Siddhartha Rao

OSMO IZDANJE

Pokriva **C++14** i **C++17**

NAUČITE SAMOSTALNO C++

jedna**lekcija**DNEVNO

kompjuter biblioteka

SAMS

Siddhartha Rao

NAUČITE SAMOSTALNO

C++

OSMO IZDANJE

jednalekcijaDNEVNO

kompjuter biblioteka

SAMS

Izdavač:

kompjuter biblioteka

Obalskih radnika 15, Beograd

Tel: 011/2520272

e-mail: kombib@gmail.com

internet: www.kombib.rs

Urednik: Mihailo J. Šolajić

Za izdavača, direktor:
Mihailo J. Šolajić

Autor: Siddhartha Rao
Prevod: Slavica Prudkov

Lektura: Miloš Jevtović

Slog : Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun

Tiraž: 500

Godina izdanja: 2017.

Broj knjige: 490

Izdanje: Prvo

ISBN: 978-86-7310-513-0

Sams Teach Yourself C++
in One Hour a Day

by Siddhartha Rao
ISBN: 978-0-7897-5774-6

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд,
се добија на захтев

O autoru

Siddhartha Rao je zadužen za Security Response u SAP SE-u za vodeći svetski trgovinski softver. Evolucija C++-a ubedila je Siddharthau da je danas moguće programiranje bržih, jednostavnijih i moćnijih aplikacija nego ikada do sada. Voli da putuje i strastveni je biciklist.

Kratak sadržaj

Sadržaj

LEKCIJA 2

LEKCIJA 3

LEKCIJA 17

LEKCIJA 18

LEKCIJA 19

LEKCIJA 22

LEKCIJA 23

LEKCIJA 24

Uvod

Za C++ su bile posebno važne 2011. i 2014. godina. Iako je C++11 označio dramatično poboljšanje predstavljanjem novih ključnih reči i struktura koje su poboljšale efikasnost programera, C++14 je doneo dodatna poboljšanja dodavanjem završne obrade za funkcije koje su predstavljene u verziji C++11.

Ova knjiga će vam pomoći da detaljno naučite C++. Pažljivo je podeljena na lekcije pomoću kojih čete naučiti osnove ovog objektno-orijentisanog programskog jezika sa praktične tačke gledišta. U zavisnosti od nivoa stručnosti, moći ćete da savladate C++, vežbajući dnevno po jedan sat.

Praktično učenje C++-a je najbolji način učenja – stoga, isprobajte različite primere koda u ovoj knjizi i poboljšajte svoju stručnost u programiranju. Ovi isečci koda su testirani pomoću najnovijih verzija dostupnih kompajlera u vreme kada je knjiga pisana, konkretno Microsoft Visual C++ kompajlera za C++ i GNU-ovog C++ kompajlera, koji obezbeđuju bogatu pokrivenost funkcija verzije C++14.

Ko treba da čita ovu knjigu?

Knjiga započinje opisom osnova C++-a. Da biste razumeli kako sve funkcioniše, potrebne su vam želja za učenjem ovog programskog jezika i radoznalost. Postojeće znanje C++ programiranja može biti prednost, ali nije preduslov. Ovo je, takođe, knjiga koju možete da koristite kao referencu ako već znate C++, ali želite da naučite i poboljšanja koja su dodata u taj jezik. Ako ste profesionalni programer, Deo III, „*Učenje o biblioteci Standard Template Library (STL)*“, pomoći će vam da kreirate bolje, praktičnije C++ aplikacije.

NAPOMENA Posetite veb sajt izdavača i registrujte ovu knjigu na adresi informit.com/register za lakši pristup ažuriranjima, fajlovima za preuzimanje i spiskovima grešaka dostupnim za ovu knjigu.

Organizacija ove knjige

U zavisnosti od aktuelnog nivoa stručnosti u C++-u, možete da izaberete odeljak od kojeg želite da započnete učenje iz ove knjige. Koncepti koje su predstavile verzije C++11 i C++14 su podeljeni u pet delova:

- Deo I, „*Osnove*“, započinje učenjem pisanja jednostavnih C++ aplikacija. Upoznaćete ključne reči koje se najčešće viđaju u C++ kodu promenljive, bez kompromitovanja bezbednosti tipa.
- Deo II, „*Osnove objektno-orijentisanog C++ programiranja*“, posvećen je konceptu klasa. Naučićete kako C++ podržava važne principe objektno-orijentisanog programiranja, kapsuliranja, apstrakcije, nasleđivanja i polimorfizma. O konceptu konstruktora move naučićete u Lekciji 9, „Klase i objekti“, a o operatoru dodele move u Lekciji 12, „Tipovi operatora i preklapanje operatora“. Ove funkcije performanse će vam pomoći da smanjite neželjene i nepotrebne korake kopiranja, poboljšavajući performansu aplikacije. Lekcija 14, „Predstavljanje makroa i šablona“, je „odskočna daska“ za pisanje generičkog C++ koda.
- Deo III, „*Učenje o biblioteci Standard Template Library (STL)*“, pomoći će vam da napišete efikasan i praktičan C++ kod, koristeći STL string klasu i kontejnere. Saznaćete kako kod std::string čini jednostavne operacije spajanja znakovnog niza bezbednim i jednostavnim i zašto više ne treba da koristite char* znakovne nizove. Moći ćete da upotrebite STL dinamičke nizove i povezane liste, umesto da programirate svoje.
- Deo IV, „*Više STL-a*“, fokusiran je na algoritme. Naučićete da pomoću iteratora koristite sort u kontejnerima, kao što je vector. U ovom delu otkrićete kako je ključna reč auto, koja je predstavljena u verziji C++11, značajno smanjila dužinu deklaracija iteratora. U Lekciji 22, „Lambda izrazi“, predstavljena je moćna nova funkcija koja značajno smanjuje kod kada se koriste STL algoritmi.
- Deo V, „*Napredni C++ koncepti*“, sadrži prikaz mogućnosti ovog jezika, kao što su pametni pokazivači i rukovanje izuzecima, koje nisu neophodne u C++ aplikacijama, ali u značajnoj meri pomažu prilikom povećanja stabilnosti i kvaliteta. Ovaj deo se završava napomenama o najboljoj praksi za pisanje dobrih C++ aplikacija i predstavljanjem novih funkcija koje se mogu očekivati u sledećoj verziji ISO standarda pod nazivom C++17.

Konvencije upotrebljene u ovoj knjizi

Unutar lekcija pronaći ćete sledeće elemente koji obezbeđuju dodatne informacije:

NAPOMENA Ovi okviri obezbeđuju dodatne informacije u vezi onoga što čitate.

PAŽNJA Ovi okviri upozoravaju na probleme ili sporedne efekte koji se mogu desiti u posebnim situacijama.

SAVET Ovi okviri predstavljaju najbolju praksu u pisanju C++ programa.

URADITE	NE RADITE
Upotrebite „Uradite/Ne radite“ okvire da biste pronašli rezime osnovnih principa lekcije.	Nemojte da previdite korisne informacije koje se nalaze u ovim okvirima.

U ovoj knjizi upotrebljeni su različiti fontovi da biste razlikovali kod i običan tekst. Kod, komande i termini koji se odnose na programiranje prikazani su računarskim fontom.

Primeri koda za ovu knjigu

Primeri koda za ovu knjigu su dostupni na Internetu za preuzimanje na veb sajtu izdavača.

LEKCIJA 1

Početak rada

Dobrodošli u Sams Teach Yourself C++ za sat vremena dnevno! Znači da ste spremni da postanete vešt C++ programer.

U ovoj lekciji otkrićete:

- zašto je C++ standard u razvoju softvera
- kako da unesete, kompajlirate i povežete prvi radni C++ program
- šta je novo u C++-u

Kratka istorija C++-a

Svrha programskog jezika je da potrošnju računarskih resursa učini jednostavnijom. C++ nije nov - to je jezik koji je popularan i neprestalno se razvija. Njegov najnovija verzija, koju je ratifikovao ISO (International Organization for Standardization), popularno se naziva C++14, a izdata je u decembru 2014. godine.

Povezanost sa C-om

C++ je projektovan da bude naslednik jezika C, od koga se razlikuje po činjenici da je dizajniran da bude objektno-orijentisani jezik, koji implementira koncepte, kao što su nasleđivanje, apstrakcija, polimorfizam i kapsuliranje. Uključuje klase koje se koriste za skladištenje podataka člana i metoda člana. Ovi metodi člana funkcionišu upotrebom podataka člana. Efekat ove organizacije je da programer modeluje podatke i akcije koje želi da izvrši. Mnogi popularni C++ kompajleri nastavili su da podržavaju i jezik C.

> **NAPOMENA** Poznavanje C programiranja nije preduslov za učenje jezika C++. Ako je vaš cilj da naučite objektno-orijentisani programski jezik, kao što je C++, ne treba da započinjete učenjem proceduralnog jezika, kao što je C.

Prednosti C++-a

C++ se smatra programskim jezikom srednjeg nivoa, što znači da omogućava programiranje aplikacija visokog nivoa i biblioteka niskog nivoa koje funkcionišu blisko sa hardverom. On obezbeđuje optimalnu kombinaciju jezika visokog nivoa koji omogućava izradu složenih aplikacija dok pruža fleksibilnost u omogućavanju programeru da izvuče najbolju performansu tačnom kontrolom potrošnje resursa i dostupnosti.

Uprkos postojanja novijih programskih jezika, kao što su Java i drugi jezici zasnovani na .NET-u, C++ je i dalje veoma popularan i još uvek se razvija. Noviji jezici obezbeđuju neke funkcije, kao što je upravljanje memorijom putem garbage kolekcije implementirane u izvršnoj komponenti, koje su ove jezike učinile omiljenim za neke programere. Ipak, C++ ostaje izabrani jezik u slučajevima gde je potrebna tačna kontrola nad potrošnjom resursa aplikacije i performansa. Slojevita arhitektura, gde veb server programiran u C++-u služi drugim komponentama programiranim u HTML-u, Java-i ili .NET-u, je uobičajena.

Evolucija C++ standarda

C++ jezik je prihvaćen i usvojen na mnogim različitim platformama koje koriste sopstvene C++ kompajlere. Njegov razvoj je izazvao odstupanja specifična za kompajlere i, samim tim, probleme interoperabilnosti i probleme promene platforme. Stoga se pojavila potreba za standardizacijom jezika i proizvođačima kompajlera su obezbeđene standardne specifikacije jezika koje treba da koriste.

ISO Committee je 1998. godine ratifikovao prvu standardnu verziju jezika C++ ISO/IEC 14882:1998. Od tada je taj standard pretrpeo velike promene, pomoću kojih je poboljšana upotrebljivost jezika i proširena podrška standardne biblioteke. U vreme pisanja ove knjige aktuelna ratifikovana verzija standarda je ISO/IEC 14882:2014, neformalno nazvana C++14.

> **NAPOMENA** Aktuelni standard možda nisu odmah ili u potpunosti podržali svi popularni kompajleri. Stoga, iako je dobro znati najnovije dodatke standarda sa akademske tačke gledišta, imajte na umu da oni nisu preduslov za pisanje dobrih, funkcionalnih C++ aplikacija.

Ko koristi programe napisane u C++-u?

Lista aplikacija, operativnih sistema, veb servisa i baza podataka i poslovnog softvera programiranih u jeziku C++ je veoma duga. Bez obzira šta ste i šta radite na računaru, postoji mogućnost da već koristite softver programiran u jeziku C++. Pored softverskih inženjera, i fizičari i matematičari često koriste C++ za istraživački rad.

Programiranje C++ aplikacije

Kada pokrenemo Notepad na Windowsu ili Terminal na Linuxu, mi, u stvari, ukazujemo procesoru da treba da pokrene izvršni fajl konkretnog programa. Izvršni fajl je završeni proizvod koji može da se pokrene i trebalo bi da izvrši ono što je programer želeo da postigne.

Koraci za generisanje izvršnog fajla

Pisanje C++ programa je prvi korak ka kreiranju izvršnog fajla koji može da se pokrene na operativnom sistemu. Osnovni koraci u kreiranju aplikacija u C++ jeziku su sledeći:

1. pisanje (ili programiranje) C++ koda pomoću editora za tekst
2. kompajliranje koda pomoću C++ kompajlera koji konvertuje kod u verziju mašinskog jezika koji se nalazi u objektnim fajlovima

3. povezivanje ispisa kompajlera pomoću programa za povezivanje da bi bio dobijen izvršni fajl (na primer, .exe u Windowsu)

Kompajliranje je korak u kojem je kod u C++ jeziku, koji je sadržan u tekstualnim fajlovima i ima ekstenziju .cpp, konvertovan u kod bajtova koji procesor može da izvrši. Kompajler konvertuje po jedan fajl koda generisanjem objektnog fajla koji ima ekstenziju .o ili .obj i ignoriše zavisnosti koje ovaj CPP fajl može imati za kod u drugom fajlu. Program za povezivanje spaja tačke i rešava ove zavisnosti. Prilikom uspešnog povezivanja program za povezivanje kreira izvršni fajl koji programer može da izvrši ili distribuira. Ceo ovaj proces se naziva i izgradnja izvršnog fajla.

Analiziranje i ispravljanje grešaka

Većina aplikacija se retko pri prvom pokretanju kompajlira i izvršava kao što bi trebalo. Velika ili složena aplikacija programirana u bilo kom jeziku (uključujući i C++) treba da se pokrene više puta, što je deo testiranja i identifikovanja grešaka u kodu, koje se nazivaju programske *greške*. Nakon što su programske greške ispravljene, izvršni fajl je ponovo izgrađen i proces testiranja se nastavlja. Stoga, pored tri koraka (programiranja, kompajliranja i povezivanja), razvoj softvera uključuje i korak koji se naziva ispravljanje grešaka, u kojem programer analizira greške u kodu i ispravlja ih. Dobra razvojna okruženja obezbeđuju alatke i funkcije koje pomažu pri ispravljanju grešaka.

Integrisano razvojno okruženje

Mnogi programeri radije koriste integrisano razvojno okruženje (IDE) u kojem su koraci programiranja, kompajliranja i povezivanja integrisani unutar jedinstvenog korisničkog interfejsa, koji, takođe, obezbeđuje funkcije za ispravljanje grešaka koje olakšavaju detektovanje grešaka i rešavanje problema.

SAVET Najbrži način da se započnu pisanje, kompajliranje i izvršavanje C++ aplikacija je upotreba online IDE-a koji se pokreću u pretraživaču. Pogledajte jednu takvu alatku na adresi http://www.tutorialspoint.com/compile_cpp_online.php. Pored toga, instalirajte jedan od mnogih besplatnih C++ IDE-ova i kompajlera. Popularni su Microsoft Visual Studio Express za Windows i GNU C++ Compiler, pod nazivom g++ za Linux. Ako programirate na Linuxu, možete da instalirate besplatan Eclipse IDE za kreiranje C++ aplikacija, koristeći g++ kompajler.

URADITE	NE RADITE
Snimite fajlove, koristeći ekstenziju .cpp. Upotrebite jednostavan editor teksta ili integrisano razvojno okruženje za pisanje koda.	**Nemojte** da upotrebite ekstenziju .c za C++ fajl, zato što će neki kompajleri prevesti ove fajlove kao C programe, umesto kao C++. **Nemojte** za pisanje koda da koristite velike editore teksta, kao što su programi za obradu teksta, zato što oni često dodaju sopstvene oznake u kod koji se programira.

Programiranje prve C++ aplikacije

Sada, kada poznajete alatke i korake koji su uključeni, vreme je da programirate prvu C++ aplikaciju koja prati tradiciju i prikazuje „Hello World!“ na ekranu.

Ako programirate na Linuxu, upotrebite jednostavan editor teksta za kreiranje CPP fajla koji ima sadržaj prikazan u programskom kodu 1.1.

Ako koristite Microsoft Visual Studio na Windowsu, pratite sledeće korake:

1. Otvorite New Project Wizard pomoću opcije menija File, New Project.
2. Za opciju Visual C++ izaberite tip Win32 Console i projektu dodelite naziv `Hello`. Kliknite na OK.
3. U odeljku Application Settings isključite opciju Precompiled Header. Kliknite na Finish.
4. Zamenite automatski generisan sadržaj u fajlu `Hello.cpp` isečkom koda koji je prikazan u programskom kodu 1.1.

PROGRAMSKI KOD 1.1 `Hello.cpp`**, program Hello World**

```
1:  #include <iostream>
 2:
 3:  int main()
 4:  {
 5:    std::cout << „Hello World!“ << std::endl;
 6:    return 0;
 7:  }
```

Ova jednostavna aplikacija samo prikazuje liniju na ekranu, koristeći liniju koda `std::cout`. Linija koda `std::endl` izdaje naredbu coutu da završi konkretnu liniju. Aplikacija se zatvara kada se operativnom sistemu vrati 0.

NAPOMENA Da biste čitali program, može biti korisno da znate kako da izgovarate specijalne karaktere i ključne reči.

Na primer, #include možete da čitate kao hash-include. Ostale verzije su sharp-include ili pound-include, u zavisnosti od toga odakle dolaze.

Slično tome, možete da čitate `std::cout` kao standard-c-out, a endl kao end-line.

PAŽNJA Važni su detalji, što znači da treba da kucate kod na potpuno isti način kao što je prikazano u programskom kodu. Kompajleri su striktni i ako slučajno unesete na kraj iskaza a :, a potrebno je da unesete a ;, možete da očekujete grešku kompajliranja i dugačak izveštaj o grešci.

Izgradnja i izvršavanje prve C++ aplikacije

Ako koristite Linux, otvorite terminal, kliknite na direktorijum koji sadrži `Hello.cpp` i otvorite g++ kompajler i program za povezivanje, koristeći komandnu liniju:

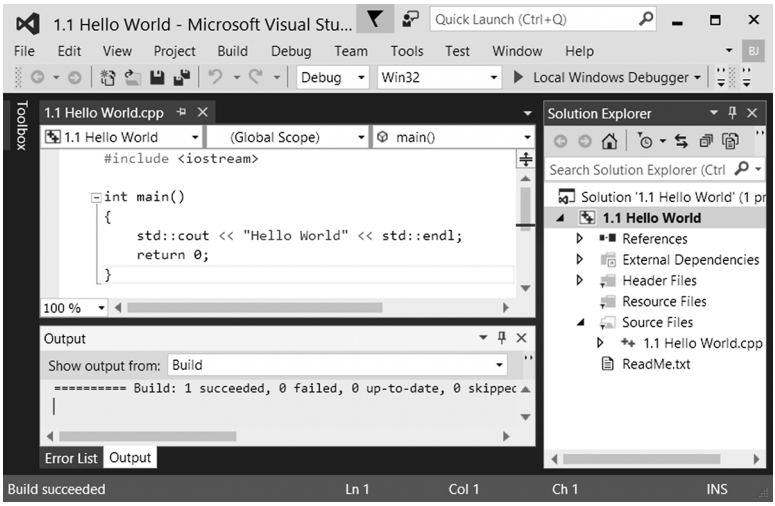
```
g++ -o hello Hello.cpp
```

Ova komanda daje naredbu kompajleru g++ da kreira izvršni fajl, pod nazivom hello, kompajliranjem C++ fajla `Hello.cpp`.

Ako koristite Microsoft Visual Studio na Windowsu, pritisnite tastere Ctrl+F5 da biste pokrenuli program direktno pomoću IDE-a. Ova prečica će kompajlirati, povezati i izvršiti aplikaciju. Alternativno, izvršite sledeće korake:

1. Kliknite desnim tasterom miša na projekat i izaberite opciju Build da biste generisali izvršni fajl `Hello.exe`.
2. Kliknite na putanju izvršnog fajla, koristeći komandnu liniju (obično se nalazi ispod direktorijuma Debug u direktorijumu projekta).
3. Pokrenite izvršni fajl, tako što ćete uneti njegov naziv.

Program kreiran u Microsoft Visual Studiou izgleda slično kao što je ilustrovano na slici 1.1.



Slika 1.1 Jednostavan „Hello World" C++ program editovan u Microsoft Visual Studio Expressu.

Izvršavanje fajla `./hello` na Linuxu ili fajla `Hello.exe` na Windowsu vraća sledeći ispis:

```
Hello World!
```

Čestitam! Započeli ste učenje jednog od najpopularnijih i najmoćnijih programskih jezika svih vremena!

ZNAČAJ C++ ISO STANDARDA

Kao što možete da vidite, usklađenost sa standardima pomaže da isečak koda u programskom kodu 1.1 bude kompajliran i izvršen na više platformi ili operativnih sistema – preduslov je dostupan C++ kompajler koji je usklađen sa standardima. Stoga, ako želite, na primer, da kreirate proizvod koji treba da se pokreće na Windowsu kao i na Linuxu, praksa programiranja usklađena sa standardima (koja ne koristi kompajler ili semantiku specifičnu za platformu) omogućava da doprete do više korisnika, bez potrebe za programom koji je specifičan za svako okruženje koje treba da se podrži. Naravno, ovo funkcioniše dobro za aplikacije koje ne zahtevaju mnogo interakcije na nivou operativnog sistema.

Razumevanje grešaka kompajlera

Kompajleri su veoma precizni u svojim zahtevima; dobri kompajleri se trude da ukažu gde je pogrešeno. Ako se suočite sa problemom prilikom kompajliranja aplikacije u programskom kodu 1.1, možda ćete dobiti grešku koja izgleda kao sledeća (namerno predstavljena greška izostavljanjem dvotačke u liniji 5):

```
hello.cpp(6): error C2143: syntax error : missing ';' before ,return'
```

Ova poruka o grešci iz Visual C++ Compilera je prilično jasna: ispisuje naziv fajla koji sadrži grešku i broj linije (u ovom slučaju 6) u kojoj je izostavljena dvotačka i daje opis same greške, zajedno sa brojem greške (u ovom slučaju C2143). Iako je znak interpunkcije izbrisan iz pete linije za ovaj primer, greška je prijavljena za sledeću liniju, zato što je postala očigledna kompajleru samo kada je analizirao vraćeni iskaz koji ukazuje da bi prethodni iskaz trebalo da bude zaustavljen pre vraćanja. Isprobajte da dodate dvotačku na početak šeste linije i videćete da će program biti kompajliran bez problema.

> **NAPOMENA** U C++ jeziku prelom linije neće automatski završiti iskaze kao u nekim drugim jezicima, kao što je VBScript.
>
> U C++ jeziku iskazi mogu da se protežu kroz više linija. Takođe je moguće da imate više iskaza u jednoj liniji, u kojoj je svaki iskaz završen znakom tačka-zarez.

Šta je novo u C++-u?

Ako ste iskusni C++ programer, možda ste primetili da osnovni C++ program u programskom kodu 1.1 nije promenjen. Iako je C++ ostao usklađen sa prethodnim verzijama C++-a, urađeno je mnogo da bi taj jezik bio jednostavniji za upotrebu i programiranje.

Najnovije ažuriranje je izdato kao deo ISO standarda koji je ratifikovan 2011. godine (popularno se naziva C++11). C++14 je izdat 2014. godine i sadrži manja poboljšanja i korekcije u odnosu na verziju C++11.

Funkcija kao što je `auto`, prvi put predstavljena u verziji C++11, omogućava da se definiše promenljiva čiji tip kompajler automatski izvodi zbijanjem razvučenih deklaracija, bez kompromitovanja bezbednosti tipa. C++14 proširuje istu ovu funkciju za vraćanje tipova. *Lambda* funkcije su funkcije bez naziva. One omogućavaju da se pišu kompaktni objekti funkcije bez dugih definicija klase, čime se značajno smanjuju linije koda.

C++ obezbeđuje programerima mogućnost pisanja prenosivih, višenitnih C++ aplikacija koje su usklađene sa standardima. Ove aplikacije, kada su pravilno izgrađene, podržavaju paralelna izvršenja obrazaca i dobro su pozicionirane u srazmeri performanse kada korisnik povećava mogućnosti konfiguracije hardvera povećavanjem broja jezgara procesora. Ovo su neka od brojnih poboljšanja u C++ jeziku koja će biti opisana u ovoj knjizi.

Nove funkcije jezika koje se očekuju u sledećoj velikoj reviziji, koja će imati naziv C++17, predstavljene su na kraju knjige, u Lekciji 29, „Napredak“.

Rezime

U ovoj lekciji naučili ste kako da programirate, kompajlirate, povezujete i izvršavate prvi C++ program. U njoj je, takođe, predstavljen kratak pregled evolucije C++ jezika i demonstrirana je efikasnost standarda prikazom da isti program može da bude kompajliran korišćenjem različitih kompajlera na različitim operativnim sistemima.

Pitanja i odgovori

P Mogu li da ignorišem poruke upozorenja kompajlera?

O U određenim slučajevima kompajler će prikazati poruke upozorenja. Upozorenja se razlikuju od grešaka po tome što je konkretna linija sintaktički tačna i može da se kompajlira. Međutim, verovatno postoji bolji način da se konkretna linija napiše, a dobri kompajleri prikazaće upozorenje, zajedno sa preporukom kako može da se ispravi linija.

Preporučena ispravka pruža bezbedniji način programiranja ili način koji omogućava da aplikacija koristi karaktere i simbole koji nisu iz latinskog jezika. Treba da obratite pažnju na ova upozorenja i da, u skladu sa njima, poboljšate aplikaciju. Nemojte ignorisati poruke upozorenja ako niste sigurni da su one tačne.

P Po čemu se razlikuje interpretirani od kompajliranog jezika?

O Jezici kao što je Windows Script su interpretirani. Ne postoji korak kompajliranja. Interpretirani jezik koristi interpreter, koji direktno čita tekstualni fajl skripta (kod) i izvršava željene akcije. Shodno tome, poželjno je da imate instaliran interpreter na mašini na kojoj skript treba da bude izvršen; međutim, kada interpreter funkcioniše kao izvršni prevodilac između mikroprocesora i napisanog koda, to obično utiče na performansu.

P Šta su greške pri izvršenju i po čemu se one razlikuju od grešaka u vreme kompajliranja?

O Greške koje se dešavaju kada izvršavate aplikaciju nazivaju se *greške pri izvršenju.* Možda ste već iskusili poznatu grešku „Access Violation“ na starijim verzijama Windowsa - to je greška pri izvršenju. Greške u vreme kompajliranja ne stižu do krajnjeg korisnika, a predstavljaju pokazatelje sintaktičkih problema; sprečavaju da programer generiše izvršni fajl.

Radionica

Radionica obezbeđuje kviz da biste proverili koliko dobro razumete ono što ste naučili u ovoj lekciji i vežbe za povećanje iskustva u upotrebi onoga što ste naučili. Pokušajte da odgovorite na pitanja u kvizu i obavite vežbe pre nego što proverite odgovore u Dodatku E.

Kviz

1. U čemu je razlika između interpretera i kompajlera?
2. Šta radi program za povezivanje?
3. Koji su koraci u normalnom ciklusu razvoja?

Vežbe

1. Pogledajte sledeći program i pokušajte da pogodite šta on izvršava, a da ga ne pokrenete:

```
 1: #include <iostream>
 2: int main()
 3: {
 4:    int x = 8;
 5:    int y = 6;
 6:    std::cout << std::endl;
 7:    std::cout << x - y << " " << x * y << " " << x + y;
 8:    std::cout << std::endl;
 9:    return 0;
10: }
```

2. Unesite program iz vežbe 1, a zatim ga kompajlirajte i povežite. Šta izvršava ovaj program? Izvršava li ono što ste mislili?

3. U čemu je greška u ovom programu:

```
1: include <iostream>
2: int main()
3: {
4:      std::cout << "Hello Buggy World \n";
5:      return 0;
6: }
```

4. Ispravite grešku u programu iz vežbe 3, kompajlirajte ga, povežite i pokrenite. Šta izvršava ovaj program?

LEKCIJA 2

Anatomija C++ programa

C++ programi su organizovani u klase, koje se sastoje od funkcija člana i promenljivih člana. Veći deo ove knjige je posvećen detaljnom opisivanju ovih delova, ali da biste bolje razumeli kako se program uklapa, morate da vidite kompletan radni program.

U ovoj lekciji naučićete:

- koji su delovi C++ programa
- kako delovi funkcionišu zajedno
- šta je funkcija i šta ona izvršava
- koje su osnovne operacije unosa i ispisa

Delovi programa Hello World

Prvi C++ program iz Lekcije 1, „Početak rada“, samo je ispisao jednostavan iskaz „Hello World“ na ekranu. Ipak, on sadrži neke od najvažnijih i osnovnih gradivnih blokova C++ programa. Upotrebite programski kod 2.1 kao početnu tačku za analiziranje komponenata koje sadrže svi C++ programi.

PROGRAMSKI KOD 2.1: HelloWorldAnalysis.cpp: analiza jednostavnog C++ programa

```
1: // Preprocessor directive that includes header iostream
 2: #include <iostream>
 3:
 4: // Start of your program: function block main()
 5: int main()
 6: {
 7:     /* Write to the screen */
 8:     std::cout << „Hello World“ << std::endl;
 9:
10:     // Return a value to the OS
11:     return 0;
12: }
```

Ovaj C++ program se može generalno razdvojiti na dva dela: na pretprocesorske komande koje započinju simbolom # i na glavno telo programa koje započinje funkcijom `int main()`.

> **NAPOMENA** Linije 1, 4, 7 i 10, koje započinju oznakom // ili /*, nazivaju se komentari i kompajler ih ignoriše. Ovi komentari su samo za korisnike koji čitaju kod.

Opisani su detaljnije u sledećem odeljku.

Pretprocesorska komanda #include

Kao što i sam naziv nagoveštava, *pretprocesor* je alatka koja se pokreće pre nego što se pokrene kompajliranje. Pretprocesorske komande su komande za pretprocesor i uvek započinju oznakom tarabe #. U liniji 2 u programskom kodu 2.1 `#include <filename>` ukazuje pretprocesoru da treba da upotrebi sadržaje fajla (u ovom slučaju `iostream`) i da ih uključi u liniju u kojoj je kreirana komanda; `iostream` je standardni fajl zaglavlja koji omogućava korišćenje komande `std::cout`, upotrebljene u liniji 8 za prikaz poruke „Hello World“ na ekranu. Drugim rečima, kompajler je mogao da kompajlira liniju 8 koja sadrži komandu `std::cout`, zato što smo izdali naredbu pretprocesoru da uključi definiciju komande `std::cout` u liniji 2.

NAPOMENA Profesionalno programirane C++ aplikacije uključuju standardna zaglavlja koja obezbeđuje razvojno okruženje i ona koje su kreirali programeri. Složene aplikacije su, obično, programirane u više fajlova u kojima neki fajlovi treba da uključuju druge. Dakle, ako artefakt deklarisan u fajlu FileA treba da se upotrebi u fajlu FileB, uključite prvi fajl u drugi. Obično se to izvršava ubacivanjem sledećeg include iskaza u fajl FileA:

```
#include "...relative path to FileB\FileB"
```

U ovom slučaju koristimo navodnike, a ne uglaste zagrade, za uključivanje samostalnog zaglavlja. Zagrade <> se, obično, koriste kada uključujemo standardna zaglavlja.

Telo programa main()

Ono što sledi iza pretprocesorske komande je telo programa karakterizovano funkcijom `main()`. Izvršenje C++ programa uvek započinje upravo ovde. Standardizovana konvencija je da je funkcija `main()` deklarisana tako da se ispred nje nalazi `int` - vraćeni tip vrednosti funkcije `main()` koji označava ceo broj.

NAPOMENA U mnogim C++ aplikacijama pronaći ćete varijantu funkcije main `()` koja izgleda ovako:

```
int main (int argc, char* argv[])
```

Ova varijanta funkcije je usklađena sa standardom i prihvatljiva je, jer vraća vrednost int. Sadržaji zagrada su argumenti koji su dodati u program. Ovaj program verovatno omogućava korisniku da ga pokrene pomoću argumenata komandne linije, kao što je

```
program.exe /DoSomethingSpecific
```

/DoSomethingSpecific argument za konkretni program (prosledio ga je operativni sistem kao parametar), kojim će se rukovati unutar funkcije `main (int argc, char* argv[])`.

Pogledajte sada liniju 8 koja ispunjava stvarnu namenu ovog programa.

```
std::cout << „Hello World" << std::endl;
```

Iskaz `cout` („console-out“, izgovara se kao si-aut) ispisuje „Hello World“ u konzoli – odnosno, ekranu. `cout` je tok definisan u standardnom `std` *imenskom* prostoru (otuda i `std::cout`) - u ovoj liniji *postavljate* tekst „Hello World“ u tok upotrebom operatora za ubacivanje toka <<. Iskaz `std::endl` je upotrebljen na kraju linije i njegovo ubacivanje u tok je slično ubacivanju znaka za početak novog reda. Ne zaboravite da se operator za ubacivanje toka << koristi svaki put kada novi objekat treba da bude dodat u tok.

U C++ jeziku različiti tipovi toka podržavaju sličnu semantiku toka za izvršavanje različitih operacija pomoću istog teksta. Na primer, za ubacivanje u fajl, umesto u konzolu, koristimo isti operator unosa << u toku `std::fstream`, umesto u toku `std::cout`. Stoga, upotreba tokova postaje intuitivna i kada se naviknete na jedan tok (kao što je `cout`, koji piše tekst u konzolu), otkrićete da je upotreba drugih tokova jednostavna (kao što je fstream, koji pomaže snimanje fajlova na disk).

Tokovi su detaljnije opisani u Lekciji 27, „Upotreba tokova za unos i ispis“.

NAPOMENA Tekst, uključujući i navodnike („Hello World“), naziva se literal znakovnog niza.

Vraćanje vrednosti

Funkcije u C++ jeziku treba da vraćaju vrednost, ako nije eksplicitno specifikovano drugačije. Funkcija `main()` je isto funkcija i uvek vraća ceo broj. Ova vrednost celog broja je vraćena u operativni sistem (OS) i, u zavisnosti od prirode aplikacije, može da bude veoma korisna, jer većina operativnih sistema obezbeđuje mogućnost slanja upita o vraćenoj vrednosti aplikacije koja je normalno zaustavljena. U mnogim slučajevima jedna aplikacija je pokrenuta drugom, a matična aplikacija (koja pokreće) „želi da zna“ da li je podređena aplikacija (koja je pokrenuta) uspešno izvršila zadatke. Programer može da upotrebi vraćenu vrednost funkcije `main()` za saopštavanje uspešnog stanja ili greške matičnoj aplikaciji pomoću koda.

NAPOMENA Obično programeri vraćaju 0 u slučaju uspeha ili -1 u slučaju greške. Međutim, vraćena vrednost je ceo broj i programer ima fleksibilnost prenosa mnogo različitih stanja uspeha ili greške, koristeći dostupan raspon vraćenih vrednosti celog broja.

PAŽNJA C++ jezik razlikuje velika i mala slova. Dakle, očekujte da će kompajliranje biti neuspešno ako napišete Int, umesto int, ili Std::Cout, umesto std::cout.

Koncept imenskih prostora

Razlog zbog kojeg ste koristili `std::cout` u programu, a ne samo count, predstavlja činjenica da se artefakt (`cout`) koji želite da pozovete nalazi u standardnom (`std`) imenskom prostoru.

Šta su imenski prostori?

Pretpostavimo da niste koristili kvalifikator imenskog prostora prilikom pozivanja artefakta `cout` i da `cout` postoji na dve lokacije koje su kompajleru poznate. To izaziva konflikt i grešku kompajliranja, naravno. U ovim situacijama imenski prostori postaju korisni. Oni su nazivi dati delovima koda koji pomažu u smanjivanju potencijalnih konflikata imenovanja. Pozivanjem koda `std::cout` ukazuje se kompajleru da treba da upotrebi jedinstveni `cout` koji je dostupan u `std` imenskom prostoru.

> **NAPOMENA** Upotrebite std (izgovara se „standardni“) imenski prostor za pozivanje funkcija, tokova i programskih alatki koje je ISO Standards Committee ratifikovao.

Mnogim programerima je dosadno da dodaju iznova specifikator `std` imenskog prostora u kod kada koriste `cout` i druge slične funkcije sadržane u istom imenskom prostoru. Deklaracija `using namespace`, kao što je prikazano u programskom kodu 2.2, pomaže da se izbegne ovo ponavljanje.

PROGRAMSKI KOD 2.2: Deklaracija `using namespace`

```
 1: // Preprocessor directive
 2: #include <iostream>
 3:
 4: // Start of your program
 5: int main()
 6: {
 7:     // Tell the compiler what namespace to search in
 8:     using namespace std;
 9:
10:     /* Write to the screen using std::cout */
11:     cout << „Hello World“ << endl;
12:
13:     // Return a value to the OS
14:    return 0;
15: }
```

ANALIZA

Pogledajte liniju 8. Ako ukažete kompajleru da koristite imenski prostor `std`, ne treba eksplicitno da spomenete imenski prostor u liniji 11 kada koristite kod `std::cout` ili `std::endl`.

Restriktivnija varijanta programskog koda 2.2 prikazana je u programskom kodu 2.3, u kojem imenski prostor nije upotrebljen u potpunosti. Uključićemo samo one artefakte koje želimo da upotrebimo.

PROGRAMSKI KOD 2.3: Još jedan prikaz ključne reči using

```
 1: // Preprocessor directive
 2: #include <iostream>
 3:
 4: // Start of your program
 5: int main()
 6: {
 7:     using std::cout;
 8:     using std::endl;
 9:
10:     /* Write to the screen using std::cout */
11:     cout << „Hello World“ << endl;
12:
13:     // Return a value to the OS
14:     return 0;
15: }
```

ANALIZA

Linija 8 u programskom kodu 2.2 je sada zamenjena linijama 7 i 8 u programskom kodu 2.3. Razlika između kodova `using namespace std` i `using std::cout` je što `using namespace std` omogućava da svi artefakti u imenskom prostoru `std` (count, `cin` i tako dalje) budu upotrebljeni bez eksplicitnog uključivanja kvalifikatora imenskog prostora `std::`. U kodu `using std::cout` imenski prostor je eksplicitno naznačen, pa je, stoga, omogućena upotreba samo artefakata `std::cout` i `std::endl`.

Komentari u C++ kodu

Linije 1, 4, 10 i 13 u programskom kodu 2.3 sadrže tekst govornog jezika (u ovom slučaju engleskog), ali, ipak, ovaj tekst ne ometa mogućnost kompajliranja programa. Ove linije, koje se nazivaju komentari, ne menjaju ispis programa. Kompajler ignoriše komentare. Komentari su popularni, jer programeri njima objašnjavaju kod – zato su napisani jasnim čitkim jezikom.

C++ podržava komentare napisane sledećim stilovima:

- // ukazuje na početak komentara koji je validan do kraja određene linije. Na primer:

```
// This is a comment - won't be compiled
```

- /* praćen */ ukazuje da je sadržani tekst komentar, čak i ako se proteže kroz više linija koda:

```
/* This is a comment
and it spans two lines */
```


NAPOMENA Možda izgleda čudno da programer treba da objašnjava svoj kod, ali što je program veći ili što je veći broj programera koji rade zajedno na određenom modulu, tim je važnije pisati kod koji može lako da se razume. Komentari pomažu programeru da dokumentuje ono što je urađeno i zbog čega je nešto urađeno na određeni način.

URADITE	NE RADITE
Dodajte komentare koji objašnjavaju rad složenih algoritama i složenih delova programa. **Sastavite** komentare, koristeći stil koji programeri saradnici mogu da razumeju.	**Nemojte** da koristite komentare za objašnjenje ili ponavljanje očiglednog. **Nemojte** da zaboravite da dodavanje komentara ne opravdava pisanje nejasnog koda. **Nemojte** da zaboravite da, kada je kod modifikovan, možda treba ažurirati i komentar.

Funkcije u C++-u

Funkcije omogućavaju da se podeli sadržaj aplikacije na funkcionalne jedinice koje mogu da budu pozvane u sekvenci po našem izboru. Funkcija, kada je pozvana, obično vraća vrednost u funkciju koja ju je pozvala. Najpoznatija funkcija je, naravno, `int main()`. Kompajler je prepoznaje kao početnu tačku C++ aplikacije i obavezno vraća `int` (ceo broj).

Kao programeri, imamo izbor i obično treba da sastavimo sopstvene funkcije. Programski kod 2.4 je jednostavna aplikacija koja koristi funkciju za prikaz iskaza na ekranu, koristeći `std::cout` sa različitim parametrima.

PROGRAMSKI KOD 2.4: Deklarisanje, definisanje i pozivanje funkcije koja prikazuje mogućnosti koje ima `std::cout`

```
1: #include <iostream>
 2: using namespace std;
 3:
 4: // Declare a function
 5: int DemoConsoleOutput();
 6:
 7: int main()
 8: {
 9:     // Call i.e. invoke the function
10:     DemoConsoleOutput();
11:
12:     return 0;
13: }
14:
15: // Define i.e. implement the previously declared function
16: int DemoConsoleOutput()
17: {
18:     cout << „This is a simple string literal“ << endl;
19:     cout << „Writing number five: ” << 5 << endl;
20:     cout << „Performing division 10 / 5 = ” << 10 / 5 << endl;
21:     cout << „Pi when approximated is 22 / 7 = ” << 22 / 7 << endl;
22:     cout << „Pi is 22 / 7 = ” << 22.0 / 7 << endl;
23:
24:     return 0;
25: }
```

ISPIS

```
This is a simple string literal
Writing number five: 5
Performing division 10 / 5 = 2
Pi when approximated is 22 / 7 = 3
Pi is 22 / 7 = 3.14286
```

ANALIZA

Linije 5, 10 i od 16 do 25 su važne. Linija 5, koja se naziva *deklaracija funkcije*, u suštini ukazuje kompajleru da želimo da kreiramo funkciju pod nazivom `DemoConsoleOutput()`, koja vraća vrednost `int` (ceo broj). Ova deklaracija omogućava kompajleru da prevede liniju 10, u kojoj je funkcija `DemoConsoleOutput()` pozvana unutar funkcije `main()`. Kompajler pretpostavlja da će se *definicija* (odnosno implementacija funkcije) pojaviti, a pojaviće se u linijama od 16 do 25.

U stvari, ova funkcija prikazuje mogućnosti artefakta `cout`. Ne samo da štampa tekst na isti način kao što je prikazan „Hello World“ u prethodnim primerima, već i rezultat jednostavnog aritmetičkog izračunavanja. Linije 21 i 22 prikazuju rezultat pi (22 / 7); linija 22 je mnogo tačnija, zato što deljenjem 22.0 sa 7 ukazujete kompajleru da treba da tretira rezultat kao realan (float u terminu C++-a), a ne kao ceo broj.

Vidite da funkcija treba da vrati ceo broj, kao što je deklarisano u liniji 5, a vraća 0. Slično tome, i funkcija `main()` vraća vrednost 0. S obzirom da je funkcija `main()` prenela sve svoje aktivnosti na funkciju `DemoConsoleOutput()`, bilo bi mnogo bolje da upotrebite vraćenu vrednost funkcije za vraćanje vrednosti iz funkcije `main()`, kao što je prikazano u programskom kodu 2.5.

PROGRAMSKI KOD 2.5: Upotreba vraćene vrednosti funkcije

```
1: #include <iostream>
 2: using namespace std;
 3:
 4: // Function declaration and definition
 5: int DemoConsoleOutput()
 6: {
 7:     cout << „This is a simple string literal“ << endl;
 8:     cout << „Writing number five: ” << 5 << endl;
 9:     cout << „Performing division 10 / 5 = ” << 10 / 5 << endl;
10:     cout << „Pi when approximated is 22 / 7 = ” << 22 / 7 << endl;
11:     cout << „Pi actually is 22 / 7 = ” << 22.0 / 7 << endl;
12:
13:     return 0;
14: }
15:
16: int main()
17: {
18:     // Function call with return used to exit
19:     return DemoConsoleOutput();
20: }
```

ANALIZA

Ispis ove aplikacije je isti kao i ispis prethodnog programskog koda. Ipak, postoje male promene u načinu na koji je aplikacija programirana. Na primer, pošto ste definisali (odnosno implementirali) funkciju ispred funkcije `main()` u liniji 5, nije vam potrebna dodatna deklaracija. Moderni C++ kompajleri upotrebiće ovu definiciju kao deklaraciju funkcije i definiciju ujedno. Funkcija `main()` je malo kraća. Linija 19 poziva funkciju `DemoConsoleOutput()` i istovremeno vraća vrednost funkcije iz aplikacije.

NAPOMENA U slučajevima kao što je ovaj, gde funkcija ne treba da donosi odluke, ili prikaže status uspeha ili neuspeha, možete da deklarišete funkciju povratnog tipa void:

```
void DemoConsoleOutput()
```

Ova funkcija ne može da vrati vrednost.

Funkcije mogu da koriste parametre, mogu da budu rekurzivne, mogu da sadrže više povratnih iskaza, mogu da budu preklopljene, kompajler može da ih proširi u liniji i tako dalje. Ovi koncepti su predstavljeni detaljnije u Lekciji 7, „Organizovanje koda pomoću funkcija“.

Osnovni unos pomoću iskaza std::cin i ispis pomoću iskaza std::cout

Računar omogućava da vršite interakciju sa aplikacijama koje su na njemu pokrenute u različitim formama i omogućava da ove aplikacije vrše interakciju sa vama u različitim formama. Možete da vršite interakciju sa aplikacijama korišćenjem tastature ili miša. Možete da prikažete informacije na ekranu kao tekst, da tekst prikažete u obliku složenih grafika, da ga odštampate na papir pomoću štampača, ili, jednostavno, da ga snimite u fajl za kasniju upotrebu. U ovom odeljku opisana je najjednostavnija forma unosa i ispisa u C++ jeziku korišćenjem konzole za pisanje i čitanje informacija.

Možete da upotrebite iskaz `std::cout` (izgovara se „standard si-aut“) za pisanje jednostavnih tekstualnih podataka u konzoli, a iskaz `std::cin` (“standard see-in”) za čitanje teksta i brojeva (unetih pomoću tastature) iz konzole. U stvari, u prikazivanju poruke „Hello World“ na ekranu već ste se susreli sa iskazom `cout`, kao što možete da vidite u programskom kodu 2.1:

```
8:     std::cout << „Hello World“ << std::endl;
```

Iskaz prikazuje `cout`, praćen operatorom unosa << (to pomaže pri ubacivanju podataka u tok ispisa). Iza operatora je dodat literal znakovnog niza „Hello World“, a nakon toga je dodata nova linija u formi `std::endl` (izgovara se „standard end-lajn“).

Upotreba iskaza `cin` je, takođe, jednostavna. Pošto je `cin` upotrebljen za unos, spojen je sa promenljivom u kojoj želite da sačuvate unete podatke:

```
std::cin >> Variable;
```

Stoga, `cin` je praćen operatorom *ekstrahovanja* >> (ekstrahuje podatke iz toka unosa), a zatim je dodata promenljiva u kojoj treba da budu sačuvani podaci. Ako unos korisnika treba da bude sačuvan u dve promenljive, koje sadrže podatke razdvojene razmakom, iskoristite jedan iskaz:

```
std::cin >> Variable1 >> Variable2;
```


Vidite da iskaz `cin` može da se upotrebi za tekst i za numeričke unose korisnika, kao što je prikazano u programskom kodu 2.6.

PROGRAMSKI KOD 2.6: Upotreba iskaza `cin` i `cout` za prikaz numeričkog i tekstualnog unosa korisnika

```
 1: #include <iostream>
 2: #include <string>
 3: using namespace std;
 4:
 5: int main()
 6: {
 7:     // Declare a variable to store an integer
 8:     int inputNumber;
 9:
10:     cout << „Enter an integer: ”;
11:
12:     // store integer given user input
13:     cin >> inputNumber;
14:
15:     // The same with text i.e. string data
16:     cout << „Enter your name: ”;
17:     string inputName;
18:     cin >> inputName;
19:
20:     cout << inputName << ” entered ” << inputNumber << endl;
21:
22:     return 0;
23: }
```

ISPIS

```
Enter an integer: 2017
Enter your name: Siddhartha
Siddhartha entered 2017
```

ANALIZA

Linija 8 prikazuje kako je deklarisana promenljiva pod nazivom `inputNumber` za čuvanje podataka tipa `int`. Korisnik treba da unese broj, koristeći iskaz `cout` u liniji 10; uneti broj je sačuvan u promenljivoj celog broja pomoću iskaza `cin` u liniji 13. Ista vežba ponovljena je za čuvanje imena korisnika, koje naravno ne može da se čuva kao ceo broj, već kao drugačiji tip pod nazivom `string`, kao što možete da vidite u linijama 17 i 18. Uključen je `<string>` u liniju 2 da bi kasnije bio upotrebljen tip string unutar funkcije `main()`. Na kraju, u liniji 20 iskaz `cout` se koristi za prikaz unetog imena sa brojem i teksta za kreiranje ispisa `Siddhartha entered 2017`.

Ovo je jednostavan primer kako funkcionišu osnovni unos i ispis u C++ jeziku. Ne brinite ako vam nije jasan koncept promenljive - detaljniji opis ćete naći u Lekciji 3, „Upotreba promenljivih i deklarisanje konstanti“.

> **NAPOMENA** Da sam prilikom izvršavanja programskog koda 2.6 uneo nekoliko reči, kao što je moje ime (na primer: Siddhartha Rao), iskaz cin bi i dalje sačuvao samo prvu reč „Siddhartha“ u znakovnom nizu. Da biste sačuvali cele linije, upotrebite funkciju getline`()`, koja je opisana u programskom kodu 4.7 u Lekciji 4, „Upravljanje nizovima i znakovnim nizovima“.

Rezime

U ovoj lekciji su predstavljeni osnovni delovi jednostavnog C++ programa. Objašnjeno je šta je funkcija `main()`, predstavljeni su imenski prostori i naučili ste osnove unosa i ispisa u konzoli. Možete sada da upotrebite ove koncepte u svakom programu koji pišete.

Pitanja i odgovori

P Šta izvršava komanda `#include`?

O Ovo je pretprocesorska komanda, koja se pokreće kada se pozove kompajler. Ova specifična komanda izaziva da sadržaji fajla naznačenog u `<>` iza `#include` budu ubačeni u određenu liniju kao da su uneti na tu lokaciju u izvornom kodu.

P U čemu je razlika između // komentara i /* komentara?

O Komentari dvostruke kose crtice (//) ističu na kraju linije. Komentari kose crtice i zvezdice (/*) su aktivni do unosa oznake za zatvaranje komentara (*/). Komentari dvostruke kose crte se nazivaju i *jednolinijski*, a komentari kose crtice i zvezdice se često nazivaju *višelinijski*. Ne zaboravite: čak ni kraj funkcije ne završava komentar

kose crtice i zvezdice; morate da postavite oznaku za zatvaranje komentara ili ćete, u suprotnom, dobiti grešku u vreme kompajliranja.

P U kom slučaju treba programirati argumente komandne linije?

O Argumenti se dodaju kada treba izvršiti unos opcija koje omogućavaju korisniku da promeni ponašanje programa. Na primer, komanda `ls` u Linuxu ili `dir` u Windowsu omogućava da vidite sadržaje unutar aktuelnog direktorijuma. Da biste pregledali fajlove u drugom direktorijumu, specifikujte putanju direktorijuma, koristeći argumente komandne linije, kao što su ls / ili dir \.

Radionica

Radionica obezbeđuje kviz da biste proverili koliko dobro razumete ono što ste naučili u ovoj lekciji i vežbe za povećanje iskustva u upotrebi onoga što ste naučili. Pokušajte da odgovorite na pitanja u kvizu i obavite vežbe pre nego što proverite odgovore u Dodatku E.

Kviz

1. U čemu je problem u deklarisanju funkcije Int `main()`?
2. Mogu li komentari biti duži od jedne linije?

Vežbe

1. BUG BUSTERS: Unesite sledeći program i kompajlirajte ga. Zašto je izvršenje neuspešno? Kako možete to da ispravite?

```
1: #include <iostream>
2: void main()
3: {
4:       std::Cout << Is there a bug here?";
5: }
```

2. Ispravite programsku grešku u vežbi 1 i ponovo kompajlirajte program, povežite ga i pokrenite.
3. Modifikujte programski kod 2.4 da biste prikazali oduzimanje, koristeći –, i množenje, koristeći *.