije moguće izračunati kvadratni korijen.

Za razliku od dosadašnjih naredbi koje su se nalazile u jednom redu, naredba koja nam omogućuje izvođenje jednog ili drugog dijela programa protežu se kroz više redova. Pogledajmo strukturu tog sustava.



Da bismo lakše shvatili način korištenja **if ... else** naredbi, napravit ćemo niz programa. Ti programi neće raditi ništa osobito korisno, već će im jedina svrha biti uvježbavanje pisanja programa unutar kojih računalo donosi odluku.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  float broj;
  cout << "Unesite broj." << endl;
  cin >> broj;
  cout << endl;
  if (broj > 0)
    {
    cout << "Broj je veci od nule." << endl;
    }
  else
    {
    cout << "Broj nije veci od nule." << endl;
    }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

```
Unesite broj.
6

Broj je veci od nule.

Press any key to continue . . .
```

```
Unesite broj.
-6

Broj nije veci od nule.

Press any key to continue . . .
```

Ovaj program samo ispituje je li unesen broj veći od nule.

Ako je broj veći od nule, izvodi se dio programa koji ispisuje tekst **Broj je veći od nule**.

Ako broj nije veći od nule, izvodi se dio programa koji ispisuje tekst **Broj nije veći od nule**.

U ovom programu ispitivat ćemo je li unesen broj jednak nuli.

Za to ispitivanje koristit ćemo dvostruki znak jednakosti.

```
Unesite broj.
0

Broj je jednak nuli.

Press any key to continue . . .
```

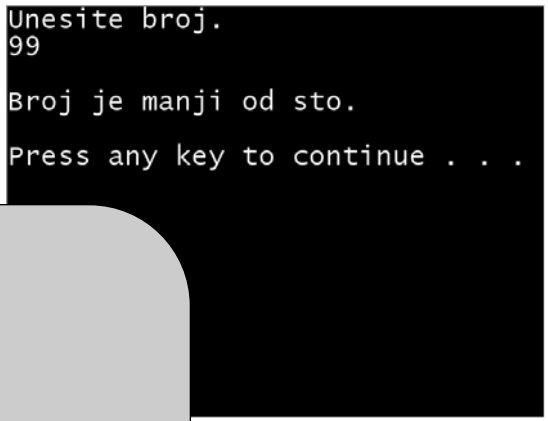
```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   float broj;
   cout << "Unesite broj." << endl;
   cin >> broj;
   cout << endl;

   if (broj == 0)
     {
       cout << "Broj je jednak nuli." << endl;
     }
   else
     {
       cout << "Broj nije jednak nuli." << endl;
     }
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Ako je odgovor na pitanje je li sadržaj varijable broj jednak nuli **DA**, izvršit će se ovaj dio programa. Ispisat će se poruka **Broj je jednak nuli**.

Ako je odgovor na pitanje je li sadržaj varijable broj jednak nuli **NE**, izvršit će se ovaj dio programa. Ispisat će se poruka **Broj nije jednak nuli**.

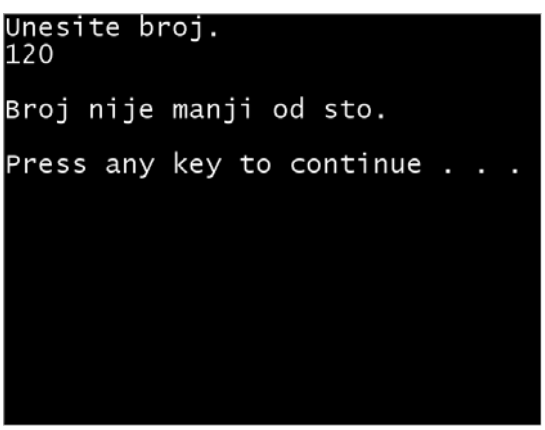
```
Unesite broj.
12

Broj nije jednak nuli.

Press any key to continue . . .
```

Da smo umjesto **broj == 0** napisali **broj = 0** program ne bi ispitivao je li sadržaj varijable broj jednak nuli, nego bi broj nula stavio u varijablu broj.




```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  float broj;
  cout << "Unesite broj." << endl;
  cin >> broj;
  cout << endl;
  if (broj < 100)
    {
      cout << "Broj je manji od sto." << endl;
    }
  else
    {
      cout << "Broj nije manji od sto." << endl;
    }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Vidjeli smo da možemo ispitivati je li varijabla veća, manja ili jednaka nekom broju.
U dosadašnjim primjerima to je bio broj nula, ali to može biti bilo koji broj, npr. broj 100 kao što je to slučaj u ovom primjeru.



Osim varijable i broja, **if** naredbom možemo uspoređivati dvije varijable.

```
Unesite prvi broj.
12

Unesite drugi broj.
12

Brojevi su jednaki.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   float prvi;
   float drugi;
   cout << "Unesite prvi broj." << endl;
   cin >> prvi;
   cout << endl;
   cout << "Unesite drugi broj." << endl;
   cin >> drugi;
   cout << endl;
   if (prvi == drugi)
     {
      cout << "Brojevi su jednaki." << endl;
     }
   else
     {
      cout << "Brojevi su razliciti." << endl;
     }
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Ovdje ispitujemo jesu li sadržaji dviju varijabli u koje smo prethodno unijeli brojeve jednaki.

Ako unesemo jednake brojeve ispisat, će se obavijest **Brojevi su jednaki**.

Ako unesemo različite, ispisat će se obavijest **Brojevi su različiti**.

```
Unesite prvi broj.
12

Unesite drugi broj.
16

Brojevi su razliciti.

Press any key to continue . . .
```

U ovom primjeru ispitujemo je li sadržaj varijable **prvi** manji od sadržaja varijable **drugi**.

```
Unesite prvi broj.
12

Unesite drugi broj.
16

Prvi broj je manji.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  float prvi;
  float drugi;
  cout << "Unesite prvi broj." << endl;
  cin >> prvi;
  cout << endl;
  cout << "Unesite drugi broj." << endl;
  cin >> drugi;
  cout << endl;
  if (prvi < drugi)
    {
      cout << "Prvi broj je manji." << endl;
    }
  else
    {
      cout << "Prvi broj nije manji." << endl;
    }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

```
Unesite prvi broj.
16

Unesite drugi broj.
12

Prvi broj nije manji.

Press any key to continue . . .
```

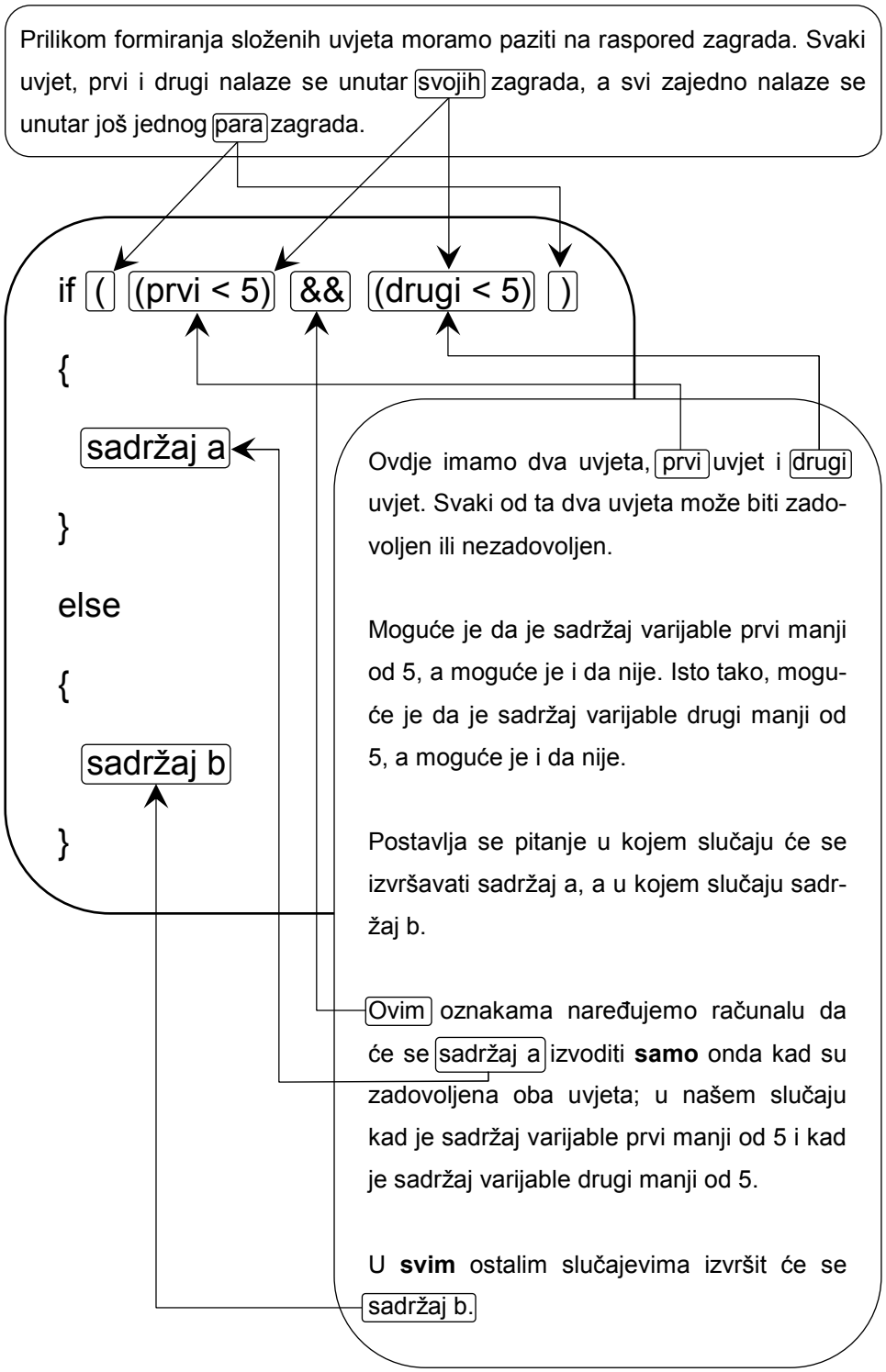
```
Unesite prvi broj.
2

Unesite drugi broj.
3

Oba broja su manja od pet.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  float prvi;
  float drugi;
  cout << "Unesite prvi broj." << endl;
  cin >> prvi;
  cout << endl;
  cout << "Unesite drugi broj." << endl;
  cin >> drugi;
  cout << endl;
  if ((prvi < 5) && (drugi < 5))
    {
      cout << "Oba broja su manja od pet." << endl;
    }
  else
    {
      cout << "Oba broja nisu manja od pet." << endl;
    }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Osim jednostavnih ispitivanja **if** naredbom, moguća su i složenija ispitivanja.

```
Unesite prvi broj.
6

Unesite drugi broj.
3

Oba broja nisu manja od pet.

Press any key to continue . . .
```

```
Unesite prvi broj.
3

Unesite drugi broj.
6

Oba broja nisu manja od pet.

Press any key to continue . . .
```


Prilikom formiranja složenih uvjeta moramo paziti na raspored zagrada. Svaki uvjet, prvi i drugi nalaze se unutar svojih zagrada, a svi zajedno nalaze se unutar još jednog para zagrada.
if ((prvi < 5) && (drugi < 5))
{
sadržaj a
}
else
{
sadržaj b
}
Ovdje imamo dva uvjeta, prvi uvjet i drugi uvjet. Svaki od ta dva uvjeta može biti zadovoljen ili nezadovoljen.
Moguće je da je sadržaj varijable prvi manji od 5, a moguće je i da nije. Isto tako, moguće je da je sadržaj varijable drugi manji od 5, a moguće je i da nije.
Postavlja se pitanje u kojem slučaju će se izvršavati sadržaj a, a u kojem slučaju sadržaj b.
Ovim oznakama naređujemo računalu da će se sadržaj a izvoditi **samo** onda kad su zadovoljena oba uvjeta; u našem slučaju kad je sadržaj varijable prvi manji od 5 i kad je sadržaj varijable drugi manji od 5.
U **svim** ostalim slučajevima izvršit će se sadržaj b.

```
Unesite prvi broj.
1

Unesite drugi broj.
4

Barem jedan je manji od 5.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  float prvi;
  float drugi;
  cout << "Unesite prvi broj." << endl;
  cin >> prvi;
  cout << endl;
  cout << "Unesite drugi broj." << endl;
  cin >> drugi;
  cout << endl;
  if ((prvi < 5) || (drugi < 5))
    {
      cout << "Barem jedan je manji od 5." << endl;
    }
  else
    {
      cout << "Niti jedan nije manji od 5." << endl;
    }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Ako umjesto oznaka && stavimo oznake || ukupni uvjet bit će zadovoljen ako je barem jedan od dva uvjeta zadovoljen.

```
Unesite prvi broj.
8

Unesite drugi broj.
4

Barem jedan je manji od 5.

Press any key to continue . . .
```

```
Unesite prvi broj.
6

Unesite drugi broj.
8

Niti jedan nije manji od 5.

Press any key to continue . . .
```

Oznaku **|** dobijemo tako da držimo lijevu **AltGr** tipku, a zatim pritisnemo **W** tipku.

Ovdje možemo vidjeti primjere jednostavnih ispitivanja na temelju kojih računalo donosi odluku koji dio programa će se izvoditi, a koji preskočiti.

Ovo su samo neki primjeri, dok je broj mogućih kombinacija praktički neograničen. Bilo koju varijablu koju koristimo u programu možemo usporediti s bilo kojim brojem.

Isto tako bilo koje dvije varijable koje se koriste u programu možemo međusobno uspoređivati.

Primjeri jednostavnih ispitivanja	
if (broj < 0)	Ispitujemo je li sadržaj varijable **broj** manji od broja nula.
if (broj > 0)	Ispitujemo je li sadržaj varijable **broj** veći od broja nula.
if (broj == 0)	Ispitujemo je li sadržaj varijable **broj** jednak broju nula.
if (broj < 50)	Ispitujemo je li sadržaj varijable **broj** manji od broja pedeset.
if (broj > 50)	Ispitujemo je li sadržaj varijable **broj** veći od broja pedeset.
if (broj == 50)	Ispitujemo je li sadržaj varijable **broj** jednak broju pedeset.
if (broj != 50)	Ispitujemo je li sadržaj varijable **broj** različit od broja pedeset.
if (prvi > drugi)	Ispitujemo je li sadržaj varijable **prvi** veći od sadržaja varijable **drugi**.
if (prvi < drugi)	Ispitujemo je li sadržaj varijable **prvi** manji od sadržaja varijable **drugi**.
if (prvi == drugi)	Ispitujemo je li sadržaj varijable **prvi** jednak sadržaju varijable **drugi**.

Osim jednostavnih ispitivanja if naredbom, u programu možemo imati i veoma složena ispitivanja.

Složena ispitivanja formiraju se korištenjem zagrada po logici sličnoj logici u matematici i uporabom simbola || i &&.

Simbol && između dva uvjeta znači da oba uvjeta moraju biti zadovoljena da bi bio zadovoljen ukupni uvjet.

Simbol || između dva uvjeta označuje da je dovoljno da je barem jedan od uvjeta zadovoljen da bi bio zadovoljen ukupni uvjet.

Broj mogućih kombinacija praktički je neograničen, a koju kombinaciju ćemo upotrijebiti ovisi o cilju koji programom želimo postići.

Primjeri složenih ispitivanja	
if ((prvi < 5) && (drugi < 5))	Uvjet je zadovoljen ako je sadržaj varijable **prvi** manji od 5 **i** ako je sadržaj varijable **drugi** manji od 5.
if ((prvi < 5) || (drugi < 5))	Uvjet je zadovoljen ako je **ili** sadržaj varijable **prvi** manji od 5, **ili** ako je sadržaj varijable **drugi** manji od 5, **ili** ako su obje varijable manje od 5.
if (((prvi < 5) || (drugi < 5)) && (treci < 5))	Uvjet je zadovoljen ako je sadržaj varijable **prvi ili** sadržaj varijable **drugi ili** sadržaj **obiju** varijabli manji od 5 **i** ako je sadržaj varijable **treci** manji od 5.
if ((prvi < 5) || (drugi < 5) || (treci < 5))	Uvjet je zadovoljen ako je barem jedna od triju varijabli manja od pet, ili ako su dvije manje od 5, ili ako su sve tri manje od 5.
if ((prvi < 5) && (drugi < 5) && (treci < 5))	Uvjet je zadovoljen ako su sve tri varijable manje od pet. Dovoljno je da jedna varijabla nije manja od 5 i uvjet nije ispunjen.

```
Unesite prvi broj.
2
Unesite drugi broj.
8
Unesite treci broj.
3
DA

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  float prvi;
  float drugi;
  float treci;
  cout << "Unesite prvi broj." <<
endl;
  cin >> prvi;
  cout << "Unesite drugi broj." << endl;
  cin >> drugi;
  cout << "Unesite treci broj." << endl;
  cin >> treci;
  if (((prvi < 5) || (drugi < 5)) && (treci < 5))
    {
      cout << "DA" << endl;
    }
  else
    {
      cout << "NE" << endl;
    }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Ovdje vidimo primjer programa koji možemo iskoristiti za isprobavanje kombinacija s prethodne stranice.

U ovom slučaju ukupni uvjet je ispunjen ako je prvi broj manji od 5 ili ako je drugi broj manji od 5 ili ako su oba manja od 5 i ako je treći broj manji od 5.

Ako je ukupni uvjet zadovoljen, izvest će se ovaj dio programa.

Ako ukupni uvjet nije zadovoljen, izvest će se ovaj dio programa.

```
Unesite prvi broj.
6
Unesite drugi broj.
8
Unesite treci broj.
2
NE

Press any key to continue . . .
```

```
Unesite broj.
-6
Broj je manji od 0.

Press any key to continue . . .
```

```
Unesite broj.
12
Broj je veci od 0.

Press any key to continue . . .
```

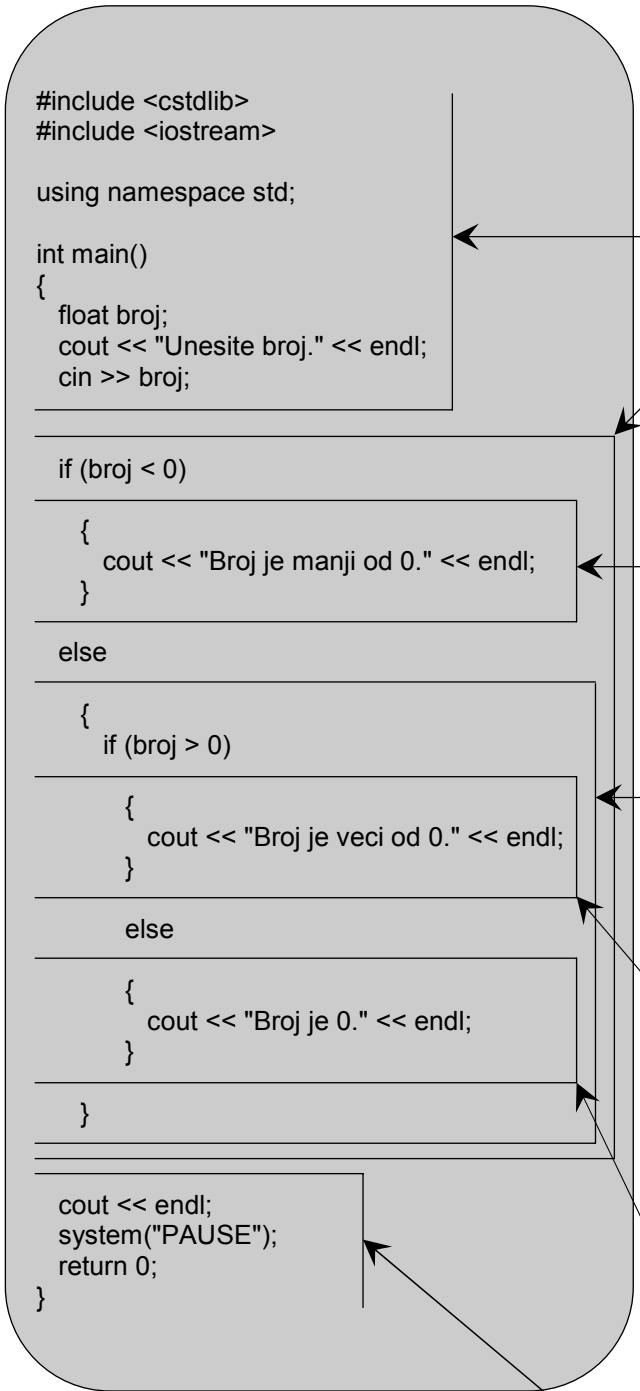
```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  float broj;
  cout << "Unesite broj." << endl;
  cin >> broj;
  if (broj < 0)
    {
      cout << "Broj je manji od 0." << endl;
    }
  else
    {
      if (broj > 0)
        {
          cout << "Broj je veci od 0." << endl;
        }
        else
        {
          cout << "Broj je 0." << endl;
        }
    }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Osim što možemo u programu imati veoma složena ispitivanja unutar if naredbe, moguće je jednu if naredbu staviti unutar druge if naredbe.

To je nužno ako želimo ispitati više stvari, npr. je li neki broj veći od nule ili jednak nuli ili je manji od nule. Takvo ispitivanje ne možemo napraviti s jednom if naredbom, nego su nužne dvije if naredbe.

Pogledajmo ovaj program.

```
Unesite broj.
0
Broj je 0.

Press any key to continue . . .
```



Pogledamo strukturu programa.

Početak programa.

Prva if naredba. Ispituje je li broj manji od 0.

Ako je uneseni broj manji od 0, izvodi se ovaj dio.

Ako uneseni broj nije manji od 0, izvodi se ovaj dio. Unutar ovog dijela nalazi se druga if naredba.

Ako je uneseni broj veći od 0, izvodi se ovaj dio.

Ako uneseni broj nije veći od 0, izvodi se ovaj dio.

Ako uneseni broj nije ni veći ni manji od nule, onda znamo da je nula.

Kraj programa.

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
float broj;
cout << "Unesite broj." << endl;
cin >> broj;
if (broj < 0)
{
cout << "Broj je manji od 0." << endl;
}
else
{
if (broj > 0)
{
cout << "Broj je veci od 0." << endl;
}
else
{
cout << "Broj je 0." << endl;
}
}
cout << endl;
system("PAUSE");
return 0;
}
```

Pogledamo li pažljivo ovaj program, vidjet ćemo da je potpuno identičan programu s prethodne stranice. Razlika je jedino u tome što pojedini redovi nisu uvučeni.

Pokušamo li pokrenuti ovaj program, vidjet ćemo da radi potpuno jednako kao i inačica s uvučenim redovima.

U čemu je onda razlika? Zašto bismo uopće uvlačili redove kad program očigledno radi i bez toga?

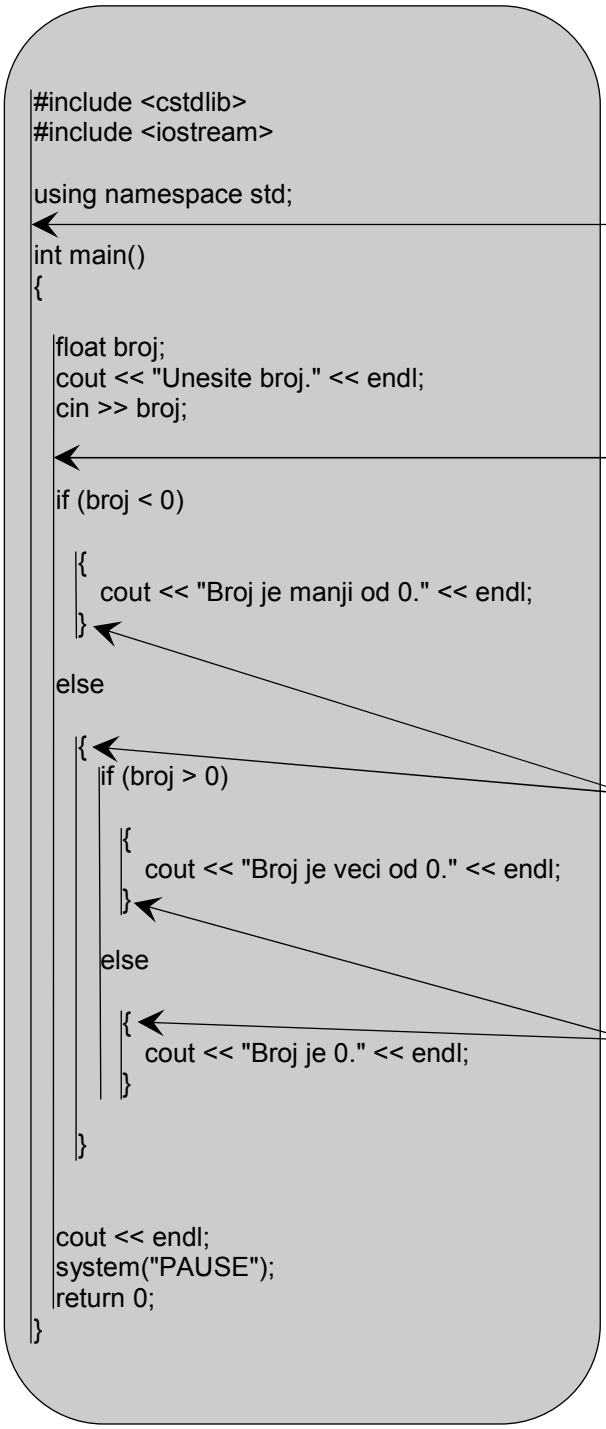
Razlika je u preglednosti. Program s uvučenim redovima daleko je pregledniji.

Usporedimo li program s uvučenim redovima na prethodnoj stranici i ovaj bez uvučenih redova, vidjet ćemo da je na inačici s uvučenim redovima daleko lakše uočiti gdje započinju, a gdje završavaju pojedine if naredbe; koje vitičaste zagrade čine par i sl.

Preglednost programa s uvlačenjima redova osobito dolazi do izražaja kod većih i složenijih programa.

Ne postoje čvrsta pravila na koji način treba uvlačiti redove. Na početku je najbolje da u vlastitim programima uvlačenja izvodimo slično kao u primjerima danima u knjizi.

Osnovna ideja je u tome da se uvuče svaka cjelina za koju bi moglo biti nejasno gdje počinje, a gdje završava.



```
#include <cstdlib>
#include <iostream>

using namespace std;
int main()
{

  float broj;
  cout << "Unesite broj." << endl;
  cin >> broj;

  if (broj < 0)

    {
      cout << "Broj je manji od 0." << endl;
    }

  else

    {
      if (broj > 0)

        {
          cout << "Broj je veci od 0." << endl;
        }

      else

        {
          cout << "Broj je 0." << endl;
        }

    }


  cout << endl;
  system("PAUSE");
  return 0;
}
```

Pogledajmo strukturu uvlačenja redova.

Bez uvlačenja piše se početak programa i dvije glavne vitičaste zagrade.

Sadržaj unutar glavnih vitičastih zagrada uvlači se u desnu stranu.

Na toj razini piše se većina jednostavnih naredbi.

Sadržaji unutar if naredbe dodatno se pomiču u desnu stranu.

To nam pomaže da lakše uočimo gdje počinje, a gdje završava if naredba.

Ako se unutar jedne if naredbe nalazi druga if naredba, njen sadržaj dodatno se pomiče u desnu stranu.

To nam pomaže da lakše uočimo gdje počinje i gdje završava druga if naredba te pomaže da njen sadržaj lakše razlikujemo od sadržaja prve if naredbe.

Switch naredba

```
Unesite 1. broj:
2
Unesite 2. broj:
3
Brojeve cemo:
1 - zbrojiti
2 - oduzeti
3 - mnoziti
4 - dijeliti
4
2 : 3 = 0.666667

Press any key to continue . . .
```

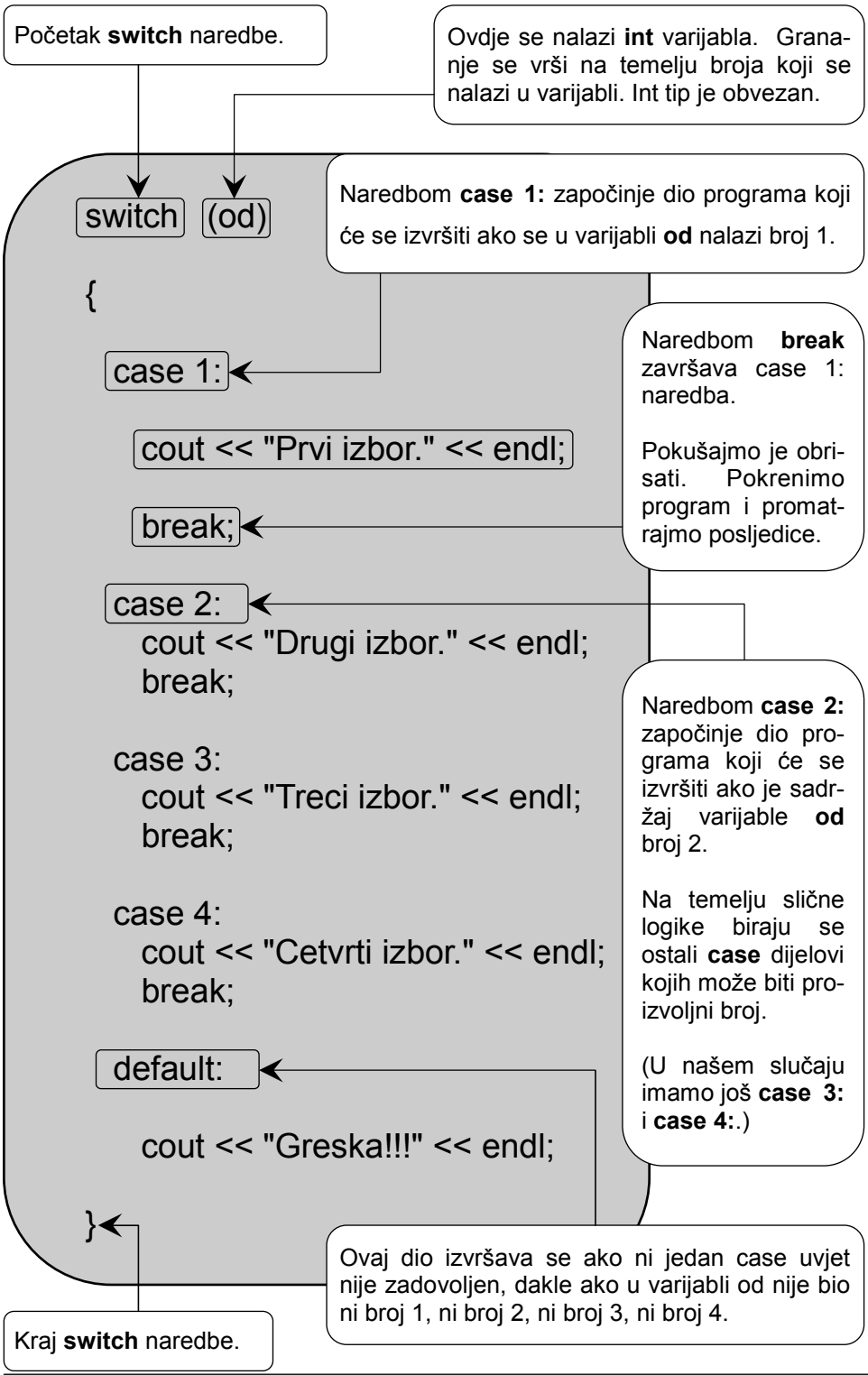
```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  float a;
  float b;
  int od;
  cout << "Unesite 1. broj:" << endl;
  cin >> a;
  cout << "Unesite 2. broj:" << endl;
  cin >> b;
  cout << "Brojeve cemo:" << endl;
  cout << "1 - zbrojiti" << endl;
  cout << "2 - oduzeti" << endl;
  cout << "3 - mnoziti" << endl;
  cout << "4 - dijeliti" << endl;
  cin >> od;
  switch (od)
  {
    case 1:
      cout << a << " + " << b << " = " << a + b << endl;
      break;
    case 2:
      cout << a << " - " << b << " = " << a - b << endl;
      break;
    case 3:
      cout << a << " x " << b << " = " << a * b << endl;
      break;
    case 4:
      cout << a << " : " << b << " = " << a / b << endl;
      break;
    default:
      cout << "Morate unijeti 1,2,3 ili 4." << endl;
  }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

If naredba nam omogućuje grananje na dvije grane. Jedna će se izvršiti ako je uvjet zadovoljen, a druga ako uvjet nije zadovoljen.

Primjena naredbe if nije elegantna ako se u programu javi potreba za grananjem u više od dvije grane.

U ovom programu biramo jednu od četiri računske operacije.

Program bismo mogli realizirati i uporabom više **if** naredbi, ali daleko je elegantniji ako upotrijebimo **switch** naredbu.


Početak **switch** naredbe.
Ovdje se nalazi **int** varijabla. Grananje se vrši na temelju broja koji se nalazi u varijabli. Int tip je obvezan.
switch (od)
{
Naredbom **case 1:** započinje dio programa koji će se izvršiti ako se u varijabli **od** nalazi broj 1.
case 1:
cout << "Prvi izbor." << endl;
break;
Naredbom **break** završava case 1: naredba.
Pokušajmo je obrisati. Pokrenimo program i promatrajmo posljedice.
case 2:
cout << "Drugi izbor." << endl;
break;
case 3:
cout << "Treci izbor." << endl;
break;
case 4:
cout << "Cetvrti izbor." << endl;
break;
Naredbom **case 2:** započinje dio programa koji će se izvršiti ako je sadržaj varijable **od** broj 2.
Na temelju slične logike biraju se ostali **case** dijelovi kojih može biti proizvoljni broj.
(U našem slučaju imamo još **case 3:** i **case 4:**.)
default:
cout << "Greska!!!" << endl;
}
Ovaj dio izvršava se ako ni jedan case uvjet nije zadovoljen, dakle ako u varijabli od nije bio ni broj 1, ni broj 2, ni broj 3, ni broj 4.
Kraj **switch** naredbe.

Petlje

For petlja

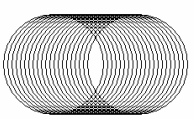
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   circle(260,240,60);
   circle(265,240,60);
   circle(270,240,60);
   circle(275,240,60);
   circle(280,240,60);
   circle(285,240,60);
   circle(290,240,60);
   circle(295,240,60);
   circle(300,240,60);
   circle(305,240,60);
   circle(310,240,60);
   circle(315,240,60);
   circle(320,240,60);
   circle(325,240,60);
   circle(330,240,60);
   circle(335,240,60);
   circle(340,240,60);
   circle(345,240,60);
   circle(350,240,60);
   circle(355,240,60);
   circle(360,240,60);
   getch();
   closegraph();
   return 0;
}
```

Ovaj program crta niz krugova jednakog promjera. Svakome je središte pomaknuto za pet točaka u desnu stranu u odnosu na središte prethodnog.

Vidimo da program funkcionira, ali nije baš elegantan budući da moramo puno puta ponoviti sličnu naredbu.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   for (x = 260; x < 365; x = x + 5 )
   {
      circle(x,240,60);
   }
   getch();
   closegraph();
   return 0;
}
```

Ovaj program radi isti posao, ali je očigledno daleko elegantniji.



Ključna naredba u novoj inačici programa je **for** naredba. Omogućuje nam izvršavanje dijela programa više puta. To višestruko ponavljanje dijela programa naziva se programska petlja. Slično kao **if** naredba, i **for** naredba je veoma složena i proteže se kroz više redova programa.

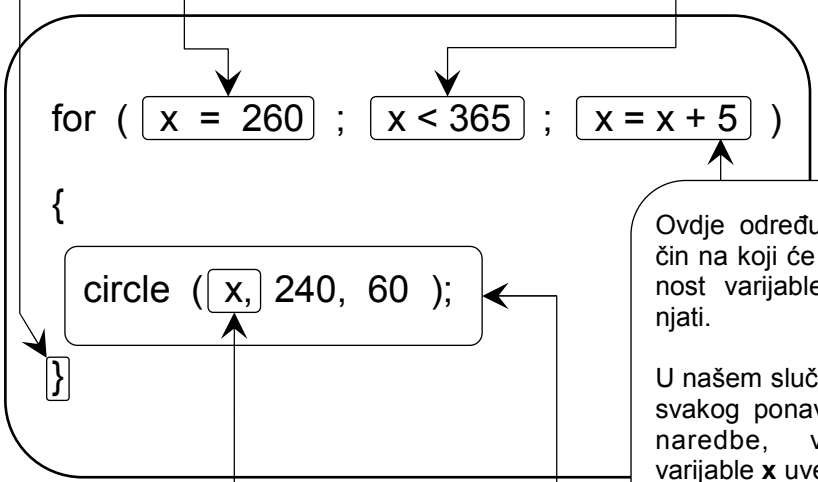
Pogledajmo sastavne elemente **for** naredbe.

Ovdje se definira početna vrijednost varijable koju koristi for naredba i koju prije for naredbe moramo formirati.

U našem slučaju početna vrijednost varijable **x** je 260.

Petlja će se vrtjeti, odnosno petlja će ponavljati dio programa, tako dugo dok je ovaj uvjet zadovoljen; u našem slučaju, tako dugo dok je sadržaj varijable **x** manji od 365.

Kad uvjet više nije zadovoljen; tj. kad **x** više ne bude manji od 365, program će nastavit izvođenje od prve naredbe nakon druge vitičaste zagrade.



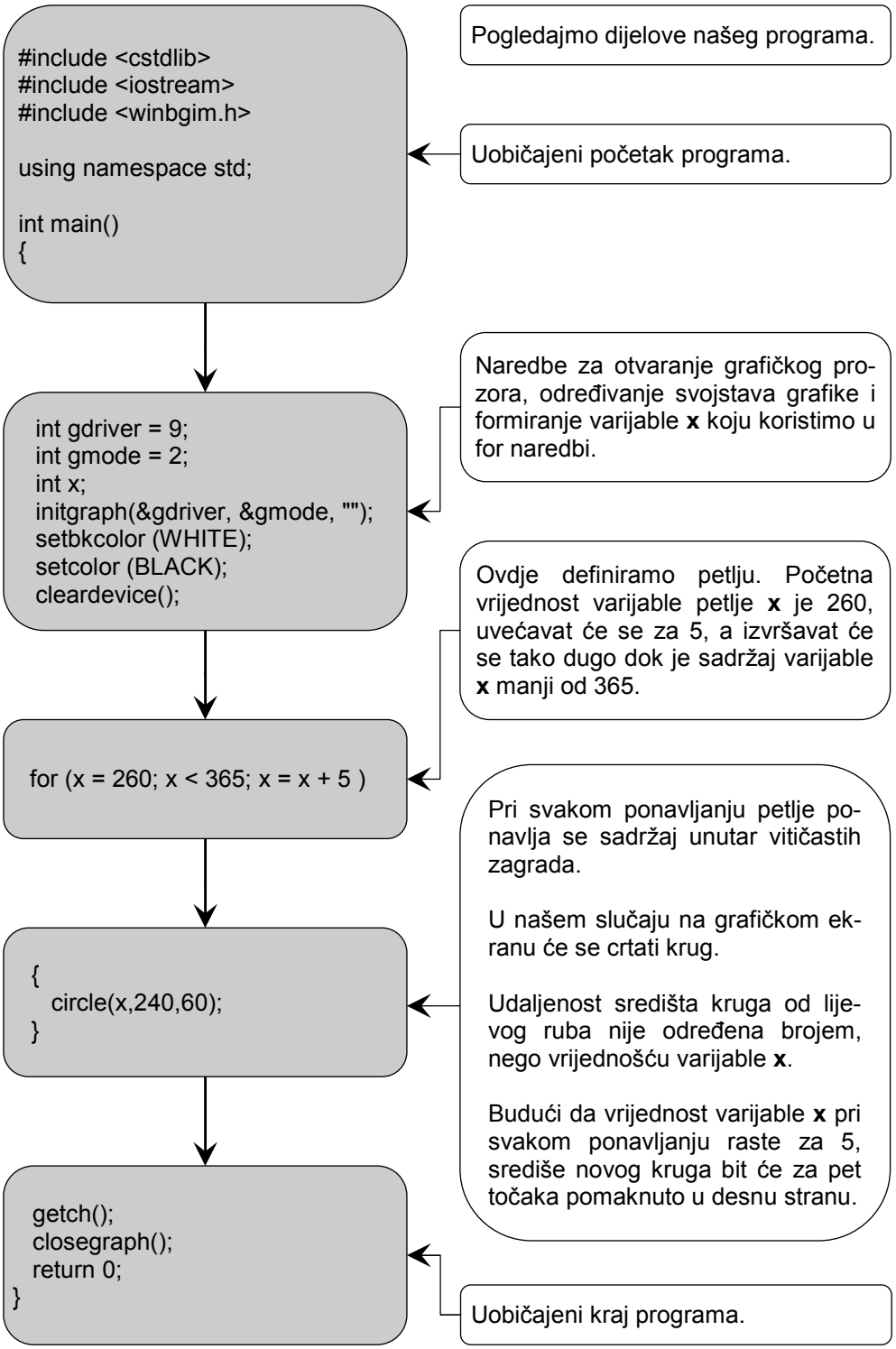
Ovdje određujemo način na koji će se vrijednost varijable **x** mijenjati.

U našem slučaju nakon svakog ponavljanja for naredbe, vrijednost varijable **x** uvećat će se za 5, a rezultat će se spremiti u varijablu **x**.

Udaljenost središta kruga od lijevog ruba nije određena brojem, nego vrijednošću varijable **x**.

Budući da se pri svakom ponovnom izvođenju sadržaj varijable **x** mijenja, jednako će se mijenjati i udaljenost središta kruga od lijevog ruba.

Sadržaj unutar navodnika ponavlja se svaki put kad for naredba uveća vrijednost varijable **x** i provjeri je li **x** manji od 365.


#include <cstdlib>
#include <iostream>
#include <winbgim.h>
using namespace std;
int main()
{
int gdriver = 9;
int gmode = 2;
int x;
initgraph(&gdriver, &gmode, "");
setbkcolor (WHITE);
setcolor (BLACK);
cleardevice();
for (x = 260; x < 365; x = x + 5)
{
circle(x,240,60);
}
getch();
closegraph();
return 0;
}
Pogledajmo dijelove našeg programa.
Uobičajeni početak programa.
Naredbe za otvaranje grafičkog prozora, određivanje svojstava grafike i formiranje varijable x koju koristimo u for naredbi.
Ovdje definiramo petlju. Početna vrijednost varijable petlje x je 260, uvećavat će se za 5, a izvršavat će se tako dugo dok je sadržaj varijable x manji od 365.
Pri svakom ponavljanju petlje ponavlja se sadržaj unutar vitičastih zagrada.
U našem slučaju na grafičkom ekranu će se crtati krug.
Udaljenost središta kruga od lijevog ruba nije određena brojem, nego vrijednošću varijable x.
Budući da vrijednost varijable x pri svakom ponavljanju raste za 5, središe novog kruga bit će za pet točaka pomaknuto u desnu stranu.
Uobičajeni kraj programa.

Nemojmo se obeshrabriti ako nam trenutno nije previše jasno ni kako **for** naredba radi, ni čemu bi to trebalo služiti. Na početku su takve nedoumice normalne i nipošto nisu dokaz da nismo talentirani za programiranje.

Takve nedoumice prevladavaju se tako da napišemo određenu količinu programa jer ćemo jedino na taj način do kraja shvatiti kako **for** naredba radi i čemu služi.

Drugačiji je problem ako nam se ne da pisati programe. Onda bismo doista mogli konstatirati da programiranje nije za nas i trebali bismo se pokušati baviti nečim intelektualno manje zahtjevnim, npr. nogometom ili politikom.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int x;
   cout << "Prije for naredbe." << endl;
   for (x = 0; x < 9; x = x + 1 )
   {
      cout << x << endl;
   }
   cout << "Nakon for naredbe." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Ovaj će nam program pomoći da lakše shvatimo funkcioniranje **if** naredbe. Vrijednost varijable **x** mijenjat će se od 0 do 9 po 1.

Jedino što ovaj program radi jest ispisivanje sadržaj varijable **x**.

To nam omogućuje lako opažanje kako se sadržaj varijable **x** mijenja, te koji dio programa se ponavlja i koliko puta.

Unošenjem varijacija u taj program najlakše ćemo shvatiti **for** naredbu.

```
Prije for naredbe.
0
1
2
3
4
5
6
7
8
Nakon for naredbe.

Press any key to continue . . .
```

Vidimo da se ponavlja sadržaj unutar vitičastih zagrada **for** naredbe.

Sadržaj prije i sadržaj poslije **for** naredbe ne ponavlja se.

```
Prije for naredbe.
0
2
4
6
8
Nakon for naredbe.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int x;
   cout << "Prije for naredbe." << endl;
   for (x = 0; x < 9; x = x + 2 )
   {
      cout << x << endl;
   }
   cout << "Nakon for naredbe." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

U prošlom primjeru vrijednost varijable **x** uvećavali smo za 1, a u primjeru prije toga za 5.

Vrijednost varijable **x** u for naredbi možemo uvećavati za bilo koju vrijednost koja nam u programu treba, npr. za 2.

U ovom slučaju vrijednost varijable **x** raste od 0 do 9 po 2.

Mijenjati možemo početnu i krajnju vrijednost petlje.

U ovom slučaju **x** će se mijenjati od 10 do 29.

```
Prije for naredbe.
10
12
14
16
18
20
22
24
26
28
Nakon for naredbe.

Press any key to continue .
```

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int x;
   cout << "Prije for naredbe." << endl;
   for (x = 10; x < 29; x = x + 2 )
   {
      cout << x << endl;
   }
   cout << "Nakon for naredbe." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

```
Prije for naredbe.
28
26
24
22
20
18
16
14
12
10
Nakon for naredbe.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int x;
   cout << "Prije for naredbe." << endl;
   for (x = 28; x > 9; x = x - 2 )
   {
      cout << x << endl;
   }
   cout << "Nakon for naredbe." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

For naredba može brojiti i unatrag.

U ovom slučaju vrijednost varijable **x** mijenja se od 28 do 9 po 2.

U **for** naredbi mogu se koristiti i negativni brojevi.

```
12
10
8
6
4
2
0
-2
-4
-6
-8
Nakon for naredbe.

Press any key to continue
```

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int x;
   cout << "Prije for naredbe." << endl;
   for (x = 18; x > -9; x = x - 2 )
   {
      cout << x << endl;
   }
   cout << "Nakon for naredbe." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

```
110642
110644
110646
110648
110650
110652
110654
110656
110658
110660
110662
110664
110666
110668
```

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int x;
   cout << "Prije for naredbe." << endl;
   for (x = 18; x > -9; x = x + 2 )
   {
      cout << x << endl;
   }
   cout << "Nakon for naredbe." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Prilikom formiranja **for** naredbe moguće je napraviti grešku koja se naziva “mrtva petlja”.

Do te greške dolazi kad uvjete za kraj petlje tako formuliramo da se oni nikad neće ispuniti, pa će se petlja vječno izvoditi.

U ovom primjeru početna vrijednost je 18. Ta početna vrijednost uvećava se za 2, a petlja će se ponavljati tako dugo dok je **x** veći od –9.

Ako 18 uvećavamo za 2, **x** će uvijek biti veći od –9 i petlja se nikad neće prekinuti.

Pogledajmo još jedan primjer “mrtve petlje”.

```
-11336
-11337
-11338
-11339
-11340
-11341
-11342
-11343
-11344
-11345
-11346
-11347
-11348
-11349
-11350
```

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int x;
   cout << "Prije for naredbe." << endl;
   for (x = 0; x < 9; x = x - 1 )
   {
      cout << x << endl;
   }
   cout << "Nakon for naredbe." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Naredba for ne postoji zato da bi se niz brojeva ispisao na zaslon računala, nego zato da bi se u programu postigli određeni efekti.

Iako to nije uvijek slučaj, najčešće se za postizanje željenih ciljeva koristi varijabla petlje. Pogledajmo nekoliko primjera.

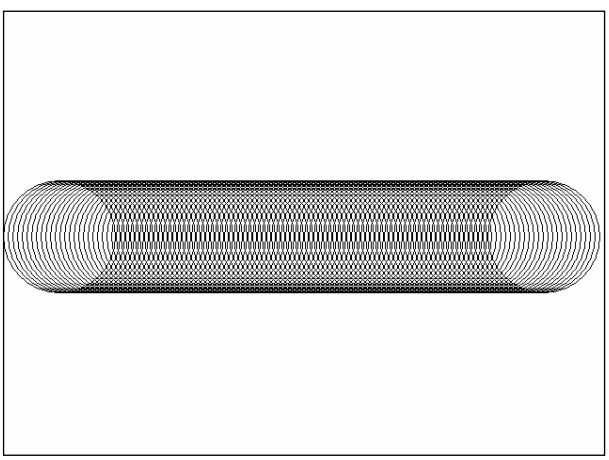
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   for (x = 60; x < 580; x = x + 5 )
   {
      circle(x,240,60);
   }
   getch();
   closegraph();
   return 0;
}
```

Ovaj primjer sličan je primjeru na početku ovog poglavlja.

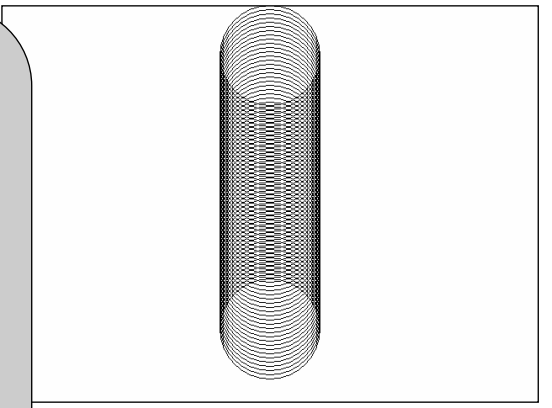
Razlika je u tome što su u ovom slučaju drugačije početna i krajnja vrijednost **x** varijable, pa smo dobili drugačiji niz krugova.

U ovom slučaju vrijednost varijable **x** mijenja se od 60 do 580 po 5.

Pokušamo sami mijenjati vrijednosti unutar **for** naredbe i promatrajmo kakve efekte će izazvati te promjene.



```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   for (x = 60; x < 390; x = x + 5 )
   {
      circle(320,x,60);
   }
   getch();
   closegraph();
   return 0;
}
```

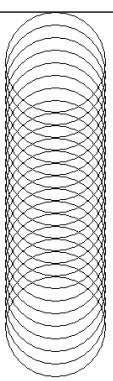


Varijablu x možemo staviti umjesto broja koji određuje udaljenost središta od gornjeg ruba, pa će se krugovi crtati od gore prema dolje.

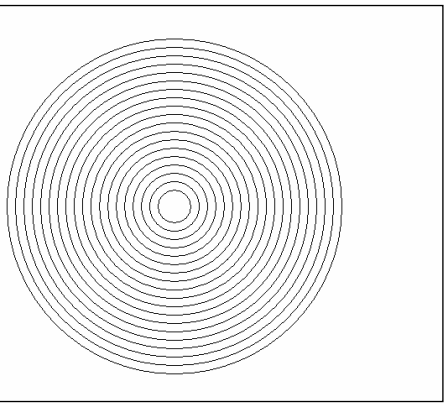
Uvećavamo li varijablu x za 15 umjesto za 5, pomak središta krugova bit će veći.

Umjesto broja 15 pokušajmo staviti 10, a zatim 20.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{

   int gdriver = 9;
   int gmode = 2;
   int x;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   for (x = 60; x < 390; x = x + 15 )
   {
      circle(320,x,60);
   }
   getch();
   closegraph();
   return 0;
}
```

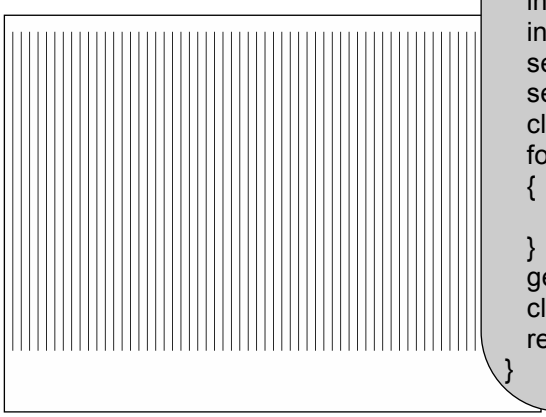


```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   for (x = 20; x < 210; x = x + 10 )
   {
      circle(320,240,x);
   }
   getch();
   closegraph();
   return 0;
}
```



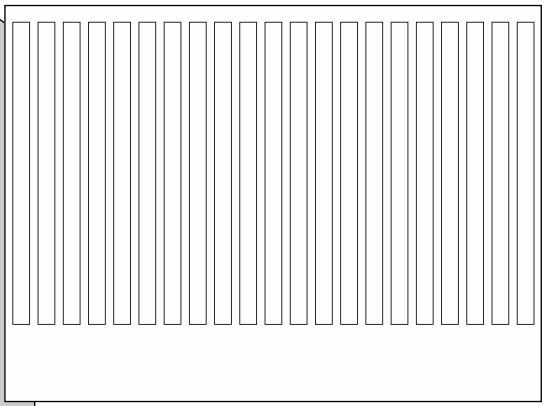
Osim pozicije središta kruga, varijablom x možemo mijenjati polumjer kruga.

Na sličan način možemo crtati nizove drugih geometrijskih oblika, npr. crta.



```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   for (x = 10; x < 640; x = x + 10 )
   {
      line (x, 20, x, 400);
   }
   getch();
   closegraph();
   return 0;
}
```

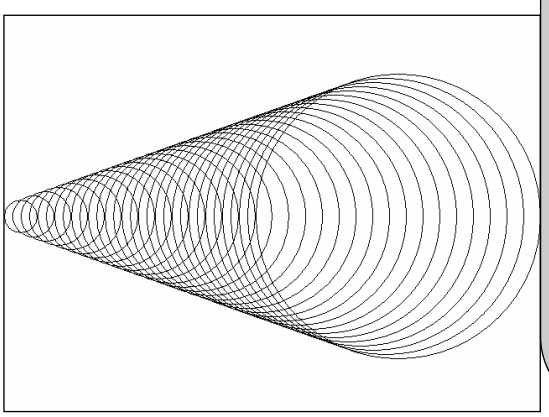
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   for (x = 10; x < 640; x = x + 30 )
   {
      rectangle (x, 20, x+20, 380);
   }
   getch();
   closegraph();
   return 0;
}
```



Ovo je primjer programa koji crta niz pravokutnika.

U dosadašnjim programima koristili smo jednu varijablu.

U ovom primjeru koristima dvije, jednom pomičemo udaljenost središta od lijevog ruba, a drugom mijenjamo promjer.



```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x;
   int r;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   r = 20;
   for (x = 20; x < 475; x = x + 15 )
   {
      circle(x,240,r);
      r = r + 5;
   }
   getch();
   closegraph();
   return 0;
}
```

Pogledajmo bitne dijelove ovog programa.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
  int gdriver = 9;
  int gmode = 2;
  int x;

  int r;

  initgraph(&gdriver, &gmode, "");
  setbkcolor (WHITE);
  setcolor (BLACK);
  cleardevice();

  r = 20;

  for (x = 20; x < 475; x = x + 15 )
  {

    circle(x,240,r);

    r = r + 5;

  }
  getch();
  closegraph();
  return 0;
}
```

Za razliku od dosadašnjih primjera, u ovom formiramo još jednu varijablu koju smo nazvali **r**.

Prije for naredbe u varijablu **r** spremimo početnu vrijednost polumjera kruga. U našem slučaju spremili smo broj 20.

For naredba mijenja vrijednost varijable **x** o 20 do 475 po 15.

U naredbi za crtanje kruga upotrijebili smo dvije varijable.

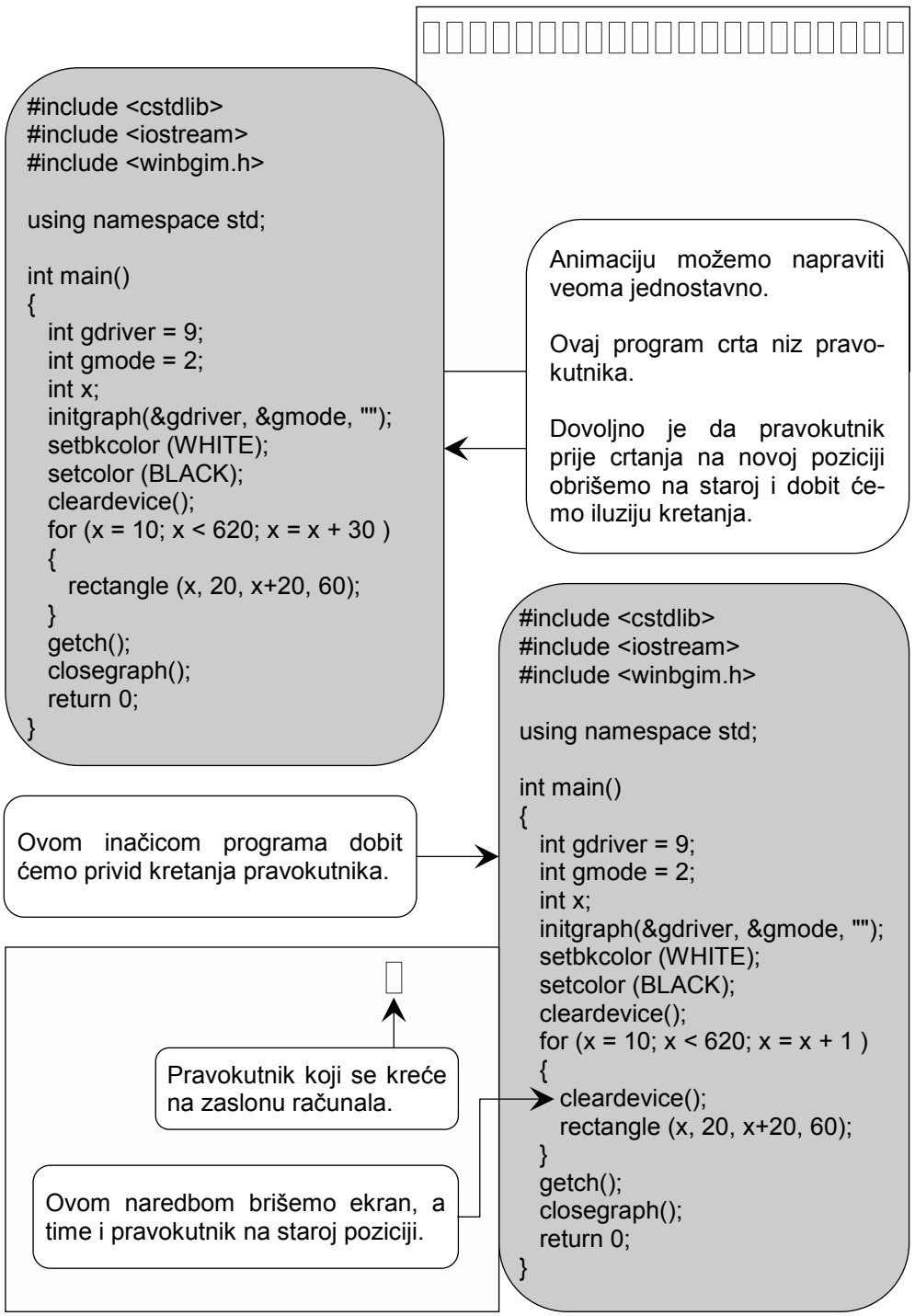
Varijablom **x** određujemo udaljenost središta kruga od lijevog ruba, a varijablom **r** polumjer kruga.

Da bi kod svakog ponavljanja polumjer kruga bio veći, nakon što smo nacrtali krug, unutar vitičastih zagrada **for** naredbe, broj u varijabli **r** povećavamo za 5.

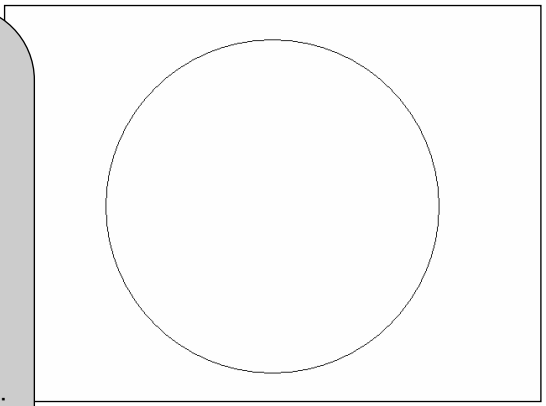
Zbog toga će pri svakom ponavljaju polumjer kruga biti veći.

I u ovom slučaju promjena pojedinih elemenata i opažanje rezultata pomoći će nam da bolje razumijemo program; npr umjesto r = r + 5 pokušajmo staviti r = r + 10; umjesto x = x + 15 pokušajmo staviti x = x + 5.

Animacija


#include <cstdlib>
#include <iostream>
#include <winbgim.h>
using namespace std;
int main()
{
int gdriver = 9;
int gmode = 2;
int x;
initgraph(&gdriver, &gmode, "");
setbkcolor (WHITE);
setcolor (BLACK);
cleardevice();
for (x = 10; x < 620; x = x + 30)
{
rectangle (x, 20, x+20, 60);
}
getch();
closegraph();
return 0;
}
Animaciju možemo napraviti veoma jednostavno.
Ovaj program crta niz pravokutnika.
Dovoljno je da pravokutnik prije crtanja na novoj poziciji obrišemo na staroj i dobit ćemo iluziju kretanja.
Ovom inačicom programa dobit ćemo privid kretanja pravokutnika.
#include <cstdlib>
#include <iostream>
#include <winbgim.h>
using namespace std;
int main()
{
int gdriver = 9;
int gmode = 2;
int x;
initgraph(&gdriver, &gmode, "");
setbkcolor (WHITE);
setcolor (BLACK);
cleardevice();
for (x = 10; x < 620; x = x + 1)
{
cleardevice();
rectangle (x, 20, x+20, 60);
}
getch();
closegraph();
return 0;
}
Pravokutnik koji se kreće na zaslonu računala.
Ovom naredbom brišemo ekran, a time i pravokutnik na staroj poziciji.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   for (x = 10; x < 200; x = x + 1 )
   {
      cleardevice();
      circle (320, 240, x);
   }
   getch();
   closegraph();
   return 0;
}
```



Sve grafičke elemente koje smo do sada mijenjali možemo animirati.
U ovom programu animiramo širenje kruga.

U ovom primjeru animiramo krug koji pada od gornjeg prema donjem rubu .

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   for (x = 10; x < 465; x = x + 1 )
   {
      cleardevice();
      circle (320, x, 10);
   }
   getch();
   closegraph();
   return 0;
}
```

Pokušajmo sami smisliti slične programe.

Petlja u petlji

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
  int gdriver = 9;
  int gmode = 2;
  int x;
  int r;
  initgraph(&gdriver, &gmode, "");
  setbkcolor (WHITE);
  setcolor (BLACK);
  cleardevice();
  for (x = 30; x < 610; x = x + 60 )
  {
    for (r = 5; r < 31; r = r + 5)
    {
      delay(100);
      cleardevice();
      circle (x, 240, r);
    }
  }
  getch();
  closegraph();
  return 0;
}
```

Slično kao što možemo unutar **if** naredbe staviti drugu **if** naredbu tako unutar **for** petlje možemo staviti drugu **for** petlju.

Kod ovakvih programa veoma je važno pravilno uvlačiti redove, jer tako lakše uočavamo početak i kraj pojedinih **for** naredbi.

Ovom naredbom naređujemo računalu da na ovom mjestu stoji onoliko milisekundi koliki broj smo stavili u zagradu.

U našem slučaju računalo će stajati 100 milisekundi.

Ovom naredbom usporavamo izvođenje animacije, ako je animacija suviše brza.

Na vašem računalu taj broj možete promijeniti i tako izvođenje animacije prilagoditi brzini svog računala. (Ako imamo sporo računalo, ovu naredbu možemo izbaciti.)

Ovdje vidimo tijek cijele animacije, iako će se tijekom animacije istovremeno vidjeti samo jedan krug.

Vanjskom petljom pomičemo središte kruga u lijevu stranu, a unutrašnjom petljom mijenjamo polumjer kruga.

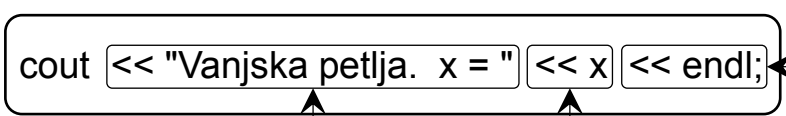
```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  int x;
  int r;
  cout << "Prije petlje." << endl;
  for (x = 0; x < 2; x = x + 1 )
  {
    cout << "Vanjska petlja.  x = " << x << endl;
    for (r = 0; r < 3; r = r + 1)
    {
      cout << "Nutarnja petlja. r = " << r << endl;
    }
  }
  cout << "Nakon petlje." << endl;
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Analiziranje ovog programa pomoći će nam da u potpunosti razumijemo način funkcioniranja petlje unutar petlje.

Pri tome nismo ograničeni na to da možemo staviti jednu petlju unutar petlje. Moguće je unutar petlje staviti petlju, a unutar te petlje još jednu petlju i tako petlje dodavati u skladu s ciljevima koje želimo postići.

```
Prije petlje.
Vanjska petlja.  x = 0
Nutarnja petlja. r = 0
Nutarnja petlja. r = 1
Nutarnja petlja. r = 2
Vanjska petlja.  x = 1
Nutarnja petlja. r = 0
Nutarnja petlja. r = 1
Nutarnja petlja. r = 2
Nakon petlje.

Press any key to continue . . .
```

Da bi rezultat rada programa bio bolje vidljiv, ovdje smo naredbu **cout** koristili na način na koji je do sada nismo koristili. U ovoj inačici sadrži tri dijela.



Tekst unutar navodnika ispisuje se na zaslon računala.

Sadržaj varijable **x** ispisuje se na zaslon računala.

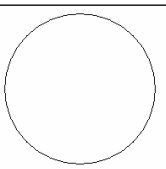
Skok u novi red.

Niz petlji

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x;
   int r;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   for (x = 465; x > 100; x = x - 1 )
   {
      cleardevice();
      circle (320, x, 10);
   }
   for (r = 10; r < 91; r = r + 1)
   {
     cleardevice();
     circle (320, x, r);
   }
   getch();
   closegraph();
   return 0;
}
```

Osim što petlju možemo staviti unutar druge petlje, petlje možemo staviti jednu iza druge.

U ovom slučaju prva petlja mali krug pomiče od dolje prema gore, a zatim druga petlja mali krug širi tako da sve zajedno tvori jednostavan prikaz vatrometa.



Analiza ovog programa pomoći će nam da lakše razumijemo funkcioniranje niza petlji.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int x;
   int r;
   cout << "Prije petlji." << endl;
   for (x = 2; x > 0; x = x - 1 )
   {
      cout << "Prva  petlja.  x = " << x << endl;
   }
   for (r = 0; r < 3; r = r + 1)
   {
     cout << "Druga petlja.  r = " << r << endl;
   }
   cout << "Nakon petlji." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

```
Prije petlji.
Prva  petlja.  x = 2
Prva  petlja.  x = 1
Druga petlja.  r = 0
Druga petlja.  r = 1
Druga petlja.  r = 2
Nakon petlji.

Press any key to contin
```

Dozvoljene i nedozvoljene kombinacije petlji

```
for (x = 30; x < 610; x = x + 60)
{
   for (r = 5; r < 31; r = r + 5)
   {
      circle  (x,  240,  r);
   }
}
```

Petlje se mogu nalaziti jedna unutar druge.

```
for (x = 465; x < 100; x = x - 1)
{
   circle  (320,  x,  10);
}

for (r = 10; r < 91; y = y + 1)
{
   circle  (320,  x,  r);
}
```

Petlje se mogu nalaziti jedna iza druge

```
for (x = 465; x < 100; x = x - 1)
{
   circle  (320,  x,  10);
   for (r = 10; r < 91; y = y + 1)
}

{
   circle  (320,  x,  r);
}
```

U ovom slučaju druga `for` naredba nalazi se unutar prve `for` naredbe.

Ovakva isprepletenost dviju for naredbi nije dozvoljena.

While petlja

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int broj;
   int kvadrat;
   int odgovor;
   odgovor = 1;
   while (odgovor == 1)
   {
      cout << "Unesite broj" << endl;
      cin >> broj;
      kvadrat = broj * broj;
      cout << "Kvadrat unesenog broja je:" << endl;
      cout << kvadrat;
      cout << endl;
      cout << "Ponovo = unesite 1." << endl;
      cout << "Kraj   = unesite 2." << endl;
      cin >> odgovor;
   }
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Do sada smo za svaki novi proračun programe morali iznova pokretati.

Daleko elegantnije bilo bi kad bi nas program nakon proračuna pitao želimo li proračun ponoviti ili želimo prekinuti izvođenje programa.

Ovaj program radi upravo tako. Unesemo li nakon proračuna broj 1, proračun ćemo moći ponoviti, a unesemo li 2, odnosno bilo koji broj različit od 1 program će se zaustaviti.

```
Unesite broj
3
Kvadrat unesenog broja je:
9
Ponovo = unesite 1.
Kraj   = unesite 2.
1
Unesite broj
4
Kvadrat unesenog broja je:
16
Ponovo = unesite 1.
Kraj   = unesite 2.
```

U ovom programu koristi se petlja, budući da se dio programa može ponoviti više puta. Ova petlje nije realizirana **for** naredbom, nego **while** naredbom.

U čemu je razlika?

Naredba **for** pogodnija je za točno određen broj ponavljanja, a naredba **while** onda kad unaprijed ne možemo znati broj ponavljanja.

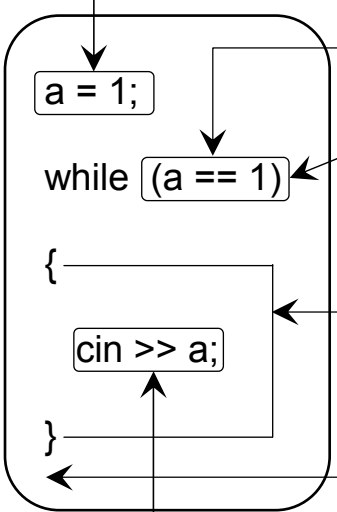
(U ovom slučaju mi ne možemo unaprijed znati koliko puta će korisnik htjeti kvadrirati neki broj.)

Ključna naredba u novoj inačici programa je **while** naredba. Logika uporabe te naredbe slična je logici uporabe for naredbe i ako bismo se potrudili svaki program realiziran for naredbom mogli bismo realizirati while naredbom i svaki program realiziran while naredbom mogli bismo realizirati for naredbom.

Zašto imamo dvije naredbe ako rade isti posao?

Zato što se neke stvari mogu lakše realizirati for naredbom, a neke while naredbom. Pogledajmo elemente while naredbe.

Prije naredbe while moramo varijabli koju će while naredba koristiti dati neku početnu vrijednost.



Da bi se ponavljanje u nekom trenutku prekinulo, unutar vitičastih zagrada varijabli koju while naredba ispituje treba promijeniti vrijednost.

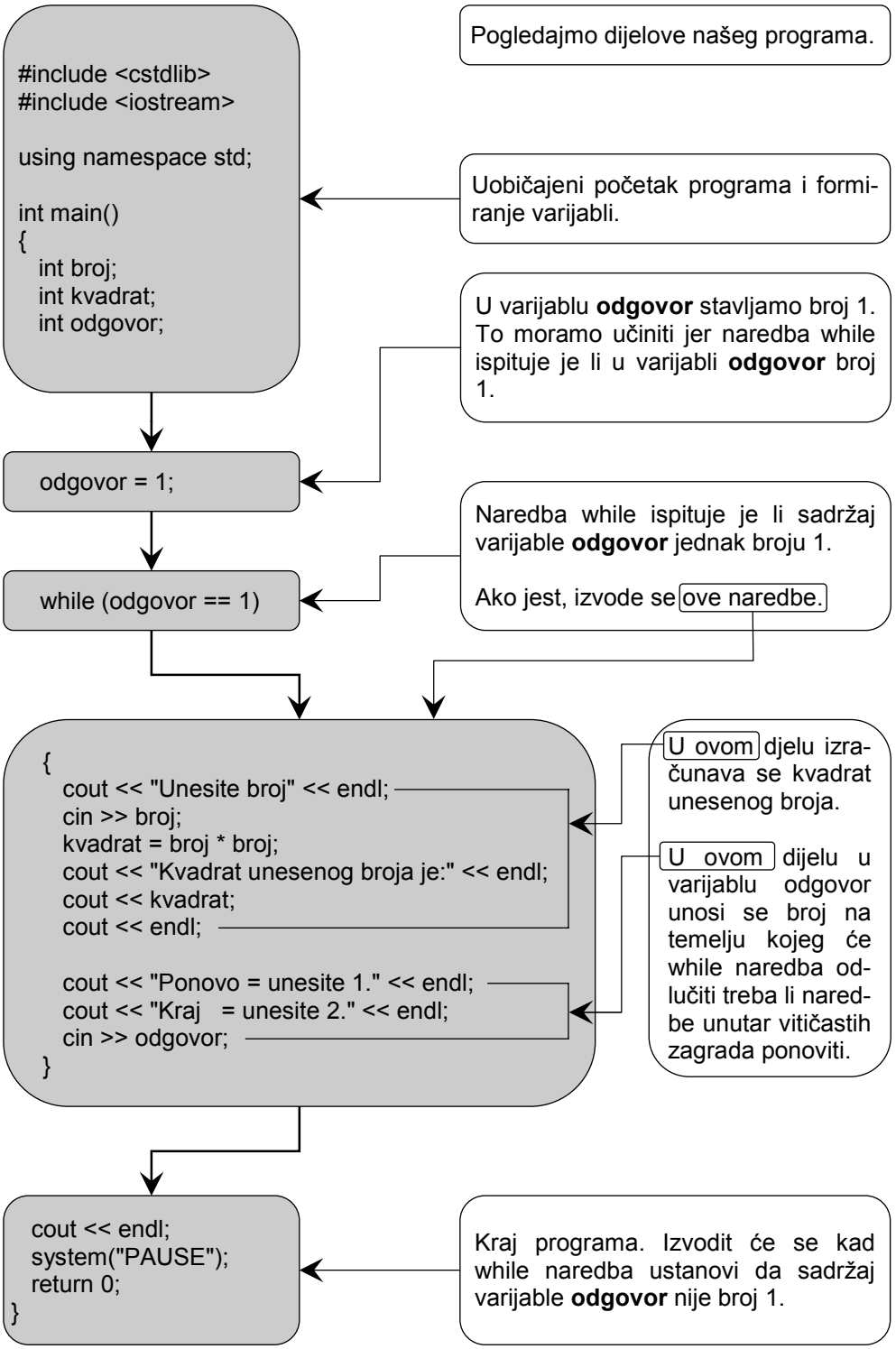
Ovdje se unutar zagrada nalazi uvjet koji **while** naredba ispituje. Uvjeti se formiraju po istoj logici kao i kod for naredbe.

Ako je uvjet zadovoljen, izvršit će se naredbe unutar vitičastih zagrada, a nakon toga se računalo vraća na while naredbu i ponovo ispituje je li uvjet zadovoljen.

Ako jest, ponovo se izvode naredbe unutar vitičastih zagrada.

Ako nije, izvodi se prva naredba poslije druge vitičaste zagrade.

U našem slučaju ispituje se je li sadržaj varijable **a** jednak broju **1**. Zato prije while naredbe moramo u varijablu **a** staviti broj jedan. Kad bi u varijabli **a** bio neki drugi broj, računalo bi skočilo na prvi red nakon druge vitičaste zagrade.


Pogledajmo dijelove našeg programa.
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
int broj;
int kvadrat;
int odgovor;
Uobičajeni početak programa i formiranje varijabli.
U varijablu odgovor stavljamo broj 1. To moramo učiniti jer naredba while ispituje je li u varijabli odgovor broj 1.
odgovor = 1;
Naredba while ispituje je li sadržaj varijable odgovor jednak broju 1.
Ako jest, izvode se ove naredbe.
while (odgovor == 1)
{
cout << "Unesite broj" << endl;
cin >> broj;
kvadrat = broj * broj;
cout << "Kvadrat unesenog broja je:" << endl;
cout << kvadrat;
cout << endl;
cout << "Ponovo = unesite 1." << endl;
cout << "Kraj = unesite 2." << endl;
cin >> odgovor;
}
U ovom djelu izračunava se kvadrat unesenog broja.
U ovom dijelu u varijablu odgovor unosi se broj na temelju kojeg će while naredba odlučiti treba li naredbe unutar vitičastih zagrada ponoviti.
cout << endl;
system("PAUSE");
return 0;
}
Kraj programa. Izvodit će se kad while naredba ustanovi da sadržaj varijable odgovor nije broj 1.

Varijablu koju će koristiti while naredba moramo prije while naredbe formirati i moramo joj dati odgovarajuću početnu vrijednost.

Zašto je u našem slučaju odgovarajuća početna vrijednost 1?

Zato što while naredba ispituje je li vrijednost varijable **odgovor** 1.

Ako bi vrijednost varijable **odgovor** bila npr. 2, nakon while naredbe program bi skočio na ovu naredbu i sadržaj unutar navodnika ne bi se izvršavao.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int odgovor;
   odgovor = 1;
   cout << "Prije while naredbe." << endl;
   while (odgovor == 1)
   {
      cout << "Prvi red bloka ponavljanja" << endl;
      cout << "1 = nastavak ili 2 = kraj." << endl;
      cin >> odgovor;
      cout << "Zadnji red bloka ponavljanja" << endl;
   }
   cout << "Nakon while naredbe." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

```
Prije while naredbe.
Prvi red bloka ponavljanja
1 = nastavak ili 2 = kraj.
1
Zadnji red bloka ponavljanja
Prvi red bloka ponavljanja
1 = nastavak ili 2 = kraj.
2
Zadnji red bloka ponavljanja
Nakon while naredbe.

Press any key to continue . . .
```

Ovdje naredbom **cin** preko tipkovnice unosimo novu vrijednost varijable odgovor i time biramo hoćemo li petlju ponoviti ili ćemo izaći iz programa.

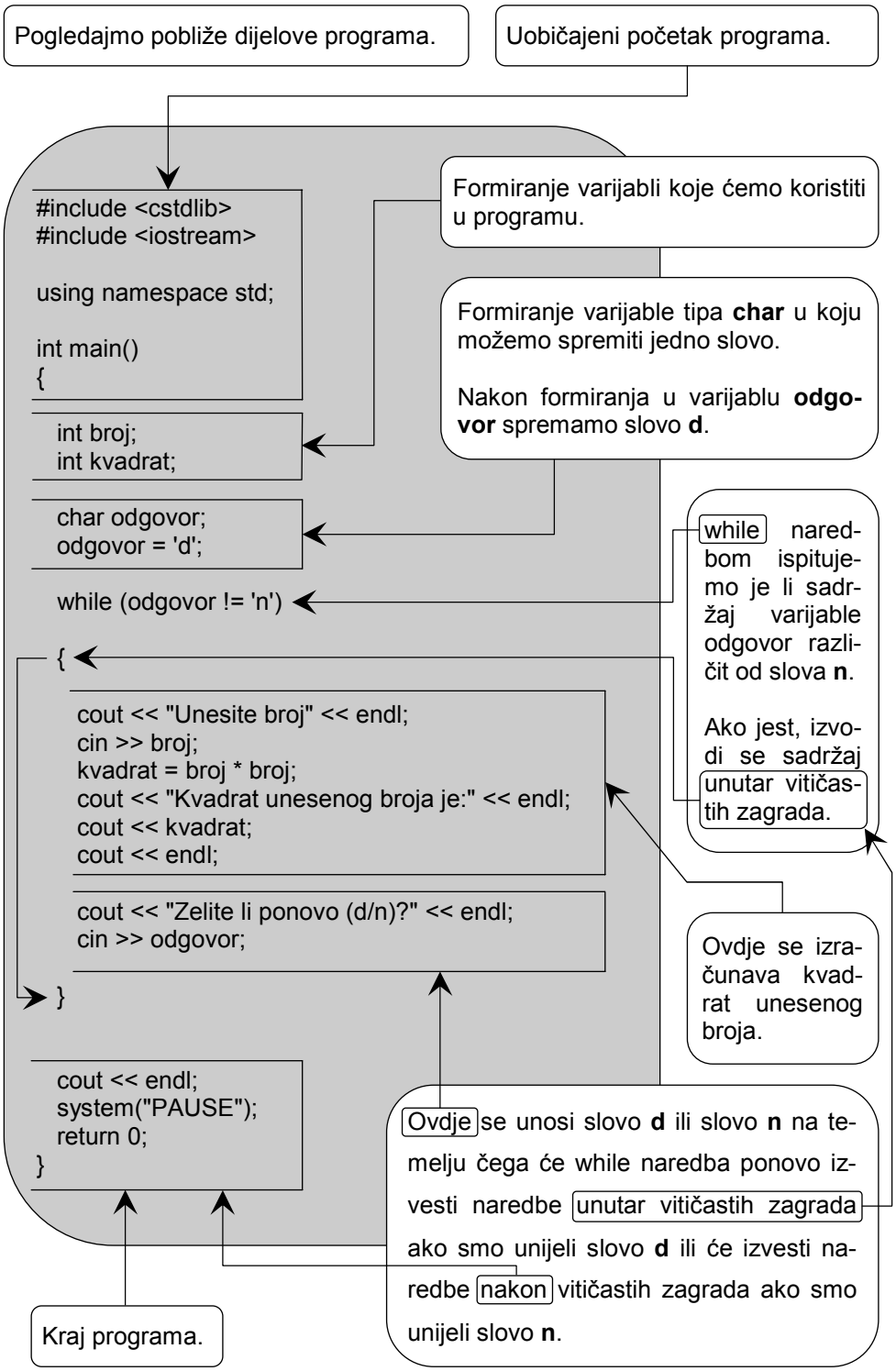
U prethodnim programima korisnik je mogao izabrati hoće li program izvoditi ponovo, ali biranje je funkcioniralo na vrlo neprirodan način. Nije baš logično unijeti broj 1 za ponavljanje proračuna ili broj 2 za prekid programa.

Ova inačica programa to biranje izvodi na elegantniji način. Umjesto unošenja brojeva kao odgovora, u ovom programu unosimo slova.

Da bi to bilo moguće, upotrijebili smo novi tip varijable, varijablu **char** tipa u koju možemo spremiti jedno slovo.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int broj;
   int kvadrat;
   char odgovor;
   odgovor = 'd';
   while (odgovor != 'n')
   {
      cout << "Unesite broj" << endl;
      cin >> broj;
      kvadrat = broj * broj;
      cout << "Kvadrat unesenog broja je:" << endl;
      cout << kvadrat;
      cout << endl;
      cout << "Zelite li ponovo (d/n)?" << endl;
      cin >> odgovor;
   }
   cout << endl;
   system("PAUSE");
   return 0;
}
```

```
Unesite broj
3
Kvadrat unesenog broja je:
9
Zelite li ponovo (d/n)?
d
Unesite broj
4
Kvadrat unesenog broja je:
16
Zelite li ponovo (d/n)?
n

Press any key to continue . . .
```


Pogledajmo pobliže dijelove programa.
Uobičajeni početak programa.
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
Formiranje varijabli koje ćemo koristiti u programu.
int broj;
int kvadrat;
Formiranje varijable tipa char u koju možemo spremiti jedno slovo.
Nakon formiranja u varijablu odgovor spremamo slovo d.
char odgovor;
odgovor = 'd';
while (odgovor != 'n')
while naredbom ispitujemo je li sadržaj varijable odgovor različit od slova n.
Ako jest, izvodi se sadržaj unutar vitičastih zagrada.
{
cout << "Unesite broj" << endl;
cin >> broj;
kvadrat = broj * broj;
cout << "Kvadrat unesenog broja je:" << endl;
cout << kvadrat;
cout << endl;
cout << "Zelite li ponovo (d/n)?" << endl;
cin >> odgovor;
}
Ovdje se izračunava kvadrat unesenog broja.
cout << endl;
system("PAUSE");
return 0;
}
Ovdje se unosi slovo d ili slovo n na temelju čega će while naredba ponovo izvesti naredbe unutar vitičastih zagrada ako smo unijeli slovo d ili će izvesti naredbe nakon vitičastih zagrada ako smo unijeli slovo n.
Kraj programa.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x;
   char odgovor;
   odgovor = 'd';
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   while (odgovor != 'n')
   {
      cleardevice();
      for (x = 10; x < 465; x = x + 1 )
      {
         cleardevice();
         circle (320, x, 10);
      }
      settextstyle(9, HORIZ_DIR, 1);
      outtextxy(20, 20, "Želite li ponovo (d/n)?");
      odgovor = getch();
   }
   closegraph();
   return 0;
}
```

While naredbu možemo iskoristiti i za višestruko pokretanje grafičkih programa.

Mali krug kreće se od gore prema dolje, a zatim nas računalo pita želimo li ponoviti animaciju.

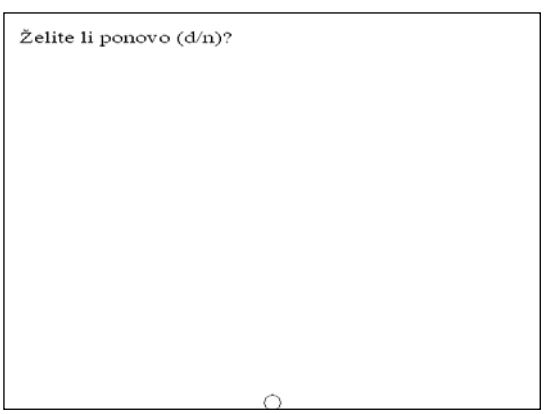
U ovom programu koristimo dvije različite vrste petlji.

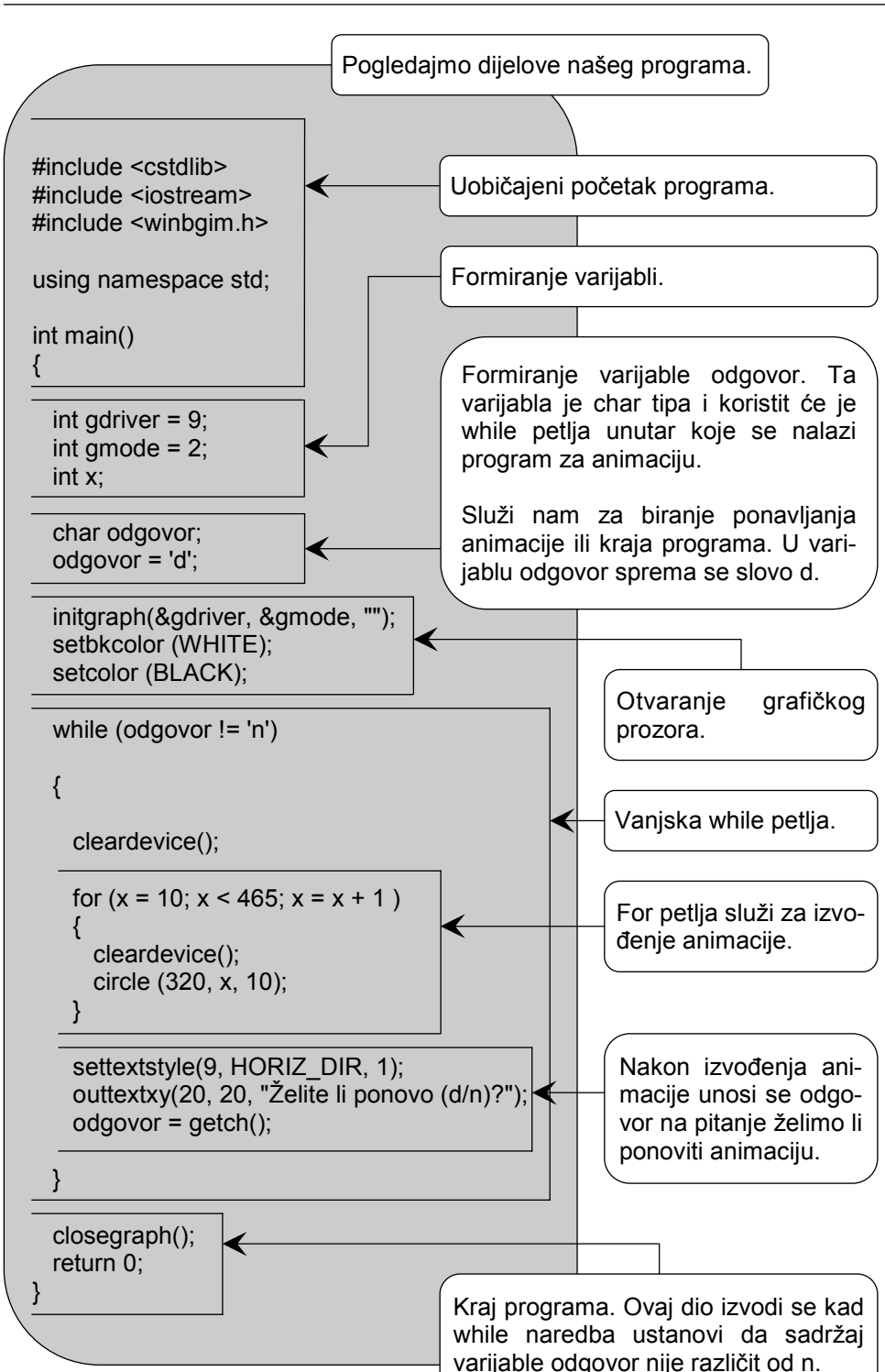
While naredbom ponavlja se animacija, a for naredbom, koja se nalazi unutar while petlje, animira se kretanje kruga.

U grafičkom prozoru za unos slova u varijablu **odgovor** ne koristi se **cin** naredba, nego **getch()** naredba.

Ako koristimo **getch()** naredbu, nakon unosa ne koristima Enter tipku.

Dovoljno je pritisnuti slovo **d** ili slovo **n**.




Pogledajmo dijelove našeg programa.
#include <cstdlib>
#include <iostream>
#include <winbgim.h>
using namespace std;
int main()
{
Uobičajeni početak programa.
int gdriver = 9;
int gmode = 2;
int x;
Formiranje varijabli.
char odgovor;
odgovor = 'd';
Formiranje varijable odgovor. Ta varijabla je char tipa i koristit će je while petlja unutar koje se nalazi program za animaciju.
Služi nam za biranje ponavljanja animacije ili kraja programa. U varijablu odgovor sprema se slovo d.
initgraph(&gdriver, &gmode, "");
setbkcolor (WHITE);
setcolor (BLACK);
Otvaranje grafičkog prozora.
while (odgovor != 'n')
{
cleardevice();
Vanjska while petlja.
for (x = 10; x < 465; x = x + 1)
{
cleardevice();
circle (320, x, 10);
}
For petlja služi za izvođenje animacije.
settextstyle(9, HORIZ_DIR, 1);
outtextxy(20, 20, "Želite li ponovo (d/n)?");
odgovor = getch();
Nakon izvođenja animacije unosi se odgovor na pitanje želimo li ponoviti animaciju.
}
closegraph();
return 0;
}
Kraj programa. Ovaj dio izvodi se kad while naredba ustanovi da sadržaj varijable odgovor nije različit od n.

Ovaj program omogućit će nam da lijevom i desnom strelicom na tipkovnici kontroliramo kretanje pravokutnika u lijevu ili desnu stranu.

Program napuštamo pritiskom na tipku **k**.

Ovaj program već bismo mogli smatrati jednim elementom primitivne igrice. Dovoljno bi bilo dodati kretanje malih krugova od gore prema dolje i brojač koji bi brojio koliko krugova smo ovim pravokutnikom uspjeli “uhvatiti”.

Umjesto:
int x;
x =290;

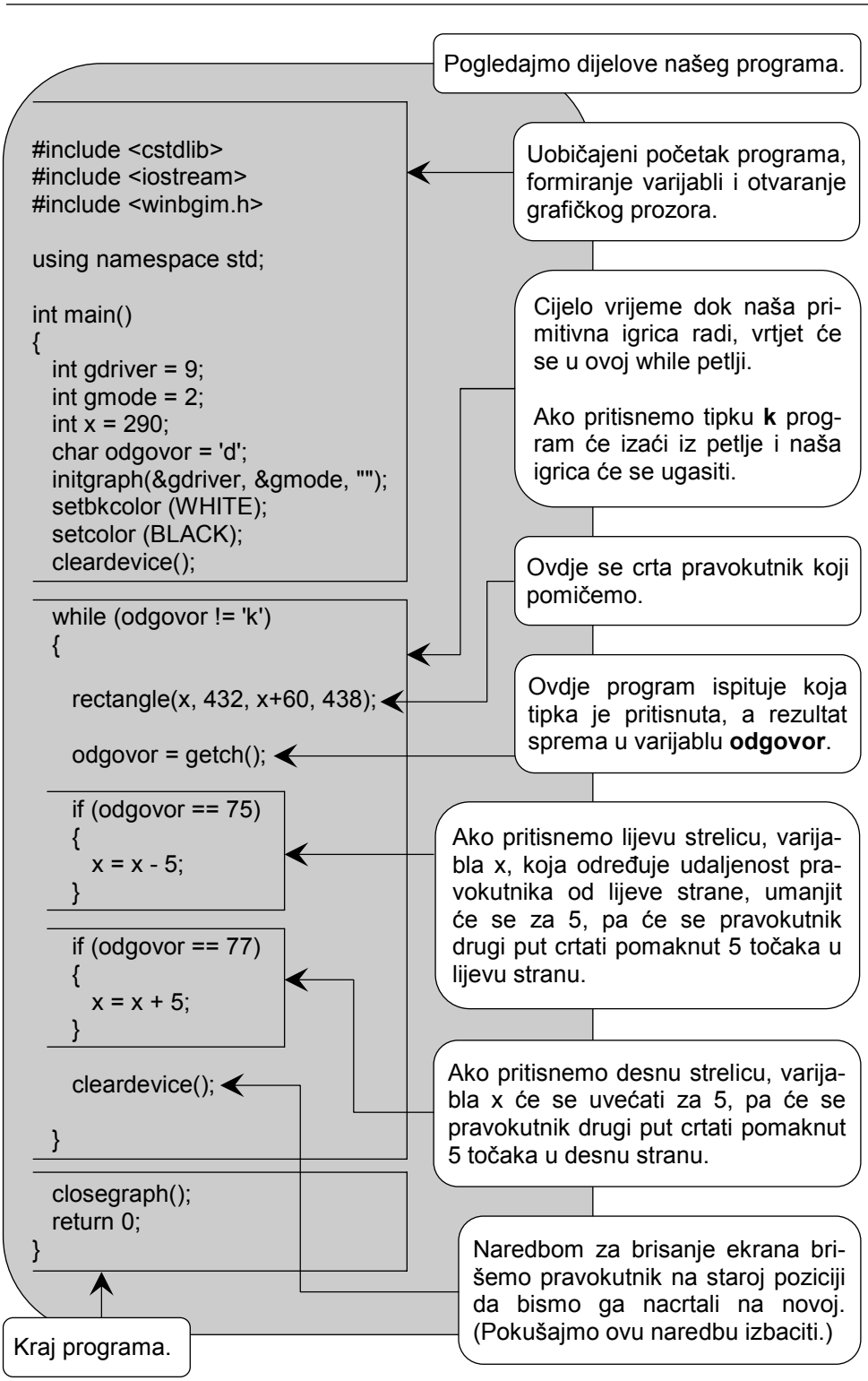
možemo napisati
int x = 290;

Umjesto:
char odgovor;
odgovor = ‘d’;

možemo napisati
char odgovor = 'd';

Pravokutnik koji možemo pomicati u lijevu i desnu stranu.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
    int x = 290;
   char odgovor = 'd';
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   while (odgovor != 'k')
   {
      rectangle(x, 432, x+60, 438);
      odgovor = getch();
      if (odgovor == 75)
      {
         x = x - 5;
      }
      if (odgovor == 77)
      {
         x = x + 5;
      }
      cleardevice();
   }
   closegraph();
   return 0;
}
```


Pogledajmo dijelove našeg programa.
#include <cstdlib>
#include <iostream>
#include <winbgim.h>
using namespace std;
int main()
{
int gdriver = 9;
int gmode = 2;
int x = 290;
char odgovor = 'd';
initgraph(&gdriver, &gmode, "");
setbkcolor (WHITE);
setcolor (BLACK);
cleardevice();
Uobičajeni početak programa, formiranje varijabli i otvaranje grafičkog prozora.
while (odgovor != 'k')
{
rectangle(x, 432, x+60, 438);
odgovor = getch();
Cijelo vrijeme dok naša primitivna igrica radi, vrtjet će se u ovoj while petlji.
Ako pritisnemo tipku k program će izaći iz petlje i naša igrica će se ugasiti.
Ovdje se crta pravokutnik koji pomičemo.
Ovdje program ispituje koja tipka je pritisnuta, a rezultat sprema u varijablu odgovor.
if (odgovor == 75)
{
x = x - 5;
}
Ako pritisnemo lijevu strelicu, varijabla x, koja određuje udaljenost pravokutnika od lijeve strane, umanjit će se za 5, pa će se pravokutnik drugi put crtati pomaknut 5 točaka u lijevu stranu.
if (odgovor == 77)
{
x = x + 5;
}
Ako pritisnemo desnu strelicu, varijabla x će se uvećati za 5, pa će se pravokutnik drugi put crtati pomaknut 5 točaka u desnu stranu.
cleardevice();
}
closegraph();
return 0;
}
Naredbom za brisanje ekrana brišemo pravokutnik na staroj poziciji da bismo ga nacrtali na novoj. (Pokušajmo ovu naredbu izbaciti.)
Kraj programa.

Dodamo li u prethodni program ispitivanje je li pritisnuta strelica prema gore ili strelica prema dolje, naš pravokutnik moći će se pomicati u sva četiri smjera.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x = 290;
   int y = 220;
   char odgovor = 'd';
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   while (odgovor != 'k')
   {
      rectangle(x, y, x+60, y+5);
      odgovor = getch();
      if (odgovor == 75)
      {
         x = x - 5;
      }
      if (odgovor == 77)
      {
         x = x + 5;
      }
      if (odgovor == 72)
      {
         y = y - 5;
      }
      if (odgovor == 80)
      {
         y = y + 5;
      }
      cleardevice();
   }
   closegraph();
   return 0;

}
```

Varijablom x određujemo udaljenost gornjeg lijevog ugla pravokutnika od lijevog ruba grafičkog prozora, a varijablom y od gornjeg ruba grafičkog prozora. Brojevi 290 i 220 određuju početnu poziciju pravokutnika. (Pokušajmo mijenjati te brojeve.)

I u ovom slučaju program će se izvoditi dok korisnik ne pritisne tipku **k**, odnosno dok je pritisnuta tipaka različita od **k**.

Ovdje smo dodali dio programa u kojem ispitujemo je li pritisnuta strelica prema gore ili strelica prema dolje.

Zašto ovdje koristimo broj 72, a ne strelicu prema gore?

Zato što strelicu prema gore ne možemo unijeti u program preko tipkovnice, a broj 72 je šifra za strelicu prema gore. (75 za lijevo, 77 za desno i 80 za dolje.)

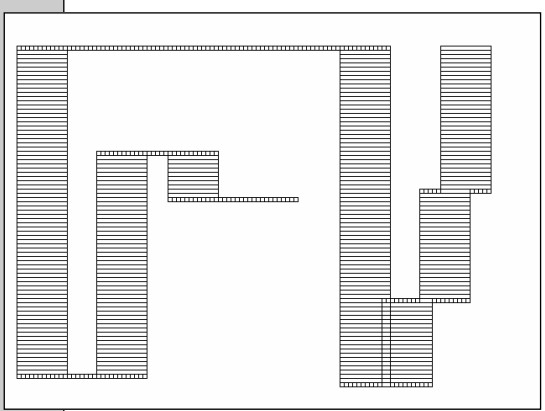
Da bismo dobili primitivni program za crtanje, dovoljno je da u prethodnom programu maknemo naredbu za brisanje zaslona računala.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
  int gdriver = 9;
  int gmode = 2;
  int x = 290;
  int y = 220;
  char odgovor = 'd';
  initgraph(&gdriver, &gmode, "");
  setbkcolor (WHITE);
  setcolor (BLACK);
  cleardevice();
  while (odgovor != 'k')
  {
    rectangle(x, y, x+60, y+5);
    odgovor = getch();
    if (odgovor == 75)
    {
      x = x - 5;
    }
    if (odgovor == 77)
    {
      x = x + 5;
    }
    if (odgovor == 72)
    {
      y = y - 5;
    }
    if (odgovor == 80)
    {
      y = y + 5;
    }
    cleardevice();
  }
  closegraph();
  return 0;

}
```

Varijablama `x i y` određena je pozicija gornjeg lijevog ugla pravokutnika. Varijabla x određuje udaljenost od lijevog ruba, a varijabla y od gornjeg ruba.

Udaljenost donjeg desnog ugla pravokutnika od lijevog ruba određena je sa `x + 60,` a udaljenost donjeg desnog ugla od gornjeg ruba određena je sa `y+5.`

Širina našeg pravokutnika je 60, a visina 5. (Pokušajmo promijeniti te brojeve.)



Ako ovu naredbu obrišemo, pravokutnik će se crtati na novoj poziciji, ali će ostati i na staroj pa će rezultat kretanja pravokutnika biti trag na zaslonu računala.

Obrišimo ovu naredbu i pokrenimo program.

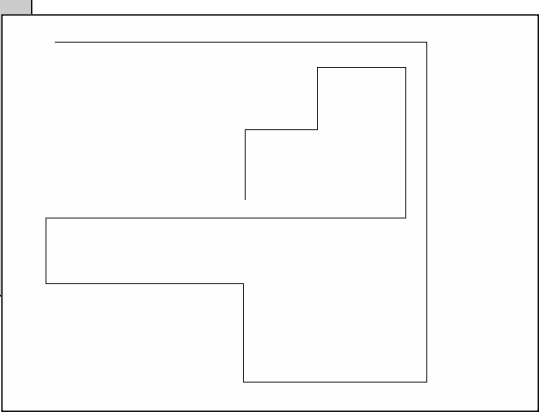
Želimo li napraviti program za crtanje, pogodnije je za crtanje koristiti točku umjesto pravokutnika.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   int x = 290;
   int y = 220;
   char odgovor = 'd';
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   while (odgovor != 'k')
   {
      putpixel(x, y, BLACK);
      odgovor = getch();
      if (odgovor == 75)
      {
         x = x - 1;
      }
      if (odgovor == 77)
      {
         x = x + 1;
      }
      if (odgovor == 72)
      {
         y = y - 1;
      }
      if (odgovor == 80)
      {
         y = y + 1;
      }
   }
   closegraph();
   return 0;
}
```

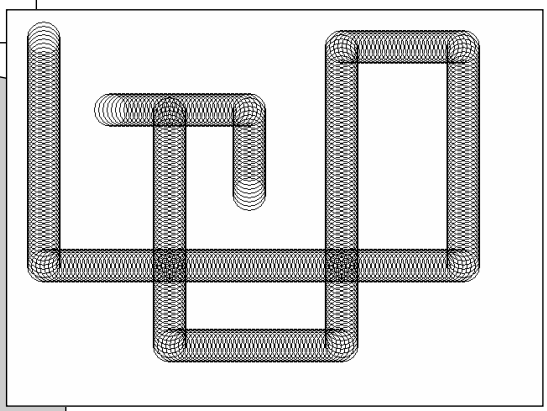
Ovom naredbom ispisujemo jednu točku na zaslon računala. Varijabla x određuje udaljenost točke od lijevog ruba, varijabla y od gornjeg ruba, a BLACK određuje boju točke.

Želimo li dobiti punu crtu, točku moramo pomicati za 1.

Pokušajmo umjesto 1 staviti 2 ili 5.



Za crtanje možemo koristiti i krug.



```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
  int gdriver = 9;
  int gmode = 2;
  int x = 290;
  int y = 220;
  char odgovor = 'd';
  initgraph(&gdriver, &gmode, "");
  setbkcolor (WHITE);
  setcolor (BLACK);
  cleardevice();
  while (odgovor != 'k')
  {
    circle(x, y, 20);
    odgovor = getch();
    if (odgovor == 75)
    {
      x = x - 5;
    }
    if (odgovor == 77)
    {
      x = x + 5;
    }
    if (odgovor == 72)
    {
      y = y - 5;
    }
    if (odgovor == 80)
    {
      y = y + 5;
    }
  }
  closegraph();
  return 0;
}
```

Ovom naredbom ispisujemo krug. Varijabla x određuje udaljenost središta kruga od lijevog ruba, varijabla y udaljenost središta od gornjeg ruba, a 20 je polumjer kruga.

Broj 5 određuje pomak novog središta u odnosu na staro.

Pokušajmo mijenjati taj broj.

Imajmo stalno na umu da ćemo razumijevanje programa i vještinu pisanja vlastitih programa steći jedino tako da svaki primjer programa koji je dat u ovoj knjizi mijenjamo na sve načine koji nam padnu na pamet i onda promatramo kakve posljedice izaziva određena promjena.

Objekte na zaslonu računala osim tipkovnicom možemo pomicati i mišem.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   while (!kbhit())
   {
   rectangle(mousex(),mousey(),mousex()+80,mousey()+80 );
   }
   closegraph();
   return 0;
}
```

Naredbom **kbhit** ispituje se je li bilo koja tipka na tipkovnici pritisnuta.

Naredbom while (!kbhit) naređujemo računalu da petlju izvodi tako dugo dok ni jedna tipka nije pritisnuta.

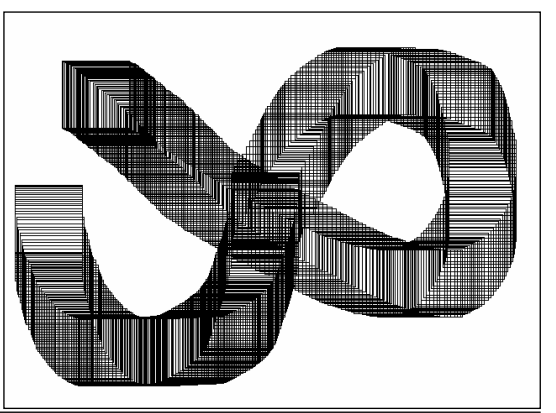
Pritisnemo li bilo koju tipku na tipkovnici, zaustavlja se izvođenje petlje i prekida se program.

Naredbom mousex() očitava se udaljenost pokazivača miša od lijevog ruba grafičkog prozora.

Naredbom mousey() očitava se udaljenost pokazivača miša od gornjeg ruba grafičkog prozora.

Gornji lijevi ugao pravokutnika određen je položajem pokazivača miša, a donji desni tako da se udaljenost pokazivača miša od desnog ruba uvećava za 80, a udaljenost pokazivača miša od gornjeg ruba uvećava isto tako za 80.

Dimenzije našeg pravokutnika su 80 puta 80.



Mišem možemo pokretati bilo koji grafički objekt, npr. krug.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
  int gdriver = 9;
  int gmode = 2;

  initgraph(&gdriver, &gmode, "");
  setbkcolor (WHITE);
  setcolor (BLACK);
  cleardevice();

  while (!kbhit())
  {
    circle(mousex(),mousey(),100);
  }

  closegraph();
  return 0;
}
```

Početak programa.

Varijable kojima određujemo tip grafičkog prozora.

Otvaranje grafičkog prozora.

Unutar while petlje stalno se očitava pozicija pokazivača miša i crta se krug čije središte određuje položaj pokazivača miša, a polumjer je 100.

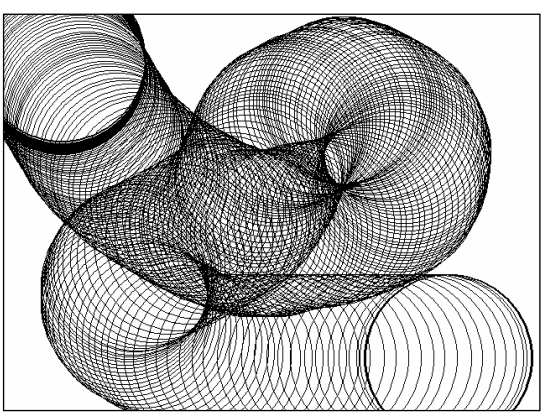
Petlja se izvodi tako dugo dok niti jedna tipka na tipkovnici nije pritisnuta.

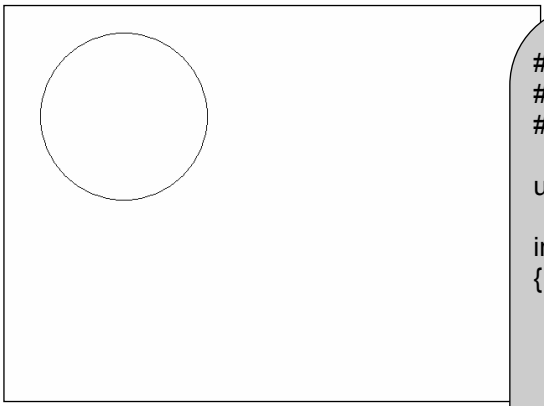
Kraj programa.

Jasno je da osim postojećih dimenzija kruga i pravokutnika možemo koristi pravokutnike i krugove drugih dimenzija.

Isto tako možemo koristiti i druge grafičke elemente, npr. točke ili crte.

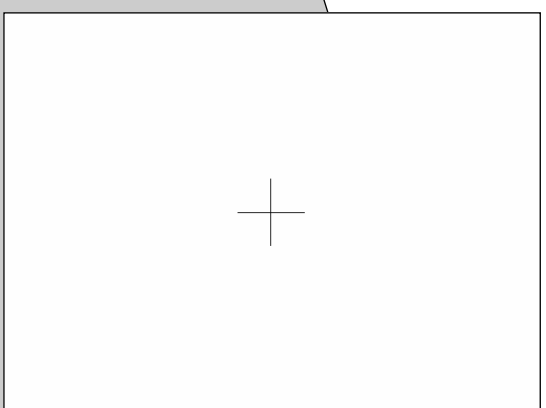
Pokušajmo sami smisliti varijacije ovakvih programa.





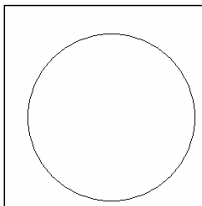
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   while (!kbhit())
   {
      cleardevice();
      circle(mousex(),mousey(),100);
   }
   closegraph();
   return 0;
}
```

Ako u prethodne programe dodamo naredbu za brisanje sadržaja grafičkog prozora, dobit ćemo program koji nam omogućuje pomicanje objekta mišem.



```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   while (!kbhit())
   {
      cleardevice();
      line (mousex()+40, mousey(), mousex()-40, mousey());
      line (mousex(), mousey()-40, mousex(), mousey()+40);
   }
   closegraph();
   return 0;
}
```

Na sličan način možemo mišem pomicati nešto drugo, npr. dvije crte koje bi u igri mogle biti nišan.



Nedostatak prethodnih programa bio je što se ekran briše i objekt ponovo crta, bez obzira je li to potrebno ili nije, odnosno bez obzira je li došlo do pomaka miša ili nije.

Ako nije došlo do pomaka miša, nema potrebe objekt brisati i ponovo crtati.

Neprestano brisanje i crtanje može na sporijim računalima izazvati treperenje slike, što je veoma neugodno.

Ovaj program ispituje je li došlo do pomaka miša ili nije. Ekran se briše i objekt crta na novoj poziciji jedino ako je došlo do pomicanja miša.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
  int gdriver = 9;
  int gmode = 2;
  initgraph(&gdriver, &gmode, "");
  setbkcolor (WHITE);
  setcolor (BLACK);
  cleardevice();
  while (!kbhit())
  {
    if (ismouseclick(WM_MOUSEMOVE))

    {
      cleardevice();
      circle(mousex(),mousey(),100);
    }

    clearmouseclick(WM_MOUSEMOVE);

  }
  closegraph();
  return 0;
}
```

Ovom naredbom ispitujemo je li došlo do pomaka miša.

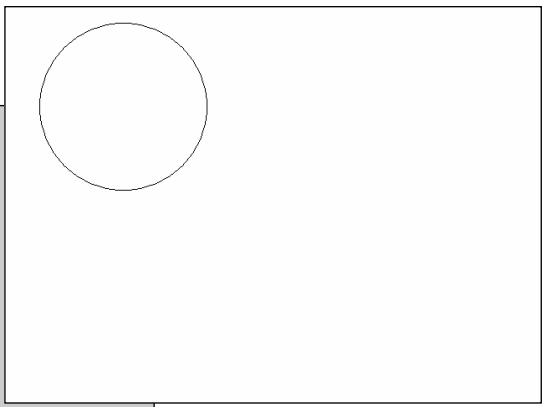
Sadržaj unutar vitičastih zagrada izvršit će se jedino ako je došlo do pomaka miša.

U tom slučaju obrisat će se sadržaj grafičkog prozora i objekt će se nacrtati na novoj poziciji.

Da bi bilo moguće novo očitavanje miša, rezultat starog očitavanja ovom se naredbom mora obrisati.

Kad ovu naredbu ne bismo upotrijebili, u programu bi bilo moguće samo jedno očitavanje miša.

Osim što možemo očitati je li došlo do pomaka miša, možemo isto tako očitati je li pritisnuta neka tipka na mišu. Ovaj program će nam omogućiti da pritiskom na lijevu tipku miša nacrtamo objekt u grafičkom prozoru.



```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   while (!kbhit())
   {
      if (ismouseclick(WM_LBUTTONDOWN))
      {
         cleardevice();
         circle(mousex(),mousey(),100);
      }
      clearmouseclick(WM_LBUTTONDOWN);
   }
   closegraph();
   return 0;
}
```

Ovom naredbom ispitujemo je li pritisnuta lijeva tipka miša.

Ako jest izvodi se sadržaj unutar vitičastih zagrada.

Ovom naredbom brišemo rezultat očitavanja lijeve tipke na mišu.

Novo očitavanje je moguće jedino nakon brisanja starog očitavanja.

Uočimo, ako smo očitavali **ismouseclick(WM_MOUSEMOVE)**, brišemo **clearmouseclick(WM_MOUSEMOVE)**,

a ako smo očitavali **ismouseclick(WM_LBUTTONDOWN)**, brišemo **clearmouseclick(WM_LBUTTONDOWN)**.

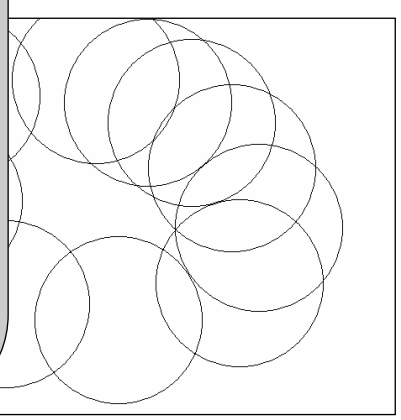
OČITAVANJE MIŠA	
ismouseclick(WM_MOUSEMOVE)	Očitavanje kretanja miša.
clearmouseclick(WM_MOUSEMOVE)	Brisanje očitavanja kretanja miša.
ismouseclick(WM_LBUTTONDOWN)	Očitavanje lijevog klika
clearmouseclick(WM_LBUTTONDOWN);	Brisanje očitavanja lijevog klika.
ismouseclick(WM_LBUTTONDBLCLK)	Očitavanje duplog lijevog klika.
clearmouseclick(WM_LBUTTONDBLCLK)	Brisanje očitavanja duplog lijevog klika.
ismouseclick(WM_RBUTTONDOWN)	Očitavanje desnog klika.
clearmouseclick(WM_RBUTTONDOWN)	Brisanje očitavanja desnog klika.
ismouseclick(WM_RBUTTONDBLCLK)	Očitavanje duplog desnog klika.
clearmouseclick(WM_RBUTTONDBLCLK)	Brisanje očitavanja duplog desnog klika.
ismouseclick(WM_MBUTTONDOWN)	Očitavanje srednjeg klika.
clearmouseclick(WM_MBUTTONDOWN)	Brisanje očitavanja srednjeg klika.
ismouseclick(WM_MBUTTONDBLCLK)	Očitavanje duplog srednjeg klika.
clearmouseclick(WM_MBUTTONDBLCLK)	Brisanje očitavanja duplog srednjeg klika.

Do while petlja

Pokrenemo li ova dva programa, vidjet ćemo da je rezultat njihovog rada potpuno jednak, iako prvi koristi **while** naredbu, a drugi **do while** naredbu.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   while (!kbhit())
   {
      if (ismouseclick(WM_LBUTTONDOWN))
      {
         circle(mousex(),mousey(),100);
      }
      clearmouseclick(WM_LBUTTONDOWN);
   }
   closegraph();
   return 0;
}
```

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   do
   {
      if (ismouseclick(WM_LBUTTONDOWN))
      {
         circle(mousex(),mousey(),100);
      }
      clearmouseclick(WM_LBUTTONDOWN);
   }
   while (!kbhit());
   closegraph();
   return 0;
}
```



Usporedimo strukture **while** petlje i **do while** petlje.

```
a = 1;

while  (a == 1)

{

   cin >> a;

}
```

Određivanje vrijednosti varijable koju će koristiti while naredba.

Ispitivanje je li uvjet zadovoljen, u našem slučaju, je li sadržaj varijable **a** jednak broju 1.

Ovaj dio unutar vitičastih zagrada ponavlja se dok je uvjet zadovoljen, u našem slučaju dok je sadržaj varijable a jednak broju 1.

Petlja **do while** započinje određivanjem vrijednosti varijable koju će koristiti while naredba.

```
a = 1;

do

{

   cin >> a;

}

while  (a == 1);
```

Ovaj tip petlje započinje naredbom **do** nakon koje slijede vitičaste zagrade unutar kojih se nalazi dio programa koji se ponavlja dok je uvjet zadovoljen

Dio koji se ponavlja i u kojem mora postojati mogućnost promjene vrijednosti varijable a.

Ovdje se ispituje je li zadovoljen uvjet po istoj logici po kojoj se ispituje i u while naredbi.

Uočimo da se ovdje na kraju while naredbe obavezno piše **;** oznaka.

U čemu je razlika između ovih dviju petlji, osim očigledne razlike u načinu pisanja? Razlika je u tome što se sadržaj **while** petlje neće izvesti niti jednom ako uvjet nije zadovoljen jer se ispitivanje nalazi prije djela koji se ponavlja. Sadržaj **do while** petlje će se bez obzira na uvjet izvesti barem jednom jer se najprije izvodi dio koji se ponavlja, a tek nakon toga ispituje se je li uvjet zadovoljen.

Generator slučajnih brojeva

Ponekad, osobito u igricama, htjeli bismo da se neki dijelovi u programu pri svakom pokretanju programa odvijaju drugačije. Ako pišemo igricu u kojoj igrač gađa "leteće tanjure", htjeli bismo da se pri svakom pokretanju igre oni pojave na drugom mjestu i da se svaki put kreću drugačijim putanjama. Igrica će nam brzo dosaditi ako se objekt koji trebamo gađati pri svakom pokretanju igrice pojavljuje na istom mjestu i kreće istom putanjom.

Takve varijacije u odvijanju programa postižemo generatorom slučajnih brojeva. Pogledajmo kako ćemo formirati program za generiranje slučajnih brojeva.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  int x;
  for (x = 0; x < 11; x = x + 1)
  {
    cout << rand() << endl;
  }
  system("PAUSE");
  return 0;
}
```

```
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
5705
Press any key to continue . . .
```

Slučajne brojeve generirat ćemo naredbom **rand()**.

Problem je u tome što će ovako napisani program pri svakom pokretanju generirati iste "slučajne brojeve".

Jasno, od takvih "slučajnih brojeva" ne možemo imati puno koristi.

Pokušajmo napisati program koji neće svaki put generirati isti niz brojeva.

```
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
5705
Press any key to continue . . .
```

Problem koji se sastoji u tome da naredba rand(); uvijek generira isti niz brojeva riješit ćemo naredbom **srand(1);**.

```
41
18467
6334
26500
19169
15724
11478
29358
26962
24464
5705
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  int x;
  srand(1);
  for (x = 0; x < 11; x = x + 1)
  {
    cout << rand() << endl;
  }
  system("PAUSE");
  return 0;
}
```

Stavimo li naredbu **srand(1)** prije naredbe rand() postići ćemo da naredba rand() generira neki niz brojeva.

Ako umjesto srand(1); stavimo srand (2); dobit ćemo neki drugi niz brojeva. Pokušajmo i vidjet ćemo da ćemo za srand(3); dobit neki treći niz brojeva, a za srand(4); neki četvrti.

Problem je u tome što ćemo za isti broj unutar zagrade dobivati uvijek isti niz. Dakle srand(6) davat će pri svakom pokretanju isti niz brojeva, ali drugačiji nego npr. srand(5). Očito bi bilo dobro kad bismo na neki način broj unutar zagrade mogli mijenjati.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  int x;
  srand(2);
  for (x = 0; x < 11; x = x + 1)
  {
    cout << rand() << endl;
  }
  system("PAUSE");
  return 0;
}
```

```
45
29216
24198
17795
29484
19650
14590
26431
10705
18316
5557
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <ctime>

using namespace std;

int main()
{
   cout << time(NULL) << endl;
   system("PAUSE");
   return 0;
}
```

Da bi naredba **time(NULL)** funkcionirala, moramo naredbom **#include <ctime>** uključiti odgovarajuću biblioteku.

Različite brojeve dobit ćemo naredbom **time(NULL)**. Iako se čini logičnim da naredba time(NULL) pokazuje trenutno vrijeme, ona to ne čini, nego prikazuje trenutno sistemsko vrijeme. To je vrijeme koje je proteklo od 1. siječnja 1970. godine.

```
1126867538
Press any key to continue . . .
```

```
1126867565
Press any key to continue . . .
```

Budući da program ne možemo više puta pokrenuti u istom trenutku, pri svakom pokretanju ove naredbe dobit ćemo drugi broj.

```
1126867586
Press any key to continue . . .
```

```
1126868268
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <ctime>

using namespace std;

int main()
{
   int x;
   srand(time(NULL));
   for (x = 0; x < 11; x = x + 1)
   {
      cout << rand() << endl;
   }
   system("PAUSE");
   return 0;
}
```

Stavimo li u naredbu **srand(1);** umjesto konkretnog broja jedan naredbu **time (NULL)** koja će u svakom trenutku generirati drugačiji broj, naredba **rand()** pri svakom će pokretanju generirati drugačiji brojčani niz. Sada smo već blizu onome što smo željeli, a to je generator slučajnih brojeva.

```
6089
26708
9130
25140
11941
25619
6841
15463
26412
21856
9577
Press any key to continue . . .
```

```
6132
2598
11988
10283
14958
11130
14535
10176
11687
7467
7654
Press any key to continue . . .
```

U ovom rješenju problematično je to što su često početni brojevi veoma slični. Prvi put smo dobili 6089, drugi put 6132, a treći put 6171.

```
6171
507
29750
4131
7661
25483
19116
7817
13218
31994
5880
Press any key to continue . . .
```

Taj problem možemo najjednostavnije eliminirati tako da prvi broj koji generira generator slučajnih brojeva ne koristimo u programu.

```
#include <cstdlib>
#include <iostream>
#include <ctime>

using namespace std;

int main()
{
  int x;
  srand(time(NULL));
  rand();
  for (x = 0; x < 11; x = x + 1)
  {
    cout << rand() << endl;
  }
  system("PAUSE");
  return 0;
}
```

Problem koji bi mogla stvarati pojava da naš generator slučajnih brojeva generira niz brojeva u kojem prvi broj u nizu ima sličnu vrijednost prilikom više pokretanja riješit ćemo tako da prvi put pozovemo naredbu rand(); i tako generirani broj nećemo ni za što koristiti.

Koristit ćemo slučajne brojeve koji će se generirati prilikom sljedećih pozivanja naredbe rand();.

Na taj način postići ćemo da će pri nizu uzastopnih pokretanja i prvi broj u svakom od nizova bit različit.

```
14351
8307
24782
10582
4197
16629
18171
21235
25952
6606
19798
Press any key to continue . . .
```

```
5694
5449
6871
7564
18685
8936
23458
3192
7572
8529
19225
Press any key to continue . . .
```

```
989
29030
1221
23914
26403
27436
9957
23021
5415
4536
20414
Press any key to continue . . .
```

Vidimo da smo uspjeli dobiti više doista različitih nizova.

Sad kad smo konačno dobili niz slučajnih brojeva, postavlja se pitanje kako postići da ti slučajni brojevi budu unutar unaprijed određenih granica. Ako želim da se objekt pojavi na slučajnoj poziciji unutar prozora širokog 640 točaka, bilo bi zgodno da generator slučajnih brojeva generira slučajni broj između 0 i 640.

```
#include <cstdlib>
#include <iostream>
#include <ctime>

using namespace std;

int main()
{
  int x;
  float sb;
  srand(time(NULL));
  rand();
  for (x = 0; x < 11; x = x + 1)
  {
    sb = rand() / RAND_MAX;
    cout << sb << endl;
  }
  system("PAUSE");
  return 0;
}
```

Najprije ćemo program preurediti tako da generira brojeve između 0 i 1. To ćemo postići tako da generirani broj podijelimo maksimalnom mogućom vrijednošću slučajnog broja. U tom slučaju, ako je generirani slučajni broj nula, rezultat će biti nula, a ako je jednak maksimalnoj mogućoj vrijednosti bit će 1. Za sve ostale vrijednosti dobit ćemo broj između nula i jedan.

Maksimalnu moguću vrijednost generiranog slučajnog broja dobit ćemo naredbom RAND_MAX.

```
0
0
0
0
0
0
0
0
0
0
0
Press any key to continue .
```

Pokrenemo li takav program kao rezultat dobit ćemo samo nule, i možda ponekad broj 1.

Zašto?

Zato što je rezultat rada naredbe rand() i naredbe RAND_MAX int tipa, tj. broj bez decimalnog zareza. Rezultat, doduše, spremamo u varijablu **sb** koja je float tipa, ali računalo radi tako da izračuna rezultat u int formatu, a tek onda broj pretvara u float.

Prvo podijeli dva int broja, makne sadržaj desno od zareza, jer int tip nema zarez, i tek onda rezultat sprema u varijablu float tipa.

```
0.83282
0.870296
0.716178
0.371349
0.123508
0.324747
0.0154118
0.55797
0.0938139
0.47383
0.437117
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <ctime>

using namespace std;

int main()
{
  int x;
  float sb;
  srand(time(NULL));
  rand();
  for (x = 0; x < 11; x = x + 1)
  {
    sb = static_cast <float> (rand()) / RAND_MAX;
    cout << sb << endl;
  }
  system("PAUSE");
  return 0;
}
```

Problem koji se javio u prethodnom programu riješit ćemo tako da računalu naredimo da broj koji će generirati naredba rand() i koji je int tipa, prije dijeljenja s RAND_MAX pretvori u float tip. To ćemo učiniti naredbom:

static_cast <float> (rand())

Nakon primjene te naredbe program će generirati slučajne brojeve između broja 0 i broja 1.

Ovom naredbom naređujemo računalu da jedan tip podatka npr. int pretvori u neki drugi tip, npr. float.

static_cast <float> (rand())

Ovdje navedemo tip u koji neki drugi tip želimo pretvoriti. Mi želimo int tip pretvoriti u float, pa smo zato ovdje napisali float.

Unutar navodnika navedemo varijablu ili izraz čiji tip želimo mijenjati. Mi ćemo mijenjati tip rezultatu izvođenja rand () naredbe.

```
#include <cstdlib>
#include <iostream>
#include <ctime>

using namespace std;

int main()
{
  int x;
  float sb;
  srand(time(NULL));
  rand();
  for (x = 0; x < 11; x = x + 1)
  {
    sb = (static_cast <float> (rand()) / RAND_MAX) * 640;
    cout << sb << endl;
  }
  system("PAUSE");
  return 0;
}
```

```
350.44
473.139
574.959
620.781
226.687
107.269
25.2156
228.132
321.318
379.426
501.832
Press any key to continue . . .
```

Kako ćemo postići da program ne generira brojeve između 0 i 1, nego između 0 i 640? Tako da slučajni broj koji računalo generira između 0 i 1 pomnožimo sa 640.

Ako je slučajni broj imao minimalnu moguću vrijednost 0, rezultat će biti nula.

Ako je slučajni broj imao maksimalnu moguću vrijednost 1, rezultat će biti 640.

Za sve ostale vrijednosti rezultat će biti između 0 i 640, a to je ono što smo htjeli postići.

Pogledajmo dio programa kojim generiramo slučajni broj između 0 i 640.

Ovaj dio pretvara int slučajni broj u float tip.

```
sb = ( static_cast <float> ( rand() ) / RAND_MAX ) * 640;
```

Množimo li broj između 0 i 1 brojem 640, dobit ćemo brojeve između 0 i 640.

Generira slučajne brojeve.

Maksimalna vrijednost koju rand() generira.

Dijeljenjem slučajnog broja maksimalnom mogućom vrijednošću dobivamo broj između broja 0 i broja 1.

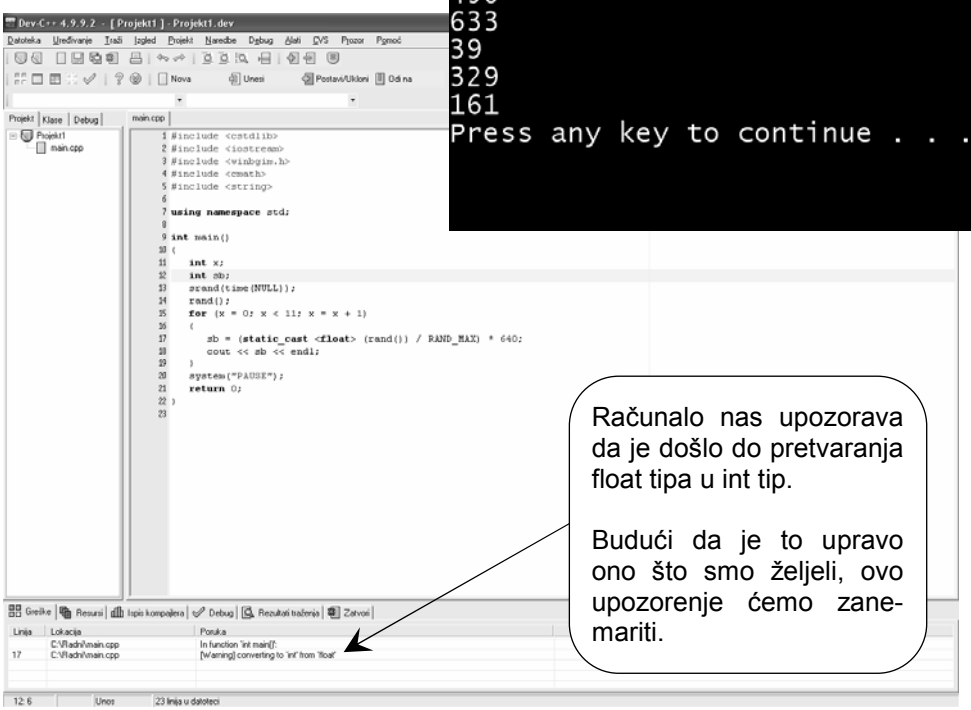
```
#include <cstdlib>
#include <iostream>
#include <ctime>

using namespace std;

int main()
{
   int x;
   int sb;
   srand(time(NULL));
   rand();
   for (x = 0; x < 11; x = x + 1)
   {
      sb = (static_cast <float> (rand()) / RAND_MAX) * 640;
      cout << sb << endl;
   }
   system("PAUSE");
   return 0;
}
```

Ako želimo da računalo generira broj između 0 i 640 bez decimalnog zareza, slučajni broj ćemo spremiti u int varijablu.

Budući da je generirani slučajni broj float tipa, računalo će nas pri prvom pokretanju programa upozoriti da je došlo do pretvaranja float tipa u int tip.

Kad float tip broja spremimo u int varijablu, računalo će ga automatski pretvoriti u int tip tako da će mu maknuti sve brojeve desno od decimalnog zareza.

```
0
454
620
553
98
576
450
633
39
329
161
Press any key to continue . . .
```



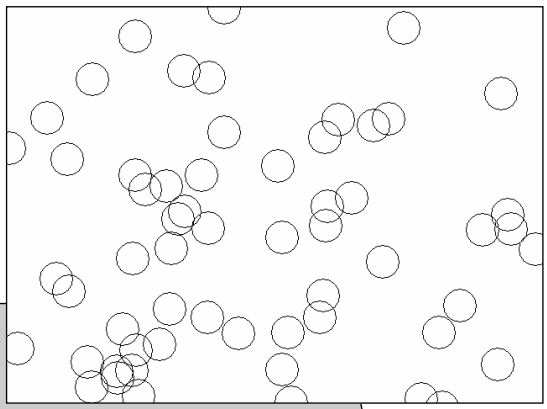
Računalo nas upozorava da je došlo do pretvaranja float tipa u int tip.

Budući da je to upravo ono što smo željeli, ovo upozorenje ćemo zanemariti.

Korištenje slučajnih brojeva

Ovaj program će generirati krugove na slučajno odabranim pozicijama.

Na taj način u igrama možemo generirati objekte na slučajno odabranim mjestima.



```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>
#include <ctime>

using namespace std;

int main()
{
   int x;
   int sx;
   int sy;
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   srand(time(NULL));
   rand();
   for (x = 1; x < 60; x = x + 1 )
   {
      sx = (static_cast <float> (rand()) / RAND_MAX) * 640;
      sy = (static_cast <float> (rand()) / RAND_MAX) * 480;
      circle(sx,sy,20);
   }
   getch();
   closegraph();
   return 0;
}
```

Budući da položaj kruga određuju dva broja, jedan koji određuje udaljenost središta od desnog ruba i drugi koji određuje udaljenost središta od gornjeg ruba generirat ćemo dva slučajna broja.

Prvi ćemo spremiti u int varijablu sx i bit će između 0 i 640. Drugi ćemo spremiti u sy i bit će između 0 i 480.

(Dimenzije grafičkog prozora su 640x480).

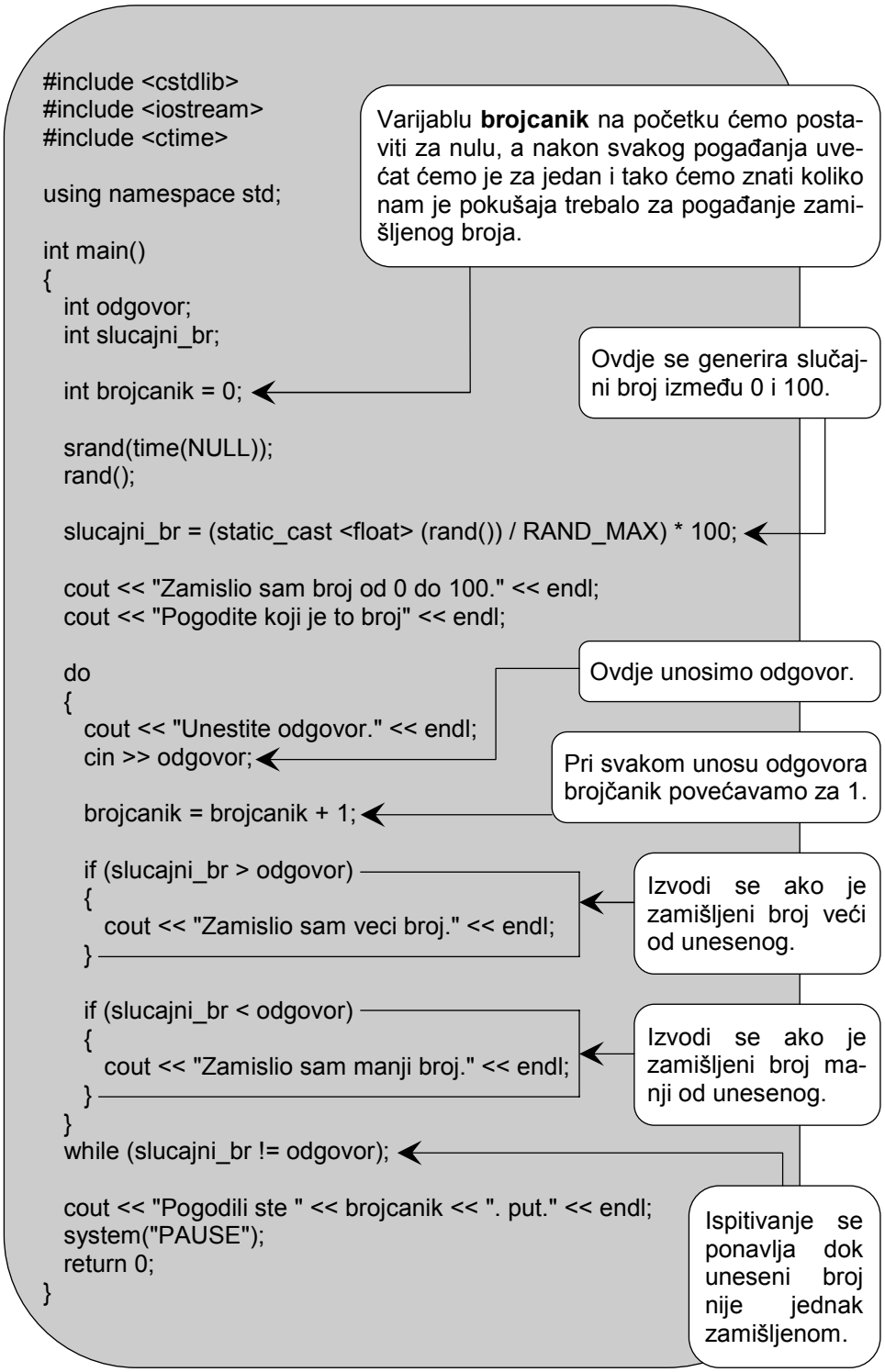
Položaj kruga ovisit će o slučajno generiranim brojevima.

Ispis kruga nalazi se unutar petlje pa će se generirati onoliko krugova koliko puta će se petlja izvesti.

Pogledajmo primjer sasvim jednostavne igrice. Računalo će generirati slučajni broj između 0 i 100, a mi ćemo pokušati pogoditi koji je to broj. Ako ne pogodimo, računalo će nas obavijestiti je li zamislilo veći ili manji broj, pa ćemo pogađati ponovo, dok ne pogodimo. Na kraju nas računalo obavještava koliko pokušaja nam je trebalo da pogodimo zamišljeni broj.

```
Zamislio sam broj od 0 do 100.
Pogodite koji je to broj
Unestite odgovor.
50
Zamislio sam manji broj.
Unestite odgovor.
25
Zamislio sam veci broj.
Unestite odgovor.
37
Pogodili ste 3. put.
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <ctime>

using namespace std;

int main()
{
  int odgovor;
  int slucajni_br;
  int brojcanik = 0;
  srand(time(NULL));
  rand();
  slucajni_br = (static_cast <float> (rand()) / RAND_MAX) * 100;
  cout << "Zamislio sam broj od 0 do 100." << endl;
  cout << "Pogodite koji je to broj" << endl;
  do
  {
    cout << "Unestite odgovor." << endl;
    cin >> odgovor;
    brojcanik = brojcanik + 1;
    if (slucajni_br > odgovor)
    {
      cout << "Zamislio sam veci broj." << endl;
    }
    if (slucajni_br < odgovor)
    {
      cout << "Zamislio sam manji broj." << endl;
    }
  }
  while (slucajni_br != odgovor);
  cout << "Pogodili ste " << brojcanik << ". put." << endl;
  system("PAUSE");
  return 0;
}
```


#include <cstdlib>
#include <iostream>
#include <ctime>
using namespace std;
int main()
{
int odgovor;
int slucajni_br;
int brojcanik = 0;
srand(time(NULL));
rand();
slucajni_br = (static_cast <float> (rand()) / RAND_MAX) * 100;
cout << "Zamislio sam broj od 0 do 100." << endl;
cout << "Pogodite koji je to broj" << endl;
do
{
cout << "Unestite odgovor." << endl;
cin >> odgovor;
brojcanik = brojcanik + 1;
if (slucajni_br > odgovor)
{
cout << "Zamislio sam veci broj." << endl;
}
if (slucajni_br < odgovor)
{
cout << "Zamislio sam manji broj." << endl;
}
}
while (slucajni_br != odgovor);
cout << "Pogodili ste " << brojcanik << ". put." << endl;
system("PAUSE");
return 0;
}
Varijablu **brojcanik** na početku ćemo postaviti za nulu, a nakon svakog pogađanja uvećat ćemo je za jedan i tako ćemo znati koliko nam je pokušaja trebalo za pogađanje zamišljenog broja.
Ovdje se generira slučajni broj između 0 i 100.
Ovdje unosimo odgovor.
Pri svakom unosu odgovora brojčanik povećavamo za 1.
Izvodi se ako je zamišljeni broj veći od unesenog.
Izvodi se ako je zamišljeni broj manji od unesenog.
Ispitivanje se ponavlja dok uneseni broj nije jednak zamišljenom.

Polja

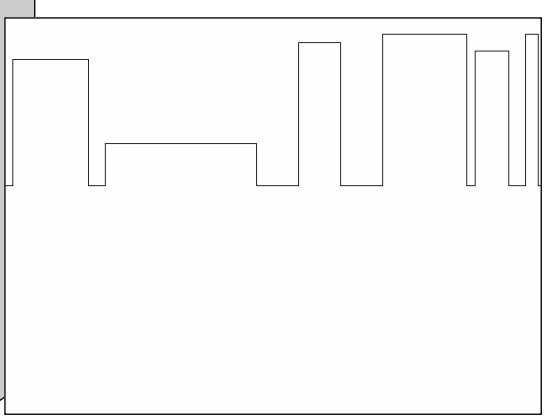
Jednodimenzionalna polja

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
   int gdriver = 9;
   int gmode = 2;
   initgraph(&gdriver, &gmode, "");
   setbkcolor (WHITE);
   setcolor (BLACK);
   cleardevice();
   moveto (0, 200);
   lineto (10, 200);
   lineto (10, 50);
   lineto (100, 50);
   lineto (100, 200);
   lineto (120, 200);
   lineto (120, 150);
   lineto (300, 150);
   lineto (300, 200);
   lineto (350, 200);
   lineto (350, 30);
   lineto (400, 30);
   lineto (400, 200);
   lineto (450, 200);
   lineto (450, 20);
   lineto (550, 20);
   lineto (550, 200);
   lineto (560, 200);
   lineto (560, 40);
   lineto (600, 40);
   lineto (600, 200);
   lineto (620, 200);
   lineto (620, 20);
   lineto (635, 20);
   lineto (635, 200);
   lineto (640, 200);
   getch();
   closegraph();
   return 0;
}
```

Zamislimo da kao pozadinu neke jednostavne igrice želimo nacrtati nešto što bi moglo asocirati na nebodere.

To možemo učiniti nizom **lineto** naredbi i to rješenje funkcionira, ali i sami vidimo da nije baš osobito elegantno.

Zamislimo koliko bi to rješenje bilo nepraktično ako bismo htjeli nacrtati crtež s više detalja, npr. prozorima.



U ovom programu pojavljuju se uglate zagrade.

Dobit ćemo ih tako da držimo pritisnutu **AltGr** tipku, a zatim kliknemo slovo **f** ili slovo **g**.

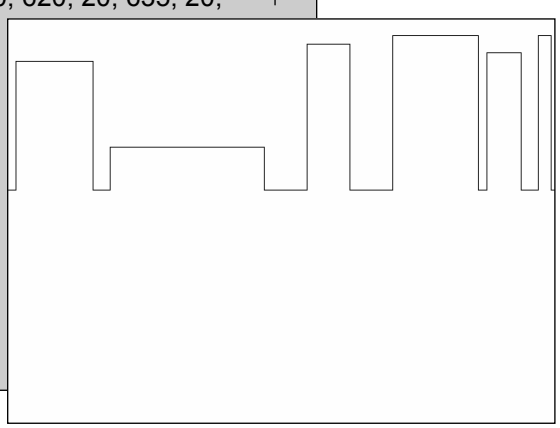
Ovaj program radi isti posao kao i prethodni, ali je daleko elegantniji od njega.

Pogledamo li pažljivo program, vidjet ćemo da su nam svi elementi poznati od prije, osim neobično definirane varijable polje.

Dok smo u dosadašnje varijable mogli spremiti jedno slovo ili broj, čini se da u ovu varijablu možemo spremiti mnoštvo brojeva.

Pažljivi analitičar možda bi uočio da broj u uglatoj zagradi odgovara količini brojeva u vitičastim zagradama.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
  int gdriver = 9;
  int gmode = 2;
  int x;
  initgraph(&gdriver, &gmode, "");
  setbkcolor (WHITE);
  setcolor (BLACK);
  cleardevice();

  int polje[54] =
  {
    0, 200, 10, 200, 10, 50, 100, 50, 100, 200,
    120, 200, 120, 150, 300, 150, 300, 200, 350, 200,
    350, 30, 400, 30, 400, 200, 450, 200, 450, 20,
    550, 20, 550, 200, 550, 200, 560, 200, 560, 40,
    600, 40, 600, 200, 620, 200, 620, 20, 635, 20,
    635, 200, 640, 200
  };
  moveto (polje[0], polje[1]);
  for (x = 2; x < 54; x = x + 2 )
  {
    lineto(polje[x], polje[x+1]);
  }
  getch();
  closegraph();
  return 0;
}
```

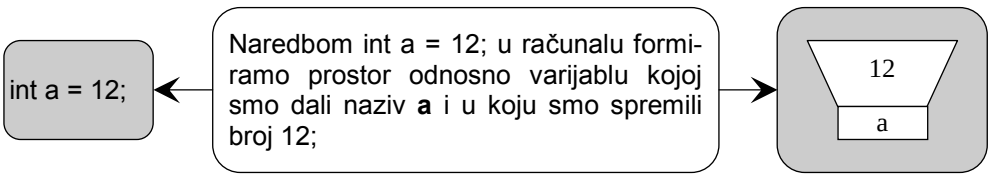


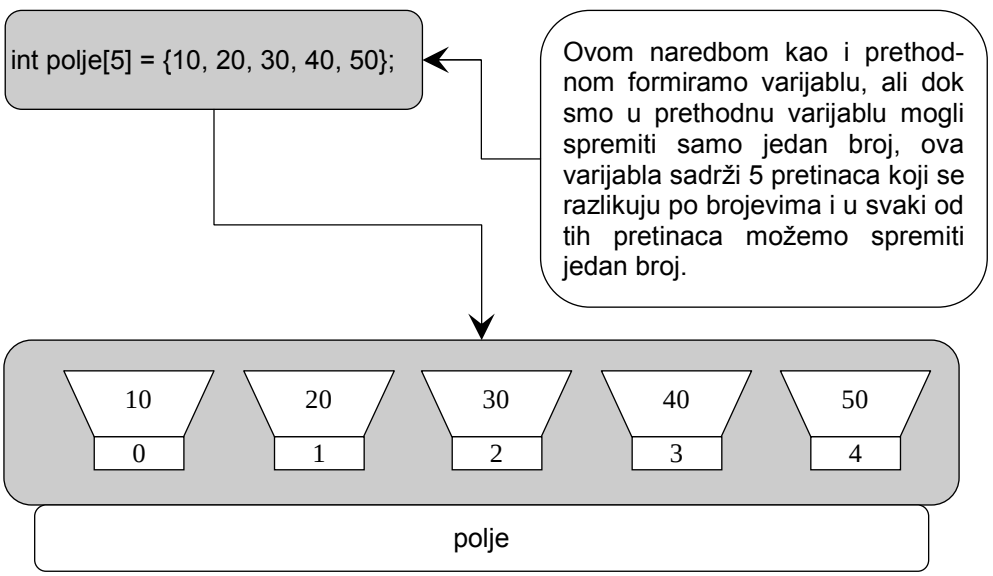
```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int polje[5] = {10, 20, 30, 40, 50};
   cout << polje[0] << endl;
   cout << polje[1] << endl;
   cout << polje[2] << endl;
   cout << polje[3] << endl;
   cout << polje[4] << endl;
   system("PAUSE");
   return 0;
}
```

```
10
20
30
40
50
Press any key to continue . . .
```

Detaljno objašnjenje funkcioniranja programa s prethodne stranice vidjet ćemo kasnije, a sada ćemo pogledati kako funkcioniraju neobične varijable koje smo uočili u prethodnom programu.

Ovaj program pomoći će nam da lakše shvatimo ovaj novi tip varijable.





Novi tip varijable ima poseban naziv, naziva se **polje**. Ćelije polja međusobno razlikujemo po brojevima, a problem donekle komplicira činjenica da prvo polje nije polje 1 nego polje 0, pa onda polje sa 5 elemenata ne sadrži polja od 1 do 5 nego od 0 do 4. U početku će nas ta činjenica često zbunjivati.

```
int polje[5] = {10, 20, 30, 40, 50};
```

U vitičastim zagradama navest ćemo vrijednosti pojedinih ćelija.

Vrijednosti ćemo međusobno odvojiti zarezima.

Za razliku od obične varijable prilikom formiranja polja u uglatim zagradama moramo navesti koliko ćelija sadrži polje. U našem slučaju polje koje smo nazvali **polje** sadrži 5 ćelija.

```
cout << polje[1] << endl;
```

Sadržaj polja ispisuje se na sličan način kao i sadržaj varijable.

Razlika je u tome da je pri korištenju varijable dovoljno navesti naziv, a ovdje osim naziva u uglatim zagradama moramo navesti broj ćelije koju želimo ispisati. U gornjem primjeru imajmo na umu da ćemo ispisati sadržaj druge ćelije, a ne sadržaj prve. Prvu bismo ispisali naredbom **cout << polje[0] << endl;**.

```
20
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int polje[5] = {10, 20, 30, 40, 50};
   cout << polje[1] << endl;
   system("PAUSE");
   return 0;
}
```

Vidimo da

cout << polje[1] << endl;

ispisuje sadržaj druge ćelije. Pokušajmo na sličan način ispisati sadržaj neke druge npr. sa:

cout << polje[0] << endl;

sadržaj prve ćelije. (U uglatoj zagradi nalazi se broj nula, a ne veliko slovo o.)

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{

   int polje[5] = {10, 20, 30, 40, 50};

   cout << polje[0] << endl;
   cout << polje[1] << endl;
   cout << polje[2] << endl;
   cout << polje[3] << endl;
   cout << polje[4] << endl;

   system("PAUSE");
   return 0;
}
```

Pogledajmo bitne dijelove ovog programa.

Ovdje formiramo polje koje se sastoji od 5 ćelija. U prvu ćeliju spremamo broj 10, u drugu 20, u treću 30, u četvrtu 40 i u petu 50.

Sadržaj polja ispisujemo tako da uz naziv polja u uglatim zagradama navodimo i broj ćelije koji želimo ispisati.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int polje[5] = {10, 20, 30, 40, 50};
   cout << polje[3] << endl;
   cout << polje[0] << endl;
   cout << polje[2] << endl;
   cout << polje[4] << endl;
   cout << polje[1] << endl;
   system("PAUSE");
   return 0;
}
```

Sadržaj polja ne mora se ispisivati po redu, nego se može ispisivati bilo kojim redoslijedom.

```
40
10
30
50
20
Press any key to continue . . .
```

Polja mogu biti float tipa.

(Uočimo da zarezom odvajamo brojeve, a točka ima funkciju decimalnog zareza.)

```
10.12
20.14
30.16
40.18
50.19
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   float polje[5] = {10.12, 20.14, 30.16, 40.18, 50.19};
   cout << polje[0] << endl;
   cout << polje[1] << endl;
   cout << polje[2] << endl;
   cout << polje[3] << endl;
   cout << polje[4] << endl;
   system("PAUSE");
   return 0;
}
```

```
P
O
L
J
A
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   char polje[5] = {'P', 'O', 'L', 'J', 'A'};
   cout << polje[0] << endl;
   cout << polje[1] << endl;
   cout << polje[2] << endl;
   cout << polje[3] << endl;
   cout << polje[4] << endl;
   system("PAUSE");
   return 0;
}
```

Polja mogu biti i **char** tipa.

Pokušajmo iz ispisa izbaciti endl naredbe:

`cout << polje[0] << endl;`

napišimo u obliku

cout << polje[0];

Samo `cout << polje[4] << endl;` nemojmo mijenjati.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int a[3] = {2, 3, 0};
   a[2] = a[0] + a[1];
   cout << a[2] << endl;
   system("PAUSE");
   return 0;
}
```

```
5
Press any key to continue . . .
```

Polja možemo koristiti na isti način kao i varijable, npr. za izračune.

Kao što je slučaj i kod varijabli, sadržaj polja ne moramo definirati prilikom formiranja polja.

U ovom slučaju sadržaj prve i druge ćelije polja unosimo cin naredbom, a u treću ćeliju spremamo rezultat zbrajanja.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int b[3];
   cin >> b[0];
   cin >> b[1];
   b[2] = b[0] + b[1];
   cout << b[2] << endl;
   system("PAUSE");
   return 0;
}
```

```
2
3
5
Press any key to continue . . .
```

Sadržaj polja možemo ispisati na ovaj način. Takav način funkcionira, ali je neelegantan. Osobito je neprikladan ako bismo htjeli ispisati veće polje.

```
10
20
30
40
50
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int polje[5] = {10, 20, 30, 40, 50};
   cout << polje[0] << endl;
   cout << polje[1] << endl;
   cout << polje[2] << endl;
   cout << polje[3] << endl;
   cout << polje[4] << endl;
   system("PAUSE");
   return 0;
}
```

Ovaj način ispisa sadržaja polja daleko je elegantniji.

Formirali smo petlju koja će se vrtjeti onoliko puta koliko ćelija petlje želimo ispisati.

Varijabla petlje mijenja se od početnog broja ćelije koju želimo ispisati, u našem slučaju od 0, do broja krajnje ćelije koju želimo ispisati, u našem slučaju to je 4.

Pri ispisu umjesto polje[1] koristimo polje[x] što znači da se ispisuje ono polje čiji broj je jednak trenutnoj vrijednosti varijable x.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int polje[5] = {10, 20, 30, 40, 50};
   int x;
   for (x = 0; x < 5; x = x + 1)
   {
      cout << polje[x] << endl;
   }
   system("PAUSE");
   return 0;
}
```

Možemo ispisati dio ćelija polja, npr. prvu, drugu i treću.

To ćemo postići tako da varijablu petlje `x` mijenjamo od jedan do tri po jedan.

```
20
30
40
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int polje[5] = {10, 20, 30, 40, 50};
   int x;
   for (x = 1; x < 4; x = x + 1)
   {
      cout << polje[x] << endl;
   }
   system("PAUSE");
   return 0;
}
```

```
50
40
30
20
10
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int polje[5] = {10, 20, 30, 40, 50};
   int x;
   for (x = 4; x > -1; x = x - 1)
   {
      cout << polje[x] << endl;
   }
   system("PAUSE");
   return 0;
}
```

Sadržaj polja možemo ispisivati i unatrag. To ćemo postići tako da varijablu petlje `x` mijenjamo unatrag od četiri do nula po jedan.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int polje[5];
   int x;
   cout << "Unos polja." << endl;

   for (x = 0; x < 5; x = x + 1)
   {
      cin >> polje[x];
   }

   cout << "Ispis polja." << endl;

   for (x = 0; x < 5; x = x + 1)
   {
      cout << polje[x] << endl;
   }

   system("PAUSE");
   return 0;
}
```

Na sličan način pomoću petlje možemo uporabom tipkovnice unijeti brojeve u pojedine ćelije.

Ovu petlju koristimo za unos brojeva u polje.

Opet umjesto cin >> polje [0] imamo cin >> polje[x], a varijablu x petlja mijenja od nula do četiri i tako će se unijeti sadržaj cijelog polja.

Kao i do sada nakon svakog unosa broja pritisnemo Enter tipku.

Koristeći isti trik ovdje ispisujemo sadržaj polja.

Umjesto cout << polje[0] stavili smo cout << polje[x], a x for petlja mijenja od nula do četiri i tako će se ispisati sadržaj cijelog polja.

```
Unos polja.
110
120
130
140
150
Ispis polja.
110
120
130
140
150
Press any key t
```

Kao što je to vrijedilo za sve elemente programa koje smo vidjeli do sada, i za polja vrijedi da ćemo ih najbolje shvatiti i usvojiti ako napravimo što veći broj varijacija primjera koje smo vidjeli do sada.

Pokušajmo sami smisliti što više sličnih primjera.

Pogledajmo program koji smo vidjeli na početku ovog poglavlja.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

int main()
{
  int gdriver = 9;
  int gmode = 2;
  int x;
  initgraph(&gdriver, &gmode, "");
  setbkcolor (WHITE);
  setcolor (BLACK);
  cleardevice();

  int polje[54] =
  {
    0, 200, 10, 200, 10, 50, 100, 50, 100, 200,
    120, 200, 120, 150, 300, 150, 300, 200, 350, 200,
    350, 30, 400, 30, 400, 200, 450, 200, 450, 20,
    550, 20, 550, 200, 550, 200, 560, 200, 560, 40,
    600, 40, 600, 200, 620, 200, 620, 20, 635, 20,
    635, 200, 640, 200
  };

  moveto (polje[0], polje[1]);

  for (x = 2; x < 54; x = x + 2 )

  {
    lineto(polje[x], polje[x+1]);
  }

  getch();
  closegraph();
  return 0;
}
```

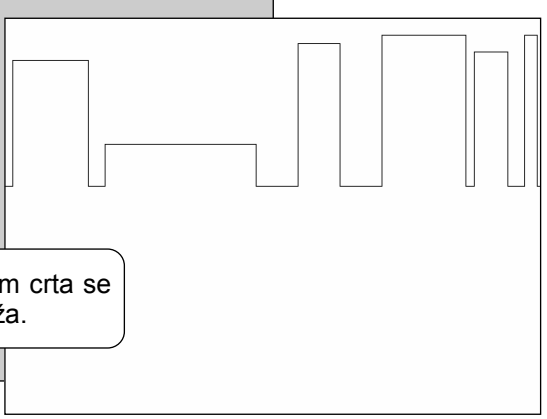
Ovdje se formira polje u kojem se nalazi položaj točaka čijim ćemo spajanjem dobiti crtež.

Na prvi pogled može nam se učiniti da smo to ovdje učinili drugačije nego do sada. Pažljivom analizom vidjet ćemo da nismo.

Najprije imamo **int polje[54] =**, a zatim u vitičastim zagradama vrijednosti pojedinih ćelija.

Ovom naredbom određuje se početak niza crta.

Ovom petljom crta se ostatak crteža.



Pogledajmo podrobnije neke detalje programa.

```
int polje[5] = {10, 20, 30, 40, 50};
```

Ovu naredbu možemo napisati i ovako.

```
int polje[5] =
{
  10, 20, 30, 40, 50
};
```

Ovaj oblik pogodniji je za korištenje kad u veliko polje želimo unijeti veliku količinu brojeva.

```
moveto (polje[0], polje[1]);
```

Ova naredba određuje početak niza crta. Udaljenost od lijevog ruba određena je sadržajem ćelije polje[0], tj. prve ćelije, a udaljenost od gornjeg ruba sa polje[1], tj. sadržajem druge ćelije.

Početna vrijednost varijable x je 2.

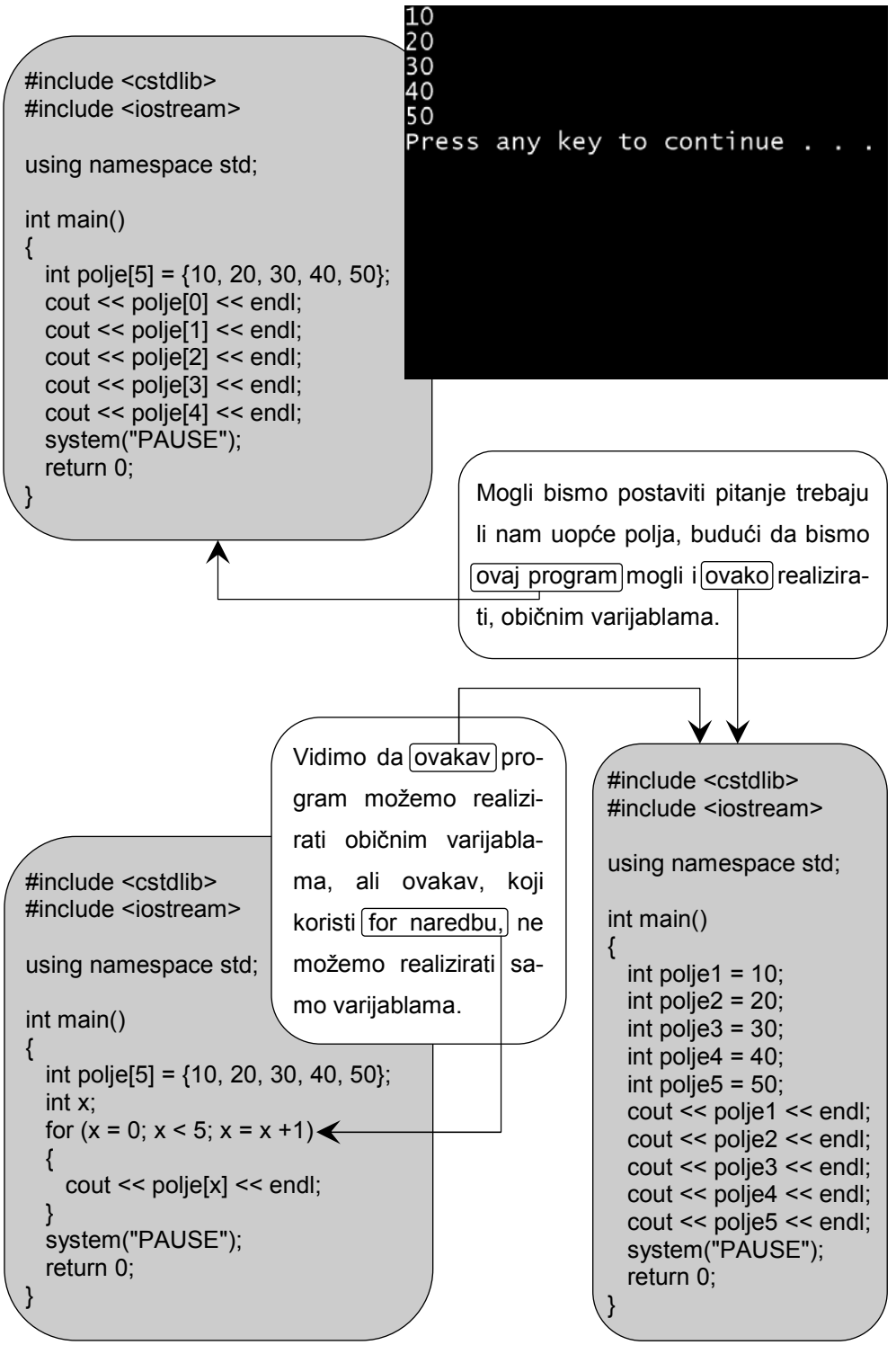
Petlja će se izvoditi dok je x manji od 54. Budući da se x uvećava za 2 vrijednost varijable x će rasti od 2 do 52.

```
for (x = 2;  x < 54;  x = x + 2 )
{
  lineto(polje[x],  polje[x+1]);
}
```

Varijabla x uvećava se za 2. To nam odgovara zato što je kraj svake crte određen s dva broja, pa kad želimo očitati poziciju novog kraja crte, moramo se u polju pomaknuti za dva člana polja unaprijed.

Udaljenost kraja crte od lijevog ruba grafičkog prozora.

Udaljenost kraja crte od gornjeg ruba grafičkog prozora.


10
20
30
40
50
Press any key to continue . . .
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
int polje[5] = {10, 20, 30, 40, 50};
cout << polje[0] << endl;
cout << polje[1] << endl;
cout << polje[2] << endl;
cout << polje[3] << endl;
cout << polje[4] << endl;
system("PAUSE");
return 0;
}
Mogli bismo postaviti pitanje trebaju li nam uopće polja, budući da bismo ovaj program mogli i ovako realizirati, običnim varijablama.
Vidimo da ovakav program možemo realizirati običnim varijablama, ali ovakav, koji koristi for naredbu, ne možemo realizirati samo varijablama.
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
int polje[5] = {10, 20, 30, 40, 50};
int x;
for (x = 0; x < 5; x = x +1)
{
cout << polje[x] << endl;
}
system("PAUSE");
return 0;
}
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
int polje1 = 10;
int polje2 = 20;
int polje3 = 30;
int polje4 = 40;
int polje5 = 50;
cout << polje1 << endl;
cout << polje2 << endl;
cout << polje3 << endl;
cout << polje4 << endl;
cout << polje5 << endl;
system("PAUSE");
return 0;
}

Ne samo da uporabom polja možemo pisati elegantnije programe, nego nam polja omogućuju realizaciju programa koje bi bilo veoma teško realizirati na neki drugi način. Pogledajmo nekoliko primjera takvih programa.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  int polje[5];
  int x;
  int r;

  cout << "Unos polja." << endl;
  for (x = 0; x < 5; x = x + 1)
  {
     cin >> polje[x];
  }

  r = polje[0];
  for (x = 1; x < 5; x = x +1)
  {
    if (r < polje[x])
    {
      r = polje[x];
    }
  }

  cout << "Najveci je." << endl;
  cout << r << endl;

  system("PAUSE");
  return 0;
}
```

Polja nam omogućuju operacije sa ćelijama polja.

Ovo je primjer programa koji pretražuje uneseno polje i ispisuje najveći broj u polju.

Unos polja.

U ovom dijelu programa pretražuje se uneseno polje i najveći broj nađen u polju sprema se u varijablu r.

U ovom dijelu ispisuje se sadržaj varijable r, odnosno najveći broj unesen u polje.

```
Unos polja.
-20
12
-5
4
7
Najveci je.
12
Press any key to continue . . .
```

Pogledajmo detaljnije najzanimljiviji dio prethodnog program, dio u kojem se pretražuje polje i najveći pronađeni broj sprema u r varijablu.

Prvi član polja, dakle polje[0] sprema se u varijablu r.

Pretraživanje se nastavlja unutar for petlje, čija se varijabla x mijenja od 1 do 5 po 1.

Zašto petlja ne počinje s nula?

Zato što smo prvu ćeliju (polje[0]) već pročitali pa je nećemo čitati unutar for petlje.

```
r = polje[0];

for (x = 1; x < 5; x = x +1)

{

   if (r < polje[x])
   {
      r = polje[x];
   }

}
```

Unutar for petlje za svaku vrijednost varijable x ispituje se je li sadržaj varijable r manji od polje[x].

Ako jest, znači da je u ćeliji polje[x] veći broj od onog u varijabli r, pa će se sadržaj ćelije polje[r] spremiti u varijablu r.

Ako nije, znači da je broj u polje[x] manji i u tom slučaju neće se izvoditi sadržaj unutar vitičastih zagrada.

Na kraju, u varijabli r ostaje najveća vrijednost nađena u polju.

Ova if naredbe je neobična. Nema else dijela. To je dozvoljeno. Ako nam else dio ne treba, možemo ga izostaviti.

U tom slučaju, ako je uvjet zadovoljen, izvodi se sadržaj u vitičastim zagradama, a ako nije, nastavlja se izvođenje programa.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  int polje[5];
  int x;
  int r;
  cout << "Unos polja." << endl;
  for (x = 0; x < 5; x = x + 1)
  {
     cin >> polje[x];
  }
  r = polje[0];
  for (x = 1; x < 5; x = x +1)
  {
    if (r > polje[x])
    {
      r = polje[x];
    }
  }
  cout << "Najmanji je." << endl;
  cout << r << endl;
  system("PAUSE");
  return 0;
}
```

Jednostavnim promjenama program koji pretražuje polje sa ciljem da nađe najveći broj u polju možemo preurediti u program koji traži najmanji broj.

Čemu služe takvi programski postupci?

Na sličan način bismo u programu za vođenje videoteke mogli pronaći korisnike koji nisu vratili film dulje od npr. tjedan dana ili korisnike koji su u prošloj godini posudili najviše filmova.

Dovoljno je da umjesto ispitivanja je li r < polje[x] stavimo ispitivanje je li r > polje[x].

Odgovarajuće preuredimo ispis obavijesti.

Pokušajmo sami korigirati program tako da pretražuje je li u polje unesen određen broj ili koliko puta se pojavljuje određen broj.

Pokušajmo napraviti slična pretraživanja slova.

```
Unos polja.
12
-18
6
-12
22
Najmanji je.
-18
Press any key to continue . . .
```

```
Unos polja.
-12
15
-3
22
8
Sortirano polje.
22
15
8
-3
-12
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  int p[5];
  int zamjena;
  int privremeni;
  int x;

  cout << "Unos polja." << endl;
  for (x = 0; x < 5; x = x + 1)
  {
     cin >> p[x];
  }

  do
  {
    zamjena = 0;
    for (x = 0; x < 4; x = x + 1)
    {
      if (p[x] < p[x+1])
      {
        privremeni = p[x];
        p[x] = p[x+1];
        p[x+1] = privremeni;
        zamjena = 1;
      }
    }
  }
  while (zamjena ==1);

  cout << "Sortirano polje." << endl;
  for (x = 0; x < 5; x = x +1)
  {
    cout << p[x] << endl;
  }

  system("PAUSE");
  return 0;
}
```

Ovo je sasvim sigurno najsloženiji program koji smo do sada vidjeli u ovoj knjizi.

Služi za sortiranje polja. Unesemo različite brojeve u polje, a program će ih sortirati tako da će u prvom polju biti najveći broj, a u posljednjem najmanji.

U ovom dijelu programa unosimo brojeve u polje.

U ovom dijelu vrši se sortiranje polja.

U ovom dijelu sortirano polje ispisuje se na zaslon računala.

Pogledamo podrobnije dio programa u kojem se vrši sortiranje.

Sortiranje se odvija u do ... while petlji. Petlja se ponavlja tako dugo dok je varijabla zamjena jednaka broju 1.

Na početku ispitivanja u varijablu zamjena stavlja se broj 0. Ako ne bude potrebno vršiti nikakve zamjene u polju, varijabla zamjena ostat će jednaka nuli i izvođenje do ... while petlje će se zaustaviti.

```
do
{
  zamjena = 0;
  for (x = 0; x < 4; x = x + 1)
  {
    if (p[x] < p[x+1])
    {
      privremeni = p[x];
      p[x] = p[x+1];
      p[x+1] = privremeni;
      zamjena = 1;
    }
  }
}
while (zamjena ==1);
```

U for petlji čita se cijelo polje, s time da se varijabla petlje mijenja za jedan manje od veličine petlje; dakle u našem slučaju ne ide do 4 nego, do 3.

If naredbom ispitujemo je li broj u nekoj ćeliji polja (p[x]) manji od broja u sljedećoj ćeliji (p [x+1]. Ako nije, znači da imamo prvo veći broj, a zatim manji. To je ono što želimo i petlja se vrti dalje.

Ako jest, znači da imamo prvo manji, a zatim veći broj, pa ih treba zamijeniti.

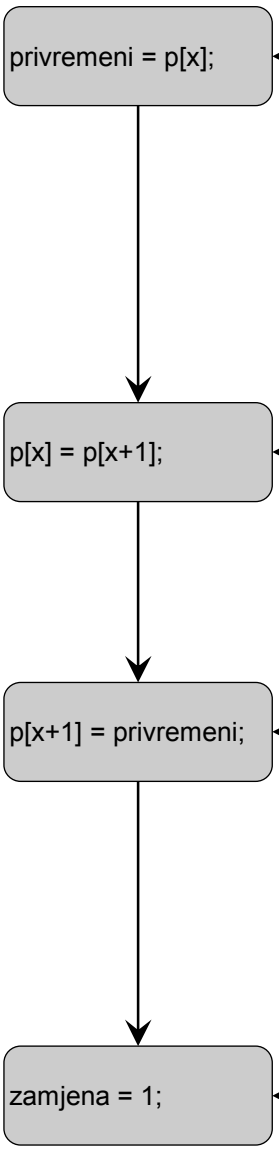
U tom slučaju u ovom dijelu će se izvršiti zamjena.

Na kraju zamjene u varijablu zamjena stavit će se broj 1.

Zbog toga do ... while petlja će cijeli postupak ponavljati tako dugo dok više ne bude obavljena ni jedna zamjena i varijabla zamjena ostane jednaka nuli.

To znači da nije pronađen ni jedan slučaj da je neki broj manji od sljedećeg tj. da je sortiranje od većeg prema manjem završeno.

Pogledajmo kako funkcionira zamjena sadržaja ćelija.



Za zamjenu sadržaja dviju susjednih ćelija u polju treba nam pomoćna varijabla; u našem slučaju to je varijabla **privremeni**.

Zašto?

Da smo odmah napravili drugu naredbu, naredbu p[x] = p[x+1]; onda bi se broj koji se nalazi u p[x+1]; nalazio i u p[x], a broj koji se je do tada nalazio u p[x] bi nestao.

Da broj koji se nalazi u p[x] ne bi nestao, privremeno ga spremamo u varijabli privremeni.

Nakon što smo u prethodnom koraku sadržaj ćelije p[x] spremili u varijablu privremeni, sadržaj ćelije p[x+1] spremamo u ćeliju p[x].

Budući da p[x] uspoređujemo sa p[x+1], x mora bit za jedan manji od broja ćelija.

Sadržaj varijable privremeni sada vraćam u polje, ali ne u ćeliju p[x] u kojoj se taj sadržaj do sada nalazio, nego u sljedeću ćeliju, u ćeliju p[x+1].

Time smo sadržaje ćelija p[x] i p[x+1] zamijenili.

Na kraju zamjene u varijablu zamjena stavljamo broj jedan.

Time smo obavijestili računalo da je bilo potrebno vršiti zamjene u polju, tj. da polje nije bilo sortirano po redu, te da do ... while petlja mora još jednom provjeriti je li polje sortirano. Kad se ni u jednom slučaju neće dogoditi da je neka ćelija manja od sljedeće, sortiranje je završeno.

Između polja mogu se vršiti matematičke operacije kao i između varijabli.

Ovaj program će zbrojiti sadržaje polja po1 i polja po2, a rezultat spremiti u polje po3 i zatim ga ispisati na zaslon računala.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
  int po1[5] = {10, 15, 20, 25, 30};
  int po2[5] = {35, 40, 35, 50, 55};
  int po3[5];
  int a;
  for (a = 0; a < 5; a = a +1)
  {
    po3[a] = po1[a] + po2[a];
    cout << po1[a] << " + " << po2[a] << " = " << po3[a] << endl;
  }
  system("PAUSE");
  return 0;
}
```

```
100
200
300
400
500
600
Inverzno polje.
600
500
400
300
200
100
Press any key to continue . . .
```

Pokušajmo napisati slične programe koji će vršiti neke operacije između polje; npr. programe koji će množiti ili dijeliti članove dvaju polja i rezultat spremati u treće polje.

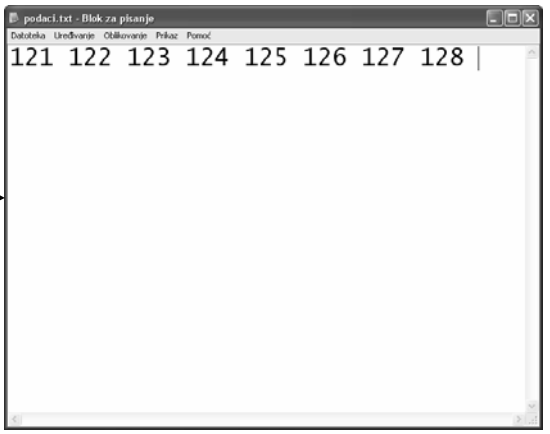
Program koji će uspoređivati pojedine ćelije dvaju polja, a zatim će u treće polje spremiti vrijednost one ćelije koja sadrži veći broj ili manji broj.

Program koji će brojiti koliko puta se neki broj pojavljuje u nekom polju.

Spremanje brojeva u datoteku

Postavlja se pitanje na koji način bismo mogli podatke umjesto preko tipkovnice unijeti u polje iz neke datoteke. U tu svrhu otvorit ćemo Nopetad (u hrvatskoj inačici Blok za pisanje) i unijet ćemo osam različitih brojeva koji će biti odvojeni praznim mjestima.

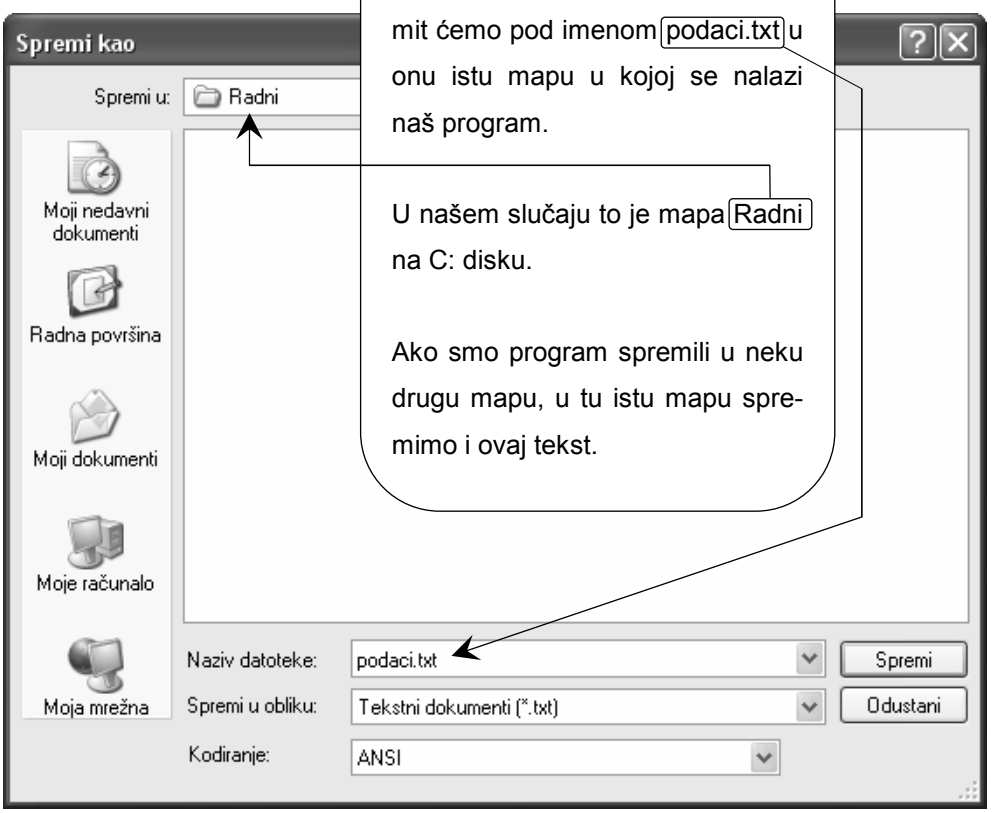
Važno je napomenuti da to **moramo** učiniti tim programom, a nipošto Wordom ili nekim sličnim programom.

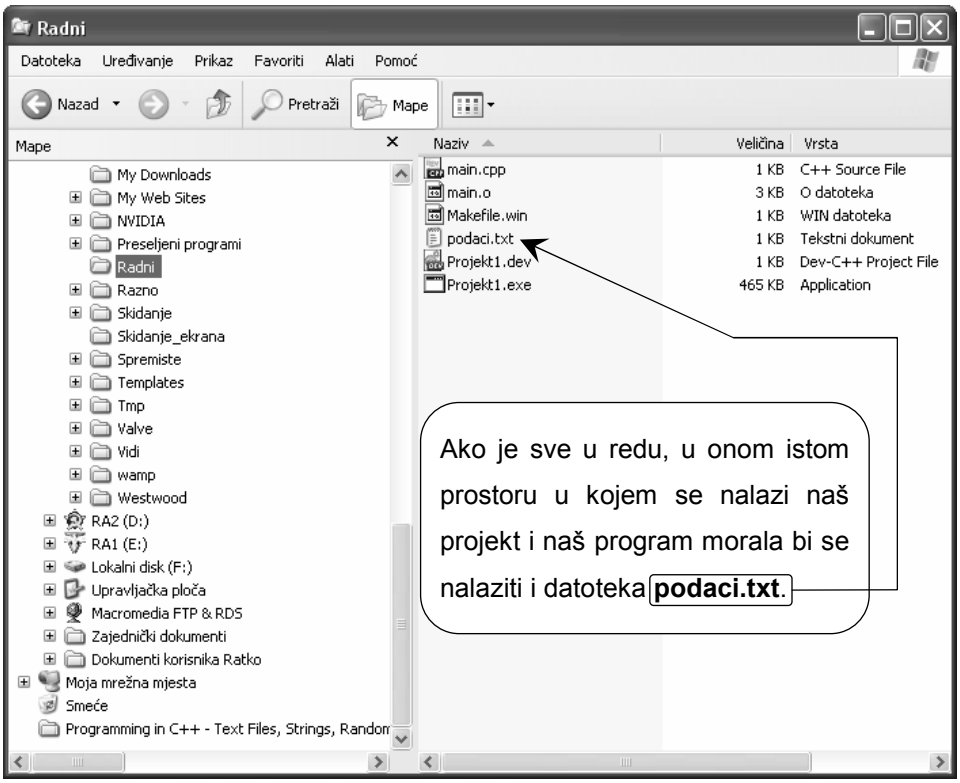


Te brojeve, odnosno taj tekst spremit ćemo pod imenom podaci.txt u onu istu mapu u kojoj se nalazi naš program.

U našem slučaju to je mapa Radni na C: disku.

Ako smo program spremili u neku drugu mapu, u tu istu mapu spremimo i ovaj tekst.





Jasno je da možemo staviti i neke druge brojeve, a možemo ih staviti više ili manje nego u našem primjeru.

Ako mijenjamo količinu brojeva, onda u programu koji će ih čitati moramo na isti način promijeniti veličinu polja i for petlju.

Moramo imati na umu i da broj koji unosimo ne može biti neograničeno velik. Koliki broj možemo unijeti u polje, odnosno u varijablu, ovisi o svojstvima programskog okruženja koje koristimo.

Int tip u slučaju da koristimo **Dev-C++** može biti od –2147483648 do 2147483647.

Float tip može imati najveću vrijednost od $3.40282 * 10^{-38}$. To se u programu piše kao 3.40285e-38.

Mogu li se u programima koristiti i veći brojevi? Naravno, ali taj problem nadilazi potrebe našeg uvodnog priručnika.

Za sada je bitno imati na umu da u int ili float varijablu ne možemo staviti neograničeno velik broj.

Ovim programom pročitat ćemo brojeve koji su spremljeni u datoteku **podaci.txt**, spremit ćemo ih u polje i na kraju ispisati na zaslon računala.

```
#include <cstdlib>
#include <iostream>
#include <fstream>

using namespace std;

int main()
{

   int x;
   int polje[8];
   ifstream ulaz("podaci.txt");

   for (x = 0; x < 8; x = x + 1)
   {
      ulaz >> polje[x];
   }

   for (x = 0; x < 8; x = x +1)
   {
     cout << polje[x] << endl;
   }

   system("PAUSE");
   return 0;
}
```

Dodat ćemo novu biblioteku.

Ovdje formiramo nešto s čime se do sada nismo sretali, a to je **objekt**. Trenutno nije važno što je to točno objekt. Za sada ćemo reći da je to element programa pomoću kojeg ćemo čitati sadržaj datoteke.

Ova petlja mora se vrtjeti onoliko puta koliko brojeva želimo pročitati, odnosno koliko naše polje ima ćelija. U našem slučaju to je osam puta.

Ovom naredbom sadržaj datoteke unosi se u polje.

Da smo ovdje napisali:

cin >> polje[x];

brojevi bi se u polje unosili pomoću tipkovnice.

ulaz >> polje[x];

je varijacija te naredbe i znači da se brojevi neće u polje unositi pomoću tipkovnice nego pomoću objekta ulaz, a prije smo definirali da objekt ulaz čita iz datoteke podaci.txt.

Ispis sadržaja polja.

```
121
122
123
124
125
126
127
128
Press any key to continue . . .
```

Naredba za formiranje objekta pomoću kojeg ćemo čitati sadržaj datoteke.

Naziv objekta. Bira se po istoj logici po kojoj biramo nazive varijabli.

```
ifstream ulaz ("podaci.txt");
```

Naziv datoteke koju ćemo čitati pomoću objekta **ulaz**.

```
Prosjek je 124.5

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
   int x;
   int polje[8];
   float a = 0;
   ifstream ulaz("podaci.txt");
   for (x = 0; x < 8; x = x + 1)
   {
      ulaz >> polje[x];
   }
   for (x = 0; x < 8; x = x +1)
   {
     a = a + polje[x];
   }
   a = a / 8;
   cout << "Prosjek je " << a << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Kada učitamo brojeve u polje, s tim brojevima možemo raditi sve ono što možemo raditi i s brojevima koje smo unijeli preko tipkovnice.

U ovom programu izračunat ćemo prosječnu vrijednost unesenih brojeva.

Budemo li eksperimentirali s drugim brojevima, imajmo na umu da njihova ukupna suma ne smije biti veća od maksimalne dozvoljene vrijednosti varijable.

Ovaj program omogućuje nam da sadržaj polja koji smo unijeli preko tipkovnice spremimo u datoteku.

```
#include <cstdlib>
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
   int x;
   int polje[8];
   ofstream izlaz("podaci.txt");

   for (x = 0; x < 8; x = x + 1)
   {
      cin >> polje[x];
   }

   for (x = 0; x < 8; x = x + 1)
   {
      izlaz << polje[x];
      izlaz << " ";
   }

   cout << "Spremanje je gotovo." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

I u ovom slučaju moramo imati uključenu **fstream** biblioteku.

Ovdje formiramo objekt izlaz pomoću kojeg ćemo brojeve iz polja spremiti u datoteku podaci.txt.

Na uobičajeni način unosimo brojeve u polje.

Ovdje sadržaj polja spremamo u datoteku podaci.txt.

Naredbom:

izlaz << polje[x];

brojeve iz polja pomoću objekta izlaz šaljemo u datoteku.

Ova naredba slična je naredbi:

cout << polje[x];

kojom brojeve šaljemo na zaslon računala.

Naredbom **izlaz << " ";** između dva broja unosimo prazno mjesto.

```
111
222
333
444
555
666
777
888
Spremanje je gotovo.

Press any key to continue . . .
```

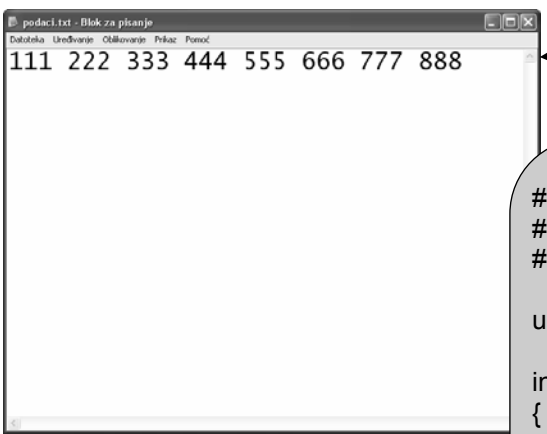
Naredba za formiranje objekta pomoću kojeg ćemo spremati sadržaj datoteke.

Naziv objekta. Bira se po istoj logici po kojoj biramo nazive varijabli.

```
ofstream izlaz ("podaci.txt");
```

Naziv datoteke u koju ćemo spremati pomoću objekta **ulaz**.

Što smo spremili možemo pogledati editorom, a možemo i programom koji sadržaj datoteke ispisuje na zaslon računala.




```
#include <cstdlib>
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
   int x;
   int polje[8];
   ifstream ulaz("podaci.txt");
   for (x = 0; x < 8; x = x + 1)
   {
      ulaz >> polje[x];
   }
   for (x = 0; x < 8; x = x +1)
   {
     cout << polje[x] << endl;
   }
   system("PAUSE");
   return 0;
}
```

```
111
222
333
444
555
666
777
888
Press any key to continue . .
```

```
#include <cstdlib>
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
   int x;
   int p[8];
   int zamjena;
   int privremeni;
   ifstream ulaz("podaci.txt");
   for (x = 0; x < 8; x = x + 1)
   {
      ulaz >> p[x];
   }
   cout << "Zapocinje sortiranje." << endl;
   do
   {
      zamjena = 0;
      for (x = 0; x < 7; x = x + 1)
      {
         if (p[x] < p[x+1])
         {
            privremeni = p[x];
            p[x] = p[x+1];
            p[x+1] = privremeni;
            zamjena = 1;
         }
      }
   }
   while (zamjena ==1);
   ofstream izlaz("podaci.txt");
   for (x = 0; x < 8; x = x + 1)
   {
      izlaz << p[x];
      izlaz << " ";
   }
   cout << "Sortiranje je zavrsilo" << endl;
   system("PAUSE");
   return 0;
}
```

U ovom programu nema ničeg novog. Samo smo kao lego kocke složili elemente koje smo do sada upoznali.

Ovaj program učitava datoteku u polje, sortira polje i zatim sortirano polje sprema natrag u datoteku.

U ovom djelu programa čitamo sadržaj datoteke i spremamo ga u polje.

Doslovno sa Copy Paste kopiran dio programa koji sortira sadržaj polja od najvećeg prema najmanjem.

U ovom djelu programa sadržaj polja sprema se u datoteku.

Ovdje vidimo i jedan jako važan element programiranja. Ne moramo svaku stvar svaki put kad nam treba iznova programirati. Segment programa koji dobro radi obilno prokomentiramo i spremimo, a zatim ga ubacujemo u programe kad nam zatreba, kao što smo ovdje ubacili segment programa koji sortira sadržaj polja.

```
#include <cstdlib>
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
  int x;
  int polje[8];
  ofstream izlaz("podaci.txt");
  for (x = 0; x < 8; x = x + 1)
  {
     cin >> polje[x];
  }
  for (x = 0; x < 8; x = x + 1)
  {
     izlaz << polje[x];
     izlaz << " ";
  }
  cout << "Spremanje je gotovo." << endl;
  cout << endl;
  system("PAUSE");
  return 0;
}
```

```
300
100
400
800
200
600
700
500
Spremanje je gotovo.

Press any key to continue . . .
```

Da bismo provjerili funkcionira li naš program, najprije ćemo programom koji smo malo prije obradili unijeti brojeve u datoteku.

```
#include <cstdlib>
#include <iostream>
#include <fstream>

using namespace std;

int main()
{
  int x;
  int polje[8];
  ifstream ulaz("podaci.txt");
  for (x = 0; x < 8; x = x + 1)
  {
     ulaz >> polje[x];
  }
  for (x = 0; x < 8; x = x +1)
  {
    cout << polje[x] << endl;
  }
  system("PAUSE");
  return 0;
}
```

```
Zapocinje sortiranje.
Sortiranje je zavrsilo
Press any key to continue . . .
```

Nakon što smo brojeve unijeli u datoteku, pokrenemo program za sortiranje datoteke koji vidimo na lijevoj stranici.

```
800
700
600
500
400
300
200
100
Press any key to continue . . .
```

Programom za čitanje sadržaja datoteke pogledat ćemo rezultat sortiranja.

Dvodimenzionalna polja

Ako postoje jednodimenzionalna polja, moraju postojati i dvodimenzionalna. (U suprotnom bismo imali situaciju kao kad je jedan novinar govorio o lokalnim stanovnicima nekog mjesta, iako nam vjerojatno ne bi mogao objasniti kako izgledaju globalni stanovnici istog mjesta.)

	1.stupac	2. stupac	3. stupac	4. stupac
1. red	11	12	13	14
2. red	21	22	23	24

Dvodimenzionalne matrice pogodne su za spremanje sadržaja tabela.

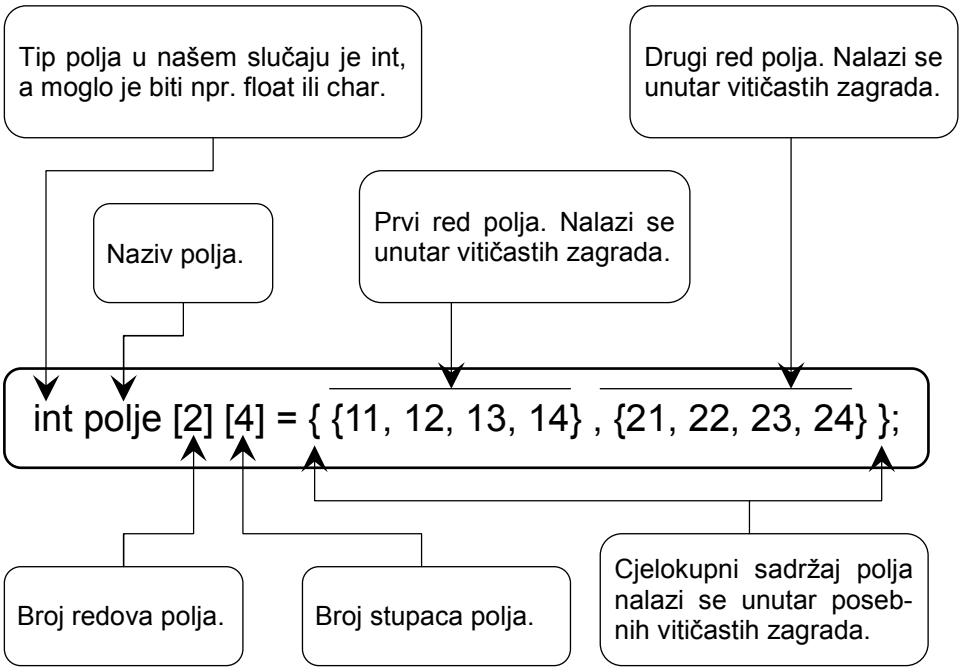
U ovom programu ćemo gornju tabelu spremiti u dvodimenzionalnu matricu, a zatim ćemo prvi i drugi red ispisati.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int polje[2] [4] = {{11, 12, 13, 14}, {21, 22, 23, 24}};
   int x;

   cout << "Prvi red." << endl;
   for (x = 0; x < 4; x = x + 1)
   {
      cout << polje[0] [x] << endl;
   }
   cout << endl;

   cout << "Drugi red." << endl;
   for (x = 0; x < 4; x = x + 1)
   {
      cout << polje[1] [x] << endl;
   }
   cout << endl;

   system("PAUSE");
   return 0;
}
```

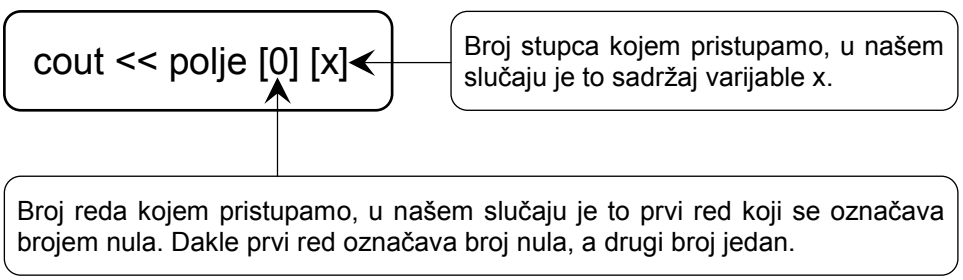
Ispis prvog reda.

Ispis drugog reda.

```
Prvi red.
11
12
13
14

Drugi red.
21
22
23
24

Press any key to continue . . .
```



Budući da su u ovom tipu polja podaci raspoređeni u redove i stupce, prilično logično je da kad želimo pristupiti nekoj ćeliji takvog polja moramo navesti u uglatim zagradama broj reda i broj stupca ćelije kojoj želimo pristupiti.



Dvodimenzionalna polja koriste se na sličan način kao jednodimenzionalna. Osobito su korisna u programima u kojima se odvijaju složeni proračuni.

Osim dvodimenzionalnih postoje i trodimenzionalna polja. Funkcioniraju slično kao dvodimenzionalna, samo što imaju još i "dubinu".

Obrada teksta

String objekti

Želimo li napisati program za vođenje telefonskog imenika ili program za vođenje videoteke, osim obrade brojeva i grafike sasvim sigurno će nam trebati obrada teksta. Iako smo i do sada koristili tekstove za ispisivanje raznih obavijesti, ostalo je nejasno na koji način bismo mogli tekstove pretraživati ili sortirati.

```
Unesite vase ime.
Stjepan

Stjepan, dobar dan!

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   string odgovor;
   cout << "Unesite vase ime." << endl;
   cin >> odgovor;
   cout << endl;
   cout << odgovor << ", dobar dan!" << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Pogledamo li ovaj program, učinit će nam se da smo za lakše baratanje tekstom upotrijebili novi tip varijable, **string** tip; te da smo u programu koristili **varijablu** koju smo nazvali odgovor.

Pogledajmo nove elemente u ovom programu.

```
string odgovor;
```

Iako ovaj red izgleda kao da formiramo varijablu string tipa koju smo nazivali **odgovor,** string nije tip varijable nego **klasa**, a odgovor nije varijabla nego **objekt**.

Time se trenutno ne moramo opterećivati i možemo objekt odgovor koristiti kao da smo formirali varijablu string tipa pod nazivom odgovor.

Tek nešto veće mogućnosti koje će nam string pružiti dat će nam naslutiti da se iza toga naziva krije kompleksnija struktura od obične varijable.

```
#include <string>
```

Da bi program mogao koristiti string objekte, ovu naredbu moramo staviti na početak programa.

U našem programskom okruženju program će raditi i bez ovog reda, ali je moguće da u nekom drugom programskom okruženju neće.

Zbog toga ćemo stavljati tu naredbu

```
cout << odgovor << ", dobar dan!" << endl;
```

Vidimo da objekt odgovor u naredbi cout koristimo kao da koristimo varijablu.

Najprije će se ispisati sadržaj objekta odgovor, zatim tekst unutar navodnika i konačno skok u novi red.

```
cin >> odgovor;
```

U naredbi cin koju koristimo za unos preko tipkovnice objekt odgovor koristimo kao što bismo koristili varijablu.

Dok smo u varijablu char tipa mogli spremiti jedno slovo, u objekt klase string možemo spremati veći tekst.

Očigledno je da u string objekt možemo spremiti tekst i postavlja se pitanje koliko najviše znakova smije sadržavati taj tekst.

U načelu računalo bi se samo trebalo brinuti da osigura dovoljno mjesta za spremanje bilo kojeg teksta. U slučaju korištenja kraćih tekstova, poput imena, naziva gradova i slično o tome ne moramo voditi brigu.

Želimo li u string objekt unijeti veću količinu teksta, da bismo bili sigurni da će sve biti u redu, možemo unaprijed naredbom **reserve** rezervirati potreban prostor.

```
Unesite vase ime.
Ivan

Ivan, dobar dan!

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   string odgovor;
   odgovor.reserve(256);
   cout << "Unesite vase ime." << endl;
   cin >> odgovor;
   cout << endl;
   cout << odgovor << ", dobar dan!" << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

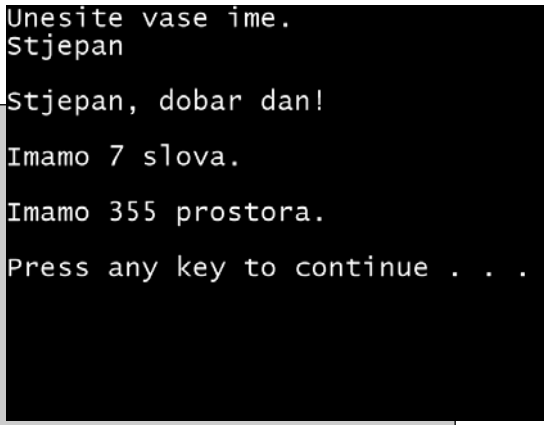
Program radi jednako kao i prethodni, osim što smo u ovom slučaju za spremanje imena osigurali veliki prostor.

Operacija koja se vrši nad objektom. Ovom naredbom se za objekt odgovor rezervira 256 mjesta za spremanje znakova.

```
odgovor.reserve(256)
```

Objekt na koji se naredba odnosi, u našem slučaju je to objekt odgovor.

Obratimo pažnju na točku koja se obvezno nalazi između oznake objekta i oznake operacije koja se nad njim vrši.




```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   string odgovor;
   odgovor.reserve(256);
   cout << "Unesite vase ime." << endl;
   cin >> odgovor;
   cout << endl;
   cout << odgovor << ", dobar dan!" << endl;
   cout << endl;
   cout << "Imamo " << odgovor.size() << " slova." << endl;
   cout << endl;
   cout << "Imamo " << odgovor.capacity() << " prostora." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

odgovor.size()

Naredba **size()** očigledno služi za određivanje broja znakova unesenog teksta. Riječ Stjepan sastoji se od sedam slova.

Objekt na koji se naredba odnosi.

odgovor.capacity()

Naredba **capacity()** služi za određivanje broja mjesta koja nam stoje na raspolaganju za unos znakova u objekt odgovor.

Zašto ih je na raspolaganju 355, ako smo naredbom odgovor.reserve(256) rezervirali 256 mjesta? Zato što je 256 mjesta samo polazište na temelju kojeg prevoditelj rezervira nešto veći prostor.

Objekt na koji se naredba odnosi.

```
Unesite vase ime.
Bunic Stjepan

Bunic, dobar dan!

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   string odgovor;
   odgovor.reserve(256);
   cout << "Unesite vase ime." << endl;
   cin >> odgovor;
   cout << endl;
   cout << odgovor << ", dobar dan!" << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Naredbom cin nije moguće u objekt odgovor unijeti više odvojenih riječi.

Iako smo napisali prezime i ime, uspjeli smo unijeti samo prezime.

```
Unesite vase ime.
Binic Stjepan

Binic Stjepan, dobar dan!

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   string odgovor;
   odgovor.reserve(256);
   cout << "Unesite vase ime." << endl;
   getline(cin, odgovor);
   cout << endl;
   cout << odgovor << ", dobar dan!" << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Ako umjesto cin naredbe upotrijebimo ovu naredbu, moći ćemo unijeti više riječi u objekt odgovor.

getline(cin, odgovor);

String objekt u koji ćemo unositi tekst. U našem slučaju to je objekt odgovor.

Naredba koja nam omogućuje unos više riječi u string objekt.

```
Unesite vase ime.

, dobar dan!

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   string odgovor;
   odgovor.reserve(256);
   cout << "Unesite vase ime." << endl;
   cin >> odgovor;
   cout << endl;
   cout << odgovor << ", dobar dan!" << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

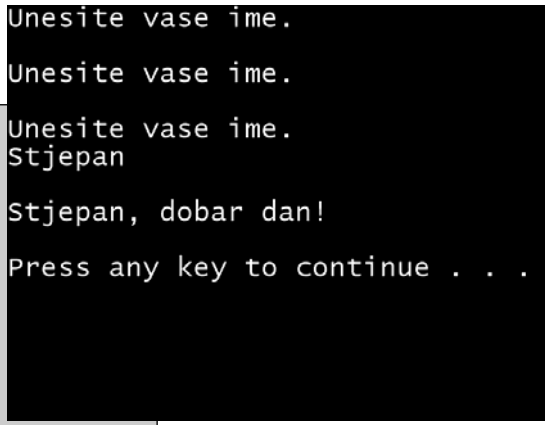
Jedan od problema pri izradi programa javlja se kad korisnik unese neodgovarajuću vrijednost.

U ovom slučaju, umjesto da unesemo ime, samo smo pritisnuli tipku enter.

U ovom slučaju, na mjestu gdje se očekivalo da ćemo unijeti tekst, mi smo unijeli brojeve.

Postavlja se pitanje na koji bismo način mogli spriječiti ili barem smanjiti mogućnost unošenja besmislenih odgovora.

```
Unesite vase ime.
1234567890

1234567890, dobar dan!

Press any key to continue . . .
```




```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
  string odgovor;

  do
  {
    cout << "Unesite vase ime." << endl;
    getline(cin, odgovor);
  }
  while (odgovor.size() == 0);

  cout << endl;
  cout << odgovor << ", dobar dan!" << endl;
  cout << endl;
  system("PAUSE");
  return 0;
}
```

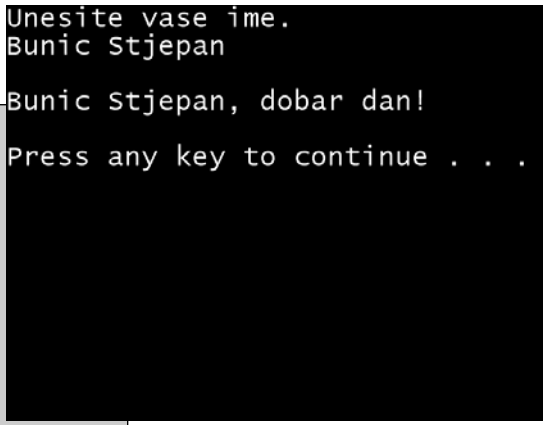
Ova inačica programa će zahtijevati da ponovimo unos ako pritisnemo tipku Enter, a da prije toga nismo ništa unijeli.

```
do
{
  cout << "Unesite vase ime." << endl;
  getline(cin, odgovor);
}
while (odgovor.size() == 0);
```

Naredbu za unos teksta u objekt odgovor stavili smo u do ... while petlju koja se izvodi tako dugo dok je broj slova koja smo unijeli jednak nuli.

Kad je taj broj različit od nule, nastavlja se izvođenje programa.

Polje char tipa




```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   char niz[255];
   cout << "Unesite vase ime." << endl;
   cin.getline(niz,255);
   cout << endl;
   cout << niz << ", dobar dan!" << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Tekst se može unijeti u program i obrađivati i na drugi način; uporabom polja char tipa.

Formiranje niza čije ćelije su char tipa. U svaku možemo staviti jedno slovo.

Tip niza. U našem slučaju niz je char tipa.

```
char niz[255];
```

Broj ćelija niza. U našem slučaju niz sadrži 255 ćelija što znači da u taj niz možemo spremiti 255 slova.

Naziv niza, u našem slučaju naziv niza je **niz**.

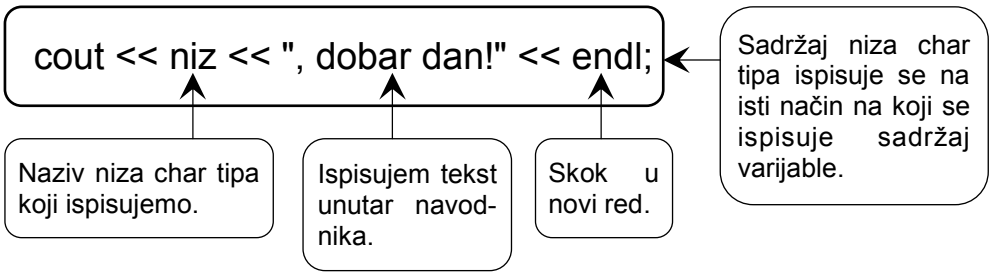
```
cin.getline( niz, 255);
```

Ovom naredbom preko tipkovnice unosimo tekst u niz char tipa.

Broj ćelija niza.

Naredbe za unos tekst u char niz.

Naziv niza.




```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   char niz[255];
   cout << "Unesite vase ime." << endl;
   cin >> niz;
   cout << endl;
   cout << niz << ", dobar dan!" << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

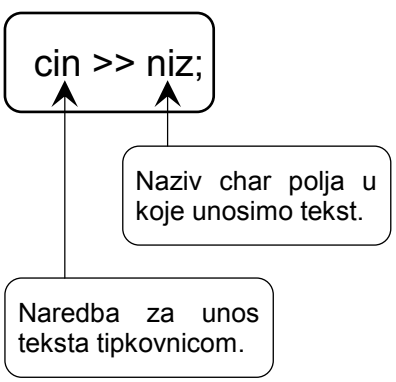
Umjesto naredbe:

cin.getline(niz,255);

možemo staviti naredbu:

cin >> niz;

ali u tom slučaju moći ćemo unijeti samo jednu riječ. Nakon pritiska razmaknice unos će se prekinuti pa, kao što možemo vidjeti na donjoj slici, iako smo napisali Bunic Stjepan, uneseno je samo prezime. Jasno, ako baš želimo unijeti samo jednu riječ onda ćemo upotrijebiti ovu naredbu jer je jednostavnija.




```
Unesite vase ime.
Bunic Stjepan

Bunic, dobar dan!

Press any key to continue . . .
```

U ovu inačicu programa nije moguće unijeti broj na mjestu na kojem očekujemo tekst. U slučaju da unesemo brojeve, računalo će zahtijevati da ponovimo upis.

```
Unesite vase ime.
1234567

Unesite vase ime.
12 12 12

Unesite vase ime.
Stjepan


Stjepan, dobar dan!

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
  char niz[255];
  int x;
  int p;
```

Početni dio programa i formiranje potrebnih varijabli.

```
  do
  {
    cout << "Unesite vase ime." << endl;
    cout << endl;
    cin.getline(niz,255);
    x = 0;
    p = 1;
    while (niz[x] != '\0')
    {
      if (isdigit(niz[x]))
      {
        p = 0;
      }
      x = x + 1;
    }
  }
  while (p == 0);
```

Ovo je glavni dio programa. Ovdje se unosi tekst i ispituje jesu li unesena slova ili brojevi.

Ako su uneseni brojevi, unos se ponavlja.

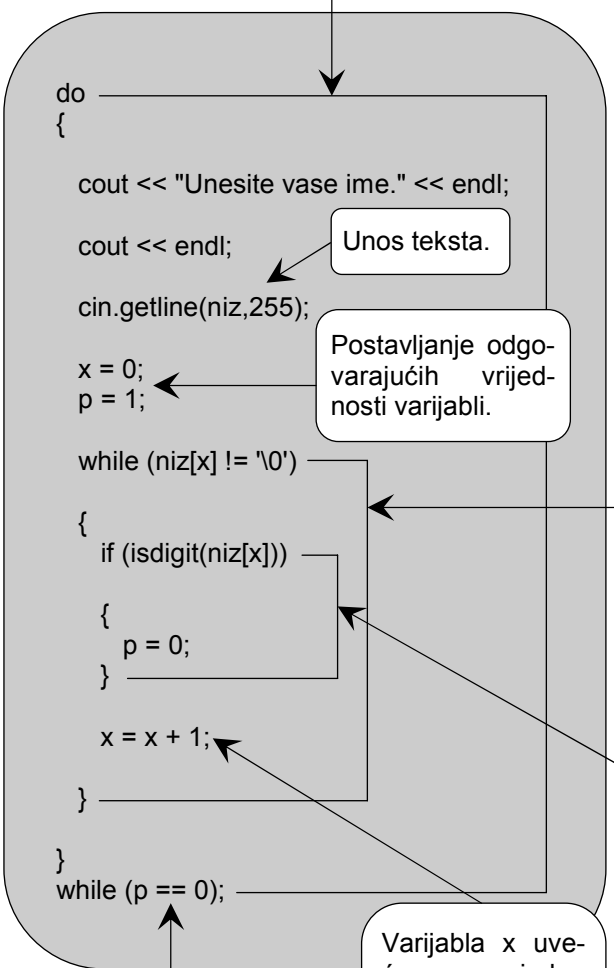
Oznaku \ dobit ćemo pritiskom na AltGr i Q.

```
  cout << endl;
  cout << niz << ", dobar dan!" << endl;
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Završni dio programa, ime se spaja s pozdravom i gasi se program.

Pogledajmo podrobnije glavni dio prethodnog programa u kojem se unosi tekst i ispituje jesu li unesena slova ili brojevi.

Cijeli ovaj dio nalazi se unutar do ... while petlji koja će se ponavljati ako korisnik umjesto slova unese brojeve.



While petlja čita niz, slovo po slovo, dok ne dođe do kraja niza.

Oznaka \0 nalazi se na kraju niza pa niz[x] != '\0' ispituje je li element niza niz[1] jednak znaku za kraj, odnosno jesmo li došli do kraja niza.

To radi tako da je na početku vrijednost varijable x nula, a zatim se uvećava po jedan

If naredbom ispitujemo je li znak u nizu broj.

Ako jest, u p varijablu se sprema broj nula što će imati za posljedicu da će se ponoviti do ... while petlja, odnosno unos.

Ako ni jedan član niza nije broj, u varijabli p će ostati broj jedan i do ... while petlja se neće ponavljati, odnosno upis se neće ponavljati.

Varijabla x uvećava se za jedan da bi se pri svakom ponavljanju naredbom niz[x] čitao slijedeći član niza.

Ovdje se ispituje je li vrijednost varijable p nula i ako jest, do ... while petlja se ponavlja.

Ovaj program je veoma složen pa ćemo odvojeno pogledati važnije elemente programa. Pogledajmo prvo glavnu do ... while petlju.

```
do
{
  cout << "Unesite vase ime." << endl;
  cout << endl;
  cin.getline(niz,255);
  x = 0;
  p = 1;
  while (niz[x] != '\0')
  {
    if (isdigit(niz[x]))
    {
      p = 0;
    }
    x = x + 1;
  }
}
while (p == 0);
```

Ovdje počinje petlja.

Ovdje se u varijablu p sprema broj jedan. To je varijabla o kojoj ovisi hoće li se do ... while petlja ponoviti ili ne.

Ovdje će se varijabla p postaviti na nula ako je jedan od znakova u nizu broj. To će imati za posljedicu da će se do ... while petlja ponoviti, odnosno da će se unos teksta ponoviti.

Ovdje se ispituje je li sadržaj varijable p jednak nuli. Ako jest do ... while petlja će se ponoviti, a to znači da će se upis ponoviti.

Na početku se x postavlja na nulu, što znači da će se prvi put sa niz[x] ispitivati niz[0] odnosno prva ćelija niza.

```
x = 0;
p = 1;
while (niz[x] != '\0')
{
  if (isdigit(niz[x]))
  {
    p = 0;
  }
  x = x + 1;
}
```

While petlja ponavlja se tako dugo dok ćelija niz[x] nije jednaka oznaci za kraj. Prvi put se ispituje prva ćelija odnosno ćelija niz[0].

Nakon što smo while naredbom ispitali jesmo li došli do kraja niza, if naredbom ispitujemo je li ćelija koju trenutno ispitujemo jednaka nuli. Ako jest, varijabla p postavlja se na nulu, što znači da će se ponoviti glavna do ... while petlja.

Ovdje se varijabla x uvećava za jedan. To znači da će pri sljedećem izvođenju while petlje x biti 1, pa će se sa niz[x] ispitivati niz[1], zatim pri sljedećem izvođenju while petlje niz[2] i tako do kraja niza.

```
if  ( isdigit ( niz [x] ) )
```

Ovom naredbom ispitujemo je li jedan znak broj. Može se ispitivati varijabla char tipa ili jedna ćelija niza char tipa.

Ispitujemo ćeliju niza char tipa. Broj ćelije određen je vrijednošću varijable x.

```
a
a nije broj.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   char slovo;
   cin >> slovo;
   if (isdigit(slovo))
   {
      cout << "Ovo je broj " << slovo <<
endl;
   }
   else
   {
      cout << slovo << " nije broj." << endl;
   }
   cout << endl;
   system("PAUSE");
   return 0;
}
```

U ovom jednostavnom programu možemo vidjeti kako funkcionira ispitivanje je li unesen znak broj ili nije.

Osim ispitivanje je li unijeti znak broj, moguća su i druga ispitivanja.

U tablici s desne strane možemo vidjeti druge mogućnosti.

```
8
Ovo je broj 8

Press any key to continue . . .
```

Pokušajmo u gornji program staviti ostala ispitivanja koja možemo vidjeti u tablici.

U tom slučaju morat ćemo modificirati obavijesti koje daje program.

Osim što možemo ispitivati je li sadržaj varijable broj, ispitivati možemo i druga svojstva sadržaja varijable char tipa.

ISPITIVANJE ZNAKOVA	
if (isdigit(slovo))	Ispituje je li sadržaj varijable slovo neki broj.
if (isalnum(slovo))	Ispituje je li sadržaj varijable slovo broj ili slovo. Odgovor je negativan ako unesemo upitnik, točku ili nešto slično.
if (isalpha(slovo))	Ispituje je li sadržaj varijable slovo neko slovo.
if (iscntrl(slovo))	Ispituje je li sadržaj varijable slovo kontrolni znak. To je znak koji se dobiva ako držimo tipku Ctrl i onda pritisnem npr. slovo v.
if (isgraph(slovo))	Je li sadržaj varijable slovo nešto što se ispisuje na zaslon računala, dakle slovo, broj ili interpunkcija. U ovu grupu ne pripada prazno mjesto između dviju riječi.
if (islower(slovo))	Je li sadržaj varijable slovo malo slovo.
if (isprint(slovo))	Je li sadržaj varijable nešto što se ispisuje. Za razliku od slične naredbe if (isgraph(slovo)) ovdje pripada i prazno mjesto između dviju riječi.
if (isupper(slovo))	Je li sadržaj varijable slovo veliko slovo.
if (ispunct(slovo))	Je li sadržaj varijable slovo neka interpunkcija.

```
Prvi znak je broj.
Drugi nije ni slovo ni broj.
Treci je slovo.
Cetvrti je kontrolni znak.
Peti nesto sto se ispisuje.
Sesti je malo slovo.
Sedmi moze biti prazno mjesto.
Osmi je veliko slovo.
Deveti je neka interpunkcija.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
  char slovo[9] = {'8','?','b','\v','*','a',' ','A','#'};
  if (isdigit(slovo[0]))
  {
    cout << "Prvi znak je broj." << endl;
  }
  if (isalnum(slovo[1]))
  {
    cout << "Drugi je slovo ili broj." << endl;
  }
  else
  {
    cout << "Drugi nije ni slovo ni broj." << endl;
  }
  if (isalpha(slovo[2]))
  {
    cout << "Treci je slovo." << endl;
  }
  if (iscntrl(slovo[3]))
  {
    cout << "Cetvrti je kontrolni znak." << endl;
  }
  if (isgraph(slovo[4]))
  {
    cout << "Peti nesto sto se ispisuje." << endl;
  }
  if (islower(slovo[5]))
  {
    cout << "Sesti je malo slovo." << endl;
  }
```

```
  if (isprint(slovo[6]))
  {
    cout << "Sedmi moze biti prazno mjesto." << endl;
  }
  if (isupper(slovo[7]))
  {
    cout << "Osmi je veliko slovo." << endl;
  }
  if (ispunct(slovo[8]))
  {
    cout << "Deveti je neka interpunkcija." << endl;
  }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Ovaj malo veći program morali smo staviti na dvije stranice.

Možemo vidjeti uporabu naredbi za ispitivanje znakova primijenjenu na polje char tipa.

Vidimo da ne možemo odjednom ispitivati cijelo polje, nego da ispitujemo pojedine ćelije u polju.

Program možemo korigirati mijenjanjem sadržaj polja ili tako da, umjesto stalnog sadržaja polja, sadržaj polja unosimo preko tipkovnice.

Uvijek imajmo u vidu da ćemo određenu naredbu usvojiti tek kad napravimo mnoštvo primjera u kojima se ta naredba koristi. Određenu naredbu nismo usvojili kad smo je shvatili. Shvaćanje naredbe je samo uvjet da bismo uopće mogli početi učiti. Naredbu smo usvojili tek kad smo je puno puta uspješno primijenili u raznim programima i na razne načine.

Pogledajmo još nekoliko detalja vezanih uz char varijable i char polja.

```
Unesite prvo slovo.
a
Unesite drugo slovo.
b
Prvo slovo je prije po abecedi.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
  char slovo1;
  char slovo2;
  cout << "Unesite prvo slovo." << endl;
  cin >> slovo1;
  cout << "Unesite drugo slovo." << endl;
  cin >> slovo2;

  if (slovo1 < slovo2)
  {
    cout << "Prvo slovo je prije po abecedi." << endl;
  }
  else
  {
    cout << "Drugo slovo je prije po abecedi." << endl;
  }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Iako to na prvi pogled nema smisla, znakovi se mogu uspoređivati kao i brojevi.

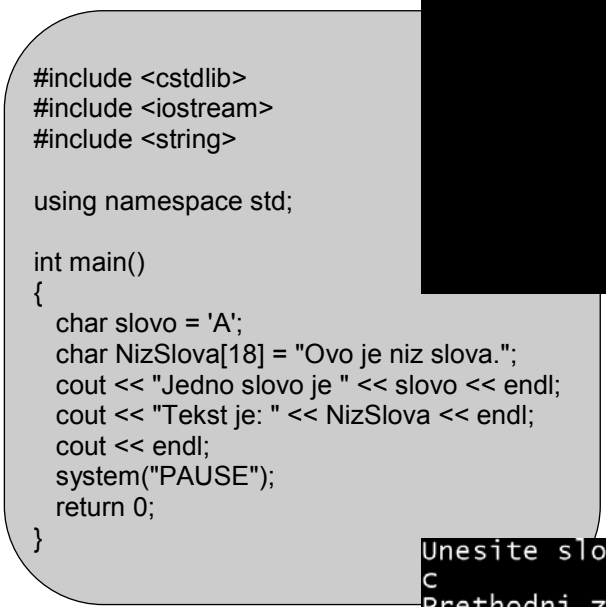
Na prvi pogled izgleda da se uspoređuju na temelju abecede, ali to nije baš tako.

Ovaj program dobro radi ako uspoređujemo samo velika ili samo mala slova.

```
Unesite prvo slovo.
b
Unesite drugo slovo.
a
Drugo slovo je prije po abecedi.

Press any key to continue . . .
```

Zašto na taj način ne možemo uspoređivati velika i mala slova?

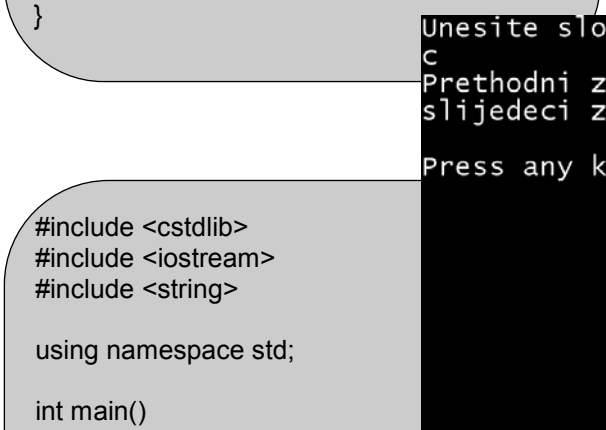
Zato što računalo smatra da velika slova dolaze prije malih. Ako želimo napisati program koji sva slova raspoređuje po abecedi, morali bismo najprije ustanoviti je li neko slovo veliko ili malo, odnosno program bi morao biti nešto složeniji. (Pokušajmo ga napisati.)




```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   char slovo = 'A';
   char NizSlova[18] = "Ovo je niz slova.";
   cout << "Jedno slovo je " << slovo << endl;
   cout << "Tekst je: " << NizSlova << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Mogla bi nas zbuniti uporaba znaka " i znaka '.

Pravilo je jednostavno, jedno slovo piše se u obliku 'a', a više slova u obliku "Ovo je tekst.".




```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   char slovo;
   cout << "Unesite slovo." << endl;
   cin >> slovo;
   slovo = slovo - 1;
   cout << "Prethodni znak je " << slovo << endl;
   slovo = slovo + 2;
   cout << "slijedeci znak je " << slovo << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Slovima možemo dodavati ili oduzimati brojeve.

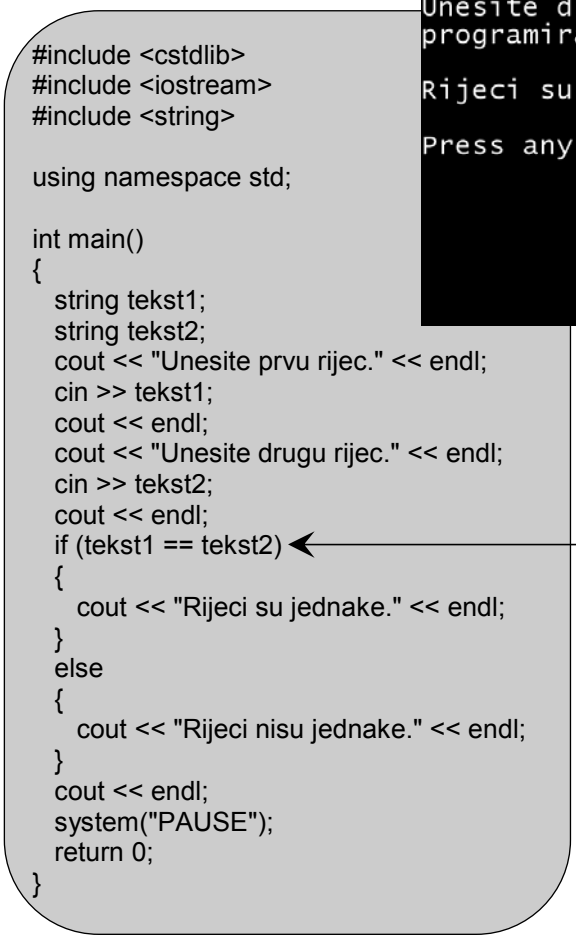
Obrada string objekata

Do sada smo uglavnom vidjeli operacije koje se mogu provoditi nad jednim slovom spremljenim u char varijablu ili char polje. Pogledajmo sada operacije nad većim tekstom spremljenim u objekt string klase.

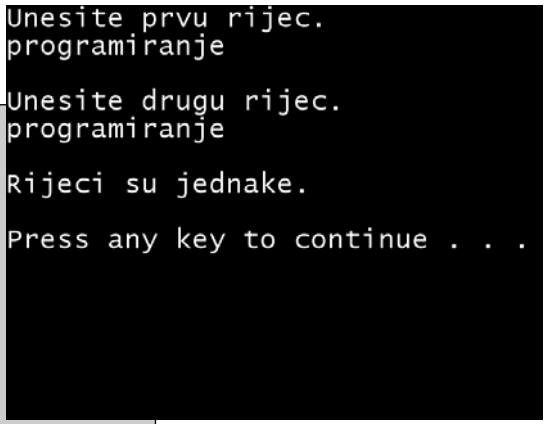
```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   string tekst1 = "Ovo je prvi tekst.";
   string tekst2 = " Ovo je drugi tekst.";
   string tekst3;
   tekst3 = tekst1 + tekst2;
   cout << tekst3 << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Objekte string klase možemo jednostavno zbrojiti i rezultat zbrajanja spremiti u treći objekt.

```
Ovo je prvi tekst. Ovo je drugi
tekst.

Press any key to continue . . .
```



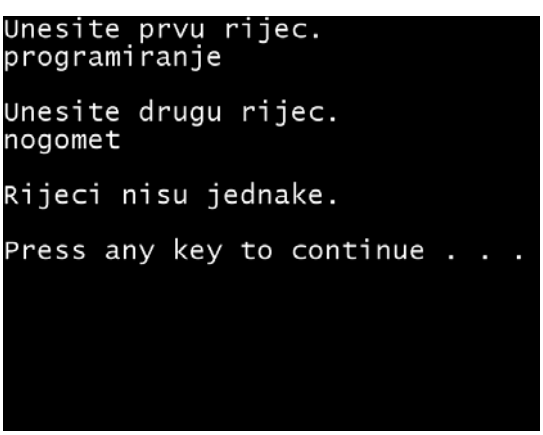
```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
  string tekst1;
  string tekst2;
  cout << "Unesite prvu rijec." << endl;
  cin >> tekst1;
  cout << endl;
  cout << "Unesite drugu rijec." << endl;
  cin >> tekst2;
  cout << endl;
  if (tekst1 == tekst2)
  {
    cout << "Rijeci su jednake." << endl;
  }
  else
  {
    cout << "Rijeci nisu jednake." << endl;
  }
  cout << endl;
  system("PAUSE");
  return 0;
}
```



```
Unesite prvu rijec.
programiranje

Unesite drugu rijec.
programiranje

Rijeci su jednake.

Press any key to continue . . .
```

Objekti klase string mogu se međusobno uspoređivati.

U ovom programu uspoređujemo jesu li jednaki.



```
Unesite prvu rijec.
programiranje

Unesite drugu rijec.
nogomet

Rijeci nisu jednake.

Press any key to continue . . .
```

```
Unesite prvu rijec.
ab

Unesite drugu rijec.
aaaaa

Prva nije manja.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
  string tekst1;
  string tekst2;
  cout << "Unesite prvu rijec." << endl;
  cin >> tekst1;
  cout << endl;
  cout << "Unesite drugu rijec." << endl;
  cin >> tekst2;
  cout << endl;
  if (tekst1 < tekst2)
  {
    cout << "Prva je manja." << endl;
  }
  else
  {
    cout << "Prva nije manja." << endl;
  }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Dva objekta klase string možemo uspoređivati i tako da ispitujemo je li prvi manji od drugog.

Ostaje nejasno u kojem smislu se ispituje je li prvi manji od drugog.

Očito se ne ispituje količina slova, nego se objekti rangiraju po abecedi, ab je po abecedi nakon aaaaa, pa u tome smislu ab nije manje nego je veće od aaaaa.

Kao i kod char polja, sustav je prilično primitivan i dobro radi jedino ako se ne miješaju velika i mala slova, budući da računalo smatra da su sva velika slova prije malih, pa su u tom smislu manja od malih.

Zato računalo smatra da je AB manje od aaaaa.

```
Unesite prvu rijec.
AB

Unesite drugu rijec.
aaaaa

Prva je manja.

Press any key to continue . . .
```

Moguće su i složenije manipulacije nad tekstom. Ponovno pogledajmo naredbu **size()** koju smo vidjeli na početku ovog poglavlja. Omogućuje nam određivanje broja znakova koje sadrži tekst. Pri tome imajmo na umu da računalo i prazno mjesto između riječi smatra jednim znakom.

```
Ovo je moj tekst.
Tekst sadrzi 17 znakova.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   string odgovor;
   getline(cin, odgovor);
   cout << "Tekst sadrzi " << odgovor.size() << " znakova." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Naredba size().

```
Ovo je drugi tekst.
Tekst sadrzi 19 znakova.

Press any key to continue . . .
```

```
Ovo je jos jedan tekst.
Tekst sadrzi 23 znakova.

Press any key to continue . . .
```

Ovaj program pretražuje sadržaj objekta tekst i traži pojavljuje li se u tom objektu sadržaj objekta trazi. Ako se pojavljuje, obavještava nas na kojoj poziciji započinje traženi tekst. (Prva pozicija nema broj jedan nego broj nula.)

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
  string tekst;
  string trazi;
  int pozicija;
  cout << "Unesite tekst." << endl;
  getline(cin, tekst);
  cout << endl;
  cout << "Sto trazim?" << endl;
  getline(cin, trazi);
  cout << endl;
  pozicija = tekst.find(trazi);
  if (pozicija != string::npos)
  {
    cout << "Pocetak je na " << pozicija << ". poziciji." << endl;
  }
  else
  {
    cout << "Nema trazene rijeci." << endl;
  }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

```
Unesite tekst.
Pliva patka preko Save ...

Sto trazim?
patka

Pocetak je na 6. poziciji.

Press any key to continue . . .
```

```
Unesite tekst.
Pliva patka preko Save ...

Sto trazim?
Drava

Nema trazene rijeci.

Press any key to continue . . .
```

Ovom naredbom utvrđujemo poziciju traženog teksta, a rezultat spremamo u varijablu pozicija.

Varijabla u koju spremamo rezultat.

```
pozicija = tekst.find(trazi);
```

Naziv objekta u kojem se nalazi tekst koji pretražujemo.

Naredba za pretraživanja.

Objekt koji pretražujemo.

Ovom naredbom provjeravamo je li tražena riječ nađena u pretraživanom objektu.

```
if (pozicija != string::npos)
```

U varijabli pozicija nalazi se rezultat pretraživanja objekta.

Ovom naredbom dobivamo oznaku koja označava da tražena riječ nije nađena.

Uspoređujemo rezultat pretraživanja u pozicija varijabli i obavijest da tražena riječ nije nađena.

```
Unesite tekst.
Ivan radi, a radi i ona.

Sto trazim?
radi

Pocetak je na 5. poziciji.

Press any key to continue . . .
```

Problem ovog programa, odnosno ove naredbe, je u tome što će pronaći samo prvo pojavljivanje neke riječi, ako se ista riječ više puta pojavljuje u tekstu.

Želimo li ispitati sva pojavljivanja neke riječi, morat ćemo napisati malo složeniji program.

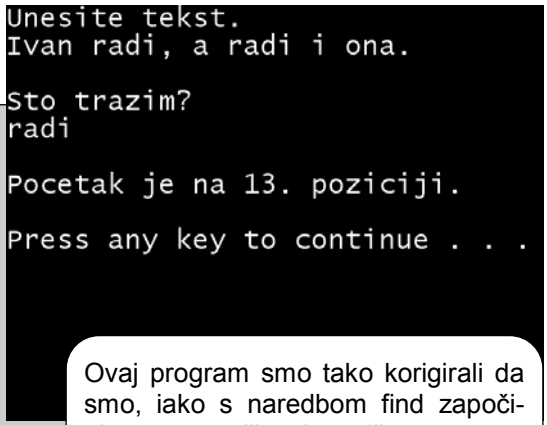
Ovaj program radi sličan posao kao i prethodni, ali pretraživanje započinje od kraja teksta. Ako tražimo riječ koja se pojavljuje više puta, ovaj program naći će posljednje pojavljivanje.

```
Unesite tekst.
Ivan radi, a radi i ona.

Sto trazim?
radi

Pocetak je na 13. poziciji.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   string tekst;
   string trazi;
   int pozicija;
   cout << "Unesite tekst." << endl;
   getline(cin, tekst);
   cout << endl;
   cout << "Sto trazim?" << endl;
   getline(cin, trazi);
   cout << endl;
   pozicija = tekst.rfind(trazi);
   if (pozicija != string::npos)
   {
      cout << "Pocetak je na " << pozicija << ". poziciji." << endl;
   }
   else
   {
      cout << "Nema trazene rijeci." << endl;
   }
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Program je u potpunosti identičan prethodnome, samo što ovdje umjesto naredbe find imamo naredbu rfind.

Time smo računalu naredili da pretraživanje započne od kraja teksta koji se nalazi u objektu tekst.



Ovaj program smo tako korigirali da smo, iako s naredbom find započinjemo pretraživanje s lijeve strane, pronašli drugo pojavljivanje tražene riječi.

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
  string tekst;
  string trazi;
  int pozicija;
  cout << "Unesite tekst." << endl;
  getline(cin, tekst);
  cout << endl;
  cout << "Sto trazim?" << endl;
  getline(cin, trazi);
  cout << endl;
  pozicija = tekst.find(trazi,6);
  if (pozicija != string::npos)
  {
    cout << "Pocetak je na " << pozicija << ". poziciji." << endl;
  }
  else
  {
    cout << "Nema trazene rijeci." << endl;
  }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

pozicija = tekst.find(trazi,6);

U već poznatu naredbu find dodali smo broj 6 čime smo naredili računalu da pretraživanje objekta tekst započne od znaka na poziciji 6.

Budući da prva riječ radi počinje na 5, računalo je našlo drugo pojavljivanje riječi radi.

Pomoću ove naredbe napravit ćemo program koji će pronaći sva pojavljivanja neke riječi, a ne samo prvo ili posljednje.

Ovaj program može pronaći svako pojavljivanje neke riječi u nekom tekstu.

```
Unesite tekst.
On ne radi, a ona radi i radi.

Sto trazim?
radi

Nalazi se na 6. mjestu.
Nalazi se na 18. mjestu.
Nalazi se na 25. mjestu.

Press any key to continue . . .
```

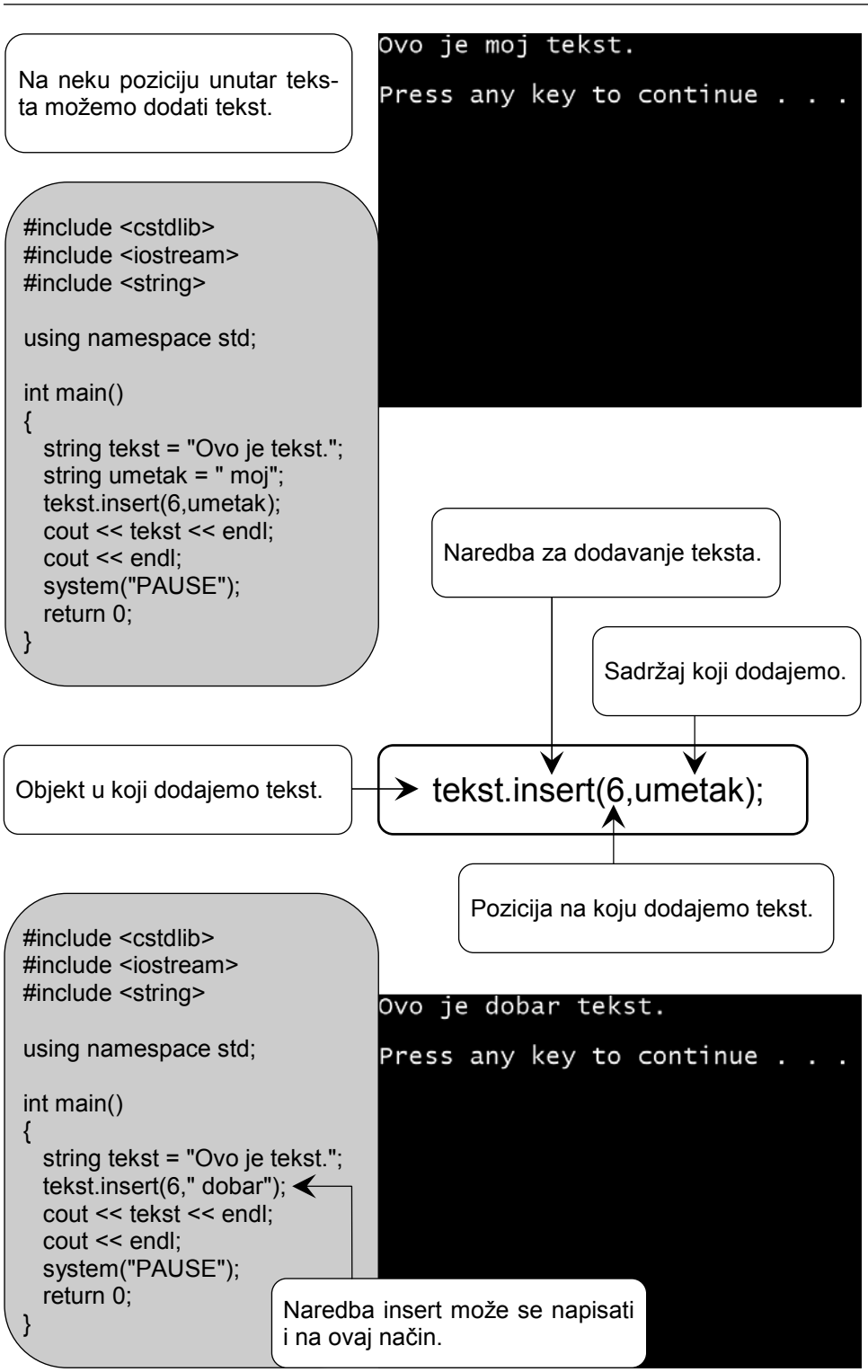
```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
  string tekst;
  string trazi;
  int duljina;
  int trenutno = 0;
  int polozaj;
  cout << "Unesite tekst." << endl;
  getline(cin, tekst);
  cout << endl;
  cout << "Sto trazim?" << endl;
  getline(cin, trazi);
  cout << endl;
  duljina = tekst.size();
  while (trenutno < (duljina + 1))
  {
    polozaj = tekst.find(trazi,trenutno);
    if (polozaj != string::npos)
    {
      cout << "Nalazi se na " << polozaj << ". mjestu." << endl;
      trenutno = polozaj + 1;
    }
    else
    {
      trenutno = duljina + 1;
    }
  }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

U ovom programu koristimo naredbu find, a pozicija početka pretraživanja nije određena brojem, nego sadržajem varijable **trenutno**.

Varijabla trenutno je na početku 0, da bi prvo pretraživanje počelo od početka teksta. Ako na nekoj poziciji pronađemo traženu riječ, varijablu trenutno uvećavamo za tu poziciju + 1, da bi novo pretraživanje počelo nakon početka već nađene riječi i da bi se eventualno pronašla nova riječ, a ne ponovo ista.

Budući da smo došavši do ove točke već prilično napredovali u vještini programiranja, ovaj put je objašnjenje funkcioniranja programa nešto skromnije.

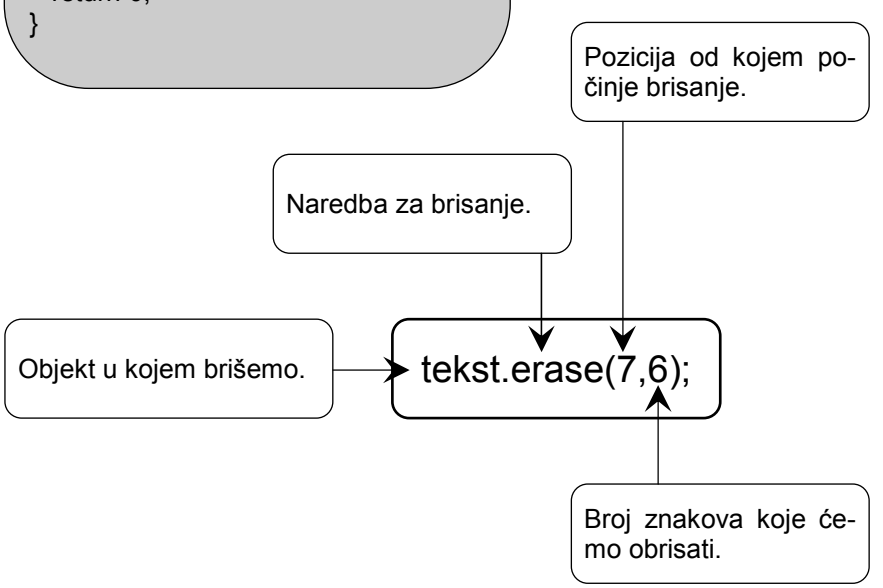
Pokušajmo shvatiti kako funkcionira program i pokušajmo ga korigirati tako da ispiše adekvatnu obavijest, ako se tražena riječ uopće ne pojavljuje. Pokušajmo isti program napisati korištenjem rfind naredbe.


Na neku poziciju unutar teksta možemo dodati tekst.
Ovo je moj tekst.
Press any key to continue . . .
#include <cstdlib>
#include <iostream>
#include <string>
using namespace std;
int main()
{
string tekst = "Ovo je tekst.";
string umetak = " moj";
tekst.insert(6,umetak);
cout << tekst << endl;
cout << endl;
system("PAUSE");
return 0;
}
Naredba za dodavanje teksta.
Sadržaj koji dodajemo.
Objekt u koji dodajemo tekst.
tekst.insert(6,umetak);
Pozicija na koju dodajemo tekst.
#include <cstdlib>
#include <iostream>
#include <string>
using namespace std;
int main()
{
string tekst = "Ovo je tekst.";
tekst.insert(6," dobar");
cout << tekst << endl;
cout << endl;
system("PAUSE");
return 0;
}
Ovo je dobar tekst.
Press any key to continue . . .
Naredba insert može se napisati i na ovaj način.

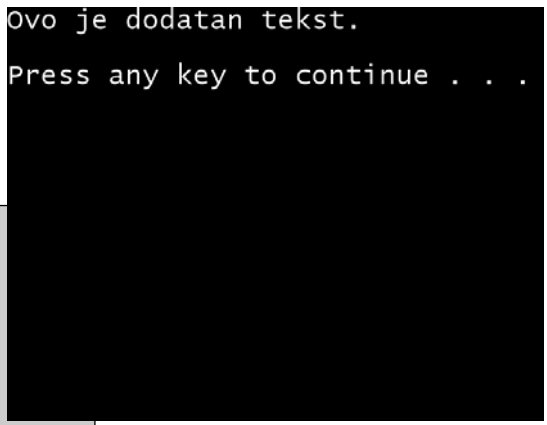
Dio teksta spremljenog u objekt klase string možemo obrisati **erase** naredbom.

```
Ovo je tekst.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   string tekst = "Ovo je dobar tekst.";
   tekst.erase(7,6);
   cout << tekst << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```



Naredbom replace možemo dio teksta koji se nalazi u objektu klase string zamijeniti nekim drugim tekstom.




```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   string tekst = "Ovo je dobar tekst.";
   string drugi = "dodatan";
   tekst.replace(7,5,drugi);
   cout << tekst << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Pozicija od koje započinje zamjena teksta.

Koliko ćemo znakova zamijeniti.

Naredba za zamjenu teksta.

Objekt u kojem mijenjamo dio teksta.

```
tekst.replace(7,5,drugi);
```

Sadržaj kojim ćemo zamijeniti tekst u objektu tekst. Zamijenit ćemo 5 znakova, počevši od sedmog sa sadržajem objekta drugi. Umjesto objekta drugi mogli bismo staviti i tekst unutar navodnika.

Premještanje teksta iz char polja u string

Postavlja se pitanje bismo li mogli tekst iz jednog oblika pretvoriti u drugi, ovisno o tome koji oblik nam je pogodniji za obradu.

```
Unesite vase ime.

123456789
Unesite vase ime.

Stjepan

Ime Stjepan sadrzi 7 slova.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>

using namespace std;

int main()
{
   char niz[255];
   string tekst;
   tekst.reserve(256);
   int x;
   int p;
   do
   {
      cout << "Unesite vase ime." << endl;
      cout << endl;
      cin >> niz;
      x = 0;
      p = 1;
      while (niz[x] != '\0')
      {
         if (isdigit(niz[x]))
         {
            p = 0;
         }
         x = x + 1;
      }
   }
   while (p == 0);
   cout << endl;
   tekst = niz;
   cout << "Ime " << tekst << " sadrzi " << tekst.size() << " slova." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Ovdje, na način koji smo već vidjeli u ovom poglavlju, ime unosimo u char polje i provjeravamo jesu li uneseni brojevi ili slova. Ako su uneseni brojevi, unos se ponavlja. Naredba **isdigit** ne bi radila da smo koristili objekt klase string.

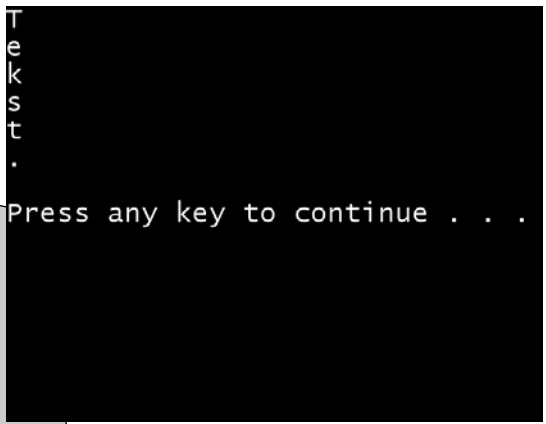
Ovdje tekst iz polja tipa char premještamo u objekt klase string koji smo nazvali tekst.

To smo učinili da bismo mogli koristiti size naredbu.

Naredba niz.size nije dozvoljena. Naredba size radi jedino s objektima, ali ne i s poljima.

Premještanje teksta iz stringa u char polje

Moguće je i tekst iz objekta klase string premjestiti u polje char tipa, ali je naredba kojom to izvodimo nešto složenija.




```
#include <cstdlib>
#include <iostream>
#include <string>


using namespace std;

int main()
{
   string tekst = "Tekst.";

   char *polje = (char*)tekst.c_str();

   cout << polje[0] << endl;
   cout << polje[1] << endl;
   cout << polje[2] << endl;
   cout << polje[3] << endl;
   cout << polje[4] << endl;
   cout << polje[5] << endl;

   cout << endl;
   system("PAUSE");
   return 0;
}
```

Formiramo objekt klase string i u njega spremamo tekst “Tekst”.

Tekst iz objekta selimo u char polje koje smo nazvali polje.

Ispisujemo sadržaj polja da bismo vidjeli nalazi li se u njemu doista “Tekst”.

```
char *polje = (char*)tekst.c_str();
```

Naziv char polja u koje ćemo spremiti tekst iz objekta. U našem slučaju naziv je **polje**.

Naziv objekta iz kojeg tekst selimo u char polje. U našem slučaju naziv objekta je **tekst**.

Spremanje teksta u datoteku

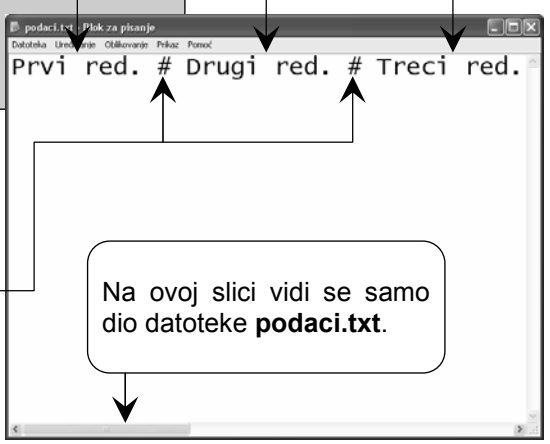
Pomoću objekata klase string možemo formirati liste i te liste možemo spremati u datoteke na sličan način na koji smo spremali brojčane liste

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main()
{
   int x;
   string tekst[8];
   ofstream izlaz("podaci.txt");
   for (x = 0; x < 8; x = x + 1)
   {
      getline(cin, tekst[x]);
   }
   for (x = 0; x < 8; x = x + 1)
   {
      izlaz << tekst[x];
      izlaz << " # ";
   }
   cout << endl;
   cout << "Spremanje je gotovo." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

```
Prvi red.
Drugi red.
Treci red.
Cetvrti red.
Peti red.
Sesti red.
Sedmi red.
Osmi red.

Spremanje je gotovo.

Press any key to continue . . .
```

Kad smo spremali brojeve, između brojeva stavljali smo prazno mjesto da bismo znali gdje završava jedan broj, a počinje drugi.

U ovom programu prazna mjesta nisu prikladna za razlikovanje sadržaja pojedinih ćelija jer se prazna mjesta nalaze između riječi.



Umjesto praznih mjesta, između sadržaja pojedinih ćelija stavljamo " # "

Ta oznaka će nam pomoći prilikom čitanja datoteke.

Na ovoj slici vidi se samo dio datoteke **podaci.txt**.

Programom sličnim programu za čitanje datoteka u kojima se nalaze brojevi čitat ćemo sadržaj datoteke u kojoj se nalaze tekstovi.

```
Prvi red.
Drugi red.
Treci red.
Cetvrti red.
Peti red.
Sesti red.
Sedmi red.
Osmi red.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <fstream>

using namespace std;

int main()
{
  int x;
  string tekst[8];
  string privremeni;
  ifstream ulaz("podaci.txt");
  for (x = 0; x < 8; x = x + 1)
  {
     do
     {
       ulaz >> privremeni;
       if (privremeni != "#")
       {
         tekst[x] = tekst[x] + privremeni + " ";
       }
     }
     while (privremeni != "#");
  }
  for (x = 0; x < 8; x = x +1)
  {
    cout << tekst[x] << endl;
  }
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Oznaku **#** koristit ćemo da bismo utvrdili jesmo li došli do kraja sadržaja jedne ćelije polja.

Ovo rješenje je jednostavno, ali je manjkavo. Ako bi netko prilikom unosa teksta unio **#** oznaku, poremetio bi ispravno funkcioniranje programa.

Zbog toga bi program za unos teksta trebalo tako modificirati da ne dozvoljava unos **#** oznake.

Objekti

Jednostavan primjer klase i objekata

U ovom trenutku mogla bi se javiti dva pitanja koja su međusobno povezana. Jedno pitanje je imamo li na raspolaganju samo one klase koje su napisali autor C++ jezika ili bismo mogli napisati svoje vlastite klase.

Pitanje zašto bismo to uopće radili usko je povezano s drugim pitanjem, a to je na koji način se pišu veliki programi. Mi smo do sada pisali male programe. Pišu li se na isti način programi koji sadrže nekoliko stotina tisuća redova? Pišu li se oni na način da počnemo s **#include <cstdlib>** napišemo nekoliko stotina tisuća redova, na kraju napišemo **}** i program je gotov.

Naravno da ne. To bi bilo nepraktično iz niza razloga. U slučaju da program pišemo na taj način (što je tehnički moguće), jako teško bismo pisanje programa raspodijelili na više osoba i u tako velikom programu jako teško bismo pronalazili greške.

Postoji još jedan problem. U većim programima često se isti posao mora obaviti na više mjesta u programu. Npr. ako u telefonski imenik želimo unijeti ime, prezime, ulicu, grad, zanimanje i tome slično, istu provjeru, npr. provjeru je li umjesto slova netko unio brojeve morali bismo ponoviti prilikom svakog od nabrojenih unosa.

Svi navedeni problemi rješavaju se tako da se program razdijeli u manje cjeline. U tom slučaju lakše se posao podijeli na više programera. Svaki radi svoju manju cjelinu. Manje cjeline lakše se pišu i u njima se lakše pronalaze greške.

Ako dio programa koji se često ponavlja izdvojimo u manju cjelinu, koju pozivamo kad nam je potrebna, dobit ćemo manji program u kojem se lakše vrše izmjene. Lakše je izmjenu unijeti u jedan dio koji se poziva na više mjesta, nego u slučaju kad se isti dio programa ponavlja na desetak mjesta. U tom slučaju trebali bi pronaći svih desetak mjesta i na svima promijeniti program na isti način, što je velik posao, a postoji i velika vjerojatnost da će doći do greške.

Vidimo da bi bilo dobro podijeliti program na manje dijelove pa se postavlja pitanje kako ćemo to učiniti. To ćemo učiniti kreiranjem vlastitih objekata ili preciznije rečeno, kreiranjem vlastitih klasa na temelju kojih ćemo formirati objekte.

Postoji još jedan razlog zašto je dobro poznavati uporabu klasa i objekata. Želimo li pisati programa koji će se umjesto u pojednostavljenom crnom prozoru odvijati u normalnom windows prozoru, apsolutno je nužno razumjeti uporabu klasa i objekata.

Imajmo u vidu da prednost uporabe objekata dolazi do izražaja u većim programima. Manji programi koje ćemo vidjeti u ovom poglavlju mogu se elegantnije napraviti bez uporabe objekata. Unatoč tome, mi ćemo vidjeti primjere uporabe objekata u malim programima jer ćemo na primjerima malih programa lakše shvatiti princip uporabe objekata.

```
Ovo je nas tekst.
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

class tekst
{
   public:
      void pisi()
      {
         cout << "Ovo je nas tekst." << endl;
      }
};

int main()
{
   tekst t;
   t.pisi();
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Unesimo u programsku okolinu i pokrenimo naš prvi program koji koristi klase i objekte.

Uočimo da taj program radi potpuno isti posao kao ovaj. Očito je da se u malim programima ne isplati koristiti klase i objekte. Njihove prednosti dolaze do izražaja u većim programima.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   cout << "Ovo je nas tekst." << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Najprije ćemo pogledati zanimljive dijelove našeg programa, a zatim funkcioniranje programa kao cjeline.

Ovaj dio programa izrazito je sličan ovom dijelu programa iz prošlog poglavlja.

```
tekst t;
t.pisi();
```

```
string tekst;
tekst.reserve(256);
```

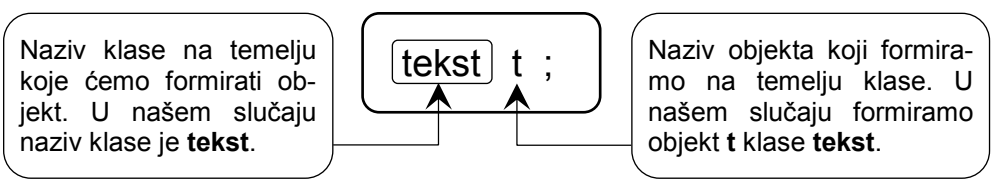
U prvom redu formira se objekt klase **tekst** koji smo nazvali **t**.

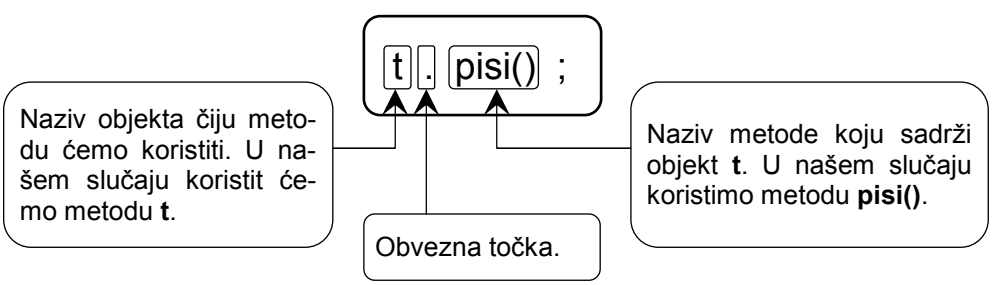
U drugom redu nad objektom **t** vrši se operacija **pisi()**. Unutar zagrada nema sadržaja, jer se ništa ne unosi u objekt **t**, ali su zagrade svejedno obvezne.

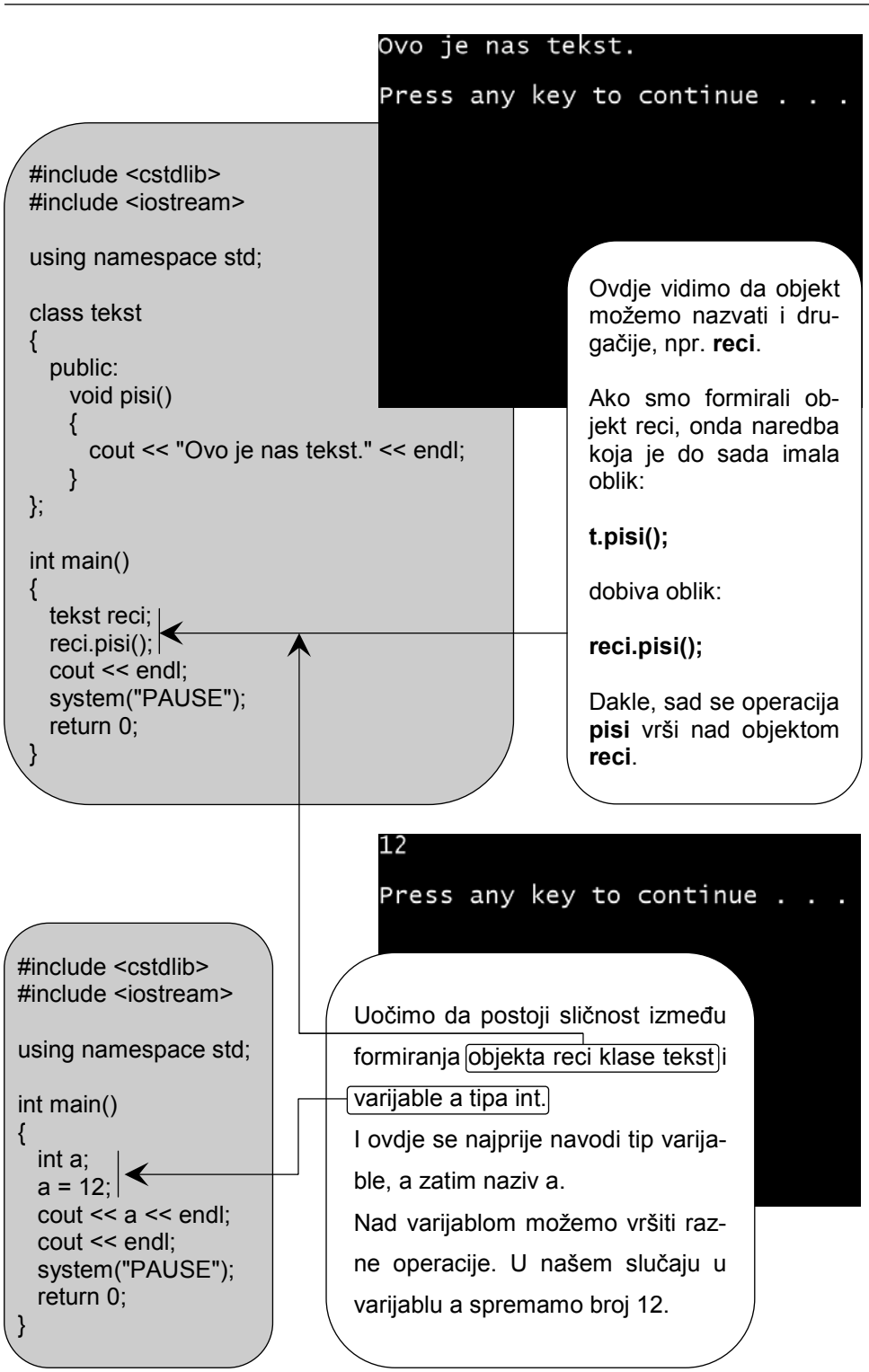
Oblik **t.pisi** bez zagrada nije ispravan.

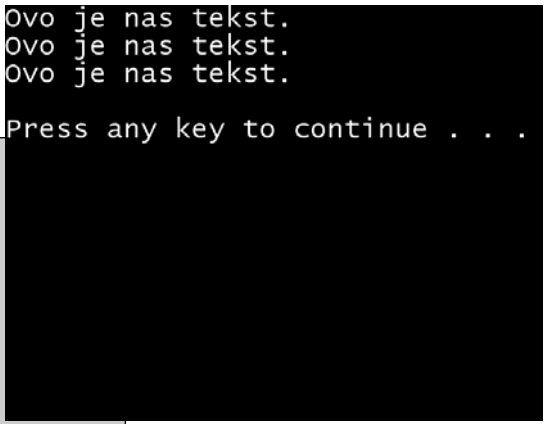
U prvom redu formirali smo objekt klase **string** koji smo nazvali **tekst**.

U drugom redu nad objektom **tekst** izvršili smo operaciju **reserve(250)** kojom smo osigurali prostor za spremanje teksta. Između naziva objekta i naziva operacije nalazi se točka.






Ovo je nas tekst.
Press any key to continue . . .
#include <cstdlib>
#include <iostream>
using namespace std;
class tekst
{
public:
void pisi()
{
cout << "Ovo je nas tekst." << endl;
}
};
int main()
{
tekst reci;
reci.pisi();
cout << endl;
system("PAUSE");
return 0;
}
Ovdje vidimo da objekt možemo nazvati i drugačije, npr. reci.
Ako smo formirali objekt reci, onda naredba koja je do sada imala oblik:
t.pisi();
dobiva oblik:
reci.pisi();
Dakle, sad se operacija pisi vrši nad objektom reci.
12
Press any key to continue . . .
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
int a;
a = 12;
cout << a << endl;
cout << endl;
system("PAUSE");
return 0;
}
Uočimo da postoji sličnost između formiranja objekta reci klase tekst i varijable a tipa int.
I ovdje se najprije navodi tip varijable, a zatim naziv a.
Nad varijablom možemo vršiti razne operacije. U našem slučaju u varijablu a spremamo broj 12.




```
#include <cstdlib>
#include <iostream>

using namespace std;

class tekst
{
   public:
      void pisi()
      {
         cout << "Ovo je nas tekst." << endl;
      }
};

int main()
{
   tekst prvi;
   tekst drugi;
   tekst treci;
   prvi.pisi();
   drugi.pisi();
   treci.pisi();
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Kao što možemo formirati više varijabli int tipa, tako možemo formirati više objekata klase tekst.

U ovom primjeru formirali smo tri objekta, objekt prvi, objekt drugi i objekt treći.

Za biranje naziva objekata vrijede slična pravila kao i za biranje naziva varijabli.

```
class tekst
{
   public:
      void pisi()
      {
         cout << "Ovo je nas tekst." << endl;
      }
};
```

Za razliku od klase string koju je već netko prije napisao, a mi je samo koristimo, klasu tekst morali smo sami napisati.

U ovom dijelu programa formira se klasa tekst.

Formiranje klase započinje naredbom class. To je naredba C++ jezika i ne može se mijenjati.

class tekst

Nakon naredbe class slijedi naziv klase. U našem slučaju klasu smo nazvali tekst, ali mogli smo i drugačije.

```
#include <cstdlib>
#include <iostream>

using namespace std;

class BrdaIDoline
{
   public:
      void pisi()
      {
         cout << "Ovo je nas tekst." << endl;
      }
};

int main()
{
   BrdaIDoline t;
   t.pisi();
   cout << endl;
   system("PAUSE");
   return 0;
}
```

U ovoj inačici programa klasu smo nazvali BrdaIDoline.

To je dozvoljeno, samo što sad naredbu koja je prije imala oblik:

tekst t;

moramo napisati u obliku:

BrdaIDoline t;

Dakle, formiramo objekt t klase BrdaIDoline.

(Drugo je što je pametno da nam naziv klase govori čemu klasa služi, pa je logičnije da se klasa za ispis teksta zove tekst, a ne BrdaIDoline.)

```
Ovo je nas tekst.

Press any key to continue . . .
```

Vidimo da je rezultat rada takvog programa potpuno jednak kao i rezultat rada programa u kojem smo formirali klasu tekst.

```
public:
  void pisi()
  {
    cout << "Ovo je nas tekst." << endl;
  }
```

U ovom djelu nalazi se popis varijabli i metoda koje posjeduje neka klasa. Metode su mudar naziv za operacije koje neka klasa može izvršiti.

U našem slučaju klasa ne sadrži varijable. Ima samo jednu metodu koju smo nazivali pisi().

Popis metoda počinje naredbom **public:** Pobliže značenje te naredbe vidjet ćemo kasnije.

```
  void pisi()
  {
    cout << "Ovo je nas tekst." << endl;
  }
```

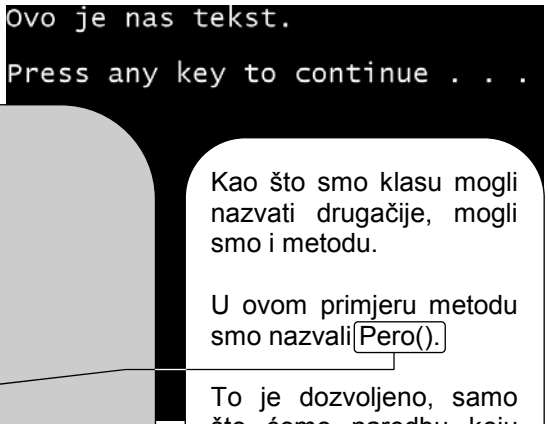
Ova naša mala klasa ne sadrži varijable i sadrži samo jednu metodu odnosno može napraviti samo jednu operaciju. Tu metodu nazvali smo **pisi ()**.

Ispred naziva metode nalazi se naredba void. Njeno značenje pobliže ćemo objasniti kasnije.

```
  {
    cout << "Ovo je nas tekst." << endl;
  }
```

Unutar vitičastih zagrada nalaze se naredbe koje pojedina metoda izvršava.

Naša jednostavna metoda samo naredbom cout ispisuje tekst unutar navodnika na zaslon računala.




```
#include <cstdlib>
#include <iostream>

using namespace std;

class tekst
{
  public:
    void Pero()
    {
      cout << "Ovo je nas tekst." << endl;
    }
};

int main()
{
  tekst t;
  t.Pero();
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Kao što smo klasu mogli nazvati drugačije, mogli smo i metodu.

U ovom primjeru metodu smo nazvali Pero().

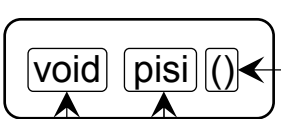
To je dozvoljeno, samo što ćemo naredbu koju smo do sada pisali u obliku:

t.pisi();

pisati u obliku:

t.Pero();

Kao i kod biranja naziva varijabli i klasa, i ovdje je pametno da nam naziv metode govori što metoda radi. Zato je naziv pisi za metodu koja ispisuje tekst bolji od naziva Pero.



Void naredba ispred naziva metode označuje da metoda ne vraća nikakvu vrijednost.

U kojem smislu ne vraća vrijednost vidjet ćemo kasnije kad ovakvu metodu usporedimo s onima koje vraćaju vrijednost.

Dok naziv metode uz određena ograničenja možemo birati po volji, ove dvije zagrade se ovdje moraju obavezno nalaziti.

Naziv metode biramo po volji, pazeći na ograničenja o kojima smo govorili kad smo govorili o biranju naziva varijabli.

Pametno je da naziv metode tako izaberemo da nam govori čemu metoda služi.

Pogledajmo strukturu našeg programa.

```
#include <cstdlib>
#include <iostream>

using namespace std;

class tekst
{
  public:
    void pisi()
    {
      cout << "Ovo je nas tekst." << endl;
    }
};

int main()
{
  tekst t;
  t.pisi();
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Početni dio programa.

U ovom dijelu nalaze se klase koje program koristi. U našem slučaju imamo samo jednu klasu, klasu pisi().

Glavni dio programa. Sada možemo vidjeti zašto smo program započinjali s **int main()** oznakom, odnosno oznakom da je to glavni dio.

Kad računalo pokrene program, najprije će se pokrenuti ovaj dio, a ostali dijelovi će se koristiti po potrebi.

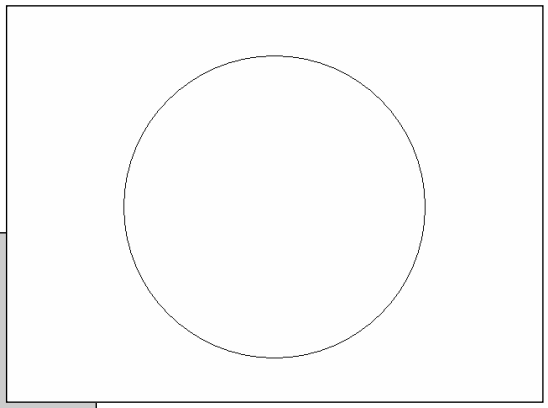
Sadrža klase nalazi se unutar vitičastih zagrada.

Klasa se može sastojati od više tipova sadržaja. Jedan od njih je **public**. Kasnije ćemo podrobnije vidjeti što oznaka public označava.

```
class tekst
{
  public:
    void pisi()
    {
      cout << "Ovo je nas tekst." << endl;
    }


};
```

U klasi se mogu nalaziti varijable i metode. U našem slučaju varijable ne koristimo, a imamo jednu metodu koju smo nazvali **pisi()**.

Metode su zapravo programi koji određuju koje operacije neka klasa može izvesti .



Unutar metode ne mora se nalaziti program za ispis. Može se nalaziti bilo što, npr. program za crtanje kruga.

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>

using namespace std;

class tekst
{
   public:
      void pisi()
      {
         int gdriver = 9;
         int gmode = 2;
         initgraph(&gdriver, &gmode, "");
         setbkcolor(WHITE);
         setcolor(BLACK);
         cleardevice();
         circle(320,240,180);
         getch();
         closegraph();
      }
};

int main()
{
   tekst t;
   t.pisi();
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Iako program i ovako radi, malo ćemo ga preurediti.

Umjesto **class tekst** stavit ćemo **class slika**.

Umjesto **void pisi()** stavit ćemo **void krug()**.

Umjesto **tekst t;** tj. umjesto naredbe formiranja objekta **t** klase tekst stavimo **slika crtaj**. Time ćemo formirati objekt crtaj klase slika.

Umjesto **t.pisi();** tj. umjesto pozivanja metode pisi() u klasi tekst stavit ćemo **crtaj.krug();**

Izbacimo naredbu **cout << endl;** i naredbu **system("PAUSE");**.
Iako program radi i s tim naredbama, bez njih će raditi nešto elegantnije. Ovaj se program odvija u grafičkom prozoru pa su nepotrebni razmak i pauza u tekstualnom prozoru.

Složeniji primjeri klasa i objekata

Klasa može sadržavati više metoda, npr. u ovom slučaju sadrži tri metode.

```
Ovo je prva  metoda.
Ovo je druga metoda.
Ovo je treca metoda.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

class tekst
{
   public:
      void prva()
      {
         cout << "Ovo je prva  metoda." << endl;
      }

      void druga()
      {
         cout << "Ovo je druga metoda." << endl;
      }

      void treca()
      {
         cout << "Ovo je treca metoda." << endl;
      }
};

int main()
{
   tekst t;
   t.prva();
   t.druga();
   t.treca();
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Prva metoda koju smo nazvali prva().

Druga metoda koju smo nazvali druga().

Treća metoda koju smo nazvali treca().

Objekt formiramo jednom, a zatim pozivamo metode po logici da najprije navedemo naziv objekta, zatim stavimo točku i konačno naziv metoda. Zagrade na kraju te ; oznaka su obvezne.

Program može sadržavati više klasa, npr. u ovom slučaju klasu **tekst1** i klasu **tekst2**.

```
Zovemo metodu klase tekst1.
Ovo je klasa tekst1.

Zovemo metodu klase tekst2.
Ovo je klasa tekst2.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

class tekst1
{
   public:
      void pisi()
      {
         cout << "Ovo je klasa tekst1." << endl;
      }
};

class tekst2
{
   public:
      void pisi()
      {
         cout << "Ovo je klasa tekst2." << endl;
      }
};

int main()
{
   cout << "Zovemo metodu klase tekst1." << endl;
   tekst1 t1;
   t1.pisi();
   cout << endl;
   cout << "Zovemo metodu klase tekst2." << endl;
   tekst2 t2;
   t2.pisi();
   cout << endl;
   system("PAUSE");
   return 0;
}
```

Prva klasa koju smo nazvali **tekst1**.

Druga klasa koju smo nazvali **tekst2**.

Ovdje na temelju svake klase formiramo objekt, a zatim pozivamo metodu od svake klase odnosno operaciju koju objekt može izvršiti

Uočimo da je dozvoljeno da se metoda pod istim nazivom **pisi()** nalazi u obje klase.

```
Objekt koristimo 1. put.
Upravo koristimo objekt.

Objekt koristimo 2. put.
Upravo koristimo objekt.

Objekt koristimo 3. put.
Upravo koristimo objekt.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

class tekst
{
  public:
    void pisi()
    {
      cout << "Upravo koristimo objekt." << endl;
    }
};

int main()
{
  tekst t;
  cout << "Objekt koristimo 1. put." << endl;
  t.pisi();
  cout << endl;
  cout << "Objekt koristimo 2. put." << endl;
  t.pisi();
  cout << endl;
  cout << "Objekt koristimo 3. put." << endl;
  t.pisi();
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Jednom formirani objekt možemo proizvoljni broj puta koristiti u programu.

To je jedna od prednosti korištenja klasa i objekata.

Operaciju koja se u programu često ponavlja stavimo u klasu, u programu formiramo objekt i pozivamo određenu operaciju kad god nam zatreba, bez potrebe da je ponovo pišemo.

Sve klase koje smo do sada formirali bile su veoma jednostavne budući da nisu komunicirale s ostatkom programa na način da bi bilo moguće u klasu unijeti neke podatke ili iz klase iščitati neke podatke, npr. brojeve ili tekst.

U nastavku ovog poglavlja vidjet ćemo na koji način možemo u klase unositi ili iz klasa iščitavati podatke.

Uporaba klasa i objekata veoma je složena ideja, i kao uvijek do sada, imajmo na umu da ćemo te programske tehnike usvojiti tek kad napravimo velik broj primjera.

Komunikacija s metodama

Pogledajmo sada program koji sadrži klasu u koju je moguće unijeti podatke iz drugih dijelova programa.

```
Povrsina pravokutnika.
Povrsina je 6
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

class pravokutnik
{
  private:
    int c;

  public:
    void naslov()
    {
      cout << "Povrsina pravokutnika." << endl;
      cout << endl;
    }

    int povrsina (int a, int b)
    {
      c = a * b;
      return (c);
    }
};

int main()
{
  pravokutnik p;
  p.naslov();
  cout << "Povrsina je ";
  cout << p.povrsina(2,3);
  cout << endl << endl;
  system("PAUSE");
  return 0;
}
```

Klasa pravokutnik sadrži jednu varijablu. To je varijabla **c int** tipa, a nalazi se u **private** grupi.

Osim varijable u **private** grupi, klasa sadrži i dvije metode u **public** grupi. To su metoda **naslov()** i metoda **povrsina (int a, int b)**.

U glavnom programu formiramo objekt **p** klase **pravokutnik** i koristimo ga.

```
Povrsina pravokutnika.

Povrsina je 6
p.pov je 6


Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

class pravokutnik
{
  private:
    int c;

  public:
    int pov;
    void naslov()
    {
      cout << "Povrsina pravokutnika." << endl;
      cout << endl;
    }

    int povrsina (int a, int b)
    {
      c = a * b;
      pov = c;
      return (c);
    }
};

int main()
{
  pravokutnik p;
  p.naslov();
  cout << "Povrsina je ";
  cout << p.povrsina(2,3);
  cout << endl;

  cout << "p.pov je " << p.pov << endl;
  // cout << "p.c je " << p.c << endl;

  cout << endl << endl;
  system("PAUSE");
  return 0;
}
```

Prethodni program malo smo modificirali da bi lakše mogli shvatiti razliku između naredbe **private** i naredbe **public**.

U public području dodali smo varijablu koju smo nazvali **pov**.

Sadržaj varijable c koja je formirana u private području i u kojoj se nalazi rezultat izračuna površine spremili smo u varijablu pov koja je formirana u public području.

Ovdje vidimo da sadržaju varijable pov možemo pristupiti u glavnom programu, iako je formirana u objektu. Pristupamo joj tako da prvo napišemo naziv objekta zatim točku i na kraju varijablu, dakle u **p.pov** obliku.

Pokušamo li na isti način pristupiti varijabli c, doći će do greške. (Želimo li probati, maknimo // oznaku na početku reda.)

Već iz ovog primjera bismo mogli zaključiti u kojem smislu je **c** varijabla **private**, a kojem smislu je **pov** varijabla **public**.

Naredba **private** označava područje u kojem se nalaze varijable i metode koje se mogu koristiti unutar klase odnosno unutar objekta, ali se njima ne može pristupiti iz glavnog programa, odnosno izvan objekta. U ovom programu u private području imamo samo varijablu **c**. Mogli bismo imati i procedure.

```
class pravokutnik
{
  private:
    int c;

  public:
    int pov;
    void naslov()
    {
      cout << "Povrsina pravokutnika." << endl;
      cout << endl;
    }

    int povrsina (int a, int b)
    {
      c = a * b;
      pov = c;
      return (c);
    }
};
```

Za razliku od private područja, svemu što formiramo u **public** području možemo pristupiti i izvan klase, odnosno izvan objekta.

To ne možemo učiniti tako da samo navedemo naziv varijable ili procedure nego tako da prvo navedemo naziv objekta, zatim točku i na kraju naziv varijable ili procedure. Ako pristupamo proceduri, moramo navesti i zagrade.

Ovdje su formirane još dvije varijable, varijabla **a** i varijabla **b**. Iako bi bilo logično da tim varijablama možemo pristupiti izvan klase odnosno izvan objekta, jer su formirane u public području, to ipak nije slučaj. Pokušamo li im pristupiti izvan objekta doći će do greške.

Dakle izvana možemo pristupiti samo onim varijablama koje su u **public** području formirane samostalno kao **pov** varijabla, a ne možemo onima koje su formirane unutar neke metode.

```
private:
  int c;
```

Unutar private područja u ovom slučaju formirali smo samo varijablu c. Mogli bismo staviti i procedure, ali u tom slučaju bismo tim procedurama mogli pristupiti samo unutar klase odnosno samo unutar objekta. Način pisanja bio bi isti kao i unutar public područja.

Ovdje formiramo pov varijablu int tipa. Varijable unutar klase formiramo na potpuno jednak način kao i unutar glavnog programa.

```
public:
  int pov;

  void naslov()
  {
    cout << "Povrsina pravokutnika." << endl;
    cout << endl;
  }

  int povrsina (int a, int b)
  {
    c = a * b;
    pov = c;
    return (c);
  }
```

Ovdje formiramo metodu za ispis teksta na zaslon računala. Takav tip procedure već smo vidjeli u prethodnim primjerima.

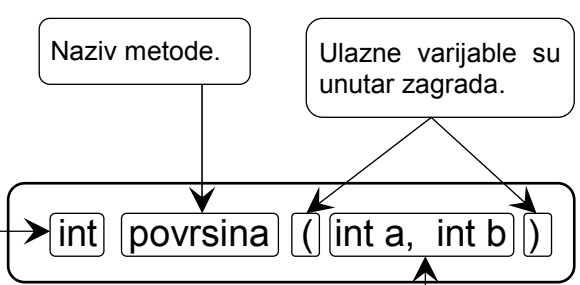
Ovdje formiramo metodu za izračun površine pravokutnika. Vidimo da joj je oblik drugačiji od oblika naslov() metode pa ćemo je pobliže pogledati.

Do sada smo formiranje metode započinjali s void. Ova metoda započinje s int.

Što to znači?

To znači da će ova metoda biti int tipa, odnosno da će rezultat njenog rada biti broj int tipa.

Naredbom:
cout << p.povrsina(2,3); ispisat ćemo rezultat rada metode povrsina na zaslon, a rezultat je broj int tipa.



Ovdje je popis varijabli koje možemo unijeti u metodu.

U ovu metodu možemo unijeti dva broja. Prvi će biti spremljen u varijablu **a**, drugi u varijablu **b**, a oba su **int** tipa.

Pogledajmo još jednom naredbu za formiranje metode koja započinje void naredbom.

Void naredba znači da metoda naslov neće vraćati nikakav rezultat rada.

Sad će netko upitati, kako ne vraća kad ispisuje tekst na zaslon.

Ispisuje, ali se ispisivanje zbiva unutar metode.

U glavnom programu moguće je napisati:

cout << p.povrsina(2,3);

ali nije moguće napisati:

cout << p.naslov();

Moguće je jedino s:

p.naslov();

pozvati metodu naslov koja onda unutar sebe ispisuje tekst na zaslon računala.

Prazne zagrade znače da u ovu metodu ništa nije moguće unijeti na način da se napiše:

p.naslov(2,3);

Moguće je unutar metode staviti cin naredbu pa nešto unijeti npr. preko tipkovnice, ali nije moguće ništa unijeti na gore opisan način.


```
c = a * b;
pov = c;
```

Unutar metode mogu se koristiti varijable koje smo koristili za unos, varijabla **a** i varijabla **b**, te se mogu koristiti sve one varijable koje su formirane u public ili private području, dakle varijabla **c** i varijabla **pov**.

```
return (c);
```

Ovom naredbom određujemo koju vrijednost će procedura vraćati u glavni program, odnosno što će naredba **cout << p.povrsina(2,3);** ispisati.

```
return(c) daje 6
return(a) daje 2
return(b) daje 4
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

class racunaj
{
  private:
    int c;
  public:
    int metoda1(int a, int b)
    {
      c = a + b;
      return (c);
    }

    int metoda2(int a, int b)
    {
      c = a + b;
      return (a);
    }

    int metoda3(int a, int b)
    {
      c = a + b;
      return (b);
    }
};

int main()
{
  racunaj r;

  cout << "return(c) daje " << r.metoda1(2,4) << endl;
  cout << "return(a) daje " << r.metoda2(2,4) << endl;
  cout << "return(b) daje " << r.metoda3(2,4) << endl;

  system("PAUSE");
  return 0;
}
```

U ovom programu možemo lijepo vidjeti kako o naredbi `return` ovisi što će se naredbom `r.metoda1(2,4)` ispisati na zaslon računala.

Ako smo u metodi upotrijebili naredbu **return (c),** naredbom `r.metoda1(2,4)` ispisat će se sadržaj varijable c.

Naredbom **return(a)** ispisat će se sadržaj varijable a, a naredbom **return(b)** ispisat će se sadržaj varijable b.

```
#include <cstdlib>
#include <iostream>

using namespace std;

class racunaj
{
  private:
    int a;
    int b;
    int c;
  public:
    void unesi(void)
    {
      cout << "Unesite stranicu a." << endl;
      cin >> a;
      cout << "Unesite stranicu b." << endl;
      cin >> b;
    }
    void povrsina(void)
    {
      c = a * b;
      cout << "Povrsina je " << c << endl;
    }
};

int main()
{
  racunaj p;
  p.unesi();
  p.povrsina();
  cout << endl;
  system("PAUSE");
  return 0;
}
```

```
Unesite stranicu a.
2
Unesite stranicu b.
3
Povrsina je 6

Press any key to continue . . .
```

U ovom programu imamo dvije void metode. Metodom **unesi** unose se brojevi u varijablu a i varijablu b. To se vrši unutar metode. Nije moguće napisati naredbu:

p.povrsina(2,3);

Moguće je jedino naredbom:

p.unesi();

pozvati metodu za unos brojeva.

Metodom povrsina izračunava se površina i ispisuje rezultat.

Rezultat se ispisuje unutar metode. Nije moguća naredba:

cout << p.povrsina();

```
Unesite stranicu a.
2
Unesite stranicu b.
3
Cijena kvadratnog metra.
4
Cijena poda je 24

Press any key to continue . . .
```

U ovom programu najprije računamo površinu, a zatim dobiveni rezultat množimo sa cijenom kvadratnog metra površine da bismo dobili cijenu površine.

```
#include <cstdlib>
#include <iostream>

using namespace std;

class racunaj
{
  private:
    int a;
    int b;
    int c;

  public:
    void unesi(void)
    {
      cout << "Unesite stranicu a." << endl;
      cin >> a;
      cout << "Unesite stranicu b." << endl;
      cin >> b;
    }
    int povrsina(void)
    {
      c = a * b;
      return (c);
    }
};

int main()
{
  int c;
  int u;
  racunaj p;
  p.unesi();
  cout << "Cijena kvadratnog metra." << endl;
  cin >> c;
  u = c * p.povrsina();
  cout << "Cijena poda je " << u << endl;
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Obratimo pažnju na činjenicu da unutar glavnog programa smijemo koristiti c varijablu, iako c varijabla postoji u klasi racunaj.

Vidjeli smo da rezultat izračuna unutar metode možemo ispisati na dva načina. Ako je metoda tipa void onda se rezultat ispisuje unutar metode. Tada naredba tipa:

cout << povrsina();

nije moguća.

Ako metoda nije void tipa, nego int ili float ili char, onda je moguće napisati gore navedenu naredbu.

Postavlja se pitanje u čemu je razlika u funkcionalnosti tih dvaju načina ispisa.

Razlika je u tome da ako metoda nije void tipa, onda metodu možemo koristiti unutar izraza poput:

u = c * p.povrsina();

```
Unesite stranicu a.
3
Unesite stranicu b.
4
Cijena kvadratnog metra.
5
Povrsina je 12
Cijena je 60

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

class racunaj
{
  private:
    int a;
    int b;
    int c;
    int p;
    int u;
  public:
    void unesi(void)
    {
      cout << "Unesite stranicu a." << endl;
      cin >> a;
      cout << "Unesite stranicu b." << endl;
      cin >> b;
      cout << "Cijena kvadratnog metra." << endl;
      cin >> c;
    }
    void cijena(void)
    {
      p = a * b;
      cout << "Povrsina je " << p << endl;
      u = p * c;
      cout << "Cijena je " << u << endl;
    }
};

int main()
{
  racunaj p;
  p.unesi();
  p.cijena();

  cout << endl;
  system("PAUSE");
  return 0;
}
```

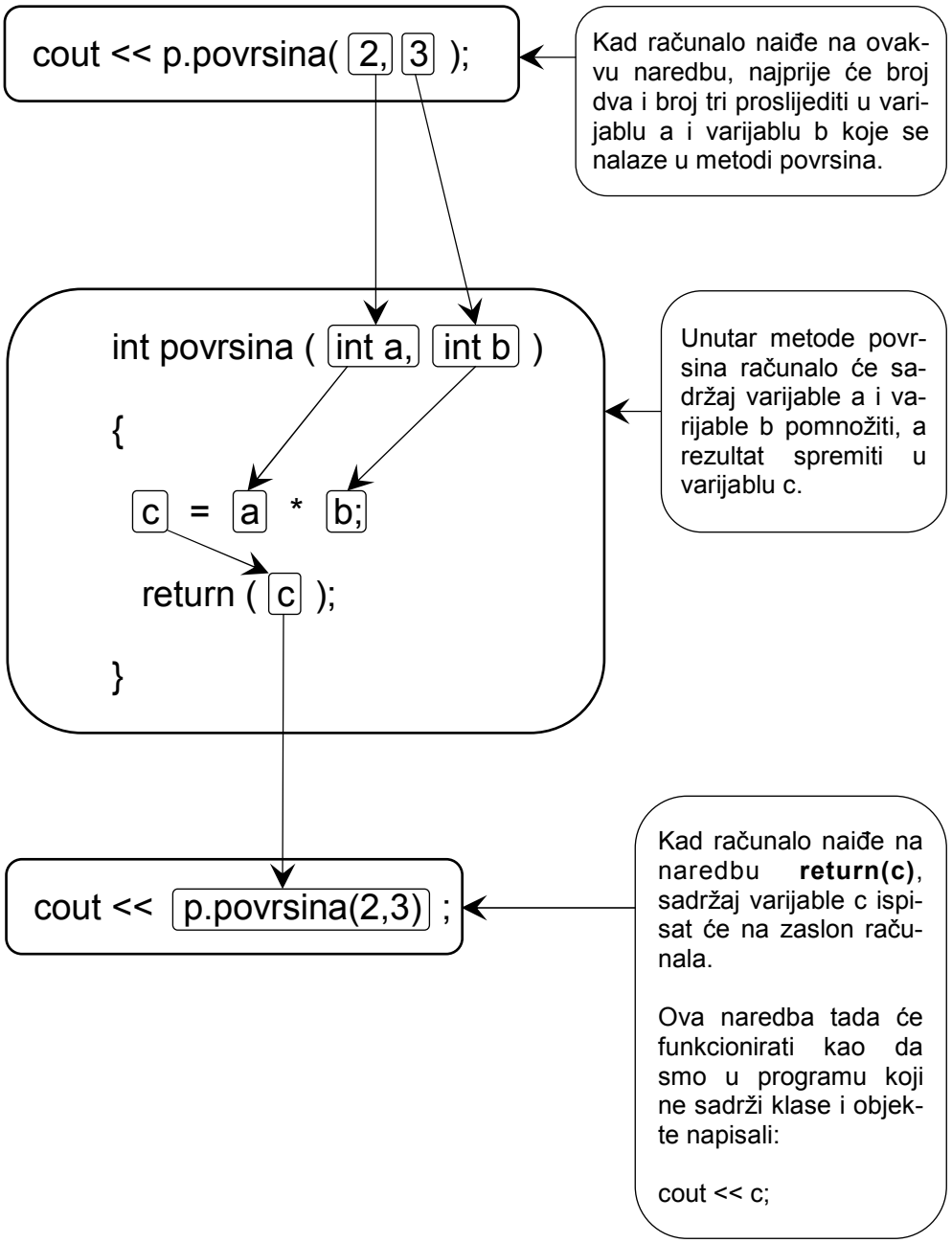
Ako za ispis koristimo void metodu pa uporaba naredbe return nije moguća, ispisom rezultata unutar metode možemo ispisati više vrijednosti.

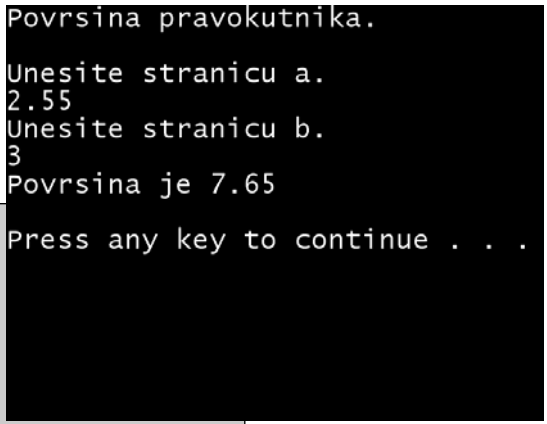
U ovom slučaju u metodi cijena mi ispisujemo površinu i cijenu.

Da je to int metoda i da rezultat vraćamo return naredbom, mogli bi napisati ili return (p) ili return(u), ali ne bismo mogli vratiti obje vrijednosti.

Ukratko, koju od metoda ispisa ćemo upotrijebiti ovisi o potrebama koje moramo zadovoljiti.

Naredba **cout << p.povrsina(2,3);** zbunjujuća je između ostalog i zato što istovremeno služi za unos broja 2 i broja 3 u metodu i za ispis rezultata množenja, odnosno rezultata rada metode.






```
#include <cstdlib>
#include <iostream>

using namespace std;

class pravokutnik
{
  private:
    float c;
  public:
    void naslov()
    {
      cout << "Povrsina pravokutnika." << endl;
      cout << endl;
    }
    float povrsina (float a, float b)
    {
      c = a * b;
      return (c);
    }
};

int main()
{
  float a1;
  float b1;
  pravokutnik p;
  p.naslov();
  cout << "Unesite stranicu a." << endl;
  cin >> a1;
  cout << "Unesite stranicu b." << endl;
  cin >> b1;
  cout << "Povrsina je ";
  cout << p.povrsina(a1,b1);
  cout << endl << endl;
  system("PAUSE");
  return 0;
}
```

Kao što smo već rekli, metode ne moraju biti int tipa. Mogu biti i nekog drugog tipa. U ovom programu imamo metodu float tipa.

Vidimo da naredbu:

p.povrsina(a1,b1)

možemo pisati i tako da unutar zagrada umjesto konkretnih brojeva stavimo varijable.

Razdvajanje deklaracije i definicije

Do sada su se metode nalazile unutar klase. U složenijim programima najčešće je deklaracija klase odvojena od definicija metoda.

U deklaraciji klase metode se samo nabroje, a definicije metoda se nalaze drugdje.

```
Povrsina pravokutnika

Povrsina je = 6

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

class pravokutnik
{
   private:
      int c;
   public:
      int povrsina (int a, int b);
};

int pravokutnik :: povrsina (int a, int b)
{
   c = a * b;
   return (c);
};

int main()
{
   cout << "Povrsina pravokutnika" << endl;
   cout << endl;
   pravokutnik p;
   cout << "Povrsina je = " << p.povrsina(2,3) << endl;
   cout << endl;
   system("PAUSE");
   return 0;
}
```

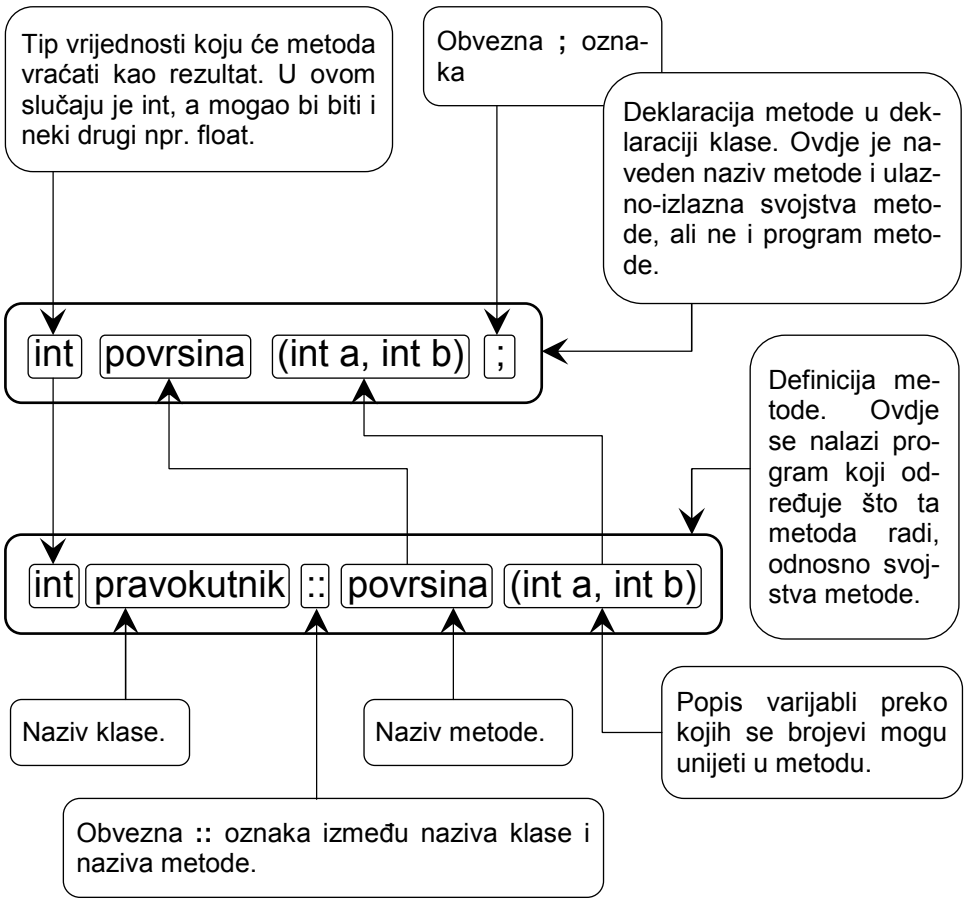
U deklaraciji klase samo su nabrojeni elementi klase i podijeljeni u private i public područje.

Ovdje se nalazi samo zaglavlje metode pravokutnik u kojem su određena ulazna i izlazna svojstva metode, ali ovdje nema sadržaja metode.

Sadržaj metode pravokutnik, odnosno dio programa kojim određujemo što će metoda pravokutnik raditi nalazi se ovdje.

Ovako napisanu klasu koristimo na isti način kao i klase koje smo do sada koristili.

Pogledajmo kako su povezane deklaracija metode i definicije metode.



U većini slučajeva do sada noviji način izrade programa bio je elegantniji od starog. U ovom slučaju to baš ne izgleda tako. Ovakav način pisanja klase očigledno je manje elegantan od prethodnog. Postoji veća vjerojatnost da će doći do greške budući da treba uskladiti deklaraciju metode i definiciju metode.

Ovaj način pisanja klasa doista je manje elegantan od načina koji smo koristili u prethodnim programima i ima ga smisla upotrebljavati samo kad želimo veći program razdijeliti u više datoteka. Ako klase pišemo na ovaj način, tada možemo u jednu datoteku staviti sve deklaracije klasa, a u druge datoteke definicije metoda. Ako se program nalazi u jednoj datoteci, bolje je definiciju metode napisati unutar deklaracije metode kao što smo to činili u prethodnim programima.

Uporaba konstruktora

```
Stranica a:
2
Stranica b:
3
Povrsina je 6.

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

class pravokutnik
{
  private:
    int a;
    int b;
  public:
    pravokutnik(void)
    {
      cout << "Stranica a:" << endl;
      cin >> a;
      cout << "Stranica b:" << endl;
      cin >> b;
    }
    int povrsina(void)
    {
      return (a * b);
    }
};

int main()
{
  pravokutnik p;
  cout << "Povrsina je " << p.povrsina() << "." << endl;
  cout << endl;
  system("PAUSE");
  return 0;
}
```

U ovom programu nalazi se nekoliko neobičnosti.

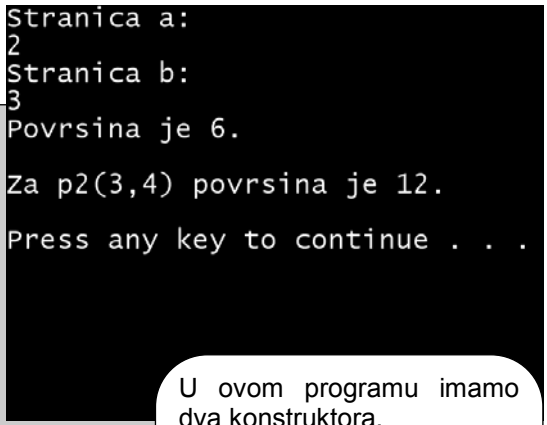
Prva neobičnost jest ova metoda. Neobično je to što ima isti naziv kao i klasa. Klasu smo nazvali pravokutnik i ova metoda zove se pravokutnik.

Druga neobičnost jest što ova metoda nema nikakvu oznaku povratne vrijednosti, nije ni void pravokutnik(), ni int pravokutnik().

Neobično je i to što je nigdje ne pozivamo.

U ovom redu formiramo objekt p klase pravokutnik, a već u sljedećem redu pozivamo metodu povrsina.

Iako nigdje u programu nije vidljivo pozivanje metode pravokutnik, njen sadržaj se očigledno izvršava. Očigledno je da prije izvođenja metode površina računalo unosi stranicu a i stranicu b. Takva metoda naziva se konstruktor, a izvršava se u trenutku formiranja objekta. Dakle, ova metoda izvršila se u trenutku kad smo naredbom pravokutnik.p formirali objekt p klase pravokutnik.




```
#include <cstdlib>
#include <iostream>

using namespace std;

class pravokutnik
{
  private:
    int a;
    int b;
  public:
    pravokutnik(void)
    {
      cout << "Stranica a:" << endl;
      cin >> a;
      cout << "Stranica b:" << endl;
      cin >> b;
    }
    pravokutnik(int a1, int b1)
    {
      a = a1;
      b = b1;
    }
    int povrsina(void)
    {
      return (a * b);
    }
};

int main()
{
  pravokutnik p1;
  cout << "Povrsina je " << p1.povrsina() << "." << endl;
  cout << endl;
  pravokutnik p2(3,4);
  cout << "Za p2(3,4) povrsina je " << p2.povrsina() << "." << endl;
  cout << endl;
  system("PAUSE");
  return 0;
}
```

U ovom programu imamo dva konstruktora.

Jednome je ulazni dio void tipa, što znači da se u njega ne može unositi vrijednost tako da napišemo:

pravokutnik p2(3,4);

Drugome je ulazni dio (int a1, int b1), što znači da se u njega može vrijednost unositi tako da napišemo:

pravokutnik p2(3,4);

Ovdje formiramo objekt klase pravokutnik bez vrijednosti u zagradi. U tom slučaju primijenit će se pravokutnik(void) konstruktor.

Objekt p2(3,4); u zagradi ima dva broja, pa će se u tom slučaju primijeniti pravokutnik(int a1, int b1) konstruktor. U tom slučaju površina će se izračunati množenjem brojeva 3 i 4, bez unosa tipkovnicom.

Nasljeđivanje

```
Unesite broj:
3
Kvadrat je 9

Unesite broj:
3
Kub je 27

Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

class kvadrat
{
  protected:
    int br;
  public:
    void unos (void)
    {
      cout << "Unesite broj:" << endl;
      cin >> br;
    }
    int izracun (void)
    {
      return (br * br);
    }
};
class kub : public kvadrat
{
  public:
    int izracun(void)
    {
      return (br * br * br);
    }
};

int main()
{
  kvadrat objekt1;
  objekt1.unos();
  cout << "Kvadrat je " << objekt1.izracun() << endl;
  cout << endl;
  kub objekt2;
  objekt2.unos();
  cout << "Kub je " << objekt2.izracun() << endl;
  cout << endl;
  system("PAUSE");
  return 0;
}
```

Klasa kvadrat napisana je na uobičajeni način.

Klasa kub napisana je na način koji do sada nismo koristili.

Naredbom:

class kub : public kvadrat

formiramo klasu kub koja nasljeđuje varijable i metode klase kvadrat. Dakle klasa kub može koristiti metodu unos, iako je ona definirana u klasi unos.

Objekt objekt2 klase kub može koristiti metodu unos koja se nalazi u klasi kvadrat kao da je njezina klasa. To je moguće jer je klasa kub naslijedila klasu kvadrat.

Metodu izracun posjeduju obje klase. U tom slučaju klasa kub koristit će svoju inačicu objekta izracun. Inačica u klasi kub prekrila je inačicu u klasi kvadrat.

Pogledajmo međusobnu povezanost bitnih dijelova programa u kojem je primijenjena tehnika nasljeđivanja klasa.

```
class kvadrat
    void unos (void)
    int izracun (void)
```

U klasi kvadrat definirane su dvije metode, metoda unos i metoda izračun.

Klasu kvadrat možemo koristiti na način koji smo koristili u dosadašnjim primjerima.

Naziv nove klase.

Naredba za nasljeđivanje klase.

Klasa koju klasa kub nasljeđuje.

```
class kub : public kvadrat
    int izracun (void)
```

Ovako deklarirana klasa kub nasljeđuje sve varijable i metode klase kvadrat.

Klasa sadrži metodu izracun, iako metodu izračun posjeduje i klasa kvadrat koju nasljeđuje.

U tom slučaju klasa kub koristit će vlastitu inačicu metode izracun.

```
kub objekt2;
```

```
objekt2.unos();
```

Ovdje koristimo metodu unos, iako je klasa kub ne posjeduje. To je moguće jer klasa kub nasljeđuje klasu kvadrat, a klasa kvadrat sadrži klasu unos.

```
objekt2.izracun()
```

Iako metodu izracun posjeduje i klasa kvadrat, objekt2 koristit će vlastitu inačicu metode izračun.

Veliki programi

Jednostavna igra u jednoj datoteci

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>
#include <ctime>

using namespace std;

class objekt
{
  private:
    int x;
    char kretanje;
  public:
    objekt()
    {
      x = 290;
    }
    void pomak(char);
    void crtaj();
};
void objekt :: pomak(char tipka)
{
  kretanje = tipka;
  if (kretanje == 75)
  {
    setcolor (WHITE);
    rectangle(x, 432, x+60, 438);
    setcolor (BLACK);
    x = x - 5;
  }
  if (kretanje == 77)
  {
    setcolor (WHITE);
    rectangle(x, 432, x+60, 438);
    setcolor (BLACK);
    x = x + 5;
  }
}
void objekt :: crtaj()
{
  rectangle(x, 432, x+60, 438);
}
```

Ovaj program nešto je veći od programa koje smo do sada pisali, pa će se zato protezati kroz nekoliko stranica.

Ovdje se nalazi deklaracija klase **objekt**. Sastoji se od dvije varijable, **x** i **kretanje**; te od dvije metode, **pomak** i **crtaj**.

Unutar deklaracije klase nalazi se konstruktor koji varijabli **x** daje početnu vrijednost **290**.

Ta varijabla određuje poziciju objekta koji pomičemo pomoću tipkovnice.

Ovdje je sadržaj metode pomak.

U metodu se unosi podatak o tome koja tipka je pritisnuta.

Metoda ispituje je li pritisnuta strelica u lijevu ili strelica u desnu stranu. Ako je neka od tih tipki pritisnuta, vrijednost varijable x koja određuje položaj objekta koji pomičemo adekvatno će se povećati ili smanjiti.

Ovdje je sadržaj metode crtaj. Ona crta objekt koji pomičemo pomoću strelica na tipkovnici.

Male programe možemo smjestiti u jednu datoteku. Za veće programe to nije praktično rješenje. Ako je veliki program u jednoj velikoj datoteci teže je pronaći grešku, teže je napraviti izmjenu programa i teže je podijeliti posao između više programera.

U ovom poglavlju vidjet ćemo kako podijeliti jedan nešto veći program u više datoteka. Najprije ćemo program napisati u jednoj velikoj datoteci, a zatim ga podijeliti u više manjih datoteka.

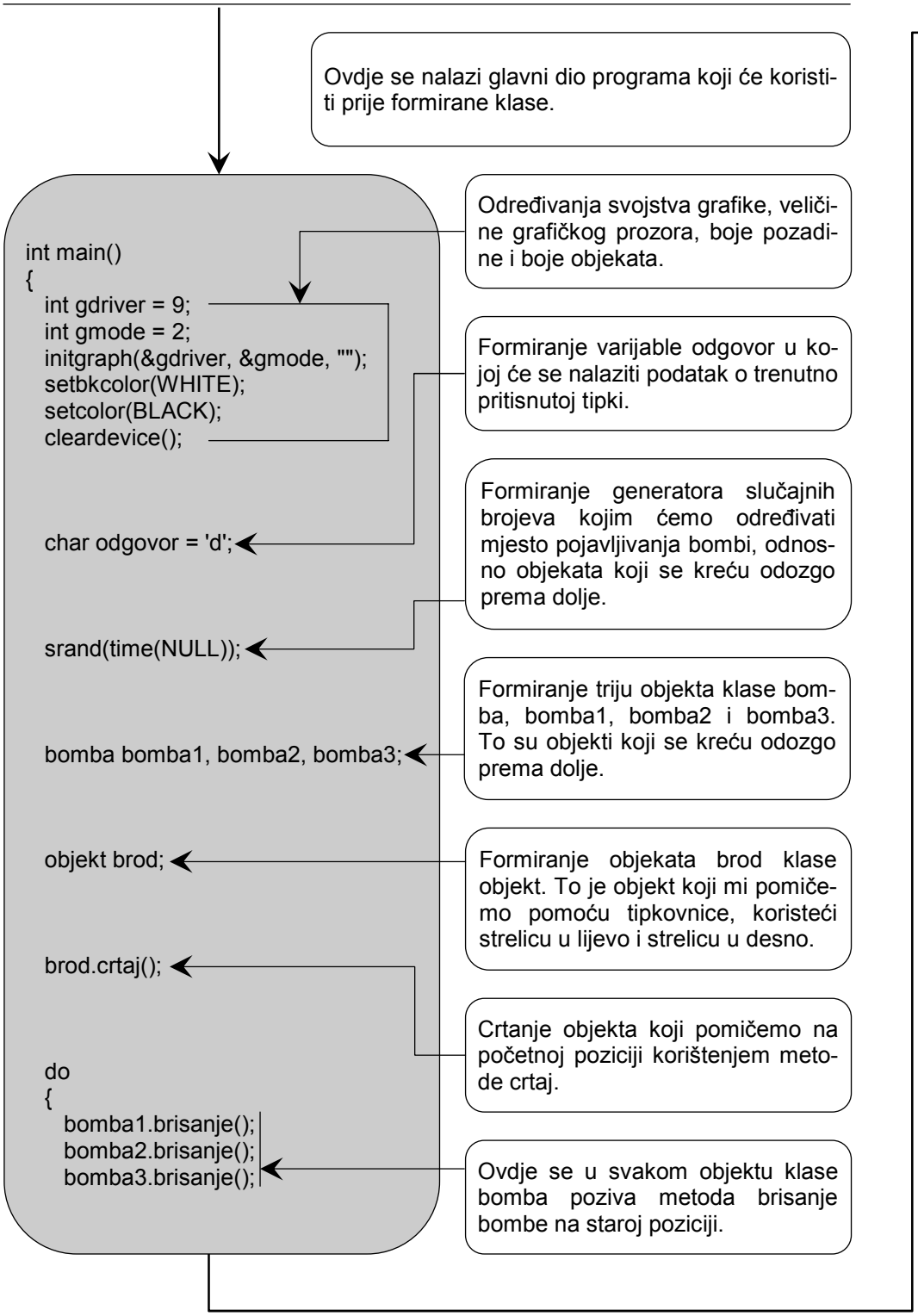
Ovdje se nalazi deklaracija klase **bomba**. Sastoji se od varijabli **x** i **y** koje određuju položaj bombe na zaslonu te metode **crtanje** koja crta bombu i od metode **brisanje** koja briše bombu da bi se dobila iluzija kretanje bombe.

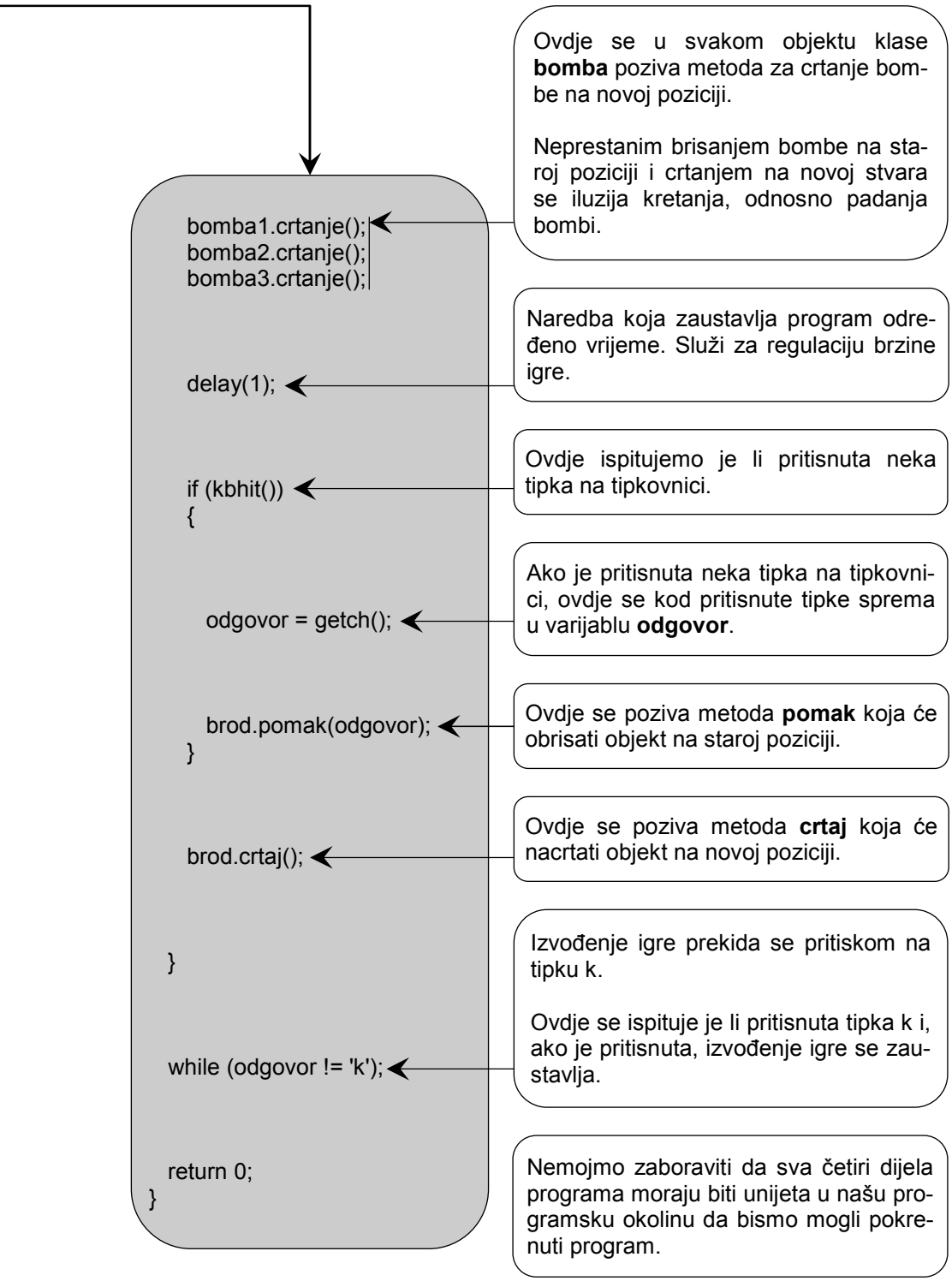
Deklaracija sadrži konstruktor koji određuje početnu poziciju bombe.

```
class bomba
{
  private:
    int x, y;
  public:
    bomba()
    {
      x = (static_cast <float> (rand()) / RAND_MAX) * 620;
      y = -((static_cast <float> (rand()) / RAND_MAX) * 480);
    }
    void crtanje();
    void brisanje();
};
void bomba :: crtanje()
{
  rectangle (x,y,(x+20),(y+5));
}
void bomba :: brisanje()
{
  setcolor(WHITE);
  rectangle (x,y,(x+20),(y+5));
  setcolor(BLACK);
  y = y + 5;
  if (y > 490)
  {
    y = -((static_cast <float> (rand()) / RAND_MAX) * 480);
    x = (static_cast <float> (rand()) / RAND_MAX) * 620;
  }
}
```

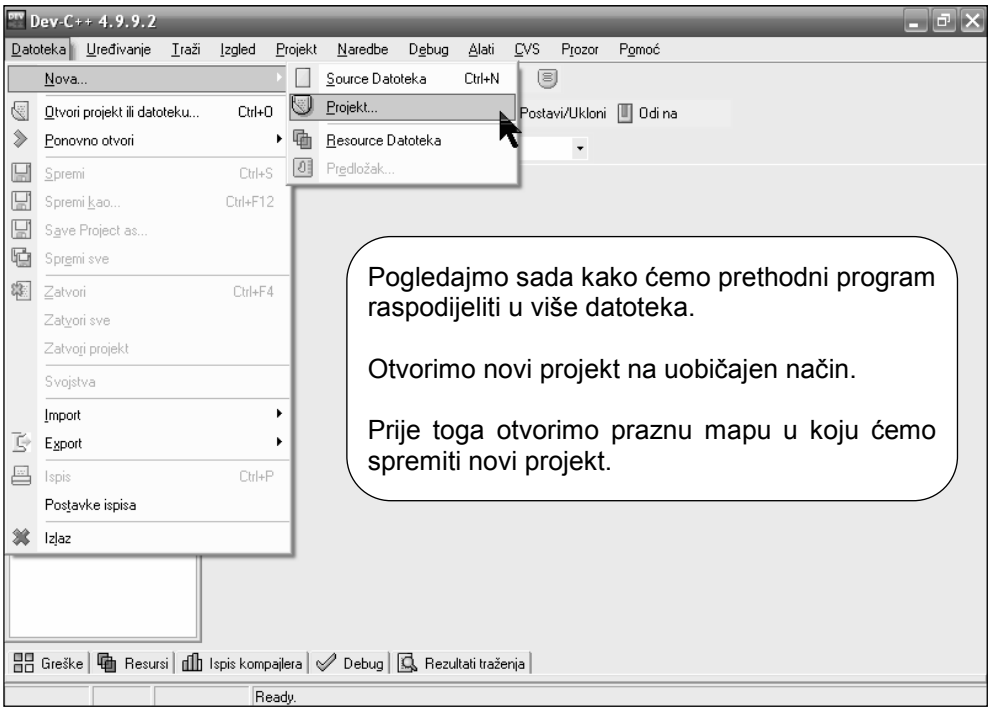
Sadržaj metode **crtanje**.

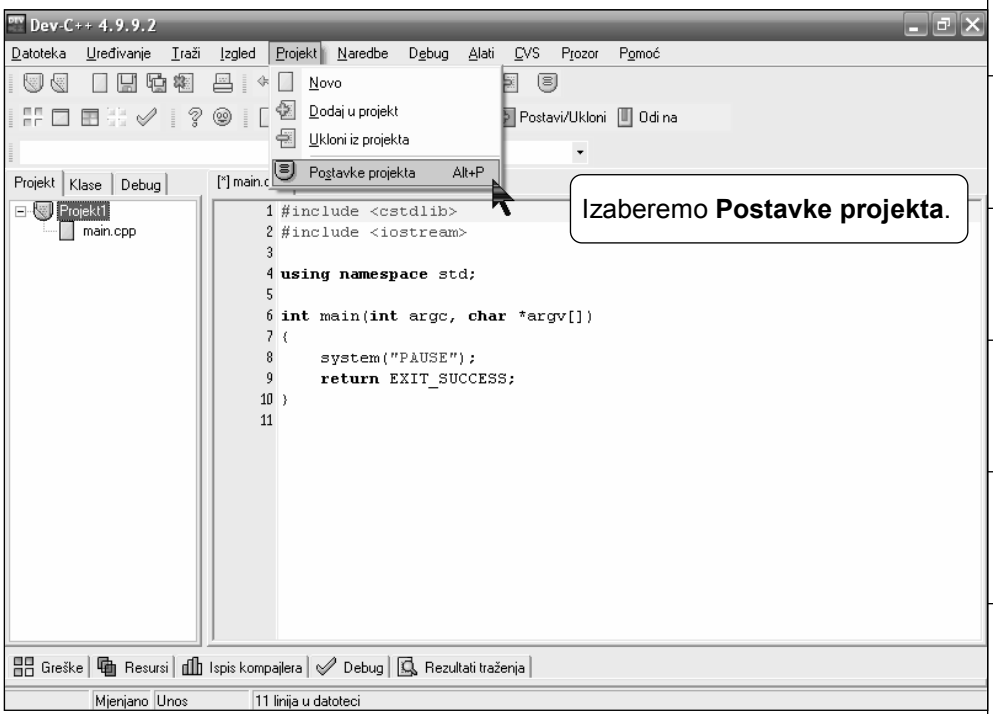
Sadržaj metode **brisanje**.

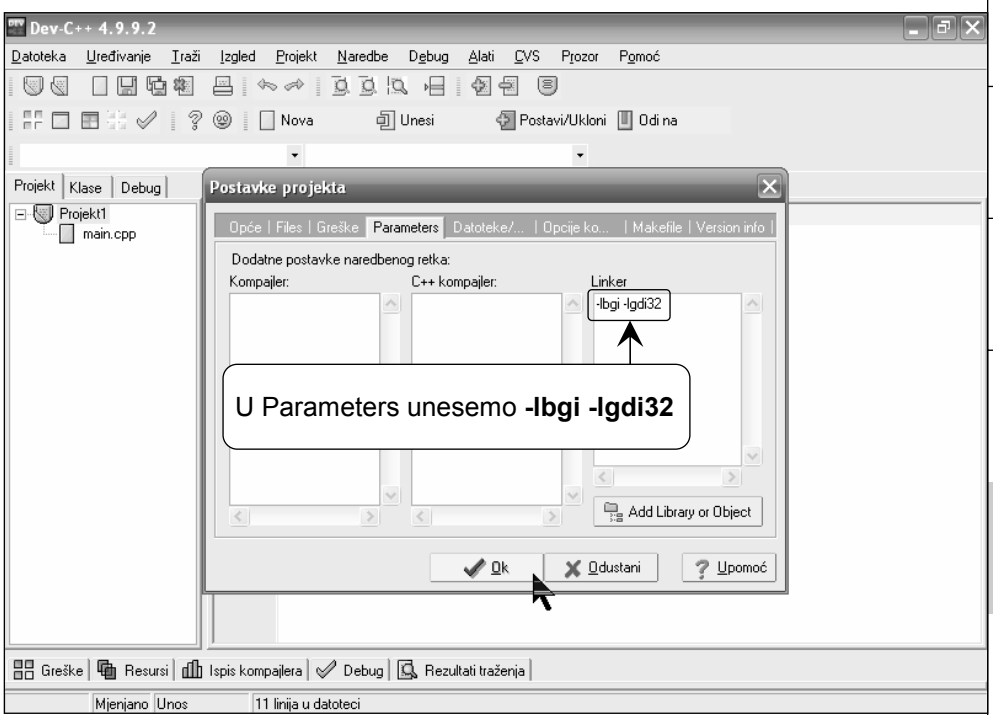

Ovdje se nalazi glavni dio programa koji će koristiti prije formirane klase.
int main()
{
int gdriver = 9;
int gmode = 2;
initgraph(&gdriver, &gmode, "");
setbkcolor(WHITE);
setcolor(BLACK);
cleardevice();
Određivanja svojstva grafike, veličine grafičkog prozora, boje pozadine i boje objekata.
char odgovor = 'd';
Formiranje varijable odgovor u kojoj će se nalaziti podatak o trenutno pritisnutoj tipki.
srand(time(NULL));
Formiranje generatora slučajnih brojeva kojim ćemo određivati mjesto pojavljivanja bombi, odnosno objekata koji se kreću odozgo prema dolje.
bomba bomba1, bomba2, bomba3;
Formiranje triju objekta klase bomba, bomba1, bomba2 i bomba3. To su objekti koji se kreću odozgo prema dolje.
objekt brod;
Formiranje objekata brod klase objekt. To je objekt koji mi pomičemo pomoću tipkovnice, koristeći strelicu u lijevo i strelicu u desno.
brod.crtaj();
Crtanje objekta koji pomičemo na početnoj poziciji korištenjem metode crtaj.
do
{
bomba1.brisanje();
bomba2.brisanje();
bomba3.brisanje();
Ovdje se u svakom objektu klase bomba poziva metoda brisanje bombe na staroj poziciji.

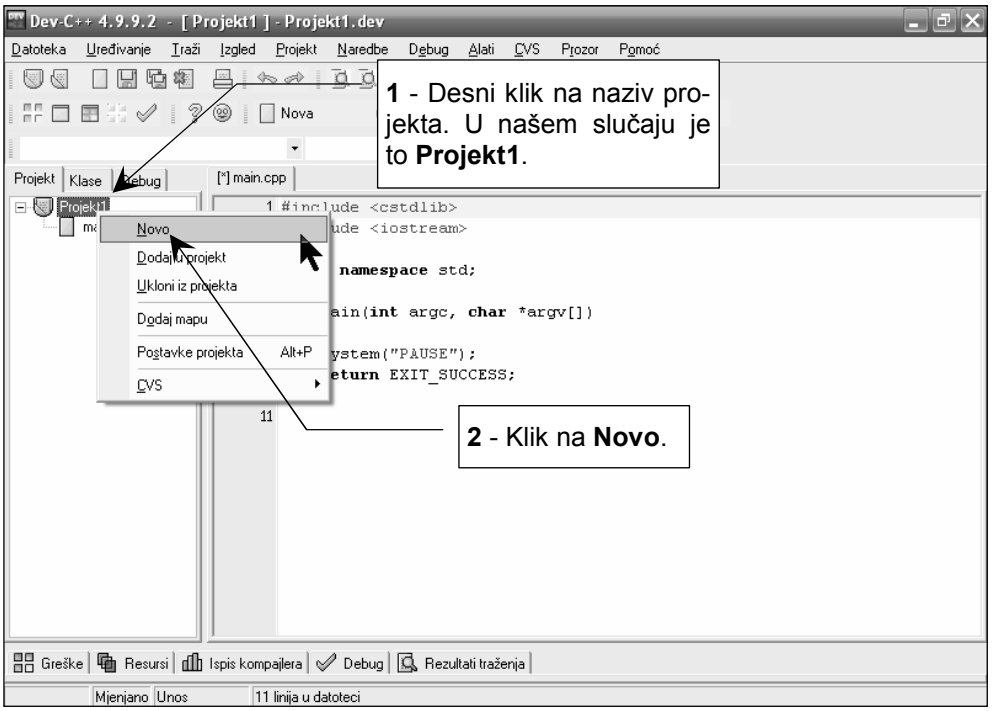

bomba1.crtanje();
bomba2.crtanje();
bomba3.crtanje();
delay(1);
if (kbhit())
{
odgovor = getch();
brod.pomak(odgovor);
}
brod.crtaj();
}
while (odgovor != 'k');
return 0;
}
Ovdje se u svakom objektu klase **bomba** poziva metoda za crtanje bombe na novoj poziciji.
Neprestanim brisanjem bombe na staroj poziciji i crtanjem na novoj stvara se iluzija kretanja, odnosno padanja bombi.
Naredba koja zaustavlja program određeno vrijeme. Služi za regulaciju brzine igre.
Ovdje ispitujemo je li pritisnuta neka tipka na tipkovnici.
Ako je pritisnuta neka tipka na tipkovnici, ovdje se kod pritisnute tipke sprema u varijablu **odgovor**.
Ovdje se poziva metoda **pomak** koja će obrisati objekt na staroj poziciji.
Ovdje se poziva metoda **crtaj** koja će nacrtati objekt na novoj poziciji.
Izvođenje igre prekida se pritiskom na tipku k.
Ovdje se ispituje je li pritisnuta tipka k i, ako je pritisnuta, izvođenje igre se zaustavlja.
Nemojmo zaboraviti da sva četiri dijela programa moraju biti unijeta u našu programsku okolinu da bismo mogli pokrenuti program.

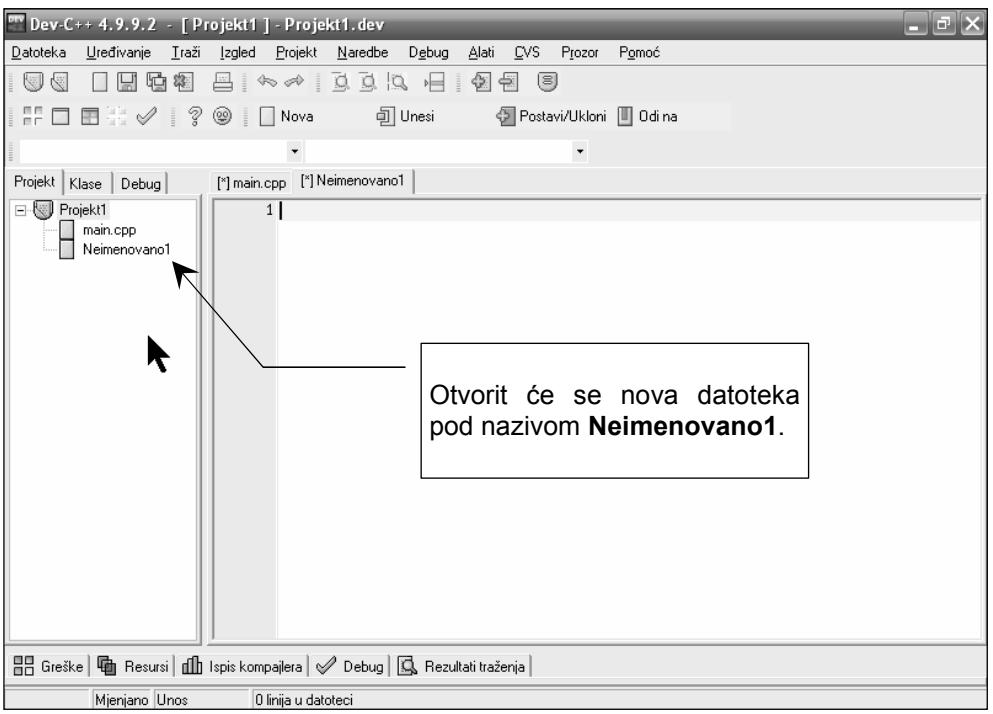
Jednostavna igra u više datoteka

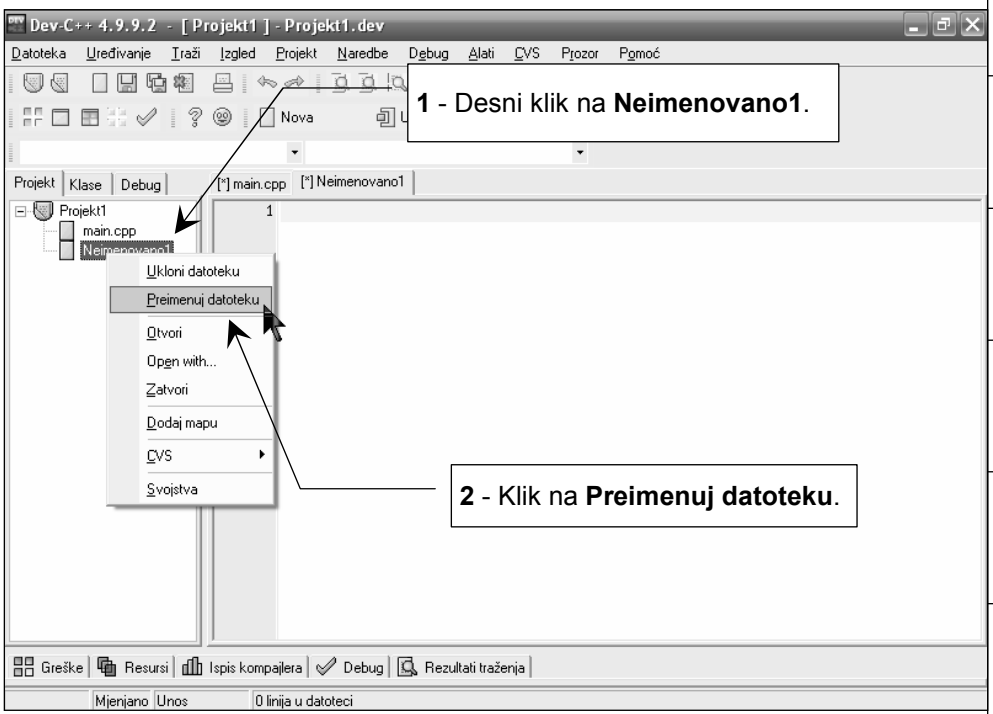


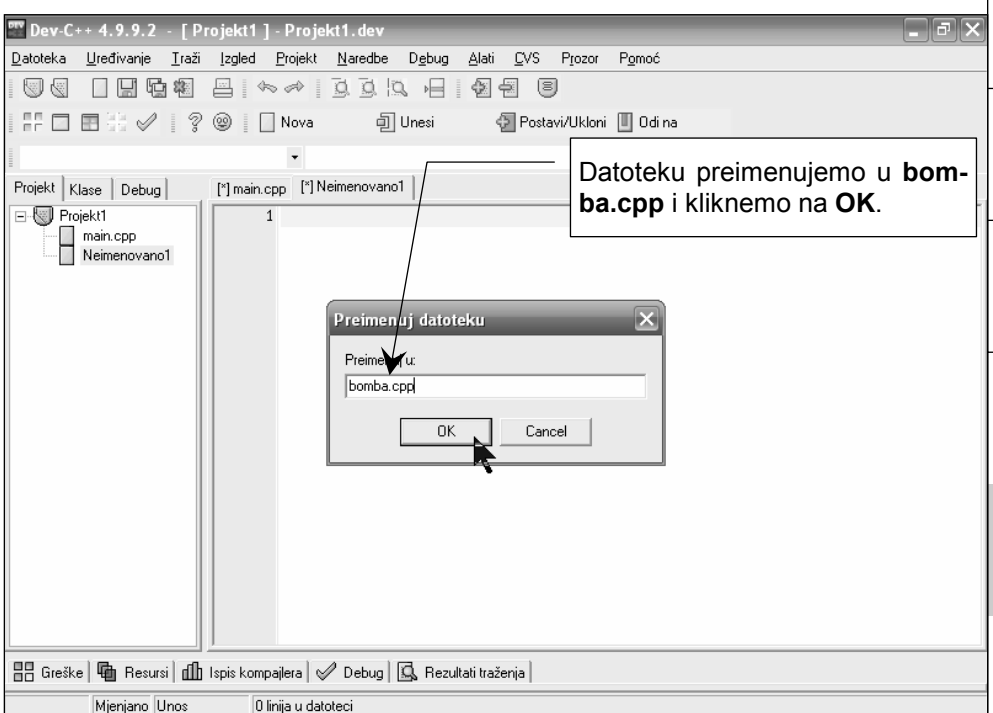

Dev-C++ 4.9.9.2
Datoteka Uređivanje Traži Izgled Projekt Naredbe Debug Alati CVS Prozor Pomoć
Novo
Dodaj u projekt
Ukloni iz projekta
Postavke projekta Alt+P
Postavi/Ukloni
Odi na
Projekt Klase Debug
Projekt1
main.cpp
#include <cstdlib>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
system("PAUSE");
return EXIT_SUCCESS;
}
Izaberemo **Postavke projekta**.
Greške Resursi Ispis kompajlera Debug Rezultati traženja
Mjenjano Unos 11 linija u datoteci

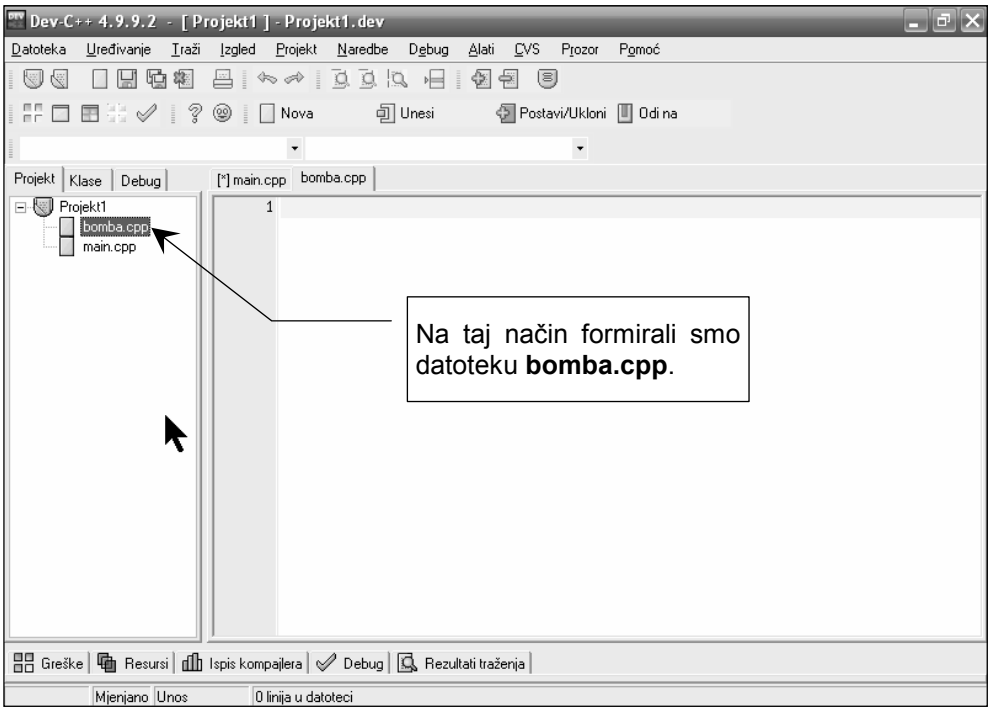

Dev-C++ 4.9.9.2
Datoteka Uređivanje Traži Izgled Projekt Naredbe Debug Alati CVS Prozor Pomoć
Nova
Unesi
Postavi/Ukloni
Odi na
Projekt Klase Debug
Projekt1
main.cpp
Postavke projekta
Opće | Files | Greške | Parameters | Datoteke/... | Opcije ko... | Makefile | Version info
Dodatne postavke naredbenog retka:
Kompajler:
C++ kompajler:
Linker
-lbgi -lgdi32
U Parameters unesemo **-lbgi -lgdi32**
Add Library or Object
Ok
Odustani
Upomoć
Greške Resursi Ispis kompajlera Debug Rezultati traženja
Mjenjano Unos 11 linija u datoteci

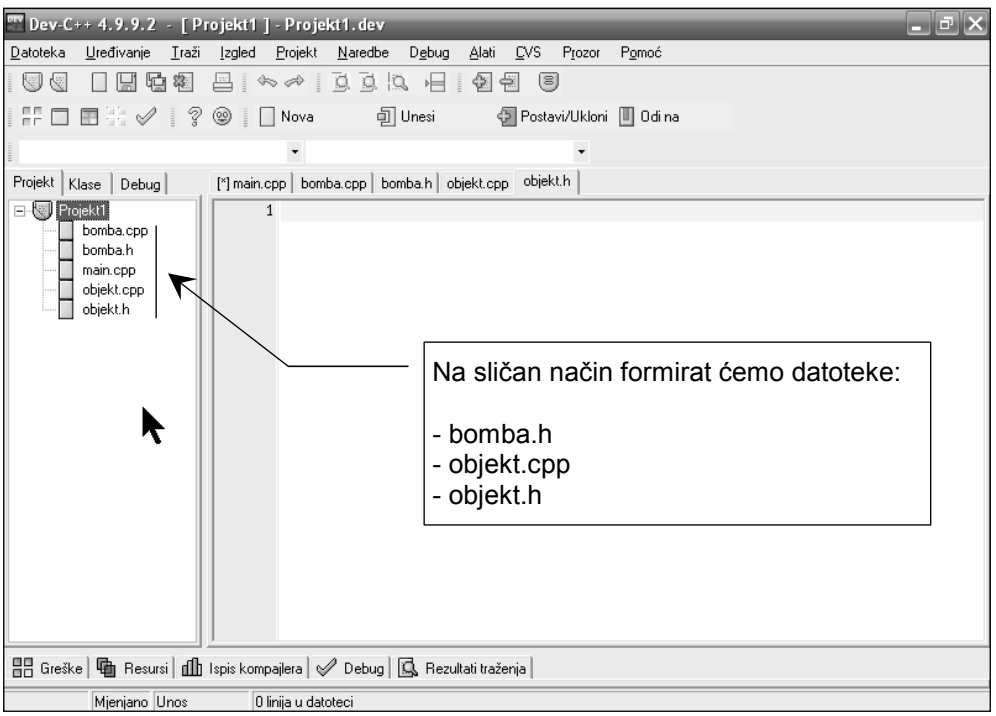

Dev-C++ 4.9.9.2 - [Projekt1] - Projekt1.dev
Datoteka Uređivanje Traži Izgled Projekt Naredbe Debug Alati CVS Prozor Pomoć
Nova
Projekt Klase Debug
[*] main.cpp
Projekt1
Novo
Dodaj u projekt
Ukloni iz projekta
Dodaj mapu
Postavke projekta Alt+P
CVS
1 #include <cstdlib>
namespace std;
ain(int argc, char *argv[])
ystem("PAUSE");
eturn EXIT_SUCCESS;
11
1 - Desni klik na naziv projekta. U našem slučaju je to Projekt1.
2 - Klik na Novo.
Greške Resursi Ispis kompajlera Debug Rezultati traženja
Mijenjano Unos
11 linija u datoteci

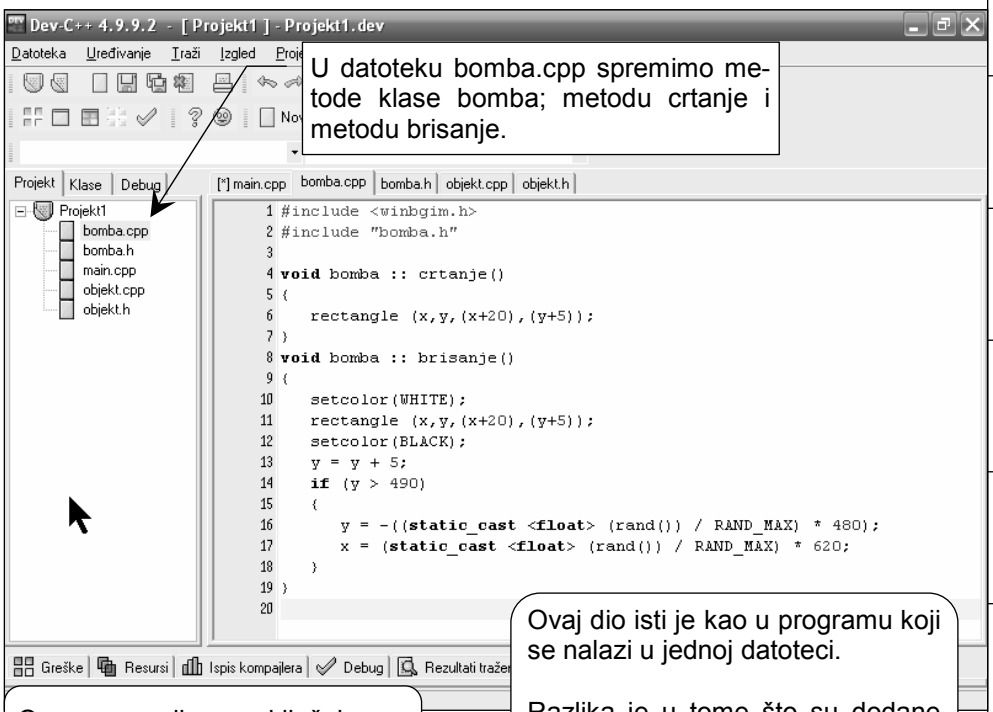

Dev-C++ 4.9.9.2 - [Projekt1] - Projekt1.dev
Datoteka Uređivanje Traži Izgled Projekt Naredbe Debug Alati CVS Prozor Pomoć
Nova
Unesi
Postavi/Ukloni
Odi na
Projekt Klase Debug
[*] main.cpp
[*] Neimenovano1
Projekt1
main.cpp
Neimenovano1
1
Otvorit će se nova datoteka pod nazivom Neimenovano1.
Greške Resursi Ispis kompajlera Debug Rezultati traženja
Mijenjano Unos
0 linija u datoteci


Dev-C++ 4.9.9.2 - [Projekt1] - Projekt1.dev
Datoteka Uređivanje Traži Izgled Projekt Naredbe Debug Alati CVS Prozor Pomoć
Nova
Projekt Klase Debug
[*] main.cpp [*] Neimenovano1
Projekt1
main.cpp
Neimenovano1
Ukloni datoteku
Preimenuj datoteku
Otvori
Open with...
Zatvori
Dodaj mapu
CVS
Svojstva
1 - Desni klik na **Neimenovano1**.
2 - Klik na **Preimenuj datoteku**.
Greške Resursi Ispis kompajlera Debug Rezultati traženja
Mjenjano Unos 0 linija u datoteci


Dev-C++ 4.9.9.2 - [Projekt1] - Projekt1.dev
Datoteka Uređivanje Traži Izgled Projekt Naredbe Debug Alati CVS Prozor Pomoć
Nova Unesi Postavi/Ukloni Odi na
Projekt Klase Debug
[*] main.cpp [*] Neimenovano1
Projekt1
main.cpp
Neimenovano1
Datoteku preimenujemo u **bomba.cpp** i kliknemo na **OK**.
Preimenuj datoteku
bomba.cpp
OK
Cancel
Greške Resursi Ispis kompajlera Debug Rezultati traženja
Mjenjano Unos 0 linija u datoteci


Dev-C++ 4.9.9.2 - [Projekt1] - Projekt1.dev
Datoteka Uređivanje Traži Izgled Projekt Naredbe Debug Alati CVS Prozor Pomoć
Nova
Unesi
Postavi/Ukloni
Odi na
Projekt
Klase
Debug
[*] main.cpp
bomba.cpp
Projekt1
bomba.cpp
main.cpp
1
Na taj način formirali smo datoteku **bomba.cpp**.
Greške
Resursi
Ispis kompajlera
Debug
Rezultati traženja
Mjenjano
Unos
0 linija u datoteci


Dev-C++ 4.9.9.2 - [Projekt1] - Projekt1.dev
Datoteka Uređivanje Traži Izgled Projekt Naredbe Debug Alati CVS Prozor Pomoć
Nova
Unesi
Postavi/Ukloni
Odi na
Projekt
Klase
Debug
[*] main.cpp
bomba.cpp
bomba.h
objekt.cpp
objekt.h
Projekt1
bomba.cpp
bomba.h
main.cpp
objekt.cpp
objekt.h
1
Na sličan način formirat ćemo datoteke:
- bomba.h
- objekt.cpp
- objekt.h
Greške
Resursi
Ispis kompajlera
Debug
Rezultati traženja
Mjenjano
Unos
0 linija u datoteci



U datoteku bomba.cpp spremimo metode klase bomba; metodu crtanje i metodu brisanje.

Ovaj dio isti je kao u programu koji se nalazi u jednoj datoteci.

Razlika je u tome što su dodane dvije include naredbe.

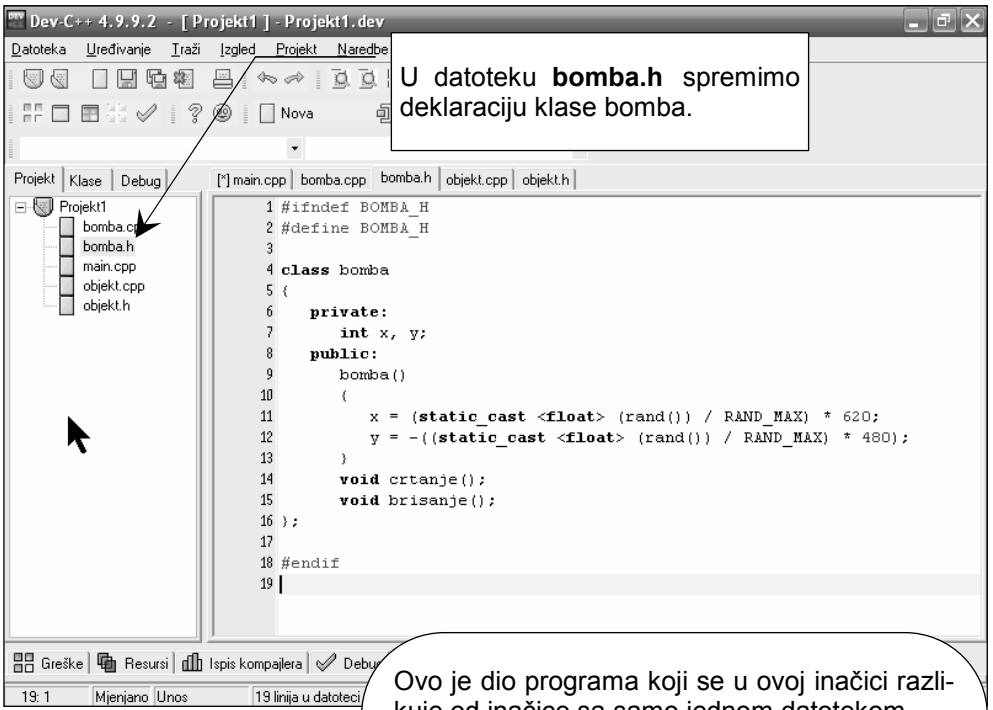
Ovom naredbom uključuje se biblioteka grafičkih naredbi.

Ovom naredbom uključuje se datoteka **bomba.h** u kojoj se nalazi deklaracija klase bomba.

Ovdje uključujmo datoteku u kojoj se nalazi deklaracija klase čije metode ovdje definiramo.

Ovaj dio programa isti je kao i u prethodnoj inačici.

```
#include <winbgim.h>
#include "bomba.h"

void bomba :: crtanje()
{
  rectangle (x,y,(x+20),(y+5));
}
void bomba :: brisanje()
{
  setcolor(WHITE);
  rectangle (x,y,(x+20),(y+5));
  setcolor(BLACK);
  y = y + 5;
  if (y > 490)
  {
    y = -((static_cast <float> (rand()) / RAND_MAX) * 480);
    x = (static_cast <float> (rand()) / RAND_MAX) * 620;
  }
}
```



U datoteku **bomba.h** spremimo deklaraciju klase bomba.

Ovo je dio programa koji se u ovoj inačici razlikuje od inačice sa samo jednom datotekom.

Ako je naziv datoteke bomba.h, ovaj dio imat će oblik:

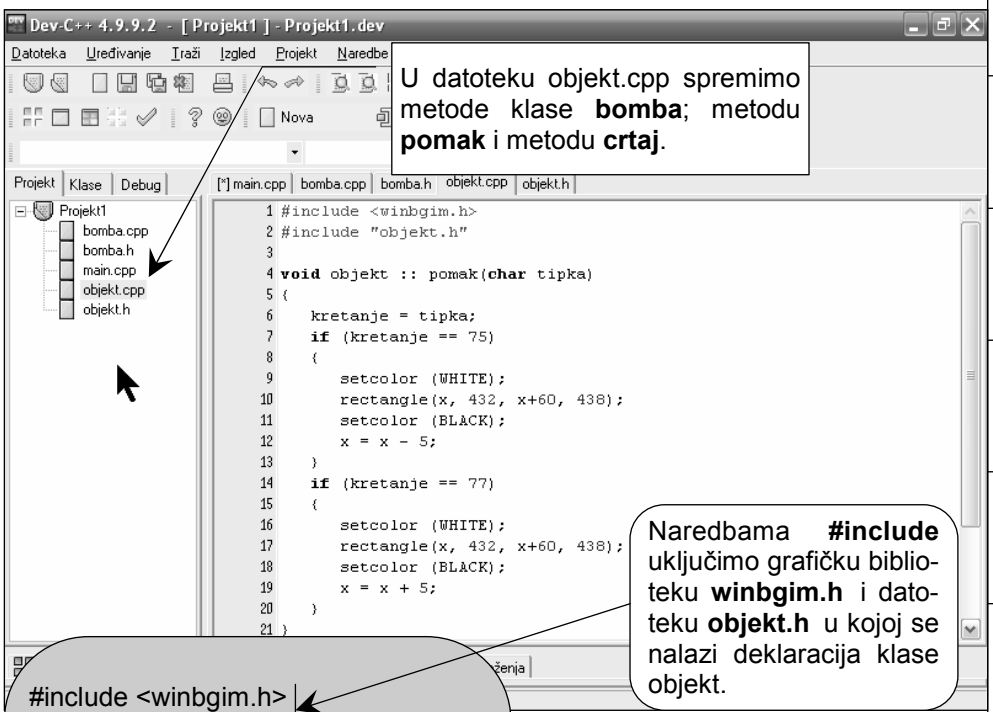
#ifndef BOMBA_H
#define BOMBA_H

Kad bi se datoteka zvala npr. datoteka.h, ovaj dio imao bi oblik:

#ifndef DATOTEKA_H
#define DATOTEKA_H

```
#ifndef BOMBA_H
#define BOMBA_H

class bomba
{
  private:
    int x, y;
  public:
    bomba()
    {
      x = (static_cast <float> (rand()) / RAND_MAX) * 620;
      y = -((static_cast <float> (rand()) / RAND_MAX) * 480);
    }
    void crtanje();
    void brisanje();
};

#endif
```

Naredbu **#endif** pisat ćemo na kraju svake datoteke koja počinje naredbom **#ifndef**.



U datoteku objekt.cpp spremimo metode klase **bomba**; metodu **pomak** i metodu **crtaj**.

Naredbama **#include** uključimo grafičku biblioteku **winbgim.h** i datoteku **objekt.h** u kojoj se nalazi deklaracija klase objekt.

```
#include <winbgim.h>
#include "objekt.h"

void objekt :: pomak(char tipka)
{
  kretanje = tipka;
  if (kretanje == 75)
  {
    setcolor (WHITE);
    rectangle(x, 432, x+60, 438);
    setcolor (BLACK);
    x = x - 5;
  }
  if (kretanje == 77)
  {
    setcolor (WHITE);
    rectangle(x, 432, x+60, 438);
    setcolor (BLACK);
    x = x + 5;
  }
}
void objekt :: crtaj()
{
  rectangle(x, 432, x+60, 438);
}
```

Kad bi se deklaracija klase nalazila u drugačije nazvanoj datoteci, npr. datoteci naziv.h, morali bismo napisati:

#include "naziv.h"

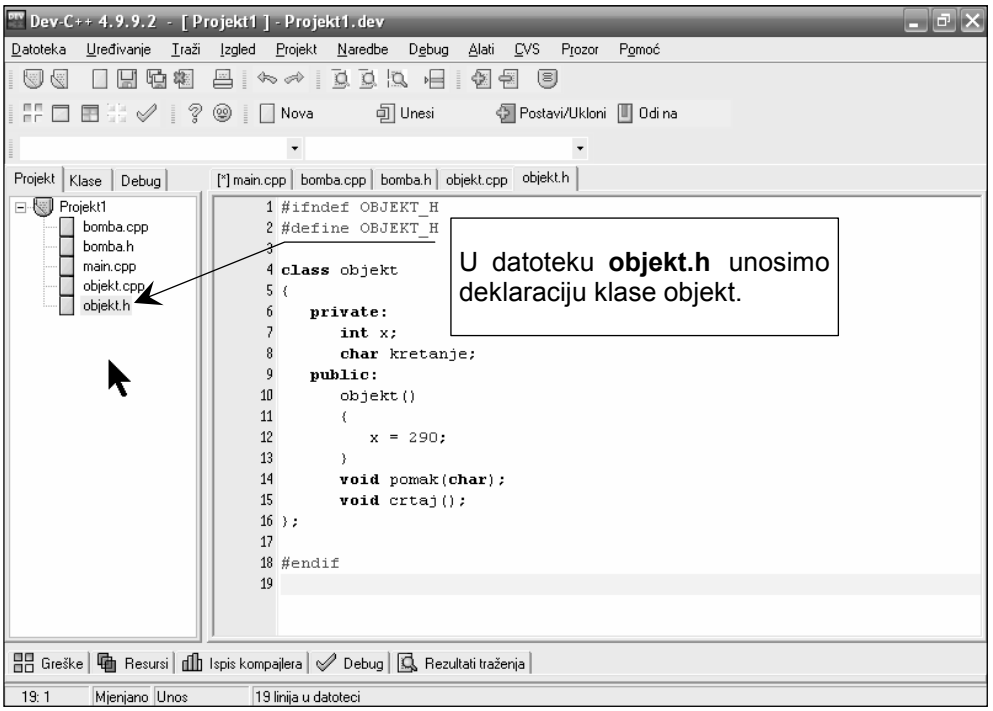
Naziv datoteka možemo birati, a nastavak **h** je obvezan. Naš program naredbom:

#include <winbgim.h>

uključuje samo grafičku biblioteku jer druge biblioteke ne koristimo. Da npr. nešto računamo morali bismo naredbom:

#include <cmath>

uključiti i matematičku biblioteku.




```
#ifndef OBJEKT_H
#define OBJEKT_H

class objekt
{
  private:
    int x;
    char kretanje;
  public:
    objekt()
    {
      x = 290;
    }
    void pomak(char);
    void crtaj();
};

#endif
```

I ova deklaracija klase započinje naredbom #ifndef i naredbom #define iza kojih se nalazi naziv datoteke u kojoj je deklaracija klase.

Budući da se deklaracija klase objekt nalazi u datoteci objekt.h iza naredbe #ifndef i naredbe #define, piše se OBJEKT_H.

Kad bi se deklaracija klase nalazila npr. u datoteci klasa.h, ovaj dio imao bi oblik:

#ifndef KLASA_H
#define KLASA_H

Naredbu **#endif** pisat ćemo na kraju svake datoteke koja počinje naredbom **#ifndef**.

Glavni program se u ovoj inačici razlikuje od inačice koja se cijela nalazi u jednoj datoteci po tome što u ovoj inačici moramo osim:

```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>
```

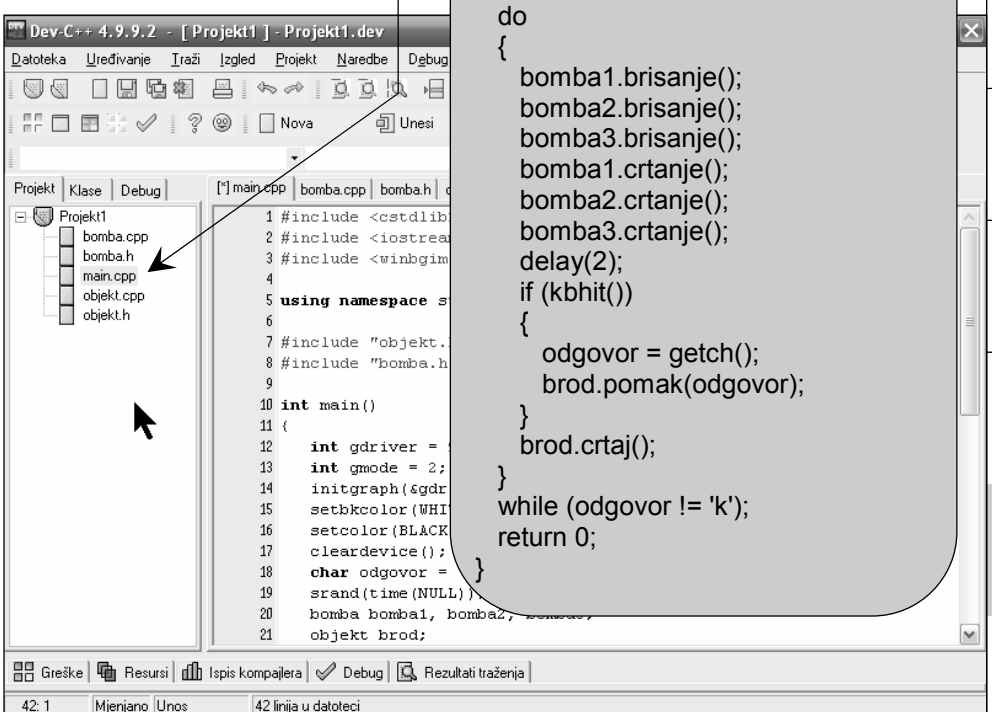
dodati i naredbe:

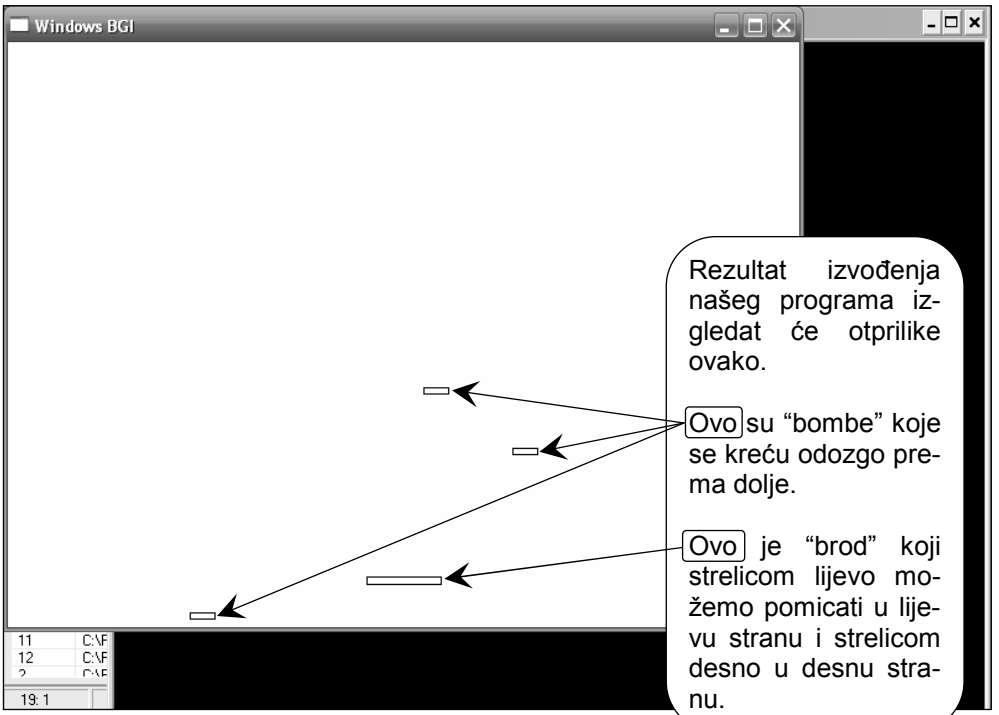
```
#include "objekt.h"
#include "bomba.h"
```

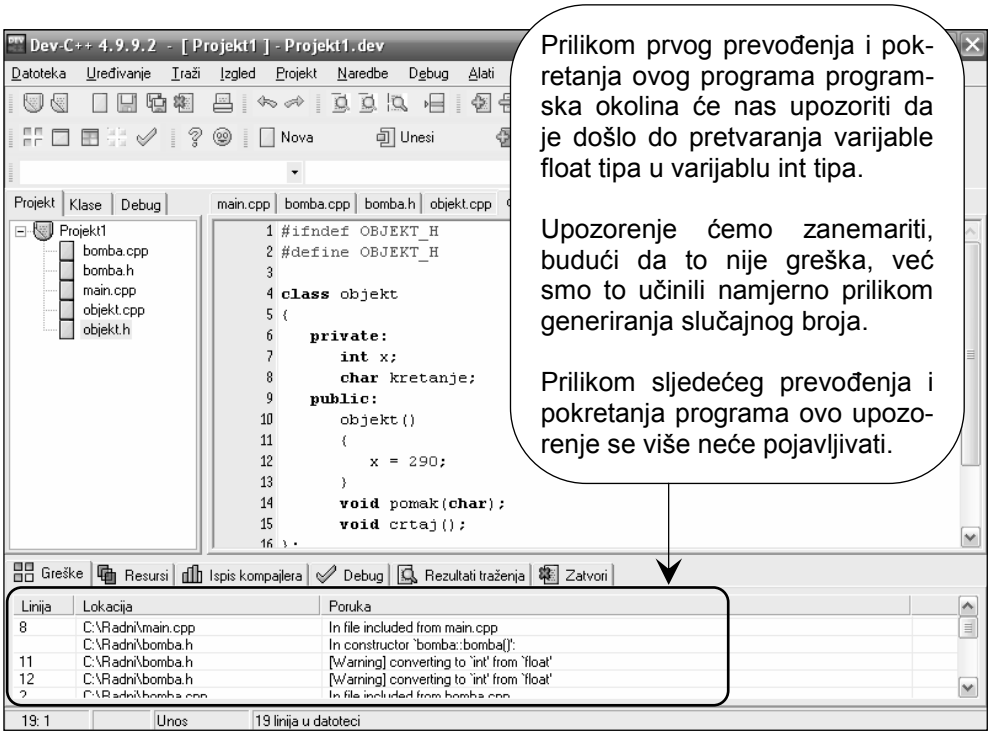
kojima se u program uključuju datoteke koje sadrže deklaracije klase **objekt** i klase **bomba**.

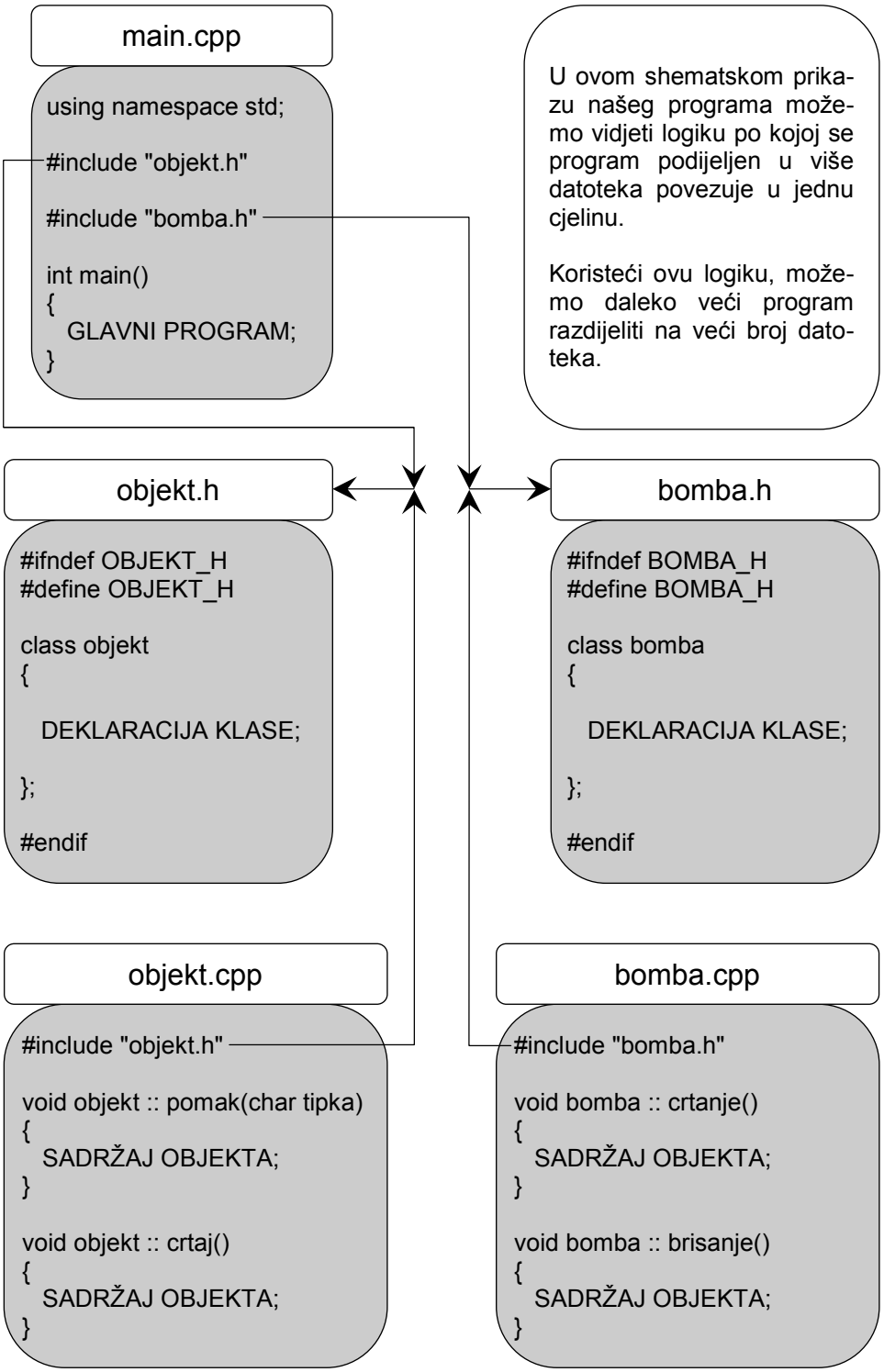
```
#include <cstdlib>
#include <iostream>
#include <winbgim.h>
#include <ctime>


using namespace std;

#include "objekt.h"
#include "bomba.h"

int main()
{
  int gdriver = 9;
  int gmode = 2;
  initgraph(&gdriver, &gmode, "");
  setbkcolor(WHITE);
  setcolor(BLACK);
  cleardevice();
  char odgovor = 'd';
  srand(time(NULL));
  bomba bomba1, bomba2, bomba3;
  objekt brod;
  brod.crtaj();
  do
  {
    bomba1.brisanje();
    bomba2.brisanje();
    bomba3.brisanje();
    bomba1.crtanje();
    bomba2.crtanje();
    bomba3.crtanje();
    delay(2);
    if (kbhit())
    {
      odgovor = getch();
      brod.pomak(odgovor);
    }
    brod.crtaj();
  }
  while (odgovor != 'k');
  return 0;
}
```

Glavni program nalazi se u datoteci mani.cpp.




Windows BGI
Rezultat izvođenja našeg programa izgledat će otprilike ovako.
Ovo su "bombe" koje se kreću odozgo prema dolje.
Ovo je "brod" koji strelicom lijevo možemo pomicati u lijevu stranu i strelicom desno u desnu stranu.


Dev-C++ 4.9.9.2 - [Projekt1] - Projekt1.dev
Prilikom prvog prevođenja i pokretanja ovog programa programska okolina će nas upozoriti da je došlo do pretvaranja varijable float tipa u varijablu int tipa.
Upozorenje ćemo zanemariti, budući da to nije greška, već smo to učinili namjerno prilikom generiranja slučajnog broja.
Prilikom sljedećeg prevođenja i pokretanja programa ovo upozorenje se više neće pojavljivati.


main.cpp
using namespace std;
#include "objekt.h"
#include "bomba.h"
int main()
{
GLAVNI PROGRAM;
}
U ovom shematskom prikazu našeg programa možemo vidjeti logiku po kojoj se program podijeljen u više datoteka povezuje u jednu cjelinu.
Koristeći ovu logiku, možemo daleko veći program razdijeliti na veći broj datoteka.
objekt.h
#ifndef OBJEKT_H
#define OBJEKT_H
class objekt
{
DEKLARACIJA KLASE;
};
#endif
bomba.h
#ifndef BOMBA_H
#define BOMBA_H
class bomba
{
DEKLARACIJA KLASE;
};
#endif
objekt.cpp
#include "objekt.h"
void objekt :: pomak(char tipka)
{
SADRŽAJ OBJEKTA;
}
void objekt :: crtaj()
{
SADRŽAJ OBJEKTA;
}
bomba.cpp
#include "bomba.h"
void bomba :: crtanje()
{
SADRŽAJ OBJEKTA;
}
void bomba :: brisanje()
{
SADRŽAJ OBJEKTA;
}

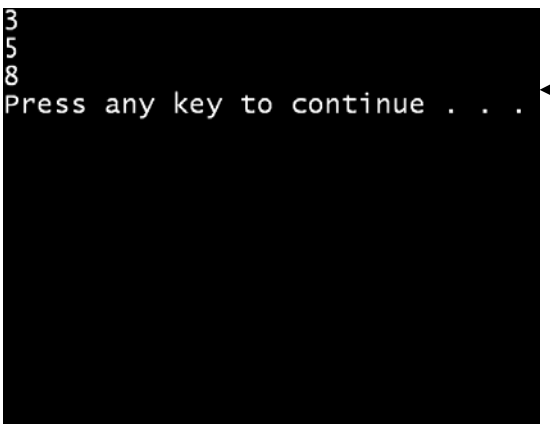
Sažimanje koda

Primjeri sažetog pisanja koda

Do sada smo, osim u nekoliko izuzetaka, programe pisali na "školski" način trudeći se da naredbe prije svega budu što razumljivije.

Mnoge naredbe mogu se, osim na način koji smo do sada vidjeli, pisati na način koji skraćuje pisanje, ponekad čak ubrzava rad programa, ali koji je početnicima možda nešto manje razumljiv.

Došavši do kraja knjige, više nismo početnici i vrijeme je da se upoznamo s nekim načinima skraćenog pisanja programa.



Obje inačice programa radit će potpuno isti pisao, zbrojit će dva broja.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
    int a;
    int b;
    int c;
    cin >> a;
    cin >> b;
    c = a + b;
    cout << c << endl;
    system("PAUSE");
    return 0;
}
```

U slučaju da formiramo više varijabli istog tipa, ne moramo svaku staviti u zasebni red, nego ih možemo formirati u jednom redu.

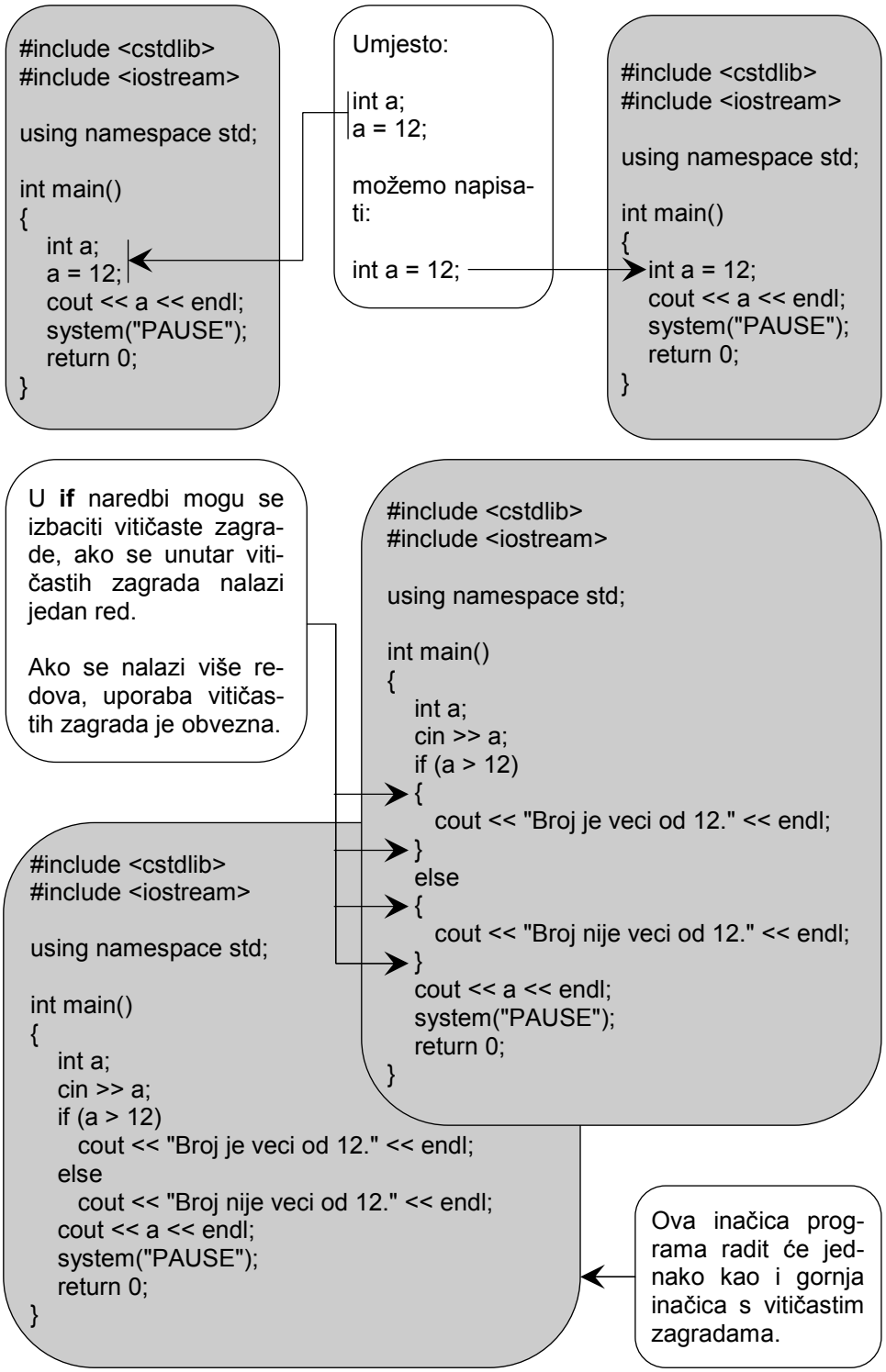
Dakle umjesto:

```
int a;
int b;
int c;
```

možemo napisati:

```
int a, b, c;
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
    int a, b, c;
    cin >> a;
    cin >> b;
    c = a + b;
    cout << c << endl;
    system("PAUSE");
    return 0;
}
```


#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
int a;
a = 12;
cout << a << endl;
system("PAUSE");
return 0;
}
Umjesto:
int a;
a = 12;
možemo napisati:
int a = 12;
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
int a = 12;
cout << a << endl;
system("PAUSE");
return 0;
}
U **if** naredbi mogu se izbaciti vitičaste zagrade, ako se unutar vitičastih zagrada nalazi jedan red.
Ako se nalazi više redova, uporaba vitičastih zagrada je obvezna.
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
int a;
cin >> a;
if (a > 12)
{
cout << "Broj je veci od 12." << endl;
}
else
{
cout << "Broj nije veci od 12." << endl;
}
cout << a << endl;
system("PAUSE");
return 0;
}
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
int a;
cin >> a;
if (a > 12)
cout << "Broj je veci od 12." << endl;
else
cout << "Broj nije veci od 12." << endl;
cout << a << endl;
system("PAUSE");
return 0;
}
Ova inačica programa radit će jednako kao i gornja inačica s vitičastim zagradama.

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
   float a, b;
   cin >> a;
   if (a < 0)
   {
      cout << "Broj je negativan." << endl;
   }
   else
   {
      b = sqrt(a);
      cout << b << endl;
   }
   system("PAUSE");
   return 0;
}
```

Ovo je program za vađenje kvadratnog korijena. Ako je broj pozitivan, izračunat će kvadratni korijen, a ako je negativan, ispisat će obavijest da je broj negativan.

U ovom dijelu programa vitičaste zagrade možemo izbaciti jer se unutar vitičastih zagrada nalazi samo jedan red.

U ovom dijelu programa vitičaste zagrade ne možemo izbaciti jer se unutar vitičastih zagrada nalazi više od jednog reda.

```
-16
Broj je negativan.
4.2039e-044
Press any key to continue . . .
```

```
#include <cstdlib>
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
   float a, b;
   cin >> a;
   if (a < 0)
   cout << "Broj je negativan." << endl;
   else
   b = sqrt(a);
   cout << b << endl;
   system("PAUSE");
   return 0;
}
```

Ako bismo ovom djelu vitičaste zagrade izbacili, ne bismo dobili obavijest o greški, ali program ne bi ispravno radio, ako bismo pokušali izračunati kvadratni korijen iz negativnog broja.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int x;
   for (x=10; x<17; x=x+1)
   {
      cout << x << endl;
   }
   system("PAUSE");
   return 0;
}
```

I u petljama se može izostaviti vitičasta zagrada ako se unutar vitičaste zagrade nalazi samo jedan red.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int x;
   for (x=10; x<17; x=x+1)
   cout << x << endl;
   system("PAUSE");
   return 0;
}
```

U ovoj inačici programa unutar **for** petlje ne koriste se vitičaste zagrade.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
   int x;
   for (x=10; x<17; x++)
   cout << x << endl;
   system("PAUSE");
   return 0;
}
```

Umjesto `x=x+1` možemo napisati:

`x++`

Pokrenemo li bilo koju od ove tri inačice programa, dobit ćemo jednak rezultat.

```
10
11
12
13
14
15
16
Press any key to continue . . .
```

Na sličan način možemo umjesto:

x = x - 1;

napisati:

x-- (Dva minusa, bez razmaka.)

Umjesto x=x+5; možemo napisati x+=5;

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
    int x;
    for (x=5; x<51; x=x+5)
    cout << x << endl;
    system("PAUSE");
    return 0;
}
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
    int x;
    for (x=5; x<51; x+=5)
    cout << x << endl;
    system("PAUSE");
    return 0;
}
```

Umjesto x=x-5 možemo napisati x-=5.

Umjesto

```
if (x != 0)
```

možemo napisati

```
if (x)
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
    int x;
    cin >> x;
    if (x != 0)
      cout << "Broje nije nula." << endl;
    else
      cout << "Broje je nula." << endl;
    system("PAUSE");
    return 0;
}
```

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main()
{
    int x;
    cin >> x;
    if (x)
      cout << "Broje nije nula." << endl;
    else
      cout << "Broje je nula." << endl;
    system("PAUSE");
    return 0;
}
```

Korisnost nekih od navedenih kraćenja prilično je dvojbena jer u nekim slučajevima zbog uštede jednog znaka narušavamo čitljivost programa.