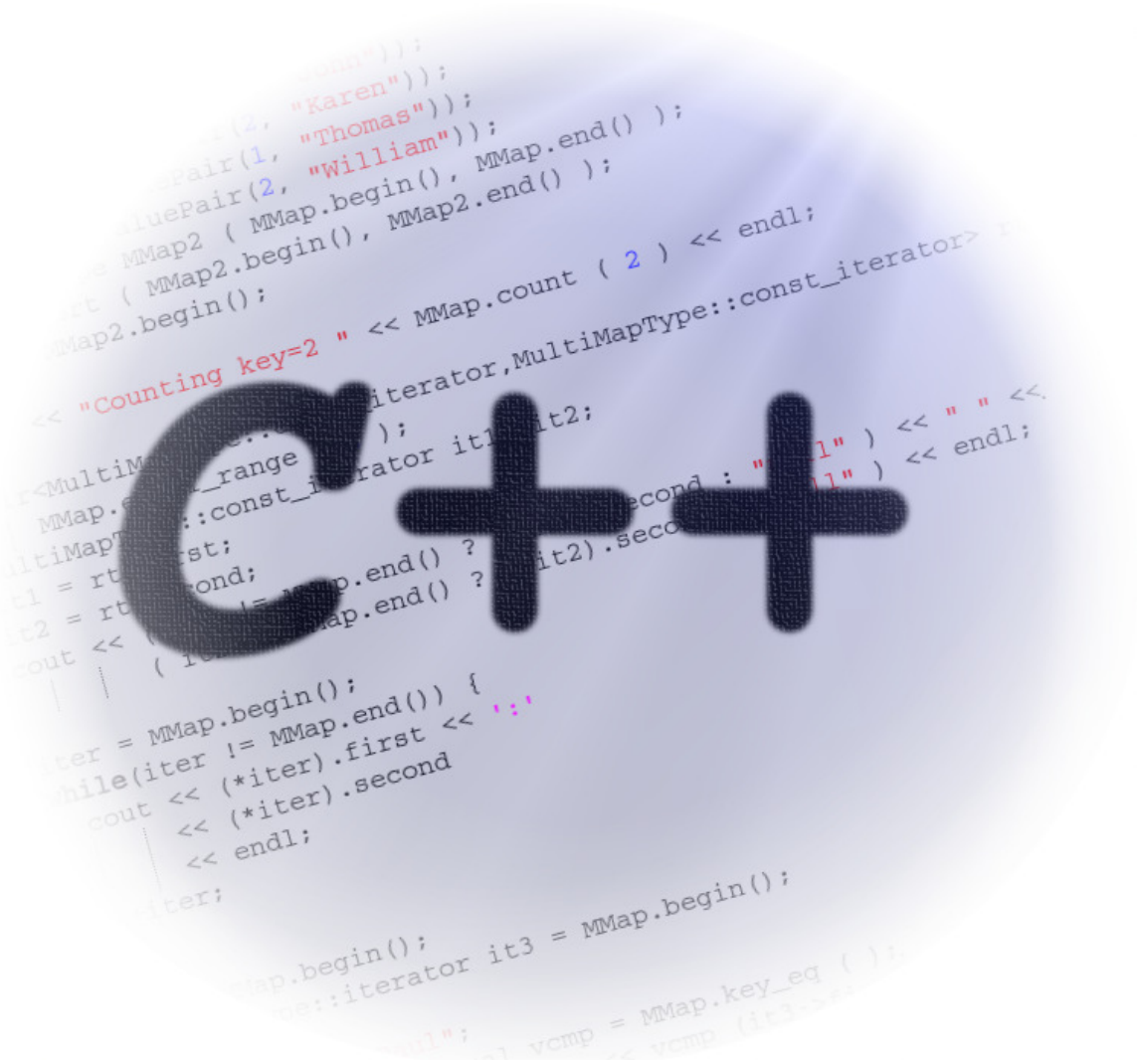


Programski jezik c++ je viši programsi jezik koji je razvijen za objektno orijentirano programiranje i bio je prvotno razvijen u Bell Labs (laboratorij telekomunikacijske tvrtke Bell) pod rukovodstvom Bjarne Stroustropa tokom 1980-tih kao proširenje programskom jeziku C, te mu je originalno ime bilo „C with classes“. Zbog velike potražnje za objektno orijentiranim jezicima te izrazitim sposobnostima, standard za programski jezik C++ ratificiran je 1998 u standardu ISO/IEC 14882.

C++ je programski jezik različitih dijalekta, kao sto jezik ima različite dijalekte. U C++ dijalekti ne postoje zbog toga što netko živi u Dalmaciji ili Slavoniji, već zato što postoje niz različiti komplailera.

Svaki od tih kompailera je malo drugačiji. Svaki bi trebao ANSI/ISO standard C++ funkcije, ali ujedno svaki kompailer će imati neke nestandardne funkcije (te funkcije su slične različitom slengu u različitim dijelovima države). Ponekad korištenje nestandardnih funkcija će stvoriti problem kada pokušate komplairati source kod sa različitim kompailerom.

```
#include<cstdlib>
#include<iostrema>
```

Ove naredbe ćemo pisati na početku svakog C++ programa. Zašto? C++ je organizovan tako da su njegove naredbe organizovane u grupe, ili kako bi programeri rekli, biblioteke i na početku programa moramo reći računaru koje biblioteke programa namjeravamo koristiti.

#include – ovo je naredba kojom na početku programa obavještavamo računar koje biblioteke naredbi želimo koristiti.

Iostream – skup naredbi koje nam omogućavaju komunikaciju sa programom, npr. ako to ne bismo uključili, ne bi bio moguć ispis teksta „Ovo je moj tekst“ na ekran.

Cstdlib – biblioteka standardnih naredbi.

```
using namespace std;
```

Postoji mnoštvo biblioteka naredbi. Moglo bi se dogoditi da se u dvije različite biblioteke nađu dvije naredbe istog naziva, a različite namjene.

Ovom naredbom obavještavamo prevodioca da ćemo koristiti standardne nazive naredbi.

```
int main ()
```

Naredba int main() označava da na ovom mjestu počinje sam program. Ono što smo pisali prije te naredbe ne smatra se programom, nego upustvima prevodiocu kako će prevesti program koji slijedi, npr. koje biblioteke naredbi će koristiti program prilikom prevođenja.

```
{
```

Ova naredba označava početak našeg programa.

```
Cout <<“Ovo je moj tekst“<<endl;
```

Ovaj tekst sadrži dvije naredbe.

Prvom naredbom naređujemo računaru da tekst koji se nalazi ispod navodnika ispiše na monitoru ekrana.

Drugom naredbom naređujemo računaru da pređe u novi red.

```
system("PAUSE");
```

Ovom naredbom naređujemo računaru da stane i čeka toliko dugo dok ne pritisnemo neku tipku na računaru.

Da bismo bolje rezumjeli ovu naredbu obrisati ćemo ovaj red u programu, a zatim ćemo prevesti i pokrenuti program klikom na Kompajliraj i pokreni.

```
return 0;
```

Ova naredba označava kraj programa. Nakon ove naredbe program se gasi.

```
}
```

Ova vitičasta zagrada označava kraj prostora unutar koga se nalazi program.

Gruba skica programa

```
#include<cstdlib>
#include<iostrema>

using namespace std;

int main ()
{
```

Ovo je početak programa. Ove naredbe pisat ćemo na početku svakog našeg programa. Kasnije ćemo ovdje dodati još neke naredbe.

Iako smo objasnili čemu služe pojedine naredbe, trenutno nije bitno da poznajemo značenje svake od njih.

Dovoljno je da znamo da te naredbe moramo napisati na početku svakog naše g programa i da ih tačno napišemo.

```
Cout <<"Ovo je moj tekst"<<endl;
```

Ovdje se nalazi glavni dio našeg programa i u ovom dijelu određujemo šta će program raditi.

U idućim vježbama gornji i donji dio uglavnom nećemo dirati, a sve bitno odvijaće se u ovom dijelu programa.

```
return 0;
system ("PAUSE");
}
```

Ovu su naredbe koje pišemo na kraju programa. Slično kao što je slučaj sa naredbama na početku programa, vrlo rijetko ćemo ih mjenjati i trenutno nije bitno da u potpunosti razumijemo značenje svake od njih.

Dovoljno je da znamo da te naredbe moramo napisati na kraju programa da bi taj program radio.

Zadatak 1. Napisati program jiji ispisuje poruku „Ovo je moj tekst“

```
#include<cstdlib>
#include<iostrema>

using namespace std;

int main ()
{
Cout <<“Ovo je moj tekst“<<endl;
return 0;
system (“PAUSE“);
}
```

Zadatak 2. Napisati program koji ispisuje zbir brojeve 2 i 3.

```
#include<cstdlib>
#include<iostrema>

using namespace std;

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

Zadatak 3. Napisati program koji ispisuje zbir za bilo koja dva unijeta broja.

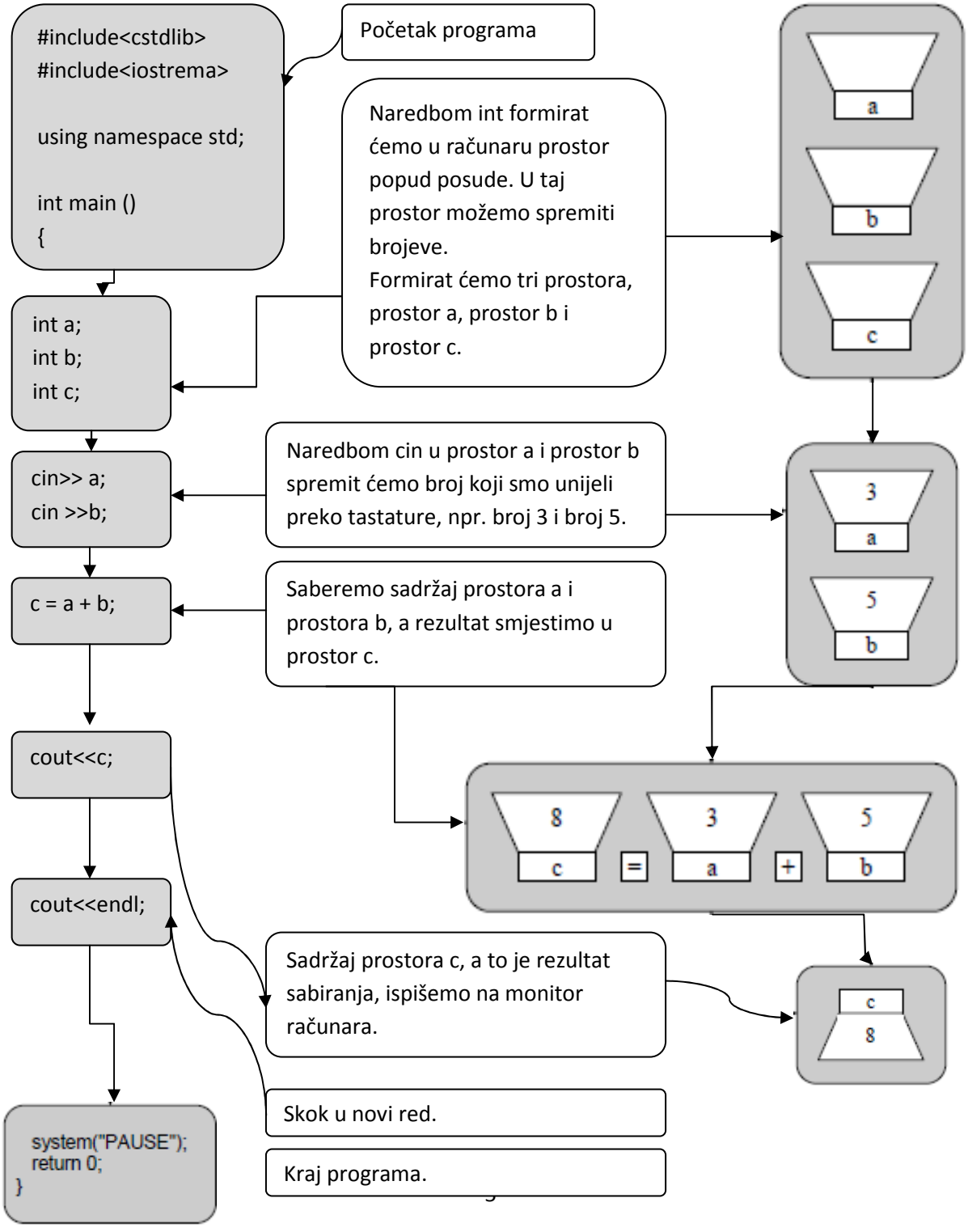
```
#include<cstdlib>
#include<iostrema>

using namespace std;

int main ()
{
Int a;
int b;
int c;
cin>> a;
cin >>b;
```

```
C=a+b;
Cout <<c;
Cout<<endl;
return 0;
system (“PAUSE“);
}
```

Analiza programa



Generisanje izvršne datoteke

Nakon što se sačuva ovaj kod, treba ga kompajlirati, tj. proizvesti izvršnu mašinsku datoteku. Ovo se izvodi korištenjem kombinacije tipki Ctrl+F9 na tastaturi, ili izborom opcije Compile u meniju Execute, ili pritiskom na ikonu u nizu alata. Nakon startovanja procesa kompajliranja, pojavljuje se prozor sa porukama koje prate proces kompajliranja. Dev-C++ daje poruku u slučaju da nađe bilo kakvu grešku u programu. U slučaju da nema grešaka, stvara se izvršna datoteka koja se naziva ime_programa.exe.

Pokratanje programa

Pokretanje programa, koji smo prethodno kompajlirali, u Dev-C++ okruženju izvodi se izborom opcije Run u meniju Execute, kombinacijom tipki Ctrl+F10 na tastaturi, ili pritiskom na ikonu. Prethodna dva procesa (kompajliranje i pokretanje) moguće je objediniti pritiskom na tipku F9 na tastaturi, izborom opcije Compile&Run u meniju Execute, ili izborom ikone . Program je moguće pokrenuti i van Dev-C++ okruženja, dvostrukim klikom na izvršnu datoteku ime_programa.exe.

Struktura programa

C++ program se sastoji od jedne ili više cjelina za prevođenje, pri čemu ove cjeline predstavljaju dio programa koji treba kompajlirati odvojeno od drugih cjelina. Tačnije, cjelina za prevođenje je rezultat primjene preliminarne faze kompilacije, koja se naziva predprocesiranje, na izvornu datoteku (source). Izvorna datoteka obično počinje sa jednom ili više (predprocesorskih) direktiva #include, pri čemu svaka od njih navodi predprocesor da kopira deklaracije entiteta (funkcija, globalnih varijabli, tipova, itd), koji su definisani u ostalim cjelinama za prevođenje. Posmatrajmo primjer iz prethodnog poglavlja:

```
#include <iostream>
using namespace std;
int main()
{
cout << “Hello, world!” << endl;
system("PAUSE");
return 0;
}
```

U liniji 1 pozvana je datoteka iostream. Prvi karakter (#) predstavlja symbol, koji daje signal predprocesoru. Svaki put kada pokrenemo kompajler, predprocesor je već pokrenut. U principu, predprocesor čita kroz izvornu datoteku i traži linije koje počinju sa ovim karakterom, tako da ih predprocesor prođe prije nego kompajler startuje sa radom. Ukratko ova linija znači: Ono što slijedi je ime datoteke. Nađi tu datoteku i odmah je pročitaj. Uglaste zagrade (<>) daju naredbu predprocesoru da nađe zadatu datoteku koja je dio standardne biblioteke (u datom primjeru to je datoteka koja sadrži definicije za ispis i upis). U slučaju kada bismo htjeli uvrstiti neku svoju datoteku, umjesto zagrada bismo koristili znake navoda. Dakle, ova linija kaže predprocesoru da nađe datoteku koja se zove iostream i da je odmah pročita. Naravno, sadržaj tražene datoteke bismo mogli upisati u izvornu datoteku bez korištenja direktive #include.

Linija 2 omogućuje pristup standardnom entitetu (namespace) koji se naziva std. Bez ove linije, linija 5 bi se morala izvršiti na drugačiji način (std::cout << ....)

Linijom 3 počinje stvarni program sa funkcijom koja se naziva main(). Svaki C++ program sadrži ovu funkciju. Funkcija predstavlja dio koda koji odrađuje određenu radnju. Inače, program može da ima proizvoljan broj funkcija, pri čemu je funkcija main() specijalna. Kada god se program starta, ona se automatski poziva. Sve funkcije počinju zagradom { i završavaju zagradom }, a sve između ovih zagrada predstavlja dio funkcije.

Glavni dio programa je linija 5, koja predstavlja neku naredbu, tj. računarski korak koji daje neku vrijednost. Kraj naredbe uvijek završava tačka-zarezom. Naredba u datom primjeru šalje string "Hello world \n" na tok cout (output stream). String je svaki niz karaktera koji se nalazi između znaka navoda. Posljednji karakter u datom stringu (\n) je karakter koji označava novi red. Stream je objekat koji izvršava ulazne i izlazne naredbe. cout je standardni izlazni stream u C++ (standardni izlazni stream je obično ekran). Simbol << je izlazni operator (usmjerivač toka) kojem je lijevi operand izlazni stream, a desni izraz, i koji uzrokuje da se ovaj posljednji posalje na prvopomenuti. Dakle, u ovom slučaju string "Hello world \n" se šalje na cout, tj. uzrokuje njegov ispis na ekranu.

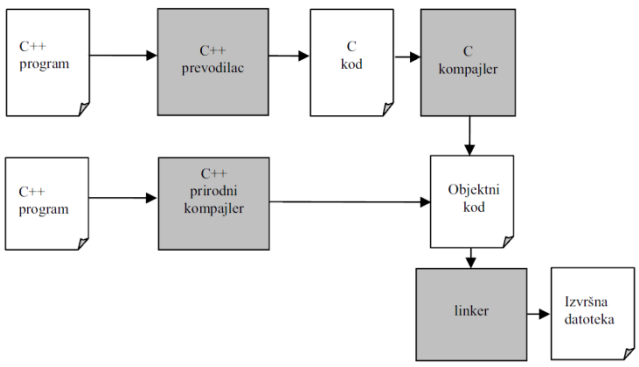
Linija 6 zaustavlja izvršenje programa, kako bismo bili u mogućnosti vidjeti rezultat njegovog rada. Bez ove linije program bi se nakon pokretanja izvršio, a konzola bi se zatvorila tako brzo da bismo imali osjećaj da program nije ništa ni uradio.

Proces kompajliranja

Kompajliranje C++ programa obuhvata nekoliko koraka, koji su većinom nevidljivi za korisnika:

- prvo, C++ predprocecsor ide kroz program i izvodi instrukcije koje su specificirane predprocesorskim direktivama (npr. #include). Rezultat ovoga je modificirani tekst programa koji više ne sadrži nikakve direktive.
- zatim, C++ kompajler prevodi programski kod. Kompajler može biti pravi C++ kompajer koji pravi osnovni (asemblerski ili mašinski) kod, ili samo prevodilac, koji kod prevodi u C jezik. U drugom slučaju, rezultujući C kod je zatim proveden kroz C kompajler kako bi se napravio osnovni kod. U oba slučaja, rezultat može biti nepotpun zbog toga što program poziva podprogramske biblioteke koje nisu definisane u samom programu.
- Na kraju, linker završava objektni kod njegovim povezivanjem sa objektnim kodom bilo kojeg modula biblioteka koji program može pozvati. Konačan rezultat je izvršna datoteka.

Slika 1. ilustruje prethodno navedene korake i za C++ prevodilac i za C++ prirodni kompajler. U praksi su sve ove komande obično izvršene jednom komandom (npr. CC), a korisnik ni ne vidi datoteke koje su se napravile u međufazama.



Varijable

Varijabla je simboličko ime za memorijsku lokaciju u koju se mogu pohraniti podaci i naknadno ih pozvati. Varijable se koriste za čuvanje vrijednosti podataka tako da se iste mogu koristiti u raznim proračunima u programu. Sve varijable imaju dvije važne osobine:

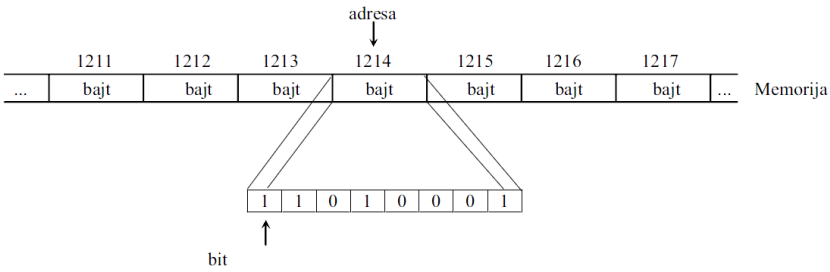
- Tip, koji se postavlja kada se varijabla definiše (npr. cijeli broj, realni broj, karakter, ...) Kada se jednom definiše, tip varijable u C++ se ne može promijeniti.
- Vrijednost, koja se može promijeniti davanjem nove vrijednosti varijabli. Vrsta vrijednosti koja se može pridružiti nekoj varijabli zavisi od njenog tipa. Na primjer, integer varijabla može da uzima samo vrijednosti cijalih brojeva (npr. -5, 13, ..)

Kada se varijabla definiše, njena vrijednost je nedefinisana sve dok joj se ne pridruži neka. Pridruživanje vrijednosti nekoj varijabli po prvi put naziva se inicijalizacija. Neophodno je obezbijediti da se svaka varijabla inicijalizira prije nego se koristi. Također je moguće da se varijabla definiše i inicializira u isto vrijeme, što je vrlo praktično. Naredni primjer pokazuje različite načine definisanja i inicijaliziranja varijabli.

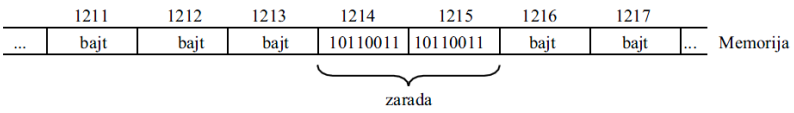
```
#include <iostream>
using namespace std;
main()
{
int a,b,c;
float x = 4.32;
int e,f,g;
char ime;
e = 4;
f = g = 12;
ime = 'C'
}
```

Memorija

Za pohranjivanje izvršnog koda kao i podataka sa kojima progam manipuliše, kompjuter ima na raspolaganju RAM memoriju (Read Access Memory). Memorija se može zamisliti kao neprekidan niz bita, od kojih svaki može da pohrani binarni broj (0 ili 1). Obično je memorija podijeljena na grupe od 8 uzastopnih bita (ovo predstavlja bajt, byte). Bajtovi su uzastopno adresirani, tako da je svaki bajt jedinstveno predstavljen svojom adresom (Slika 2.).



C++ kompajler generiše izvršni kod, koji mapira ulazne veličine na memorijske lokacije. Na primjer, definicija varijable `int zarada = 500;`
navodi kompajler da alocira nekoliko bajta kako bi predstavio varijablu `zarada`. Tačan broj bajta koji je alociran i metod koji se koristi za binarnu reprezentaciju cijelog broja zavisi od specifičnosti C++ implementacije, ali uzmimo da se radi o 2 bajta. Kompajler koristi adresu prvog bajta na koju se alocira zarada kako bi označio varijablu. Prethodna jednakost uzrokuje da se vrijednost `500` pohrani u ova dva bajta koja su alocirana (Slika 3.)



Treba napomenuti da je organizacija memorije i korištenje adresa koji se odnose na podatke veoma važno za programera, dok tačna binarna reprezentacija podataka koje on koristi to nije.

Ulazno/izlazne naredbe

Najčešći način na koji program komunicira sa vanjskim svijetom je preko jednostavnih ulazno/izlaznih (IO) operacija. C++ omogućuje dva korisna operatora za ovu svrhu: `>>` za ulaz, i `<<` za izlaz. U ranijem tekstu pokazana je upotreba operatora `<<`. Naredni primjer pokazuje upotrebu operatora `>>`.

```
#include <iostream>
using namespace std;
int main (void)
{
int radniDani = 22;
float radniSati = 7.5;
float satnica, plata;
cout << "Kolika je satnica? ";
cin >> satnica;
plata = radniDani * radniSati * satnica;
cout << "Plata = ";
cout << plata;
cout << '\n';
}
```

Linija 9 čita ulaznu vrijednost, koju unosi korisnik i kopira je u `satnica`. Ulazni operator `>>` uzimaulazni `stream` kao lijevi operand (`cin` je standardni C++ ulazni stream koji odgovara podacima unesenim pomoću tastature), a varijablu (na koju se kopira ulazni podatak) kao desni operand.

Komentari

Komentar je dio opisnog teksta koji objašnjava neke aspekte programa. Kompajler u potpunosti ignoriše komentare u programu, tako da je jedina svrha koju komentar ima, da pomognu onome koji će čitati program. C++ daje dvije mogućnosti pisanja komentara:

- Bilo šta napisano nakon `//`, pa do kraja date linije smatra se komentarom
- Bilo šta napisano između `/*` i `*/` smatra se komentarom.

```
#include <iostream>
using namespace std;
/* Ovaj program racina ukupnu platu radnika, koja se zasniva na ukupnom
broju
radnih sati i satnici. */
int main (void)
{
int radniDani = 22; // Broj radnih dana u mjesecu
float radniSati = 7.5; // Broj radnih sati u danu
float satnica = 33.50; // Satnica
float plata; // Ukupna mjesecna plata
plata = radniDani * radniSati * satnica;
```

```
cout << "Plata = " << plata << '\n';
}
```

Jasno da je da se prvi primjer može koristiti za komentar jedne i samo jedne linije (ili dijela jedne linije), dok se posljednjim može komentarisati tekts upisan u više linija.

Imena

Programski jezici koriste imena kako bi se označile različite cjeline koje čine program. Osim imena varijabli, ovdje spadaju i imena funkcija, tipova, te makroa. C++ postavlja sljedeća pravila za pravilno kreiranje imena (ili identifikatora). Ime traba da se sastoji od jednog ili više karaktera, od kojih bilo koji može biti slovo (tj, slova engleske abecede a-z i A-Z), broj (0-9) i znak "_", pri čemu na prvom mjestu ne može da bude broj. Uz to, velika i mala slova se razlikuju, tako da se, na primjer, varijable zarada i Zarada razlikuju. C++ ne postavlja nikakvo ograničenje na broj karaktera u nekom identifikatoru. Međutim, većina implementacija ima ovo ograničenje, ali je ono toliko veliko da ne predstavlja nikakav problem (npr. i do 255 karaktera). Treba imati na umu da postoje odrežene riječi u C++ koje su rezervisane, tako da identifikatori ne mogu uzimati njihova imena. Te riječi se nazivaju rezervisane ili ključne riječi i date su u tabeli.

Tabela II.2.1 Ključne (rezervisane) riječi u C++

asm	continue	float	new	signed	try
auto	default	for	operator	sizeof	typedef
break	delete	friend	private	static	union
case	do	goto	protected	struct	unsigned
catch	double	if	public	switch	virtual
char	else	inline	register	template	void
class	enum	int	return	this	volatile
const	extern	long	short	throw	while

Cijeli brojevi

Cijeli broj (integer) se može definisati pomoću tipova `short, int i long`. Jedina razlika je u tome da `int` koristi više ili barem isto bajta kao `short`, a `long` koristi više ili barem isto bajta nego `int`. Ovo zavisi od računara na kojem se radi.
Konvencija je da cjelobrojne varijable uzimaju u obzir i pozitivne i negativne brojeve (kaže se da su `signed`). Međutim, u slučaju kada se koriste kao `unsigned`, onda mogu uzeti samo pozitivne vrijednosti (sa `0`).

Realni brojevi

Realni broj se može definisati sa tipom `float i double`. `double` koristi više bajta i time omogućuje veću opseg i veću tačnost pri predstavljanju realnih brojeva.

Karakteri

Varijabla karakter se definiše tipom `char`. Ona obuhvata jedan jedini bajt koji sadrži kod datog karaktera. Ovaj broj je numerička vrijednost i zavisi od sistema kodiranja karaktera koji se koristi (to je zavisno od mašine). Najčešći sistem je ACSII (Amercan Standard Code for Information Interchange). Na primjer, karakter A ima ASCII kod 65, a karakter a 97.
Kao i interger i karakter može da bude `signed` i `unsigned`. Tako, signed karakter može da sadrži numeričke vrijednosti između -128 i 127, a `unsigned` 0 do 255. Karakteri koji imaju posebnu namjenu (ne predstavljaju karaktere koji se ispisuju) predstavljaju se pomoću *escape* sekvenci, kao npr:
'\n' – novi red
'\t' – novi tabulator
'\v' – novi vertikalni tabulator
'\b' – backspace

‘\’’ – znak navoda (apostrof)
‘\”’ – dvostruki znak navoda
‘\\’ – backslash (/)
‘\a’ – zvučni signal

Stringovi

String je uzastopni niz karaktera koji se završavaju nultim karakterom. Tako je, barem, bilo u jeziku C. Ipak, u C++ uveden je novi tip podataka koji se naziva C++ string klasa, pa je moguće na mnogo prirodniji način manipulisati varijablama sa više karaktera. Sljedeći primjer pokazuje njenu upotrebu.

```
#include <iostream>
using namespace std;
main()
{
char ch;
do {
cout << "Pritisnite K ili k za kraj, a bilo koju tipku za
nastavak \n";
cin >> ch;
if (ch != 'K' && ch != 'k')
cout << "Zelite nastaviti?\n";
else
cout << "Kraj programa";
} while (ch != 'K' && ch != 'k');
}
```

Ovaj primjer dobro funkcioniše jer završava program ako se unese "k" ili "K", odnosno program se nastavlja ako se unese bilo koji drugi karakter. Problem nastaje ako korisnik programa pritisne samo tipku ENTER. U tom slučaju objekat "cin" očekuje da se unese neka vrijednost, pa tek onda pritisne ENTER.

Operatori

Ovo poglavlje obrađuje ugrađene C++ operatore koji se koriste za stvaranje izraza, pri čemu izraz predstavlja bilo kakav proračun koji daje neku vrijednost. C++ nudi operatore za izvršavanje aritmetičkih, ralacijskih, logičkih, bitwise i uslovnih izraza. Također nudi veoma korisne “popratne efekte” (side-effect) kao što su pridruživanje, inkrement i dekrement.

Aritmetički operatori

C++ nudi pet osnovnih operatora, koji su sumirani u Tabeli II.3.1.

Tabela II.3.1 Aritmetički operatori

Operator	Ime	Primjer
+	Sabiranje	`12 + 4.9  // daje 16.9`
-	Oduzimanje	`3.98 - 4  // daje -0.02`
*	Množenje	`2 * 3.4  // daje 6.8`
/	Dijeljenje	`9 / 2.0  // daje 4.5`
%	Ostatak pri dijeljenju	`13 % 3  // daje 1`

Osim ostatka pri djeljenju (`%`) svi aritmetički operatori prihvataju miješanje cijelih i realnih brojeva. Općenito, ako su oba operanda cijeli brojevi, i rezultat je cijeli broj. Međutim, ako je jedan od operanada realan, onda je i rezulat realan (tipa `double`).
Kada su oba operanda pri dijeljenju cijeli brojevi, rezultat je također cijeli broj (tzv. Cjelobrojno dijeljenje). U tom slučaju rezultat se zaokružuje na donju vrijednost, tj.

```
 9 / 2    // daje 4, a ne 4.5!
-9 / 2    // daje -5,a ne -4.5!
```

S obzirom da neželjeno cjelobrojno dijeljenje predstavlja jednu od najčešćih greški u programiranju, neophodno je da promijenimo jedan od operanada da bude realan broj, kao npr.

```
int cijena = 100;
int volumen = 80;
double jedinicnaCijena = cijena / (double) volumen; // daje 1.25
```

Operator `%` daje ostatak pri dijeljenju dva cijela broja (oba operanda moraju biti cijeli brojevi), npr. `13%3 daje 1`

Relacijski operatori

C++ nudi 6 relacijskih operatora za računanje brojnih veličina (Tabela II.3.2)

Tabela II.3.2 Relacijski operatori

Operator	Ime	Primjer	
==	Jednakost	`5 == 5`	`// daje 1`
!=	Nejednakost	`5 != 5`	`// daje 0`
<	Manje od	`5 < 5.5`	`// daje 1`
<=	Manje ili jednako	`5 <= 5`	`// daje 1`
>	Veće od	`5 > 5.5`	`// daje 0`
>=	Veće ili jednako	`6.3 >= 5`	`// daje 1`

Treba zapamtiti da se operatori `<=` i `>=` mogu koristiti samo u tom obliku, a da `=<` i `=>` ne znače ništa u ovom kontekstu.
Operandi nekog relacijskog operatora moraju biti brojevi. No, i karakteri su ispravni operandi pošto predstavljaju brojnu vrijednost (sjetimo se ASCII tabele).
Relacijski operaotri se ne smiju koristiti ze poređenje stringova, pošto se u tom slučaju porede njihove adrese, a ne sadržaj. U tom slučaju, rezlutat je neodređen. Ipak, postoje funkcije koje mogu porediti i leksikografsku razliku dva stringa.

Logički operatori

Za kombinovanje logičkih izraza C++ nudi tri logička operatora (Tabela II.3.3). Slično relacijskim operatorima, rezultat pri korištenju logičkih operatora je `0` (`false`) ili `1` (`true`).

Tabela II.3.3 Logički operatori

Operator	Ime	Primjer	
!	Logička negacija	`!(5 == 5)`	`// daje 0`
&&	Logičko i	`5 < 6 && 6 < 6`	`// daje 1`
||	Logičko ili	`5 < 6 || 6 < 5`	`// daje 1`

Logička negacija je unarni opearator, tj. ima samo jedan operand kojem daje negativnu vrijednost.

Inkrementalni i dekrementalni operatori

Takozvani auto inkremetalni (`++`) i auto dekrementalni (`--`) operatori obezbijeđuju prigodan način za povećavanje, odnosno smanjivanje brojne varijable za `1`. Upotreba ovih operatora je sumirana u Tabeli II.3.4., pri čemu se predpostavlja da je

```
int k = 5;
```

Tabela II.3.4 Inkrement i dekrement operatori.

Operator	Ime	Primjer	
++	Inkrement (prefiks)	`++k + 10`	`// daje 16`
++	Inkrement (postfiks)	`k++ + 10`	`// daje 15`
--	Dekrement (prefiks)	`--k + 10`	`// daje 14`
--	Dekrement (postfiks)	`k-- + 10`	`// daje 15`

Kao što se vidi, oba operatora se mogu koristiti u prefiksnom ili postfiksnom obliku. Razlika je velika, jer kada se operator koristi u prefiksnom obliku prvo se primijenjuje operator, a onda se u

izrazu koristi rezultat. Kada se koristi u postfiksnom obliku, prvo se računa izraz, a onda se primijenjuje operator. Oba operatora se mogu primijeniti kako na cjelobrojne, tako i na realne brojeve, iako se ova karakteristika veoma rijetko koristi na realnim brojevima.

Operatori pridruživanja

Operator pridruživanja se koristi za pohranjivanje vrijednosti na neku memorijsku lokaciju (koja je obično pridružena nekoj varijabli). Lijevi operand operatora treba biti neka `lijeva_vrijednost`, dok desni operand može biti proizvoljni izraz. Desni operand se izračuna i pridruži lijevoj strani. Pri tome `lijeva_vrijednost` predstavlja bilo šta što zauzima neku memorijsku lokaciju na koju se može pohraniti neka veličina, može biti varijabla, te zasnovana na pointerima i referencama. Operator pridruživanja može imati mnogo varijanti, koje se dobijaju njegovim kombinovanjem sa aritmetičkim i *bitwise* operatorima. Ove varijante su date u sljedećoj tabeli.

Tabela 2.3.5 Operatori pridruživanja

Operator	Primjer	Ekivalentno sa
=	`n = 25`	
+=	`n += 25`	`n = n + 25`
-=	`n -= 25`	`n = n - 25`
*=	`n *= 25`	`n = n * 25`
/=	`n /= 25`	`n = n / 25`
%=	`n %= 25`	`n = n % 25`

Kako operator pridruživanja sam po sebi predstavlja izraz čija se vrijednost pohranjuje u lijevi operand, on se može korisititi kao desni operand za narednu operaciju pridruživanja, odnosno može se napisati:

```
int m, n, p;
m = n = p = 100; // znači: n = (m = (p = 100));
m = (n = p = 100) + 2; // znači: m = (n = (p = 100)) + 2;
```

ili

```
m = 100;
m += n = p = 10; // znači: m = m + (n = p = 10);
```

Uslovni (ternarni) operator

Uslovni operator treba tri operanda (odatle ime ternarni). On ima opštu formulu:

```
operand1 ? operand2 : operand3
```

`operand1` se izračunava, i tretira se kao logički uslov. Ako je rezultat različit od nule, tada se izračunava `operand2`. U suprotnom, izračunava se `operand3`. Na primjer:

```
int m = 1, n = 2;
int min = (m < n ? m : n); // min dobija vrijednost 1
```

Provjeriti šta je rezultat sljedeće upotrebe uslovnog operatora:

```
int min = (m < n ? m++ : n++);
```

Postoji još nekoliko vrsta operatora (npr. zarez operator, *sizeof* operator), ali o njema neće biti riječi u ovom kursu.

Naredbe

Ovo poglavlje opisuje razne oblike C++ naredbi koje služe za pisanje programa. Kao i većina ostalih programskih jezika, C++ nudi različite vrste naredbi koje se koriste u različite svrhe. Tako se deklaracione naredbe koriste za definisanje varijabli, naredbe pridruživanja za jednostavne proračune, itd. U narednom teksu biće objašnjene neke od njih.

Jednostavne i složene naredbe

Jednostavna naredba je svaka naredba koja završava tačka-zarezom. Definicije varijabli i izrazi koji završavaju sa tačka-zarezom su neki primjeri:

```
int i; // deklaraciona naredba
++i; // naredba sa popratnim efektom
double d = 10.5; // deklaraciona naredba
d + 5; // beskorisna naredba
```

Posljednji primjer pokazuje beskorisnu naredbu, jer nema nikakvih popratnih efekata. Najjednostavniji oblik naredbe je linija koja sadrži samo tačka-zarez, tzv. *null*-naredba. No, i ovakva naredba ponekad ima smisla, što će se pokazati u kasnijem tekstu. Mnogostruke naredbe se mogu kombinovati u složene naredbe kada se grupišu između velikih zagrada (`{}`), kao na primjer

```
{ int min, i = 10, j = 20;
min = (i < j ? i : j);
cout << min << '\n';
}
```

Ovakve naredbe su korisne iz dva razloga: a) omogućuju da se mnogostruka naredba postavi tamo gdje bi inače mogla da se postavi samo jedna, i b) omogućuju da se u program uvede *scope* (prostor). *Scope* predstavlja dio programa unutar kojeg varijabla ostaje definisana. Izvan *scope*-a ona to više to nije. Ovo je veoma važna osobina o kojoj će više biti riječi kada se budu objašnjavale funkcije.

Naredba if

Ponekad je poželjno da se izvrši određena naredba koja zavisi od ispunjenja nekog uslova. Upravo tu mogućnost pruža `if` naredba, čiji je opšti oblik:

```
if (izraz)
 naredba;
```

Prvo se izvršava `izraz`, i ako je rezultat različit od nule izvršava se `naredba`. U suprotnom, ništa se ne dešava.
Na primjer, ako bismo željeli provjeriti da li je pri djeljenju djelilac različit od nule, imali bismo:

```
if (djelilac != 0)
Kolicnik=djelitelj/djelilac;
```

Da bismo izvršili više naredbi koje ovisi o nekom istom uslovu, koristimo složenu naredbu, tj. Sve naredbe stavljamo između zagrada. Varijanta `if` naredbe koja omogućuje da se specificiraju dvije alternativne naredbe, jedna koja se izvršava kada je uslov ispunjen i druga kada nije, se naziva `if-else` naredba i ima oblik:

```
if (izraz)
naredba1;
else
naredba2;
```

Ovdje se najprije izvršava `izraz`, i ako je rezultat različit od nule, izvršava se `naredba1`. U suprotnom, izvršava se `naredba2`, što pokazuje i sljedeći primjer:

```
#include <iostream>
using namespace std;
main()
{
int x;
cout << "Unesite neki broj";
```

```
cin >> x;
if (x % 2 == 0)
cout << "Broj je paran" << endl;
else
cout << "Broj je neparan" << endl;
}
```

Pored prethodno navedenih varijanti, postoji i ugniježdena `if` naredba, u kojoj se javljaju više od dvije alternative. Primjer takve varijente je:

```
if (callHour > 6) {
if (duzinaPoziva <= 5)
cijena = duzinaPoziva * tarifa1;
else
cijena = 5 * tarifa + (duzinaPoziva – 5) * tarifa2;
} else
cijena = osnovnaCijena;
```

Naredba switch

`switch` naredba omogućuje izbor između više alternativa, koje su zasnovane na vrijednosti izraza. Opšti oblik `switch` naredbe je:

```
switch (izraz) {
case konstanta_1:
naredbe;
...
case konstanta_n:
naredbe;
default:
naredbe;
}
```

Prvo se računa `izraz` (koji se naziva *switch tag*), a zatim se rezultat poredi sa svakom od numeričkih konstanti (koje se nazivaju labele), po redu kako se javljaju, dok se ne poklopi sa jednom od komponenti. Nakon toga se izvršavaju naredbe koje slijede. Izvršavanje se izvodi sve dok se ne naiđe na naredbu `break` ili dok se ne izvrše sve naknadne naredbe. Posljednji slučaj (`default`) može, a i ne mora da se koristi, i pokreće se ako nijedna od prethodnih konstanti nije zadovoljena.
Klasični primjer ocijenjivanja nekog rada na osnovu osvojenih bodova dat je u daljem teksu:

```
#include <iostream>
using namespace std;
main()
{
int ocj;
cout << "Unesite ocjenu: ";
cin >> ocj;
switch (ocj)
{
case 5:
cout << "Imate 90 – 100 bodova" << endl;
break;
case 4:
cout << "Imate 80 – 89 bodova" << endl;
break;
case 3:
cout << "Imate 70 – 79 bodova" << endl;
break;
case 2:
```

```
cout << "Imate 60 - 69 bodova" << endl;
break;
default:
cout << "Imate ispod 60 bodova" << endl;
}
}
```

Naredba while

Naredba `while` (naziva se i `while` petlja) omogućuje ponavljanje neke naredbe sve dok je ispunjen neki uslov. Opšti oblik ove naredbe je:

```
while (izraz)
naredba;
```

Prvo se izračunava `izraz` (naziva se i uslov petlje). Ako je rezultat različit on nule tada se izvršava naredba (naziva se i tijelo petlje) i cijeli proces se ponavlja. U suprotnom, proces se zaustavlja.
Na primjer, ako bismo željeli izračunati zbir svih brojeva od `1` do `n`, upotreba while naredbe bi izgledala kao:

```
i = 1;
sum = 0;
while (i <= n)
sum += i++;
```

Interesantno je da nije neuobičajeno za while naredbu da ima prazno tijelo petlje, tj. null-naredbu.
Takav primjer je problem nalaženja najvećeg neparnog faktora nekog broja

```
while (n % 2 == 0 && n /= 2) ;
```

Ovdje uslov petlje izvršava sve neophodne kalkulacije, tako da nema potrebe za tijelom.

Naredba do

Naredba `do` (naziva se i `do` petlja) je slična naredbi `while`, osim što se prvo izvršava tijelo petlje, a zatim se provjerava uslov. Opšti oblik naredbe je:

```
do
      naredba;
while (izraz);
```

Prvo se izvršava naredba, a zatim provjerava `izraz`. Ako je `izraz` različit od nule cijeli proces se ponavlja. U suprotnom, petlja se zaustavlja.
`do` petlja se manje koristi nego `while` petlja. Obično se koristi kada se tijelo petlje mora izvrćiti najmanje jedanput bez obzira na ispunjenje uslova. Takav primjer je ponovljeno unošenje nekog broja i izračunavanje njegovog kvadrata dok se ne unese `0`:

```
do {
cin >> n;
cout << n * n << '\n';
} while (n != 0);
```

Za razliku od `while` petlje, `do` petlja se nikada ne koristi sa praznim tijelom prvenstveno zbog jasnoće.

Naredba for

Naredba `for` (`for` petlja) je slična naredbi `while`, ali ima dvije dodatne komponente: izraz koji se izračunava samo jednom prije svega, i izraz koji se izračunava jednom na kraju svake iteracije. Opšti oblik naredbe `for` je:

```
for (izraz1; izraz2; izraz3)
naredba;
```

Prvo se izračunava `izraz1`. Svakim prolazom proz petlju se izračunava `izraz2`. Ako je rezultat različit od nule izračunava se `izraz3`. U suprotnom petlja se zaustavlja. Oblik `while` petlje koja je ekvivalenta `do` petlji je:

```
izraz1;
while (izraz2) {
naredba;
izraz3;
}
```

`for` petlja se najćešće koristi u situacijama kada se neka promjenljiva povećava ili smanjuje za neku veličinu u svakoj iteraciji, odnosno kada je broj iteracija unaprijed poznat. Sljedeći primjer računa zbir svih brojeva od `1` do `n`:

```
sum = 0;
for (i = 1; i <= n; ++i)
sum += i;
```

Bilo koja od komponenti u petlji može biti prazna. Na primjer, ako se uklone prvi i treći izraz, onda `do` petlja liči na `while` petlju:

```
for (; i != 0;) // je ekvivalentno sa: while (i != 0)
bilo-sta; // bilo-sta;
```

Uklanjanje svih izraza u petlji daje beskonačnu petlju:

```
for (;;) // beskonačna petlja
bilo-sta;
```

Pošto petlje predstavljaju naredbe, mogu se pojaviti unutar drugih petlji (tzv. Ugniježdene petlje).

Na primjer:

```
for (int i = 1; i <= 3; ++i)
for (int j = 1; j <= 3; ++j)
cout << '(' << i << ',' << j << ")\n";
```

daje parove skupa {1,2,3}

Funkcije

Ovo poglavlje opisuje funkcije, koje definiše korisnik, kao jedan od glavnih građevinskih blokova u C++ programiranju. Funkcije obezbjeđuju prikladan način upakivanja nekog numeričkog recepta, koji se može koristiti koliko god je to puta potrebno.

Definicija funkcije

Definicija funkcije se sastoji od dva glavna dijela: zaglavlja ili interfejsa, i tijela funkcije. Interfejs (neki ga nazivaju i prototip) definiše kako se funkcija može koristiti. On se sastoji od tri dijela:

- Imena. Ovo je, u stvari, jedinstveni identifikator.
- Parametara (ili potpisa funkcije). Ovo je niz od nula ili više identifikatora nekog tipa koji se koriste za proslijeđivanje vrijednosti u i iz funkcije.

- Tipa funkcije. Ovo specificira tip vrijednosti koji funkcija vraća. Funkcija koja ne vraća nijednu vrijednost bi trebala da ima tip void.

Tijelo funkcije sadrži računske korake (naredbe) koji čine neku funkciju.
Korištenje funkcije se izvodi njenim pozivanjem. Poziv funkcije se sastoji od imena funkcije, praćenim zagradama za pozivanje `()`. Unutar ovih zagrada se pojavljuje nula ili više argumenata koji se odvajaju zarezom. Broj argumenata bi trebao odgovarati broju parametara funkcije. Svaki argument je izraz čiji tip bi trebao odgovarati tipu odgovarajućeg parametra u interfejsu funkcije. Kada se izvršava poziv funkcije, prvo se računaju argumenti i njihove rezultujuće vrijednosti se pridružuju odgovarajućim parametrima. Nakon toga se izvršava tijelo funkcije. Na kraju, funkcija vraća vrijednost (ako ista postoji) pozivu.
Sljedeći primjer ilustrativno pokazuje definiciju jednostavne funkcije koja izračunava vrijednost stepen cijelog broja na neki cijeli broj.