1

Rad sa znakovnim nizovima

U poglavlju 0 detaljno smo proučili mali program, koji nam je otkrio zapanjujuće velik broj elemenata jezika C++: komentare, standardna zaglavlja, oblasti važenja, imenske prostore, izraze, iskaze, literale znakovnih nizova i izlazni tok podataka. Nastavljamo da se bavimo osnovnim konceptima i predstavljamo jednostavan program koji koristi znakovne nizove. Pored toga, upoznaćemo vas sa deklaracijama, promenljivama, inicijalizacijom i bibliotekom `string`. Programi u ovom poglavlju su jednostavni, pa ne sadrže kontrolne strukture (poglavlje 2).

1.1 Ulaz

Pošto smo naučili da ispisujemo tekst, sledeći korak je, logično, čitanje teksta. Na primer, program `Zdravo, svete!` možemo izmeniti tako da pozdravlja određenu osobu:

```
// pita korisnika za ime, zatim ga pozdravlja
#include <iostream>
#include <string>

int main()
{
  // pita korisnika za ime
  std::cout << "Unesite ime: ";

  // učitava ime
  std::string name;      // definiše promenljivu name
  std::cin >> name;      // učitava sadržaj promenljive name

  // ispisuje pozdrav
  std::cout << "Zdravo, " << name  << "!" << std::endl;
return 0;
}
```

Kada se izvrši, program na standardni izlazni tok ispisuje:

```
Unesite ime:
```

Upišimo, na primer, ime:

```
Marija
```

Program će ispisati:

```
Zdravo, Marija!
```

Pogledajmo šta se ovde dešava. Da bismo pročitali ulazni tekst, moramo ga negde smestiti. To mesto zovemo ***promenljiva*** (engl. *variable*). Promenljiva je ***objekat*** sa imenom. Objekat je, s druge strane, deo memorije kome je pridružen tip. Razlika između objekata i promenljivih je važna jer objekti ne moraju da imaju ime (odeljci 3.2.2, 4.2.3 i 10.6.1).

Ako hoćete da upotrebite promenljivu, morate saopštiti prevodiocu njeno ime i tip da bi proizveo efikasan mašinski kôd programa. Ime i tip u definiciji promenljive neophodni su i da bi prevodilac pronašao greške u pisanju imena promenljivih (koje mogu promaći jedino ako se pogrešno uneto ime slučajno poklapa sa već upotrebljenim imenom u programu).

U ovom primeru, ime promenljive je *name*, a tip `std::string`. Kao što smo videli u odeljcima 0.5 i 0.7, prefiks `std::` ukazuje da ime koje sledi (u ovom slučaju `string`) pripada standardnoj biblioteci, ne jezičkom jezgru ili nekoj drugoj biblioteci. Kao i svaki element standardne biblioteke, imenu *std::string* pridruženo je zaglavlje, `<string>`, zbog čega smo u program uključili i odgovarajuću naredbu `#include`.

Prvi iskaz

```
std::cout << "Unesite ime: ";
```

treblo bi da vam je poznat: ispisuje molbu za unos imena korisnika. U ovom slučaju izostavljen je manipulator `std::endl`. Zbog toga se ne prelazi u novi red nakon ispisivanja poruke. Umesto toga, program će, pošto ispiše molbu, zadržati kursor u istom redu očekujući da unesete ime.

Sledeći iskaz

```
std::string name;       // definiše promenljivu name
```

jeste ***definicija*** (engl. *definition*), kojom pravimo promenljivu `name` tipa `std::string`. U pitanju je ***lokalna promenljiva*** jer se pojavljuje unutar tela funkcije. Lokalna promenljiva postoji samo dok se izvršava deo programa unutar vitičastih zagrada. Kada računar naiđe na desnu vitičastu zagradu (}), uništiće promenljivu `name` i osloboditi memoriju koju je ona zauzimala. Pošto je postojanje lokalnih promenljivih ograničeno, važno je razlikovati promenljive i objekte.

Tip objekta posredno određuje ***interfejs*** (engl. *interface*) objekta, odnosno skup operacija koje su izvodljive nad objektima tog tipa. Time što promenljivoj (dakle imenovanom objektu) dodeljujemo tip `string`, podrazumevamo da s njom želimo raditi sve što biblioteka dozvoljava da se radi sa znakovnim nizovima.

Jedna od operacija koje se sprovode nad znakovnim nizovima jeste ***inicijalizacija*** (engl. *initialization*). Definisanje promenljive tipa `string` podrazumeva i inicijalizaciju jer standardna biblioteka traži da svaki objekat tipa `string` ima vrednost čim se napravi. Uskoro ćemo videti da i sami možemo da dodelimo početnu vrednost prilikom stvaranja znakovnog niza. Ako to ne učinimo, znakovni niz neće sadržati nijedan znak. Takav znakovni niz je ***prazan***.

Kada definišemo promenljivu *name*, izvršavamo iskaz

```
std::cin >> name;       // upisuje u promenljivu name
```

koji čita tok podataka `std::cin` i sadržaj upisuje u promenljivu *name*. Kao što se za izlaz upotrebljavaju operator << i tok podataka `std::cout`, biblioteka za ulaz koristi **operator >>**. U navedenom primeru, operator >> čita znakovni niz sa standardnog ulaznog

toka, a zatim ga smešta u objekat *name*. Kada zatražimo od biblioteke da učita znakovni niz, ona prvo zanemaruje ***razmake*** (odnosno razmaknicu, tabulator, znak za brisanje unazad ili kraj reda) u ulaznom toku podataka, a zatim u promenljivu `name` upisuje znakove dok ne naiđe na nov razmak ili kraj datoteke. Rezultat izraza `std::cin >> name` jeste čitanje sa standardnog ulaznog toka i smeštanje pročitane reči u promenljivu `name`.

Sporedni efekat operacije unošenja podataka jeste ispis zahteva za korisničkim imenom na izlaznom uređaju računara. Ulazno/izlazna biblioteka čuva ulazno-izlazne vrednosti u unutrašnjoj strukturi podataka nazvanoj ***bafer*** (engl. *buffer*) kojom optimizuje izlazne operacije. Većini sistema treba mnogo vremena da ispišu znakove na izlazni uređaj, bez obzira na njihov broj. Da bi ubrzala ispisivanje, biblioteka u baferu nagomilava znakove koji treba da se ispišu i po potrebi ga ***prazni*** ispisujući njegov sadržaj na izlazni uređaj. Na taj način biblioteka kombinuje nekoliko operacija u jednom ispisu.

Postoje tri situacije u kojima sistem prazni bafer. Ako je bafer pun, biblioteka ga automatski prazni. Dalje, ukoliko biblioteka dobije zahtev da čita iz standardnog ulaznog toka podataka, isprazniće bafer i ako nije pun. Na kraju, biblioteka prazni bafer ako joj izričito saopštimo da to uradi.

Kada program upiše zahtev u objekat `cout`, taj tekst se smešta u bafer pridružen standardnom izlaznom toku podataka. Čitanjem iz toka `cin` praznimo bafer toka `cout`, zbog čega će korisnik videti poruku.

Sledećim iskazom koji stvara izlazni tekst izričito nalažemo biblioteci da isprazni bafer. Ovaj iskaz je nešto komplikovaniji od iskaza koji je ispisivao poruku. Ovde ćemo nakon literala znakovnog niza, `"Zdravo, "` napisati vrednost promenljive `name` tipa `string`, a potom i `std::endl`, kojim završavamo izlazni red. Posle toga se bafer prazni i sistem trenutno ispisuje sadržaj izlaznog toka podataka.

Pražnjenje izlaznih bafera u pogodnim trenucima bi trebalo da vam postane navika ako pišete programe koji se dugo izvršavaju. Ako ne praznite bafere, može se desiti da neki izlazni podaci programa „životare“ u sistemskim baferima dugo pre nego što ih korisnik vidi.

1.2 Uokvirite ime

Program je dosad bio prilično ćutljiv. Ako ga malo izmenimo, dobrodošlica će biti toplija, a ulazno/izlazni tok podataka izgledaće ovako:

```
Unesite ime: Ana
* * * * * * * * * * * * * *
*                         *
*   Zdravo, Ana!          *
*                         *
* * * * * * * * * * * * * *
```

Program ispisuje rezultat u pet redova. Prvi red, tj. gornju ivicu okvira čini niz zvezdica (*) duži od imena za onoliko znakova koliko ima pozdrav (`"Zdravo, "`) (na to treba dodati još i razmak i zvezdicu (*) na kraju svakog reda). Drugi red sadrži odgovarajući broj razmaka između dve zvezdice. Treći red sadrži zvezdicu, razmak, poruku, razmak i zvezdicu. Poslednja dva reda su ista kao drugi, odnosno prvi red.

Poželjno je da izlazni tekst pravite deo po deo. Prvo čitamo ime koje ćemo upotrebiti da bismo napravili pozdrav, zatim pomoću pozdrava gradimo jedan po jedan izlazni red. Evo programa koji ostvaruje ovu ideju:

```
// traži ime korisnika, a zatim ispisuje uokviren pozdrav
#include <iostream>
#include <string>

int main()
{
  std::cout << "Unesite ime: ";
  std::string name;
  std::cin >> name;

  // pravi poruku koju hoćemo da ispišemo
  const std::string greeting = "Zdravo, " + name + "!";

  // pravi drugi i četvrti izlazni red
  const std::string spaces(greeting.size(), ' ');
  const std::string second = "* " + spaces + " *";

  // pravi prvi i peti izlazni red
  const std::string first(second.size(), '*');

  // ispisuje sve
  std::cout << std::endl;
  std::cout << first << std::endl;
  std::cout << second << std::endl;
  std::cout << "* " << greeting << " *" << std::endl;
  std::cout << second << std::endl;
  std::cout << first << std::endl;

  return 0;
}
```

Program prvo traži ime korisnika, a zatim ga upisuje u promenljivu `name`. Nakon toga definiše promenljivu `greeting` za čuvanje poruke koju namerava da ispiše. Zatim definiše promenljivu `spaces`, sa razmacima u dužini teksta u promenljivoj `greetings`. Pomoću promenljive `spaces`, definiše se promenljiva `second`, koja će sadržati drugi izlazni red. Onda se pravi promenljiva `first` i u nju se smešta znakovni niz od zvezdica (*) iste dužine kao promenljiva `second`. Na kraju, program ispisuje jedan po jedan izlazni red.

Naredba `#include` i prva tri reda programa su vam poznati. Definicija promenljive `greeting`, s druge strane, otkriva nam tri nove ideje.

Prvo, promenljivu možete istovremeno i definisati i dodeliti joj vrednost, tako što ćete u definiciji između imena promenljive i tačke i zareza dopisati znak jednakosti (=) i vrednost promenljive. Ako promenljiva i vrednost nisu istog tipa, kao što se, na primer, razlikuju tip `string` i literal znakovnog niza (odeljak 10.2), prevodilac će ***pretvoriti*** tip početne vrednosti u tip promenljive.

Dalje, pomoću simbola + možete ***nadovezati*** znakovni niz u objektu tipa `string` i literal znakovnog niza, ili dva objekta tipa `string` (ali ne i dva literala znakovnog niza). U poglavlju 0 pomenuli smo da izraz `3 + 4` ima vrednost 7. Ovde znak + ima sasvim drugačije značenje. U svakom slučaju, možemo utvrditi šta operator + radi na osnovu tipa operanada. Za operator čije se značenje menja u zavisnosti od tipa operanada kažemo da je ***preklopljen*** (engl. *overloaded*).

Iz navedenog primera možemo još naučiti šta u definiciji promenljive znači rezervisana reč **`const`**. Ako promenljivu definišemo kao konstantnu, saopštavamo prevodiocu da njenu vrednost nikada nećemo menjati. U pomenutom primeru nismo ništa dobili time što smo upotrebili rezervisanu reč `const`, osim što smo, da biste lakše razumeli program, ukazali na promenljive čija se vrednost neće menjati.

Konstantnu promenljivu morate inicijalizovati dok je definišete jer kasnije neće biti prilike za to. Važno je da znate i to da početna vrednost konstantne promenljive ne mora da bude konstanta. U našem primeru, vrednost promenljive `greeting` ne znamo dok je ne upišemo u promenljivu `name`, što je moguće tek kada se program pokrene. Ne možemo da kažemo da je promenljiva `name` konstantna jer se njena vrednost može menjati svaki put kada u nju nešto upisujemo.

Asocijativnost operatora je nepromenljiva. U poglavlju 0, naučili ste da je operator `<<` levo asocijativan, tako da izraz `std::cout << s <<t` ima isto značenje kao i izraz `(std::cout << s) <<t`. Svojstvo leve asocijativnosti imaju i operatori + i >>. U skladu s tim, vrednost izraza `"Zdravo, " + name + " ! "` rezultat je nadovezivanja literala `"Zdravo, "` s promenljivom `name`, a potom sa `" ! "`. Na primer, ako promenljiva `name` sadrži znakovni niz `Ana`, vrednost izraza `"Zdravo, " + name + " ! "` biće `"Zdravo, Ana!"`.

Pošto smo saznali ime osobe i tekst odgovarajućeg pozdrava smestili u promenljivu `greeting`, treba da napravimo okvir za pozdrav. To radimo pomoću sledećeg iskaza u kome koristimo tri nova elementa jezika:

```
std::string spaces(greeting.size(), ' ');
```

Kada smo definisali promenljivu `greeting`, upotrebili smo znak = da bismo je inicijalizovali. U prethodnom iskazu smo nakon imena promenljive `spaces` u zagradama dopisali dva izraza razdvojena zarezom. Ako upotrebite znak =, izričito saopštavate vrednost promenljive. Zagradama u definiciji ukazujete prevodiocu da treba da konstruiše promenljivu (`spaces`, u ovom slučaju) na osnovu izraza, poštujući tip promenljive. Drugim rečima, ovu definiciju nećete razumeti ako vam nije jasno šta znači konstruisati objekat tipa `string` pomoću dva izraza.

Konstruisanje promenljive zavisi potpuno od njenog tipa. U ovom slučaju pravimo znakovni niz (još samo kad bismo znali iz čega?). Oba izraza imaju oblik koji do sada nismo videli. Šta ovi izrazi znače?

Prvi izraz, `greeting.size()`, primer je pozivanja ***funkcije članice*** (engl. *member function*). Objekat `greetings` sadrži funkciju *size*, koju pozivamo da bismo dobili neku vrednost. Promenljiva `greeting` je tipa `std::string`, pa je rezultat funkcije `greeting.size()` celobrojna vrednost koja predstavlja broj znakova u promenljivoj `greeting`.

Drugi izraz, `' '`, predstavlja ***literal znaka***. Literali znaka se potpuno razlikuju od literala znakovnog niza. Prvi se pišu između dva polunavodnika, a drugi pod navodnicima. Tip literala znaka je ugrađeni tip, **`char`**, dok je tip literala znakovnog niza složeniji (odeljak 10.2). Literal znaka predstavlja jedan znak. Simboli koji u literalima znakovnog niza imaju posebna značenja tumače se na isti način i u literalima znaka.

Ako nam je potreban znak ' ili \, pre njega treba dopisati obrnutu kosu crtu (\). Značenje izlaznih sekvenci, poput '\n', '\t', '\"' isto je kao i u literalima znakovnih nizova (poglavlje 0).

Da biste potpuno razumeli promenljivu spaces, zapamtite da se objekat tipa string, koji je napravljen pomoću celog broja i znaka, sastoji samo od prosleđenog znaka u dužini koja je zadata prvim parametrom. Pogledajmo sledeću definiciju:

```
std::string stars(10, '*');
```

Rezultat funkcije stars.size() je 10, što znači da promenljiva stars sadrži 10 zvezdica (**********).

Promenljiva spaces, znači, sadrži isti broj znakova (u njenom slučaju razmaka) kao i promenljiva greeting.

Nisu potrebna dodatna objašnjenja da biste razumeli definiciju promenljive second: nadovezali smo literal znakovnog niza "* ", objekat tipa string koji sadrži razmake i literal " *", da bismo dobili uokvirenu poruku. Lako ćete razumeti i definiciju promenljive first, u kojoj smo dodelili onoliko znakova * koliko sadrži promenljiva second.

Ostatak programa izgleda poznato jer ispisuje znakovne nizove kao što smo naveli u odeljku 1.1.

1.3 Detalji

Tipovi:

char	Ugrađen tip za obične znakove koje koristi izvršno okruženje.
wchar_t	Ugrađen tip namenjen "širokom znakovima", dovoljno velik za znakove zahtevnih jezika kakav je japanski.

Tip string je definisan u standardnom zaglavlju <string>. Objekat tipa string sadrži niz proizvoljnog broja znakova. Neka je n tipa integer, c je tipa char, promenljiva is predstavlja ulazni tok podataka, a os izlazni tok podataka.

Evo nekih bitnijih operacija nad znakovnim nizovima:

`std::string s;`
: Definiše promenljivu s tipa std::string, koja je prazna.

`std::string t=s;`
: Definiše promenljivu t tipa std::string čija je početna vrednost kopija znakova iz promenljive s, pri čemu *s* može da bude objekat tipa string ili literal znakovnog niza.

`std::string z(n, c);`
: Definiše promenljivu z tipa std::string čija se početna vrednost sastoji od n kopija znaka c (c ovde mora biti tipa char, nikako string ili literal znakovnog niza).

`os << s`
: Znakove iz promenljive s upisuje u izlazni tok podataka koji označen sa os. Rezultat izraza je os.

`is >> s`
: Čita znakove iz ulaznog toka podataka s. Zanemaruje sve razmake. Zatim sve znakove iz is do prvog razmaka upisuje uzastopno preko postojećih znakova u s. Rezultat izraza je is.

`s + t` Rezultat ovog izraza je znakovni niz tipa `std::string`, koji sadrži kopiju znakova iz promenljive `s` na koju je nadovezana kopija znakova iz `t`; `s` ili `t` (ne istovremeno) mogu biti literal znakovnog niza ili vrednost tipa `char`.

`s.size()` Vraća dužinu znakovnog niza `s`.

Promenljive možete definisati na tri načina:

```
std::string hello = "Zdravo";      // definiše promenljivu izričito
                                   // joj dodeljujući početnu vrednost

std::string stars (100, '*');      // pravi promenljivu u skladu s njenim
                                   // tipom i datim izrazom

std::string name;                  // definiše promenljivu, posredno je
                                   // inicijalizuje zavisno od njenog tipa
```

Promenljive definisane unutar vitičastih zagrada su lokalne i traju dok se ne izvrši deo programa u vitičastim zagradama. Kada prevodilac naiđe na desnu vitičastu zagradu, uništava lokalne promenljive i oslobađa memoriju koju su zauzimale.

Ako promenljivu definišete pomoću rezervisane reči `const`, obavezujete se da joj se vrednost neće menjati dok postoji. Konstantnu promenljivu morate inicijalizovati kada je i definišete, jer kasnije neće biti prilike da to uradite.

Čitanje: Izraz `std::cin >> v` zanemaruje razmake u standardnom ulaznom toku podataka, zatim upisuje reč iz standardnog ulaznog toka u promenljivu `v`. Rezultat izraza je `std::cin`, tipa `istream`, čime se omogućava nadovezivanje drugih operacija vezanih za ulaz.

Vežbe

1-0. Prevedite, izvršite i proverite programe iz ovog poglavlja.

1-1. Da li su sledeće definicije prihvatljive? Obrazložite odgovor.

```
const std::string hello = "Zdravo!";
const std::string message = hello + ", svete" + "!";
```

1-2. Šta mislite o valjanosti sledećih definicija? Obrazložite odgovor.

```
const std::string exclam = "!";
const std::string message = "Zdravo" + ", svete" + exclam;
```

1-3. Ima li grešaka u sledećem programu? Obrazložite mišljenje.

```
#include <iostream>
#include <string>

int main()
{
  { const std::string s = "znakovni niz";
    std::cout << s << std::endl;}

  { const std::string s = "drugi znakovni niz";
    std::cout << s << std::endl;}
  return 0;
}
```

1-4. Da li je sledeći program dobar? Šta će se desiti ako u trećem redu odozdo umesto }} napišemo };}?

```
#include <iostream>
#include <string>

int main()
{
  { const std::string s = "znakovni niz";
    std::cout << s << std::endl;
  { const std::string s = "drugi znakovni niz";
    std::cout << s << std::endl;}}
  return 0;
}
```

1-5. Da li ima grešaka u sledećem programu? Obrazložite odgovor. Ako ima grešaka, ispravite ih.

```
#include <iostream>
#include <string>

int main()
{
  {std::string s = "znakovni niz";
  {std::string x = s + ", zaista";
    std::cout << s << std::endl;}
    std::cout << x << std::endl;
  }
  return 0;
}
```

1-6. Objasnite šta sledeći program radi ako, upišete ime od dve reči (na primer `Vlade Divac`)? Pre nego što pokrenete program, pokušajte da predvidite šta će se desiti, nakon toga ga prevedite i izvršite.

```
#include <iostream>
#include <string>

int main()
{
  std::cout << "Kako se zovete? ";
  std::string name;
  std::cin >> name;
  std::cout << "Zdravo, " << name
            << std::endl << "A kako se vi zovete? ";
  std::cin >> name;
  std::cout << "Zdravo, " << name
            << "; drago mi je što smo se upoznali!" << std::endl;
  return 0;
}
```