Osnove inženjerske informatike II.

Uvod u programiranje

Vaš prvi program

K. F. & V. B.

> The only way to learn a new programming language is by writing programs in it.
>
> Kernighan & Ritchie

> . . . and it is well known that a vital ingredient of success is not knowing that what you're attempting can't be done.
>
> T. Pratchett

Prvi program koji ćemo napisati bit će

> *na zaslon ispisati tekst* `Goodbye!`
> *preći u novi redak*

Pojmom *program* označavaju se i *izvorni kôd* (engl. *source code*) koji piše programer i *izvršni program* (engl. *executable program*) koji računalo može izvesti. Izvorni program napisan u jeziku *C++* pohranjuje se najčešće u datoteku sa sufiksima `.cc`, `.cpp` ili, ako operacijski sustav razlikuje velika i mala slova, `.C` (sufiks `.c` označava programski jezik *C*), primjerice `goodbye.cc`, `goodbye.cpp` ili `goodbye.C`. Da bi ga računalo moglo izvesti, program treba prevesti u strojni jezik.

Izvorni je kôd u jeziku *C++*:

```
#include<iostream>

using namespace std;

int main() {
  cout << "Goodbye!";
  cout << endl;
  return 0;
}
```

Nalikuje možda pomalo na crnu magiju. Postupno će, nadamo se, pojedine magične riječi i rituali kojima ih povezujemo u smislene cjeline postajati razumljivijima. Objašnjavanje, međutim, neće teći od retka do retka, slijedom kojim su napisani. Počet ćemo „iznutra", između vitičastih zagrada. Uz to, ponešto će ostati nedorečeno pa ćete neke pojedinosti morati zasad jednostavno prihvatiti.

Programi napisani u jezicima *C* i *C++* (i ne samo u njima) nizovi su *funkcija* koje barataju *podacima*. Funkcije su pak sastavljene od *naredbi* koje podrobno propisuju postupak izvođenja određene zadaće. Može se reći da su funkcije aktivni, a podaci pasivni dio programa.

U našem smo programčiću napisali samo jednu funkciju: funkciju `main()`. Naredbe koje neka funkcija — `main()` ili bilo koja druga — sadrži ograđuju se vitičastim zagradama { i }. (Za znalce *Pascal*a: vitičaste zagrade imaju ulogu ključnih riječi `begin` i `end`.)

Podatak kojim naša funkcija `main()` barata doslovno je navedeni niz znakova `Goodbye!`. Doslovno navedeni nizovi znakova se u kôdu u jezicima *C* i *C++* navode između para dvostrukih gornjih navodnika `"`.

Funkcija `main()` obično poziva druge funkcije koje umjesto nje obavljaju veći dio posla. U našem primjeru, na prvi pogled, nema pozivâ funkcija. Međutim, izrazi u kojima se pojavljuju ope-

ratori prikriveni su pozivi funkcija. I u matematici je, primjerice, zbrajanje, označeno operatorom $+$, funkcija koja paru brojeva, recimo realnih, pridružuje treći broj:

$$+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}, \qquad + : (a, b) \mapsto c = a + b.$$

Operatori samo pružaju nešto sažetiji i čitljiviji zapis.

Poput operatora +, i operator $<<$ ima dva operanda,[1] ali oni nisu istoga tipa. Označava li operator $<<$ ispis, lijevi je operand izlazni tôk, a desni nešto što treba ispisati. Tako u izrazu

```
cout << "Goodbye!"
```

u izlazni tôk izlijevamo niz znakova „`Goodbye!`”. Zasad je malo teže objasniti što je točno `endl` u

```
cout << endl
```

Nećete pogriješiti smatrate li ga nalogom koji od izlaznoga tôk zahtijeva da sljedeći ispis započne u novom retku (`endl` je skraćeno od *end of line*, što znači „kraj retka”; četvrti je znak u `endl` slovo ℓ, a ne znamenka 1).

Ulaznim i izlaznim tôkovima (engl. *input/output stream, I/O stream*) struje nizovi znakova kojima su podaci zapisani. Tôkovi povezuju program s tipkovnicom, zaslonom, datotekama podataka... Izlazni tôk `cout` povezan je najčešće sa zaslonom ili s „prozorom” koji operacijski sustav pridružuje našem programu.

Tôk `cout`, nalog `endl` i operator $<<$ definirani su u upisno/ispisnom dijelu *standardne biblioteke*; ne pripadaju, drugim riječima, jezgri jezika *C++*. Stoga prevodilac ne zna što nazivi `cout` i `endl` i niz od dva uzastopna znaka $<$ znače.[2] Da bi ih prevodilac mogao prepoznati, moramo u program, prije nego što ih upotrijebimo, uvesti njihovo značenje. To postižemo uključivanjem *standardne datoteka zaglavljâ* iostream koja sadrži potrebne informacije o ulaznim i izlaznim tôkovima i o pridruženim operatorima. Datoteku zaglavljâ uključujemo pretprocesorskim nalogom #include koji prevodiocu nalaže da privremeno prekine čitanje tekuće datoteke (naše datoteke `goodbye.cc` s izvornim kôdom) kako bi pročitao sadržaj datoteke[3] čiji je naziv naveden između < i > (iostream u našem primjeru). Pošto završi čitanje datoteke zaglavljâ, prevodilac nastavlja čitanje početne datoteke u retku koji slijedi iza retka s nalogom #include (to znači da se jednim nalogom #include može uključiti samo jedna datoteka zaglavljâ, te da u jednom retku smije biti samo jedan nalog #include; uz to, # mora u retku biti prvi znak različit od praznine).

Na ovom mjestu treba otvoriti poveću zagradu i dati opću, iako površnu i nepotpunu, sliku postupka prevođenja programa. Za različite se operacijske sustave i za različite prevodioce pojedinosti razlikuju, no, okvirno, osnovni su koraci uvijek:

1. *Pretprocesor* (engl. *preprocessor*) obrađuje takozvane pretprocesorske naloge — retke programa koji počinju sa #.

[1] Operator s dva operanda — dakle, sažetiji način zapisa funkcije dviju varijabli — naziva se binarnim operatorom. Uobičajeno pisanje binarnoga operatora između njegovih dvaju operanada je takozvani infiksni zapis.

[2] Operator $<<$ ima u jeziku *C++* nekoliko značenja. Operacija ispisa, rekosmo, nije definirana u jezgri jezika već u standardnoj biblioteci. Osnovno, u jezik ugrađeno značenje nije primjenjivo na izlazne tôkove, nego na neke druge tipove operanada; to nas značenje, međutim, ne zanima.

[3] Prema standardu jezika *standardne* datoteke zaglavljâ ne moraju biti posebne datoteke, ali posljedice izvršavanja naloga #include moraju biti takve *kao da* je datoteka pročitana.

2. *C++ prevodilac* (engl. *compiler*) prevodi izvorni u *ciljni kôd* (engl. *object code*). Ciljna datoteka (najčešće sa sufiksima `.o` ili `.obj`) sadrži mješavinu instrukcija u strojnom jeziku i simboličkih informacija kojima se u sljedećem koraku koristi povezivač.
3. *Povezivač* (engl. *linker*) ciljnoj datoteci dodaje modul koji omogućava pokretanje programa. Uz to, povezivač će, ako treba, pretražiti standardnu i, možda, neke druge biblioteke (datoteke sa sufiksima `.a` ili `.lib` te `.so` ili `.dll`; usput, na engleskom je biblioteka *library*), te u izvršni program ugraditi potrebne cjeline (u našem primjeru modul za ispis).
   Izvorni se kôd većih programa često dijeli u nekoliko datoteka. Štoviše, neki dijelovi mogu biti napisani u mnemoničkom jeziku (*assembler*-u) ili u nekom drugom višem programskom jeziku. Povezivač će tada sve ciljne datoteke, nastale prevođenjem tih izvornih datoteka, povezati u jedan izvršni program.

O pojedinim koracima obično ne trebate sami voditi računa. Sve će ih umjesto vas izvesti kontrolni program (koji ćemo, kao i do sada, jednostavnosti radi nazivati prevodiocem, a cjelokupni navedeni niz koraka prevođenjem programa). Različiti se prevodioci upotrebljavaju na različite načine. U novije su vrijeme prevodioci uklopljeni u takozvane integrirane razvojne okoline (engl. *integrated development environment, IDE*). Pojedinosti ćete morati potražiti u priručniku vašega prevodioca.

Prije nego što zatvorimo zagradu, još nekoliko riječi o odnosu datoteka zaglavljâ i biblioteka. Datoteke zaglavljâ su sučelja biblioteka, ona samo uvode nazive: tako će prevodilac nakon uključivanja sadržaja datoteke zaglavljâ iostream znati da je `cout` naziv nekog izlaznoga tôka (to jest, objekta tipa **`ostream`**) i da postoji binarni operator $<<$ kojemu je izlazni tôk lijevi, a niz znakova desni operand. Ali prevodilac neće znati, ali niti ne mora znati, što taj operator radi i kako to radi, pa čak ni što je zapravo izlazni tôk — za njega je i **`ostream`** tek naziv izvedenoga tipa. Kôd koji oblikuje izvedene tipove i propisuje stvarni rad funkcija (i operatora) nalazi se, preveden, u bibliotekama.

Kad već pričamo o bibliotekama, razjasnit ćemo odmah i redak

```
using namespace std;
```

Biblioteka može imati naziv. Taj naziv označava okružje (*namespace*) u kojemu se nalaze i u kojem su „skriveni" svi drugi nazivi koje ta biblioteka uvodi. Takvo skrivanje nazivâ omogućava autorima različitih biblioteka da različite stvari nazovu istim imenima; tako naziv **`vector`**, koji u standardnoj biblioteci označava niz, može u nekoj matematičkoj biblioteci označavati tip koji je bliži pojmu vektora poznatom iz linearne algebre.

Okružje standardne biblioteke nazvano je **`std`**. Budući da su nazivi iz standardne biblioteke skriveni u njezinom okružju, akoli želimo upotrijebiti ono što imenuju, moramo navesti i naziv biblioteke: primjerice, **`std`**`::cout` ili **`std`**`::endl`. Potrebu za stalnim navođenjem „prefiksa" **`std`**`::` možemo izbjeći tako da naredbom

```
using namespace std;
```

u program uvedemo sve nazive iz standardne biblioteke. Treba napomenuti da to baš i nije najbolji način dohvaćanja naziva iz standardne biblioteke, jer njime uvodimo sve nazive, one koje ćemo upotrijebiti, ali i one koji nam neće trebati, a time otvaramo vrata mogućnosti „sudara" istih nazivâ. No, u manjim programima, posebice ako upotrebljamo samo standardnu biblioteku, taj se način, kao najjednostavniji, može primjenjivati. Manje je opasno izrijekom navesti samo one nazive koje ćemo

zaista trebati:

```
using std::cout;
using std::endl;
```

Napokon se možemo vratiti našem programu i izrazu

```
cout << "Goodbye!" .
```

Kao što operator zbrajanja + paru brojeva pridružuje broj — njihov zbroj, tako i operator << svojim operandima pridružuje rezultat operacije; u (pseudo)matematičkom zapisu:

```
<< : ostream × niz znakova → ostream,
<< : ( cout, "Goodbye!" ) ↦ cout .
```

Operator <<, dakle, kao rezultat operacije „vraća" svoj lijevi operand, izlazni tôk. Ta činjenica omogućava uzastopnu primjenu operatora <<:

```
cout << "Goodbye!" << endl .
```

Prevodilac prethodni izraz interpretira kao

```
(cout << "Goodbye!") << endl .
```

Prvo se izvodi operacija u zagradama. Njen je rezultat lijevi operand, tôk `cout`. Taj tôk postaje potom lijevim operandom u sljedećoj primjeni operatora <<, u kojoj je desni operand `endl`. Prema tome, naš programčić možemo napisati i u malo sažetijem obliku:

```
#include<iostream>

using namespace std;

int main() {
  cout << "Goodbye!" << endl;
  return 0;
}
```

Postoji još jedna važna razlika između operatora zbrajanja + i operatora za ispis <<. Dok operator +, kao i drugi aritmetički operatori, samo „vraća" neki rezultat, operator << ima i popratni učinak — ispis na zaslon ili u prozor. Štoviše, operator << primjenjujemo upravo zbog njegova popratnog učinka (to da taj operator vraća izlazni tôk tek je pogodnost koja omogućava sažetiji zapis niza operacija). *Popratnim učinkom* (engl. *side effect*) nazivamo promjenu u stanju okoline u kojoj se program izvodi; tako operator za ispis mijenja prikaz na zaslonu ili u prozoru. U nastavku ćemo se često se susretati s još jednim operatorom koji se primjenjuje ponajprije zbog popratnoga učinka: s operatorom pridruživanja = koji varijabli pridružuje vrijednost; on, dakle, mijenja sadržaj memorijskoga prostora koji je pridružen varijabli. Aritmetički pak operatori ništa ne mijenjaju; rezultat operacije tek treba ispisati ili pridružiti nekoj varijabli.

Vrijeme je da obratite pozornost i na točke sa zarezom (`;`) na krajevima redaka u funkciji `main()`. Dok je

```
cout << "Goodbye!" << endl
```

izraz,

```
cout << "Goodbye!" << endl;
```

je naredba. *Izraz* je zapis operacije, a *naredba* je zahtjev za njenim izvođenjem. Izraz postaje postaje naredbom tek kad na njegovu kraju napišete točku sa zarezom. (U jeziku *C++* se, kao i u jezicima *C* i *Pascal*, a za razliku od *Fortran*a, nekoliko naredbi može navesti u jednom retku ili se jedna naredba može protegnuti kroz nekoliko redaka, bez posebne oznake za nastavljanje. No, dok u *Pascal*u razdvaja naredbe, točka sa zarezom u *C*/*C++*-u završava naredbu; prema tome, iza *svake* naredbe, pa i ako je jedina, ili posljednja u nizu, *mora biti* točka sa zarezom.)

Jedinu funkciju u našem programu, koju smo sami napisali, nismo slučajno nazvali `main()`. Ona zaista jest glavna funkcija: svaki *C*/*C++* program *mora* imati *jednu*, *i samo jednu*, funkciju `main()`. Program se počinje izvoditi na njezinu početku.

Početak definicije funkcije, prije otvorene vitičaste zagrade {, nazivamo njezinim zaglavljem. Sastavljeno je od tri dijela: naziva funkcije, deklaracija njezinih parametara, navedenih između okruglih zagrada, i tipa njezine vrijednosti (ne u navedenom redoslijedu). U zaglavlju

```
double power (double x, int n)
```

`power` je naziv funkcije, `double x` i `int n` (unutar zagrada) deklaracije su parametara, a `double` lijevo od naziva funkcije tip je njezine vrijednosti. Vrijednost funkcije je vrijednost koju funkcija „vraća" kao rezultat izvođenja. Tako naša funkcija `power()` izračunava i vraća broj $y = x^n$ (u zapisu s pokretnim zarezom, uz dvostruku točnost). Tip vrijednosti funkcije u programu odgovara kodomeni matematičke funkcije, a tipovi parametara odgovaraju njezinoj domeni:

$$\text{power} : \mathbb{R} \times \mathbb{Z} \to \mathbb{R}$$

(ipak, to je površna analogija: tipovi `int` i `double` nisu skupovi cijelih i realnih brojeva, nego tek njihovi konačni, vrlo ograničeni podskupovi).

Kako u zaglavlju funkcije `main()`,

```
int main()
```

između zagrada nema ničega, podrobniju priču o deklaracijama parametara možemo odložiti za kasnije. Prazne zagrade znače da funkcija nema parametara; u matematičkoj analogiji

$$\texttt{main} : \emptyset \to \mathbb{Z}.$$

Ipak, zagrade moraju postojati — tek s njima `main` postaje nazivom funkcije.

Vrijednost je naše funkcije `main` tipa `int`; ta vrijednost mora biti tipa `int`, standard jezika ne dopušta niti jedan drugi tip vrijednosti. Tu vrijednost prevedeni program nakon izvođenja vraća operacijskom sustavu (ili nekom drugom programu koji ga je pokrenuo) kao izvještaj o uspješnosti izvođenja. Dogovorno, vrijednost 0 znači da je sve u redu; bilo koja druga vrijednost znači da je nešto pošlo krivo.

Vrijednost se iz funkcije vraća naredbom `return`. Pritom se prekida izvođenje funkcije te upravljanje preuzima okolina koja je pozvala funkciju, u našem slučaju operacijski sustav.