Neka vrsta magije

Programiranje je čin osmišljavanja i pisanja računalnih programa. No što su, zapravo, računalni programi, možda se pitate? Pokušajmo to objasniti ovako: recimo da vam je nekom providnošću poznata magija kojom ćete prizvati drijadu[1], ili možda troglavog Kerbera[2]. Ta stvorenja, kao i sva mitska stvorenja, ne postoje u prirodi, upravo kako ne postoje ni brojevi – nigdje u prirodi ne možemo pronaći ni brojeve, ni drijade. I drijade i brojevi samo su najobičnija ljudska izmišljotina. Pih!

Ipak, brojeve je moguće zapisati brojkama! Brojke, to su oni čudnovati znakići, sastavljeni od linija punih zavijutaka i kutova, posudismo ih od Arapa, a izgledaju otprilike ovako: 1; 4; –11; 117; 43,56; 3,14. Eto, kako brojeve zapisujemo brojkama, tako programerske magije, koje od milja nazivamo algoritmima, zapisujemo s pomoću vrlo posebnih, štoviše čarobnih jezika. S obzirom na to da je riječ o vrsti magije, algoritmi ne mogu postojati. Međutim, programi su zapisi algoritama, neka njihova pojavnost, baš kao što mitska stvorenja svoju pojavnost mogu dobiti u kipovima i slikama nadahnutih umjetnika.

Usprkos tomu, programi mogu biti vrlo korisni. Ova knjiga, osim što će vas naučiti pisati programe, pokazat će vam kako ih možete oživjeti, i od cijele te izmišljotine dobiti pokoji konkretan rezultat – u obliku kakvog duha ili magičnog dima. U današnje doba magični dim ipak nije od prevelikog interesa, pa programe koristimo da oni, umjesto nas, nadziru rad raznih složenih procesa. Tako programi nadziru rad vašeg automobila, televizora, klima-uređaja, mlaznog aviona i nuklearne elektrane. Ne zadovoljavaju li vas ti primjeri zbog svoje banalnosti, recimo da je mnogo zanimljivije što programi upravljaju složenim igračkama, poput *Spirita* (hrv. Duh) – autića za istraživanje Marsa – i računalnim igrama.

Nedavno su ljudi osmislili čudnovate strojeve koje nazivamo računalima. Računalo može napraviti da programi u njemu ožive u obliku razigranih duhova, pa kroz svoju igru i druženje proizvode raznovrsne čudnovate pojave.

Magični dim vrlo je važna sastavnica računala. Brojni ljudi nisu uvjereni u njegovo postojanje, unatoč silnim dokazima u suprotno. Tako, na primjer, svaki integrirani krug, koji ponekad jednostavnije nazivamo čipom, sadržava magični dim. Magični dim može izaći iz čipa u obliku bijelog oblačića. Kada se takvo što dogodi, čip ne može ispravno raditi pa kažemo da je čip pokvaren!

U redu, ovaj prethodni odlomak bio je samo mala šala! Unatoč tomu, mislite li da se šalimo, varate se! Bez obzira na to što volimo koristiti raznolike, neobične i ponekad šaljive metafore, jer ih smatramo učinkovitim načinom tumačenja, želimo da nas shvatite vrlo ozbiljno, uključujući i upravo spomenute neobičnosti. Kroz ovu knjigu pokušali smo dati pristupačan, ali također sustavan i stručan uvod u programiranje. Za nas to ne znači da moramo biti

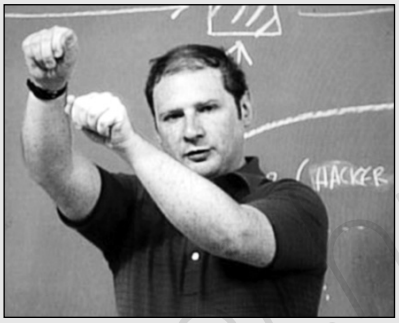
[1] U starogrčkoj mitologiji, šumske nimfe. Od grč. dryos – drvo, hrast.
[2] Troglavi zli pas iz grčke mitologije s repom i grivom zmaja – čuva ulaz u podzemni svijet.

dosadno suhoparni, štoviše, smatramo da je programiranje samo po sebi zabavna aktivnost – po mnogočemu nalik magiji[3], kako smo to već natuknuli, pa se može lako rastumačiti na zabavan način, što smo i pokušali napraviti – vi ocijenite jesmo li uspjeli.

Vrlo je važno pri radu s magijama pridržavati se naputaka Čarobnjačkog ceha, u kojem mudruju, i mudrovali su, najugledniji čarobnjaci! Događalo se već da čarobnjaci-početnici svojim nemuštim čarolijama namjesto vile prizovu sedmeroglavu hidru[4], tada, jao! Nema druge nego iz duboke meditacije buditi čarobnjake-majstore da svijet spase utjelovljenjem Herkula[5]!

Računalni duhovi, iako neopipljivi i nevidljivi, veoma su stvarne i mnogostruke pojavnosti: obitavajući u računalima diljem svijeta čine naš rad za računalom uglavnom ugodnim iskustvom omogućujući pisanje vrlo urednih školskih zadaća s automatskom provjerom pravopisa, lako izračunavanje nekih nepravedno teških jednadžbi iz matematike ili nam pružaju nebrojene sate zabave kroz veoma edukativno nastrojene računalne igre. Ipak, svi oni su, izravno ili posredno, potekli iz ljudskog uma. Dakle, ti duhovi u računalima naše su umotvorine, stvorene čarima u obliku pomnjivo sazdanih simboličkih izraza negovornih jezika, poznatih kao **programski jezici**.

Gerald Jay Sussman (Džerald Đej Sasmn), čarobnjak-majstor, profesor pri MIT-u (Massachusetts Institute of Technology), koautor programskog jezika *Scheme*.

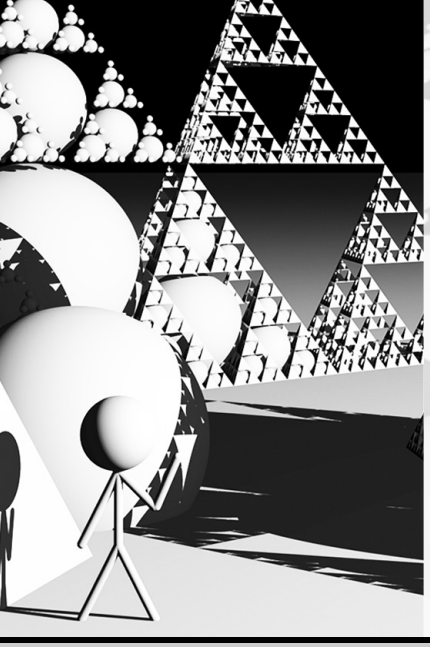


Harold Abelson (Herld Ejblson), čarobnjak-majstor, profesor pri MIT-u. Primio je nekoliko nagrada za doprinos pedagogiji i poučavanju računarstva.

Postoji mnogo, mnogo programskih jezika, a među njima je i dalje mnogo onih koji bi bili prikladni za praksu čarobnjaka-početnika. Odvagnuvši njihove prednosti i nedostatke, i uzevši u obzir brojne činitelje, među kojima čak i količinu vode na planeti Zemlji, zaključili smo da bi bilo najprikladnije koristiti programski jezik zvan **C++** (ce plus plus).

Kako su nam poznate muke kroz koje mladi šegrti računalnog čarobnjaštva prolaze učeći arhaične i ispočetka teško razumljive jezike, nas dvoje meštara čarobnjaštva odlučismo samo za vas sastaviti ovu malu knjigu čarobnjačkih tajni. Kroz nju ćete spoznati pravo naličje tih nestašnih vilenjaka i patuljaka u vašim računalima, a možda i tajnu pronalaženja četverolisnih djetelina.

[3] Ova parabola preuzeta je sa poznate serije predavanja SICP (engl. *Structure and interpretation of computer programs*) sa sveučilišta MIT.
[4] U starogrčkoj mitologiji, neimenovana zmijolika morska neman, s mnogo glava.
[5] U starogrčkoj mitologiji, božanski junak, sin Zeusa i Alkmene, ubio je Hidru.



Prvi dio

www.element.hr

1. Prve čarolije

Učenje programskog jezika uvelike je nalik učenju stranog jezika. U ovom poglavlju upoznat ćemo vas s osnovnim sastavnicama programskog jezika "C++". Također, vidjet ćete kako pisati i izvršavati čitave programe koristeći ništa više osim olovke i komada papira.

Poput svakog drugog jezika, rečenice programskog jezika "C++" sačinjene su od riječi. S obzirom da je C++ jezik namijenjen računanju nije iznenađujuće što su najčešće riječi u ovom jeziku brojevi. Riječi ove vrste mogu se spajati u "rečenice" koristeći aritmetičke operatore poput zbrajanja (+), množenja (*), oduzimanja (–) i dijeljenja (/). U programskom jeziku C++-u, spojevi brojeva i operatora nalikuju uobičajenim aritmetičkim izrazima:

```
5+3.14
```

ili

```
7+2*6
```

Ovakav spoj brojeva i operatora nazivamo **izraz** (engl. *expression*).

Kao što svaka rečenica govornog jezika ima značenje, tako ga imaju i izrazi programskih jezika. Značenje izraza je opis postupka kojim se dolazi do vrijednosti izraza, i taj se postupak naziva **vrednovanjem** (engl. *evaluation*). Dakle, vrednovanje izraza je postupak izračunavanja njegove vrijednosti. Koristeći ovo nazivlje možemo, dakle, reći da su vrijednosti prethodnih dvaju izraza sljedeće:

`5+3.14` po vrednovanju rezultira brojem `8.14`

`7+2*6` po vrednovanju rezultira brojem `19`

Sada moramo istaknuti činjenicu da izrazi nisu potpuno ispravne rečenice jezika "C++". Oni nisu ispravne rečenice u istom smislu kao što ni gramatički ispravan slijed riječi nije ispravna rečenica hrvatskog jezika ako nije završen jednim od dozvoljenih pravopisnih znakova. U jeziku C++-u postoji samo jedan takav pravopisni znak, a to je točka sa zarezom (;). Sve što je stoga potrebno napraviti da prethodna dva izraza pretvorimo u ispravne rečenice jezika C++-a jest da ih završimo točkom sa zarezom ovako:

```
5+3.14;
7+2*6;
```

Izraz završen točkom sa zarezom naziva se **naredbom** (engl. *command*)[6]. U jeziku C++-u su upravo naredbe, a ne izrazi, najmanji samostalni dijelovi ispravno napisanog programa. No, zacijelo se pitate što je to program. Pokažimo to primjerom.

[6] Lažemo, i to namjerno. U C++-u se dotična gramatička struktura naziva **izjavom** (engl. *statement*). Ipak ćemo u ovoj knjizi koristiti pojam "naredba" jer je on smisleno bliži konceptu koji pokušavamo objasniti, i koji se u sličnom obliku koristi u mnogim drugim programskim jezicima. A ovo je ipak prvenstveno knjiga o programiranju, a ne o C++-u.

Prvi program

Svi programi koje budemo pisali u C++-u bit će sastavljeni od slijeda naredbi (poput onih koje smo upravo naveli) zaokruženog komadičkom magije koji zovemo **funkcija "main"**. Engleska riječ "main" znači: glavna, a izgovara se kao "mejn".

Na primjer, želimo li načiniti program od naredbe `5+3.14;`, dovoljno je napisati sljedeće:

```
int main()
{
    5+3.14;
}
```

Ovime smo naredbu `5+3.14;` učinili dijelom funkcije "main". Ova funkcija sačinjava glavni (i u ovom slučaju jedini) dio našeg programa. No, što je rezultat rada programa koji smo upravo napisali?

Odgovor na navedeno pitanje može se dobiti postupkom nazvanim **izvršavanje** (engl. *execution*). Da bismo izvršili program jezika C++-a, moramo izvršiti sve naredbe unutar funkcije `main`. U ovom slučaju, u toj funkciji nalazimo samo jednu naredbu.

U jeziku C++-u, svaka naredba sastoji se od dvaju dijelova: izraz i završnik (točka sa zarezom). Da bismo izvršili naredbu, trebamo izvršiti oba njena sastavna dijela. Izvršavanje izraza je jednostavno: naprosto ga vrednujemo. Dakle, rezultat izvršavanja izraza nije ništa drugo nego sama vrijednost tog izraza: u ovom se slučaju izraz `5+3.14` vrednuje u broj 8,14.

Ali, koje je značenje točke sa zarezom? Osim što njome završavamo izraze i razdvajamo naredbe, točka sa zarezom nalaže čitatelju ili izvršitelju programa da odbaci vrijednost izraza koji je tom točkom sa zarezom završen.

Ovo vam zacijelo zvuči totalno bizarno! Zašto uopće računati vrijednost nečega što ćemo na kraju jednostavno odbaciti?

Sagledajmo stvar iz malo drukčijeg kuta. Pretpostavimo da svom prijatelju kažete: "Izračunaj mi 5+3,14!". Za očekivati je da će vam odgovoriti rezultatom: "8,14!" Ipak, u malo vjerojatnom slučaju da on ne shvati što se od njega očekuje, uvijek možete pojasniti situaciju kazujući: "A sad mi reci rezultat!".

U programiranju ipak ne možemo uživati u luksuzu nedorečenih želja. Želimo li imati ikakvu sigurnost da će nas čitatelj ispravno shvatiti, moramo se dodatno potruditi te jasno i glasno kazati da želimo znati rezultat dotičnog izračuna. Kako u prethodnom programu nismo učinili ništa tomu nalik, rezultat računanja zbroja `5+3.14` biva odbačen, a čitav naš trud napravljen beskorisnim.

Standardni izlaz

Naš je sljedeći korak napisati program koji će priopćiti rezultate svojih računa. Da bismo ovo izveli, morat ćemo opet uvesti komadičak magije koji ćemo nazvati **standardni izlaz**. Standardni izlaz je donekle napredna ideja za čije ćete potpuno objašnjenje morati pričekati do druge polovine ove knjige. Za potrebe ovog poglavlja pretpostavit ćemo da je standardni izlaz list papira na koji će programi zapisivati svoje rezultate. Na vrhu tog papira možemo napisati **'STDOUT'** da bismo naznačili njegov smisao.

U tekstu programa pisanom u jeziku C++-u, standardni izlaz se naznačava riječju `cout` (od engl. *character output*). Bez daljnjeg odugovlačenja, predstavljamo vam izmjenu prethodnog programa tako da on sada ispisuje svoj rezultat na standardni izlaz:

```
int main()
{
    cout << 5 + 4;
}
```

Osim nove riječi `cout`, uočavamo još jednu novinu u ovom programu: neobičan dvoznak `<<`. Ovaj dvoznak je zapravo operator, poznat pod nazivom **operator umetanja** (engl. *insertion operator*). Ovaj operator "uzima" vrijednost izraza sa svoje desne strane i šalje je na standardni izlaz, naznačen riječju `cout`.

Izvršimo sada zajedno ovaj program. Za to će vam trebati olovka i komad papira koji će predstavljati naš standardni izlaz.

Izvršavanje svakog programa jezika C++-a započinje prvom naredbom u funkciji `main`. U ovom slučaju nalazimo samo jednu naredbu za izvršiti. Prvo izvršavamo izraz 5+4, kojemu je rezultat broj 9. Zatim šaljemo taj rezultat na standardni izlaz, odnosno na naš komad papira na kojem bi se sad trebao pojaviti broj 9.

```
STDOUT:

9
```

Stringovi

Ponekad je u životu potrebno podrobno objasniti zašto smo nešto učinili kako bi ljudi mogli u potpunosti cijeniti naša dostignuća. Isto bi se moglo reći i za programe: promatrač koji bi stajao sa strane za vrijeme izvršavanja prethodnog programa ne bi baš mogao imati jasnu sliku što se upravo dogodilo, odnosno saznati koji je bio smisao izvršenog programa. Da bismo ispravili ovu nepravdu, izmijenit ćemo naš program ispisujući poruku na standardni izlaz. Poruka će slučajnom promatraču objasniti što se naš program trsio postići:

```
int main()
{
    cout << "Ovo je moj treci program!" << endl;
    cout << "5+4=" << 5+4 << endl;
}
```

Možemo uočiti nekoliko novina u ovom programu. Prva novina je da tekst umetnut unutar dvostrukih znakova navodnika, poput `"Ovo je moj treci program!"` nazivamo **string**. Za razliku od izraza, stringovi se ne izračunavaju već se šalju na standardni izlaz točno onako kako su navedeni. Drugim riječima, vrijednost stringa je upravo tekst između dvostrukih znakova navodnika.

Druga novina je riječ: `endl` (od engl. *end line* – završetak retka). Ovu riječ koristimo da bismo na standardnom izlazu završili ispis trenutačnog retka i prešli u idući, novi redak.

Ovaj se program sada sastoji od dviju naredbi koje su obje složenije nego u prethodnom primjeru, pa uočavamo nekoliko operatora umetanja. Ovakve složene naredbe izvršavamo korak po korak, slijeva na desno.

Počevši s prvom naredbom, najprije moramo ispisati string `"Ovo je moj treci program!"`. Nastavljajući dalje slijeva na desno, nailazimo na riječ `endl` koju je također potrebno "ispisati", to jest izvršiti. Rezultat je završetak ispisa trenutačnog retka i prelazak u novi redak. Učinivši ovo, u potpunosti smo izvršili prvu naredbu:

```
STDOUT:

Ovo je moj treći program!
```

Ispis će se nastaviti u ovom retku.

Izvršimo sada drugu naredbu programa. Idući slijeva na desno, prva stvar na koju nailazimo jest string `"5+4="`. Na prvi se pogled doima da je posrijedi izraz unutar stringa, no ovo je samo privid. U jeziku C++-u stringovi nikada ne sadržavaju izraze, već samo tekst. Kao što smo već rekli, upravo je ovaj tekst unutar znakova navodnika vrijednost stringa koju je potrebno ispisati na standardni izlaz:

```
STDOUT:

Ovo je moj treći program!
5+4=
```

Nastavljajući dalje duž naredbe, nailazimo na izraz `5+4` koji najprije izračunavamo, a zatim rezultat tog izračuna zapisujemo na standardni izlaz. Dakle, na našem komadu papira pojavit će se broj 9. Konačno, "ispišemo" i riječ `endl` čime završavamo ispis ovog retka i prelazimo u novi:

```
STDOUT:

Ovo je moj treći program!
5+4=9
```

Naredni ispis nastavio bi se u ovom retku.

Budući da izvršavanje programa završava zadnjom izvršenom naredbom, time smo ovaj program uspješno priveli kraju.

Zaključit ćemo dvama kratkim komentarima. Čitajući ovaj program mogli ste primijetiti popriličan broj operatora umetanja. Ne ulazeći u detalje, kazat ćemo da su operatori umetanja neophodni za razdvajanje različitih elemenata jezika: stringova od čarobnih riječi (`cout` i `endl`), stringova od izraza ili izraza od čarobnih riječi. U suštini, ti su operatori samo još jedan primjer glagola u jeziku C++-u, poput aritmetičkih operatora kojima smo gradili izraze.

Drugo, jeste li obratili pažnju na redoslijed kojim smo izvršili naredbe ovog programa? U jeziku C++-u naredbe se uvijek izvršavaju strogim redoslijedom: od prve naredbe na vrhu programa do posljednje naredbe na dnu programa. Želimo li zvučati malo pametnije, možemo reći da tekst programa jezika C++-a u sebi implicitno sadržava pojam vremena, a zbog toga poredak naredbi ima primjetan učinak na rezultat izvršavanja programa.

Izrazi i operatori

Kao što smo već spomenuli, programski jezik C++ nam dopušta korištenje svih osnovnih aritmetičkih operatora: za zbrajanje, oduzimanje, množenje i dijeljenje koristeći uobičajene logograme[7]: `+`, `-`, `*`, `/`. Znak zvjedice, poznat i kao **asterisk**, jest **operator množenja**, dok je kosa crta (engl. *forward slash*) **operator dijeljenja**. Za sve ove operatore vrijede uobičajena pravila prioriteta: operatori množenja i dijeljenja imaju primat nad operatorima zbrajanja i oduzimanja. Stoga:

```
cout << 20-3*4;
```

... za rezultat daje:

```
STDOUT:

8
```

Moguće je izračunati i više izraza unutar jedne naredbe:

```
cout << 7+5 << 20/5;
```

Ova bi naredba rezultirala ovakvim ispisom na standardnom izlazu:

```
STDOUT:

124
```

Ali kako? Odakle sad broj 124? Pa, zapravo je jednostavno: 7+5 je 12, a 20/5 je 4. Stoga, prvo što biva ispisano je broj 12 nakon kojeg odmah slijedi broj 4. Ovo nam se čini kao broj 124 zato što nismo koristili bjeline prilikom ispisa da bismo razdvojili ova dva broja. To možemo učiniti ovako:

[7] Logogram je znak koji stoji za čitavu riječ ili pak morfem. Na primjer, $ (dolar) i £ (funta) su dva poznata logograma. Brojke i matematički operatori također su logogrami.

```
cout << 7+5 << " " << 20/5;
```

... i rezultat je sada:

```
STDOUT:

12 4
```

Zagrade

Dozvoljeno je koristiti oble zagrade za grupiranje izraza:

```
cout << (3 + (28-2)*5)*2;
```

Rezultat rada ove naredbe je:

```
STDOUT:

266
```

Ipak, moramo vas upozoriti da nije dozvoljeno koristiti uglate i vitičaste zagrade u svrhu grupiranja izraza:

```
cout << [3 +(28-2)*5]*2;    // POGRESNO!!!!
```

Spomenuli smo već da je vrijednost stringa tekst sadržan unutar dvostrukih navodnika. Razlog tome je činjenica da je string sam po sebi primjer izraza koji zovemo **doslovni izraz** (engl. *litteral*). Doslovni izrazi su oni izrazi kojima je vrijednost izravno navedena. Još jedan primjer doslovnih izraza su samostalni brojevi. U sljedećem programu dajemo za primjer i broj kao doslovni izraz:

```
int main()
{
    cout << "Ovaj string je doslovni izraz" << endl;
    cout << "Ovaj broj je takoder doslovni izraz: " << 5;
}
```

```
STDOUT:

Ovaj string je doslovni izraz
Ovaj broj je također doslovni izraz: 5
```

Da zaključimo: Izraz je ili **doslovan** ili **složen**. Složene izraze dobivamo spajanjem dvaju izraza nekim operatorom. Primjerice, izraz `5+2*5` je složeni izraz. Sastoji se od doslovnog izraza `5`, operatora `+` i izraza `2*5`. Složeni izraz `2*5` se, pak, sastoji od doslovnog izraza `2`, operatora množenja i doslovnog izraza `5`.

Urednost

Evo programa koji računa broj sati u tjednu:

```
int main()
{ cout << "U tjednu ima " << 24*7 << " sati." << endl; }
```

Možda je neobično što smo vitičaste zagrade stavili u isti redak s prvom naredbom, ali to nema baš nikakvu važnost. Moguće je cijeli program napisati u jednom retku, ovako:

```
int main(){ cout << "U tjednu ima " << 24*7 << " sati." << endl; }
```

... ili u više redaka, ovako:

```
int
main(
    ){ cout     <<
"U tjednu ima "<<24
    *7 <<               " sati."
<<
endl;
    }
```

Što se bjelina i redaka tiče, C++ je programski jezik **slobodnog oblika** (engl. *free form*), što znači da program možemo dijeliti u retke kako god nam se to sviđalo. Ipak, Čarobnjački ceh izdao je naputak da programe treba uredno pisati, da budu što jasniji i čitljiviji, ne bi li se time smanjio broj katastrofalnih pogrešaka. Ovaj posljednji primjer u suprotnosti je s tim naputkom, pa programe nećemo tako pisati. To bi moralo vrijediti i za vas, slažete li se?

Kao rezultat rada ovog programa ispisat će se:

```
STDOUT:

U tjednu ima 168 sati.
```

Greške

Evo još jednog programa:

```
int main() { cout<<(5+(6-4)*2)*10 }
```

Ovaj program ne može se izvršiti. Problem je u tomu što je program neispravno napisan – tekst programa sadržava pravopisnu pogrešku. Da, i programski jezici imaju svoj pravopis kao i prirodni jezici. Greška je jednostavna – zaboravili smo staviti točku sa zarezom na kraju naredbe `cout`.

Kada popravimo grešku:

```
int main() { cout<<(5+(6-4)*2)*10; }
```

... tad možemo pomoću računaljke, ili uz pomoć olovke i papira (ili čak bez njih, za one koji dobro računaju), zaključiti da je rezultat izvršavanja programa:

```
STDOUT:

90
```

Recite, gdje je greška u sljedećem programu?

```
int main() { cout<<[5+(6-4)*2]*10; }
```

Ponavljanje

Posljednji program u ovom uvodu objasnit će naredbu ponavljanja. Već smo spomenuli da se naredbe izvršavaju redom. Međutim, postoje naredbe koje utječu na redoslijed izvršavanja, a među takvim naredbama je i naredba ponavljanja.

Naredba ponavljanja se u jeziku C++-u naziva `for`, a slično je i u mnogim drugim programskim jezicima. Na prvi pogled, naredba `for` ima pomalo složen način pisanja, ili, drukčije rečeno, naredba `for` ima složenu sintaksu. No, pogledajmo kako to izgleda:

```
int main()
{
    cout << "I pjevam:" << endl;
    for (double i=1;i<=3;i++)
        {
        cout << "Plovi plovi plovi" << endl;
        cout << "moja barka mala" << endl;
        }
    cout << "A ja ribu lovim!" << endl;
}
```

Naredbu `for` slijede oble zagrade koje unutar sebe sadržavaju neke čarobne riječi. Zasad nećemo detaljno objašnjavati značenje tih riječi, osim što možete primijetiti broj 3, koji označava koliki je broj ponavljanja. Dakle, ova naredba `for` zadaje 3 ponavljanja.

> Primijetite da iza naredbe `for` ne slijedi točka sa zarezom!

Slijed naredbi sadržanih unutar vitičastih zagrada naziva se **blok naredbi** (engl. *statement block*). Naredba `for` upravlja blokom naredbi koji slijedi naredbu `for` tako da ga ponavlja zadani broj puta.

Naredbe u ovom programu izvršavaju se ovim redoslijedom:

- Najprije se izvršava naredba koja ispisuje "I pjevam:". Nakon toga nailazimo na naredbu `for`, kojom zadajemo da se naredni blok naredbi ponavlja 3 puta. Zatim se taj blok naredbi izvršava prvi put te se ispisuje "Plovi plovi plovi", a zatim "moja barka mala".

- Zatim se izvršava zadani blok naredbi drugi put: ispisuje se "Plovi plovi plovi", a zatim "moja barka mala".
- Zatim se izvršava zadani blok naredbi treći put: ispisuje se "Plovi plovi plovi", a zatim "moja barka mala". Uh, ovo ponavljanje je uistinu… ponavljajuće.
- Ovim smo izvršili naredbu `for`. Izvršavanje programa nastavlja se idućom naredbom, koja na standardni izlaz ispisuje "A ja ribu lovim!"

Rezultat izvršavanja programa je:

```
STDOUT:

I pjevam:
Plovi plovi plovi
moja barka mala
Plovi plovi plovi
moja barka mala
Plovi plovi plovi
moja barka mala
A ja ribu lovim!
```

2. Bez alata nema zanata

Ovo poglavlje je nešto drukčije od ostalih poglavlja. Govori se manje o programiranju, a više o alatima, kojima svaki šegrt mora znati rukovati želi li napredovati u ovom zanatu.

Iako bismo mogli pisati programe na papiru i ručno ih izvršavati, ljudi su osmislili alate koji uvelike olakšavaju oba procesa. Alat koji se najčešće koristi kao pomoć pri izvršavanju programa naziva se **računalo** (engl. *computer*). Ovaj, u stvarnosti sasvim bezuman stroj, u posljednjim je desetljećima raznim poboljšanjima tehnologije prilično uznapredovao u pogledu svoje praktičnosti, te čini nezamjenljiv alat koji pomaže pri pisanju programa – koje i sam može izvršavati.

Samim računalom ne bismo mnogo olakšali sebi rad. Samo po sebi, računalo radi samo s vrlo primitivnim jezikom zvanim **strojni jezik** (engl. *machine language*), koji je potpuno neprilagođen čovjekovim potrebama prilikom programiranja, ili za izvršavanje bilo kakvoga drugog zadatka. Prvi dodatak računalu, koji je potreban da bismo mogli izvršiti ikoji zadatak uz njegovu pomoć naziva se **operativni sustav** (engl. *operating system*). Operativni sustav je softver koji u sebi sadržava cijeli niz funkcionalnosti koje olakšavaju rad s računalom, kao što su:

- pohrana podataka i njihova organizacija
- pojednostavnjeno upravljanje i pristup hardveru računala
- upravljanje drugim instaliranim softverskim komponentama
- upravljanje memorijom i dodjela procesorskog vremena
- sustav dozvola i prava pristupa resursima računala
- korisničko sučelje za jednostavan odabir dostupnih akcija
- itd…

Bez operativnog sustava bilo bi vrlo teško raditi s računalom, stoga danas gotovo sva računala imaju barem jedan, obično postavljen prije kupnje računala. Dakle, osim operativnog sustava, što nam je još potrebno da bismo lakše programirali?

Pa, potreban nam je skup alata koji se naziva **objedinjena razvojna okolina – ORO** (engl. *integrated development environment – IDE*) za jezik koji želimo koristiti, a to je C++. Ovi alati nisu fizički već softverski, dakle riječ je o skupu programa. Možemo reći ovako – koristimo programe da nam olakšaju pisanje programa. ORO je skup programa, među kojima se obično nalaze barem:

- **Prevodilac** (engl. *compiler*) – ključni element koji prevodi program napisan u jednom jeziku u program napisan u drugom jeziku. U našem slučaju riječ je o prevodiocu iz jezika C++-a u strojni jezik procesora zasnovanog na skupu naredbi zvanom x86 (iks osam šest).

- **Uređivač teksta** (engl. *text editor*) – koristimo ga za unošenje teksta programa. Tekst programa naziva se **izvornim kôdom** (engl. *source code*). Uređivači teksta koji se koriste pri programiranju najčešće su posebno prilagođeni tom zadatku, tako da npr. samoinicijativno boje tekst raznim bojicama da bi pojedini dijelovi izvornog kôda bili bolje uočljivi.
- **Istrebljivač** (engl. *debugger*) – nudi mogućnost zaustavljanja programa u tijeku izvršavanja radi jednostavnijeg pronalaženja grešaka.
- I drugi...

Ne znamo zašto smo se ovoliko raspričali, jer možda ne trebate ništa od ovoga ni znati. A možda se i varamo. Za sada, sve što trebate napraviti jest postaviti objedinjenu razvojnu okolinu na vaše računalo i pokrenuti je.

Izbor objedinjene razvojne okoline ovisi o operativnom sustavu koji koristite i o programskom jeziku u kojem želite pisati programe. U ovoj su knjizi autori odabrali operacijski sustav Windows kao platformu na kojoj će pisati i izvršavati programe. U trenutku pisanja ove knjige, operacijski sustav Windows još uvijek drži prvo mjesto po zastupljenosti na našim prostorima, a učenicima i studentima je često dostupan besplatno u sklopu nekog od proizvođačevih školskih programa suradnje.

> Mnogo ćete bolje usvojiti znanja i vještine predstavljene u ovoj knjizi ako sjednete za računalo i **odradite** ovo i sva naredna poglavlja. Ako ste planirali ovu knjigu pročitati kao kakvu beletristiku, naše je mišljenje da ovo i nije baš najbolja knjiga takve vrste, naime, nema čak ni zaplet.

Za operativne sustave klase Windows i za potrebe ove knjige preporučujemo objedinjenu razvojnu okolinu *Microsoft Visual C++ 2010 Express*, koja je ujedno i besplatna. Upravo ćemo na ovoj ORO pisati i izvršavati sve programe ove knjige.

Ipak, ako više volite operativne sustave iz klase GNU Linux ili pak OS X preporučili bismo vam ORO *Codeblocks* koja je u mnogočemu vrlo slična razvojnoj okolini *Microsoft Visual C++ 2010 Express* pa ne biste trebali imati problema s praćenjem primjera u ovoj knjizi.

ORO Codeblocks je dostupna na adresi: http://www.codeblocks.org/downloads.

Preporučujemo da se opredijelite za opciju *Download the binary release* i odaberete odgovarajuću plaftormu.

Instalacija ORO *Microsoft Visual C++ 2010 Express*

Najjednostavnije je da ***Microsoft Visual C++ 2010 Express*** dobavite s internetskih stranica proizvođača, na adresi:

http://www.microsoft.com/visualstudio/en-us/express-cpp/overview

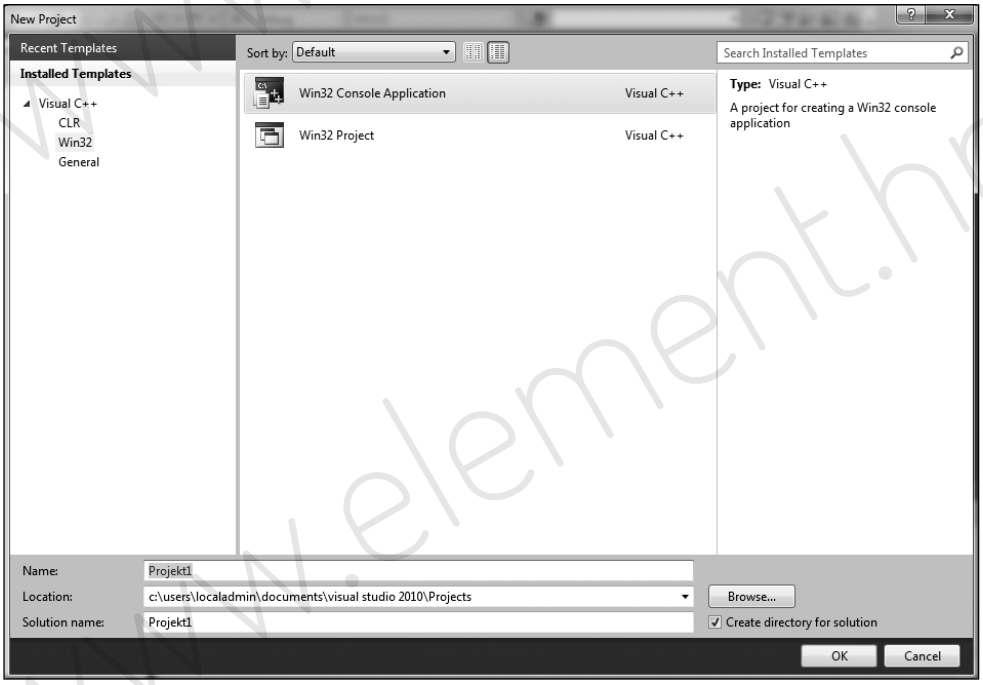
Odaberite stavku ***Install now – English***, a potom odaberite ***Run*** i slijedite naredne upute. Instalacija je vrlo jednostavna i sastoji se uglavnom od toga da pritišćete dugmad za nastavak. Nakon što prihvatite ugovor korištenja njihovih proizvoda, slijedi korak u kojem vas se pita želite li instalirati *Microsoft SQL Server 2010 Express Edition* i *Microsoft Silverlight* proizvode, pa tada, kako biste malo ubrzali proces instalacije, možete ukloniti odgovarajuće kvačice i time izuzeti spomenute pakete.

Stvaranje projekta

Pokrenite ORO ***Microsoft Visual C++ 2010 Express***, na uobičajen način pomoću izbornika ***Start***.

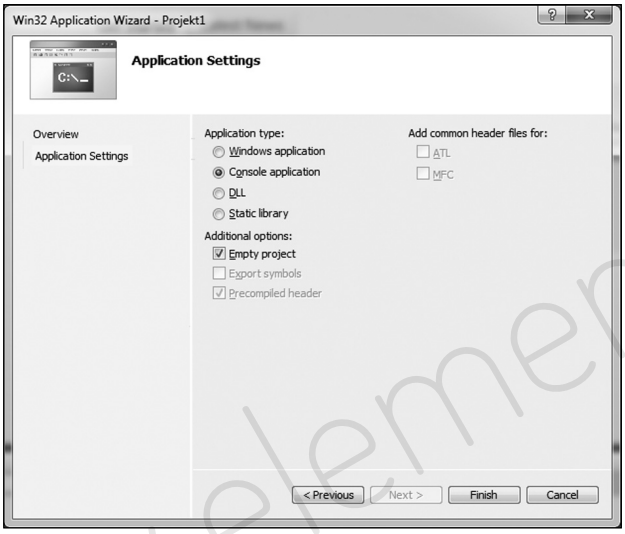
Nužno je najprije stvoriti novi projekt. Projekt je jedinica za upravljanje datotekama s izvornim kôdom i za upravljanje parametrima unutar ORO.

Za stvaranje novog projekta, odaberite izbornik ***File***, zatim ***New***, pa ***Project***, zatim odaberite ikonicu ***Win32 Console Application***, unesite naziv projekta, recimo *Projekt1*, uklonite kvačicu ***Create directory for solution***, a potom kliknite na dugme ***Ok***.



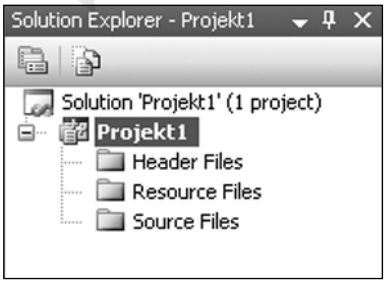
Slika 2.1.

U sljedećem prozoru odaberite stavku ***Application Settings***, a potom odaberite ***Empty project***:



Slika 2.2.

Ovim ste stvorili projekt, a kao rezultat će u razvojnoj okolini biti otvoren novi projekt imena *Projekt1*. Primijetite s lijeve strane okvir s nazivom *Solution explorer – Projekt*. Unutar tog okvira na raspolaganju ćemo imati pregledan prikaz svih datoteka koje budemo stvarali prilikom izrade naših računarskih programa.



Slika 2.3.

Za početak, u mapu pod nazivom ***Source Files*** dodat ćemo jednu datoteku za sadržavanje izvornog kôda. Kliknite desnom tipkom miša na mapu ***Source Files***, odaberite ***Add***, a potom ***New Item***. Provjerite da je odabrana ikonica ***C++ file (.cpp)*** i upišite naziv datoteke, recimo ***mojprogram***, a potom kliknite dugme ***OK***. U uređivaču teksta razvojne okoline otvorit će se datoteka ***mojprogram.cpp*** u koju možete pisati izvorni kôd vašeg programa.

Pokretanje programa

Sada bismo htjeli da računalo izvrši posljednji program iz prethodnog poglavlja. To postižemo "pokretanjem" programa. Ali, najprije nekoliko napomena.

Da biste program mogli pokrenuti, potreban mu je jedan maleni dodatak. Na početku programa potrebno je dodati sljedeća dva retka:

```
#include <iostream>
using namespace std;
```

Preskočit ćemo objašnjenje o svrsi ovog dodatka; u svakom slučaju, bez njega ne bismo mogli pokrenuti program. Za sada ćemo samo otprilike "prevesti" riječi koje se koriste. U prvom retku, engleska riječ *include* može se prevesti kao: uključiti. `iostream` je skraćeno od engleskog *input/output stream*, što znači ulazno-izlazni tok. Dakle, *"uključiti ulazno-izlazni tok"*. Drugi redak se može prevesti kao *"koristi uobičajeni prostor imena"*.

Sveukupno, posljednji program neka izgleda ovako:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "I pjevam:" << endl;
    for (double i=1;i<=3;i++)
        {
        cout << "Plovi plovi plovi" << endl;
        cout << "moja barka mala." << endl;
        }
    cout << "A ja ribu lovim!" << endl;
}
```

Prepišite navedeni program u prozor uređivača teksta objedinjene razvojne okoline.

Da biste pokrenuli program, u izborniku odaberite ***Debug –> Start without debugging*** (ili jednostavno pritisnite kombinaciju tipki **Ctrl-F5**). Ako se pojavi okvir *This project is out of date ... Would you like to build it?*, odaberite *Yes*.

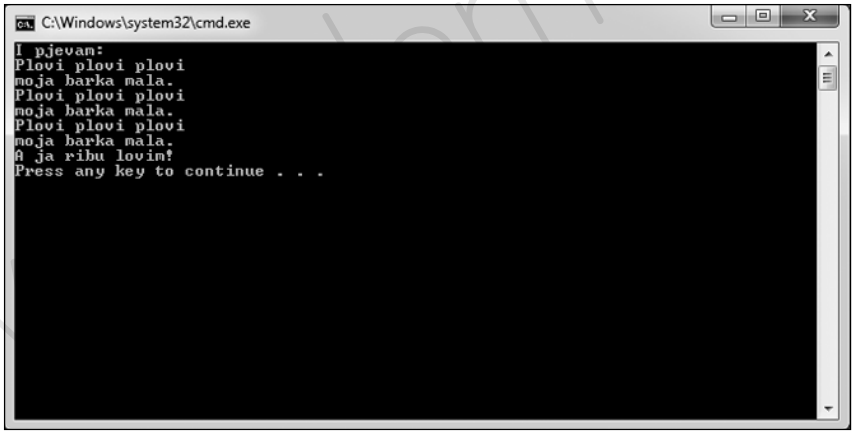
Kako utipkati znakove `{ i }`**?**

Znak `{` možete dobiti pritiskanjem **AltGr-B**, a znak `}` pritiskanjem **AltGr-N**.

A što je to **AltGr-B**?

Pritisnete tipku **AltGr** i, dok je držite pritisnutu, pritisnite tipku **B**.

Kao rezultat izvršavanja ovoga programa morali biste ugledati ovakav prozor:



Slika 2.4.

Voilà!

Pritisnite bilo koju tipku da biste završili izvršavanje programa.

Pokušajmo izmijeniti program tako da se refren ponavlja više puta. U izvornom kôdu programa, u naredbi `for`, promijenite brojku 3 u brojku 30. I, naravno, izvršite program.

Slika 2.5.

Nažalost, ovaj prozor nam, očito, prikazuje samo ograničeni broj redaka našega književnog remek-djela, što je velika i neprocjenljiva šteta. Da biste mogli vidjeti ostatak teksta, koristite traku za pomicanje s desne strane prozora.

Griješio sam, griješit ću

Sada ćemo u naš program namjerno unijeti jednu grešku da bismo vidjeli što će se dogoditi. Posljednju naredbu programa promijenite u

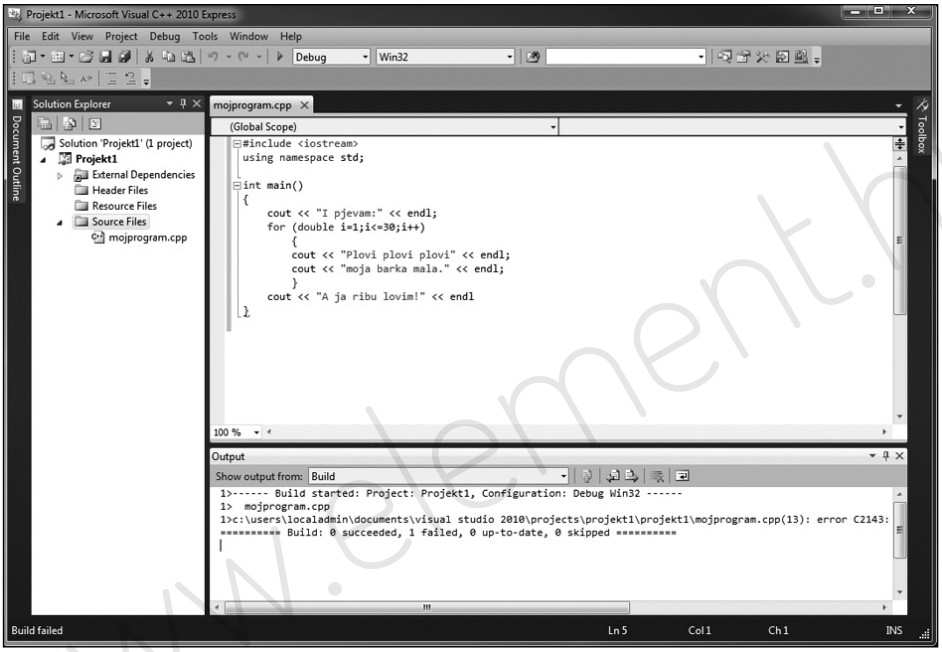
```
    cout << "A ja ribu lovim!" << endl
```

... odnosno izbrišite točku sa zarezom s kraja naredbe. Pokušajte pokrenuti program (s ***Debug –> Start without debugging***).

Vaša ORO će tada prikazati poruku da u programu postoji greška. U okviru koji se prikaže kliknite ***No***.

Program koji sadržava pravopisne pogreške nije moguće pokrenuti. Pravopisne pogreške u programima nazivaju se **sintaktičke pogreške** (engl. *syntax error*).

Greška je, naravno, otkrivena. U prozoru *Output* na dnu ORO nalazi se poruka o pogrešci, a sve zajedno izgleda otprilike ovako:



Slika 2.6.

Poruka o pogrešci u prozoru *Output* glasi otprilike ovako:

`c:\users\localadmin\documents\visual studio 2010\projects\projekt1\projekt1\mojprogram.cpp(13): error C2143: syntax error : missing ';' before '}'`	Ime i lokacija datoteke u kojoj se nalazi greška. 13 je redni broj retka s pogreškom. greška C2143: sintaktička pogreška: nedostaje znak ';' prije znaka '}'.

Vidimo da je prevodilac dosta dobro uspio zaključiti o kakvoj je pogrešci riječ – nedostaje točka sa zarezom, prije vitičaste zagrade.

Ako u prozoru *Output* dvokliknemo na redak s porukom o pogrešci, on će pomodrjeti, a u uređivaču teksta će se tamnozelenim znakićem označiti redak koji sadržava pogrešku. Napravite to, i pokušajte uočiti tamnozeleni znak.

Ipak, neće biti označen redak u kojem nedostaje točka sa zarezom, nego sljedeći redak, ali recimo da je to dovoljno blizu. Prevodilac ne može uvijek točno ustanoviti u kojem je retku pogreška i o kakvoj je točno pogrešci riječ, ali može barem otprilike, što je od velike pomoći, i o tomu će nas uvijek pokušati što detaljnije obavijestiti.

Eto, sada znate i kako izgleda kada učinite pogrešku u tipkanju (sintaktičku pogrešku).

Sve sintaktičke pogreške uvijek će biti otkrivene. Prevodiocu je program s pogreškom nerazumljiv, pa ga ne može prevesti. Tek kada se iz programa uklone sve sintaktičke pogreške, tako da program postane gramatički potpuno ispravan, tek se takav program može prevesti, a potom izvršiti uz pomoć računala.

Standardni tokovi

U prethodnom poglavlju napomenusmo da naredba `cout` ispisuje tekst i da se tada tekst ispisuje na standardni izlaz, a standardni izlaz je ime jednog od standardnih tokova… uh…

Standardni tokovi su:

- standardni izlaz (engl. *standard output*),
- standardni ulaz (engl. *standard input*),
- standardni tok za greške (engl. *standard error*), koji je izlazni tok, kao što je i standardni izlaz.

Standardni tokovi su (naravno) standardni načini na koje programi komuniciraju sa svojom okolinom. Kako smo naučili, s pomoću naredbe `cout` nalažemo ispis na standardni izlaz. Standarni izlaz možemo zamisliti kao nekakvu cijev koja vodi iz programa. Ta cijev može biti jako dugačka, i sa stajališta programa uopće nije moguće zaključiti kamo ta cijev vodi.

Ono što bude naloženo za ispis naredbom `cout`, program će gurnuti u cijev zvanu standardni izlaz. I ono što je gurnuto krenut će prema drugom kraju cijevi, a gdje završava drugi kraj, pitanje je sad.

Ako nije drukčije zadano, drugi kraj cijevi zvane standardni izlaz operativni sustav dužan je, na neki način, povezati s monitorom.

Slično tomu, u program ulazi jedan kraj cijevi zvane standardni ulaz, čiji će početak, ako to nije drukčije zadano, biti na neki način povezan s tipkovnicom.

Ili da to kažemo ovako: standardni izlaz je monitor, a standardni ulaz je tipkovnica. Eto, nije baš tako, al' je l' sada bar jasnije?

Po potrebi se standardni tokovi mogu preusmjeriti. Primjerice, standardni izlaz može se preusmjeriti tako da ide prema pisaču, ili prema datoteci.

Suma sumarum, naredba `cout` ispisuje na standardni izlaz. A to je, kao, monitor. Zapravo, to je onaj crni prozor koji se pojavi kada pokrenemo program. Taj crni prozor naziva se **konzola** (engl. *console*).

Ovime završavamo poglavlje o alatima koji su nam potrebni u svrhu olakšavanja rada s računalom. Procedura je zasigurno bila pomalo naporna, jer smo se umorili samo pišući o ovim detaljima. Ipak, trud koji ste uložili u odrađivanje ovog poglavlja sigurno će se isplatiti, jer sada imate postavljene i podešene sve potrebne alate. Korištenje ovih alata je jednostavno, te sada možemo barem ponekad računalu prepustiti izvršavanje naših programa.

U sljedećem poglavlju ponovno se vraćamo programiranju.