1 SEDMICA

Ukratko

Tokom priprema za prvu nedelju učenja programiranja pomoću programskom jeziku C++ biće vam potrebno nekoliko stvari: kompajler, editor i ova knjiga. Ukoliko nemate C++ kompajler i editor, možete koristiti ovu knjigu, ali bez vežbanja.

Najbolji način da naučite programiranje jeste da pravite program! Na kraju svakog poglavlja nalazi se odeljak koji sadrži pitanja i vežbe. Odvojite dovoljno vremena da odgovorite na sva pitanja i objektivno ocenite koliko ste naučili. Svaka naredna lekcija nadovezuje se na prethodnu, te se potrudite da u potpunosti razumete materiju pre nego što pređete dalje.

Napomena programerima koji koriste programski jezik C

Materijal koji se nalazi u prvih pet poglavlja već vam je poznat. Ipak, obnovite znanje i uradite vežbe kako biste se bez problema uključili u izlaganje u poglavlju Dan 6, "Osnovne klase". Ukoliko poznajete programski jezik C ali ga niste koristili pod Linuxom, pažljivo pročitajte uvod u GNU kompajler koji se nalazu u prvih nekoliko poglavlja.

Šta vas očekuje

U prvoj ne1delji nalazi se materijal koji je neophodan da biste počeli da programirate (naročito za programiranje pomoću programskog jezika C++). U poglavljima Dan 1, "Pripreme za rad", i Dan 2, "Delovi C++ programa", naučićete osnovne koncepte programiranja i toka programa. U poglavlju Dan 3, "Promenljive i konstante", naučićete šta su promenljive i konstante i kako da u programima koristite podatke. U poglavlju Dan 4, "Izrazi i iskazi", naučićete kako se program grana na osnovu podataka i uslova u kojima se izvršava. U poglavlju Dan 5, "Funkcije", naučićete šta su funkcije i kako da ih koristite, a u poglavju Dan 6, "Osnovne klase", upoznaćete se sa klasama i objektima. U poglavlju Dan 7, "Tok programa", naučićete kako se izvršava program, a na kraju prve nedelje napravićete prave objektno orijentisane programe.

Dan 1

Pripreme za rad

Dobrodošli u Samsovu knjigu *Naučite sami programski jezik C++ za Linux za 21 dan!* Danas ćete započeti učenje koje će vas učiniti veštim C++ programerom. Naučićete:

- Šta su GNU kompajleri i kakav je međusobni odnos GNU kompajlera i Linuxa.
- Zašto je C++ standard koji se razvija za pravljenje programa.
- Korake pravljenja C++ programa.
- Osnove korišćenja GNU kompajlera.
- Kako da unesete, kompajlirate i povežete vaš prvi C++ program.

Šta je GNU?

Skraćenica GNU označava "GNU nije UNIX" (engl. GNU is Not UNIX). To je ime za niz korisnih softverskih paketa koji se uobičajeno mogu pronaći u UNIX okruženju koje distribuira GNU projekat MIT univerziteta. Paketi su besplatni i mogu se pronaći na raznim lokacijama na Internetu (pakete plaćate samo ukoliko želite da ih dobijete na flopi disku ili traci). Razvoj paketa obavljaju mnogi volonteri. Čitav posao vodi Ričard M. Stalman (Richard M. Stallman), jedan od programera EMACS editora.

Linux je operativni sistem koji je veoma sličan UNIX-u (ali nije u suprotnosti sa zakonom o autorskim pravima komercijalnih verzija UNIX-a). Međutim, Linux je uglavnom samo kernel, tj.osnovni deo operativnog sistema.

Većina komandi koje koristite pod Linuxom deo su GNU projekta, čiji je sponzor Free Software Foundation. Dvadeset i osam procenata tipične verzije Linuxa koja se distribuira je GNU, a samo tri posto je zaista Linux.

NAPOMENA Da, GNU je skraćenica za samu sebe ali je namerno takva. Ovakvo imenovanje ima istorijat koji je povezan sa UNIX-om, ali nije obuhvaćen njegovim autorskim pravima. Operativni sistemi koji nisu komercijalni, a slični su UNIX-u, poznati su kao XINU: XINU nije UNIX (engl. XINU is Not UNIX).

Dakle, vi koristite GNU usluge kada mislite da zadajete Linux komande. Ljudi koji rade na Linuxu nisu našli ni jedan valjan razlog zbog kojeg bi duplirali posao koji su obavili ljudi koji rade na GNU projektu. To je deo filozofije UNIX-a: ponovo koristi, nemoj ponovo praviti.

Ukoliko pažljivo pročitate dokumentaciju koju dobijate uz Linux sistem, naići ćete na GNU.

GNU projekat obuhvata C i C++ kompajlere (kao i mnoge druge alate, programske jezike i pomoćne programe). C kompajler je poznat kao gcc, dok je C++ kompajler poznat kao g++ (pod Linuxom) ili gxx (pod drugim operativnim sistemom kakav je DOS koji ne dozvoljava da se u imenu datoteke koristi znak plus).

GNU alati postoje za mnoge platforme, a ne samo za Linux. Dakle, GNU kompajler možete nabaviti za Linux, DOS, DEC (Compaq) Alpha ili neki drugi operativni sistem. To znači da možete naučiti da koristite kompajler u jednom operativnom sistemu a zatim da ga koristite i u mnogim drugim operativnim sistemima. Takođe, možete kod kuće raditi na C++ projektu pod Linuxom i nastaviti da na njemu radite na poslu, kada bi inače trebalo da radite za svog poslodavca, čak i kada na poslu imate računar sa Microsoft Windowsom ili DOS-om. Naravno, nikada nećete uraditi tako nešto!

Nabavite GNU alate

Najlakši način da nabavite GNU kompajler i druge alate jeste da instalirate kopiju Linuxa koja se nalazi na propratnom CD ROM-u. Da biste dobili detaljne informacije o instaliranju, morate pročitati instrukcije koje se nalaze na propratnom CD ROM-u jer je tema ove knjige C++ a ne Linux. SAMS je objavio nekoliko odličnih knjiga o Linuxu.

Kao što je ranije pomenuto, možete nabaviti kompajler za druge operativne sisteme i platforme.

Ukoliko ste već instalirali Linux, verovatno već imate GNU kompajlere. Ako imate stariju verziju.

Najbolje je da najnovije verzije kompajlera potražite na adresi `http://www.gnu.org`. Sa te adrese možete besplatno preuzeti GNU alate i kompajlere za različite platforme. Takođe, na CD ROM-u možete naći informacije o kupovini GNU softvera. Bilo kako da nabavite alate, trebalo bi da pošaljete novac jer pravljenje Web stranica i štampanje dokumentacije ipak košta. Ja sam naručio sve datoteke i alate za sva platforme da bih mogao da ih uporedim. (Ne dobijam nikakav novac kada dotiram ili kupujem od Free Software Foundationa; ja samo verujem u ono što rade.)

Za unapređivanje koda potrebno je vreme. Ono je potrebno i za snimanje na CD ROM-u, pravljenje kopija i isporuku. Rezultati rada su različite verzije za različite platforme i različite verzije za različite medije za distribuciju. Možete preko Interneta preuzeti najnoviju verziju ili kupiti nešto stariju.

Na propratnom CD ROM-u nalazi se verzija 2.9.5 GNU kompajlera (najnovija verzija u trenutku kada je pisana knjiga). Verzija kompajlera koja se može naći na CD ROM-u Free Software Foundationa jeste verzija 2.7.2. U svakom slučaju, noviji kompajler je bolji.

Primeri koji se nalaze u ovoj knjizi mogu se kompajlirati pomoću obe verzije kompajlera, izuzev ako nije drugačije naznačeno.

Novije verzije kompajlera su deo EGCS-a (Experimental GNU Compiler System). Veći deo posla na pravljenju novog kompajlera obavljaju ljudi u Cygnusu (`http://www.cygnus.com`). Cygnus je kompanija koja zvanično održava GNU kompajlere (zbog omaške upravnog odbora).

Kratki istorijat programskog jezika C++

Programski jezici su pretrpeli dramatičnu evoluciju od nastanka prvih računara koji su se koristili za izračunavanje trajektorije artiljerijskih granata tokom Drugog svetskog rata. U prošlosti su programeri radili sa najprimitivnijim instrukcijama: mašinskim jezikom. Ove instrukcije su se predstavljale pomoću dugačkih nizova nula i jedinica. Ubrzo su napravljeni asembleri kako bi se mašinske instrukcije zamenile mnemonicima koji su bili razumljiviji (takvi mnemonici su recimo ADD i MOV).

Vremenom su napravljeni programski jezici višeg nivoa, kakvi su recimo BASIC i COBOL. Ovi programski jezici su omogućili ljudima da rade sa nečim što je blisko rečima i rečenicama (recimo, Let I = 100). Ove instrukcije su se prevodile u mašinski jezik pomoću interpretatora i kompajlera. *Interpretator* prevodi program dok ga čita, pri čemu programske instrukcije, odnosno kod, prevodi direktno u akcije koje treba obaviti. *Kompajler* (engl. compiler) prevodi kod u posredan oblik. Ovaj korak se naziva kompajliranje, pri čemu se pravi objektna datoteka. Kompajler zatim poziva linker, koji objektnu datoteku pretvara u izvršnu datoteku.

Pošto interpretatori čitaju kod onako kako je napisan i odmah ga izvršavaju, programerima je lakše da ih koriste. Kompajleri uvode dodatni korak kompajliranja i povezivanja koda, što nije zgodno za rad. Međutim, kompajleri prave program koji se svaki put veoma brzo izvršava jer je posao prevođenja, za koji je potrebno vreme, već obavljen.

Druga prednost koju imaju mnogi programski jezici poput C++, koji koriste kompajler, jeste to što program možete distribuirati ljudima koji nemaju kompajler. Ako koristite programske jezike koji koriste interpreter, onda i vi morate imati interpreter da biste mogli da pokrenete program.

Neki programski jezici, kakav je Visual Basic, interpreter nazivaju izvršnom bibliotekom. U programskom jeziku Java, interpreter se naziva virtuelna mašina (engl. Virtual Machine, VM), ali u slučaju ovog programskog jezika, VM je deo pretraživača (recimo, Internet Explorera ili Netscapea).

Tokom godina osnovni cilj programera bio je pravljenje kratkih delova koda koji će se brzo izvršavati. Programi su morali da budu mali jer je memorija bila skupa, a morali su da budu brzi jer je bilo skupo procesorsko vreme. Pošto su računari postali manji, jeftiniji i brži i pošto je cena memorije niža, prioriteti su se promenili. Danas je cena utrošenog vremena programera veća od cene mnogih računara koji se koriste u poslovnom svetu. Dobro napisan kod koji se lako održava prava je premija. Lak za održavanje znači da se program može poboljšati i proširiti bez mnogo napora kako bi odgovorio na zahteve koji se pred njega postavljaju.

Programi

Reč *program* ima dvojako značenje: označava instrukcije (ili izvorni kod) koje je napravio programer, i opisuje ceo izvršni softver. Ova dvoznačnost može napraviti veliku zabunu, te ćemo mi pokušati da napravimo razliku između izvornog koda s jedne strane i izvršne datoteke sa druge strane.

Program se može definisati kao skup instrukcija koje je napisao programer ili kao izvršna datoteka softvera.

Izvorni kod se može pretvoriti u izvršnu datoteku na dva načina: interpretatori prevode izvorni kod u instrukcije računara, koje računar odmah izvršava. Drugi način je da se pomoću kompajlera izvorni kod prevede u program koji se kasnije može pokrenuti. Iako je lakše raditi sa interpretatorima, za ozbiljnije programiranje koriste se kompajleri, jer se kompajlirani kod izvršava mnogo brže. C++ je programski jezik koji se kompajlira.

Rešavanje problema

Problemi koje programeri treba da reše stalno se menjaju. Pre dvadeset godina pravili su se programi koji su radili sa velikim količinama podataka. Ljudi koji su pisali kod i ljudi koji su koristili program bili su iskusni korisnici računara. Danas računare koristi daleko veći broj ljudi, od kojih većina malo zna kako računari i programi rade. Računari su alat koji ljudi koriste da bi rešili poslovne probleme, a ne da bi se borili s njima.

Ironija je da su programi postali složeniji kako bi novim korisnicima olakšali rad na računaru. Davno su prošla vremena kada su korisnici unosili tajnovite komande na ezoteričnim komandnim linijama samo da bi dobili gomilu podataka. Savremeni programi koriste složene korisničke interfejse koji obuhvataju prozore, menije, okvire za dijalog i mnoštvo metafora na koje smo navikli. Programi koji su napravljeni tako da koriste ovaj pristup daleko su složeniji od programa koji su pravljeni pre deset godina.

Sa razvojem Weba, računari su ušli u novu eru marketinga; njih koristi više ljudi nego ikad ranije, a očekivanja su im veoma velika. U poslednjih nekoliko godina programi su postali veliki i složeni, pa je očigledna potreba za tehnikama objektno orijentisanog programiranja kako bi se takvi programi održavali. Promena zahteva programiranja dovela je do napredovanja programskih jezika i tehnika za pravljenje programa. Mada je celokupan istorijat zaista fascinantan, ova knjiga se bavi prelazom sa proceduralnog na objektno orijentisano programiranje.

Proceduralno, strukturno i objektno orijentisano programiranje

Donedavno su programi bili niz procedura koje su radile sa podacima. *Procedura*, ili funkcija, jeste skup specifičnih instrukcija koje se izvršavaju jedna za drugom. Podaci su odvojeni od procedure, a programiranja se sastojalo u praćenju koja od funkcija poziva druge funkcije i koji podaci su promenjeni. Da bi se uveo red u ovu potencijalno zbunjujuću situaciju, stvoreno je strukturno programiranje.

Osnovna ideja strukturnog programiranja je jednostavna koliko i princip podeli i vladaj. Program računara se može posmatrati kao skup zadataka koje treba obaviti. Svaki zadatak koji je složen da bi se na jednostavan način opisao deli se u skup manjih zadataka, sve dok oni ne postanu toliko mali da se lako mogu shvatiti.

Kao primer uzmimo izračunavanje prosečne zarade zaposlenih jedne kompanije, što je prilično složen zadatak. Ovaj zadatak možete podeliti u nekoliko manjih:

1. Saznajte koliko svaki zaposleni zarađuje.
2. Saznajte koliko ima zaposlenih.
3. Saberite sve plate.
4. Podelite dobijeni rezultat brojem zaposlenih.

Izračunavanje zbira svih plata može se podeliti u nekoliko koraka:

1. Pronađite slog svakog zaposlenog.
2. Iz sloga izdvojite platu.
3. Visinu plate dodajte sumi plata.
4. Pronađite sledeći slog zaposlenog.

Pronalaženje sloga svakog zaposlenog radnika može se podeliti u sledeće korake:

1. Otvorite datoteku koja sadrži podatke o zaposlenima.
2. Pronađite odgovarajući slog.
3. Pročitajte podatke sa hard diska.

Strukturno programiranje je i dalje veoma efikasan pristup rešavanju složenih problema. Krajem 80-tih godina 20. veka neki nedostaci strukturnog programiranja postali su očigledni.

Prvo, prirodna je težnja da se podaci (na primer, slogovi o zaposlenima) i ono što sa njima možete uraditi (na primer, sortiranje, editovanje i drugo) tretiraju kao celina. Proceduralno programiranje je ovaj problem rešavalo tako što su se podaci odvajali od funkcija koje su obrađivale podatke.

Drugo, programeri su stalno pravili nova rešenja za poznate probleme. Ovaj postupak se često naziva "ponovno pronalaženje točka", a to je suprotno od ideje ponovnog korišćenja. Ideja ponovnog korišćenja podrazumeva pravljenje komponenti koje imaju svojstva koja su poznata i upotrebu tih komponenti u programu. Ovakav model programiranja je inspirisan svetom hardvera – kada je inženjeru potreban novi tranzistor, on ga obično ne pravi; inženjer uzima tranzistor koji ima karakteristike koje mu odgovaraju ili ga menja kako bi dobio ono što želi. Inženjeru softvera nije bila na raspolaganju ni jedna slična opcija.

NOVI TERMIN Način na koji sada koristimo računare – pomoću menija, kontrola i prozora – podržava interaktivnije, događajima vođeno programiranje. Vođeno događajima (engl. event-driven) znači da na određeni događaj – recimo, kada korisnik klikne mišem ili otvori meni – računar mora da odgovori. Programi postaju sve više interaktivni i veoma je važno napraviti program koji ima takvu funkcionalnost.

Staromodni programi su korisnika primoravali da postepeno prolazi kroz niz okvira za dijalog. Moderni, događajima vođeni programi, prikazuju odjednom sve svoje opcije i reaguju na akcije korisnika.

NOVI TERMIN Objektno orijentisano (engl. object-oriented) programiranje pokušava da odgovori na ove zahteve obezbeđujući tehnike za upravljanje složenim celinama, omogućavajući ponovno korišćenje komponenti softvera i grupišući podatke i zadatke u celinu.

Suština objektno orijentisanog programiranja je tretiranje podataka i procedura koje ih obrađuju kao jednog objekta – samostalni objekat koji ima identitet i sopstvene karakteristike.

C++ i objektno orijentisano programiranje

Programski jezik C++ 1u potpunosti podržava objektno orijentisano programiranje, uključujući tri osnovne ideje objektno orijentisanog programiranja: enkapsulaciju, nasleđivanje i polimorfizam.

Enkapsulacija

Kada je inženjeru potreban otpornik za uređaj koji pravi, on obično ne projektuje potpuno nov otpornik. Inženjer odlazi do skladišta, pogleda obojene trake kako bi saznao karakteristike otpornika i uzima onaj koji mu je potreban. Otpornik je "crna kutija" sa tačke gledišta inženjera – ne interesuje ga mnogo kako otpornik radi sve dok radi po specifikacijama; nije potrebno da pogleda šta se nalazi unutar otpornika da bi mogao da ga koristi.

Osobina da je jedinica samostalna zove se *enkapsulacija*. Pomoću enkapsulacije možemo sakriti podatke. *Sakrivanje podataka* (engl. data hiding) je značajna karakteristika, jer se objekat može upotrebljavati a da korisnik ne mora da zna ni da brine kako on interno funkcioniše. Baš kao što možete koristiti frižider a da ne znate kako radi kompresor, tako možete koristiti objekat a da ne znate njegove interne podatke.

Slično, kada inženjer koristi otpornik, on ne mora ništa da zna o internom stanju tog otpornika. Sve njegove osobine su smeštene u objekat otpornik; te osobine nisu razbacane po električnoj šemi. Nije neophodno razumeti način na koji otpornik radi da bi se koristio. Njegovi podaci su sakriveni u kućištu otpornika.

C++ podržava svojstva enkapsulacije pomoću korisnički definisanih tipova podataka koji se nazivaju klase. U poglavlju Dan 6, "Osnovne klase", naučićete kako da napravite klase. Pošto napravite klasu, ona se ponaša kao potpuno enkapsuliran entitet – koristi se kao celina. Stvarni posao koji se obavlja unutar klase treba da bude sakriven. Korisnici klasa ne moraju da znaju kako klasa radi; treba samo da znaju kako da je koriste.

Nasleđivanje i ponovno korišćenje

Kad inženjeri Acme Moto1rsa žele da naprave nov automobil, oni to mogu uraditi na dva načina: mogu početi od nule ili mogu izmeniti postojeći model. Pretpostavimo da je njihov model Star skoro savršen, ali da žele da dodaju turbo punjač i šestostepeni menjač.

Glavni inženjer ne bi počeo od nule, već bi rekao: "Hajde da napravimo drugačiji model Stara, tako što ćemo dodati nove mogućnosti. Novi model ćemo nazvati Quasar." Quasar je vrsta Stara, ali unapređena novim mogućnostima.

C++ podržava nasleđivanje. Može se definisati novi tip koji se dobija proširivanjem starog tipa. Za takvu novu potklasu se kaže da je izvedena iz postojećeg tipa, a ponekad se naziva izvedeni tip. Quasar je izveden iz Stara i stoga nasleđuje sve njegove kvalitete kojima može dodati nove. Nasleđivanje i njegova primena u C++ se razmatraju u poglavlju Dan 11, "Nasleđivanje" i poglavlju 15, "Napredno nasleđivanje".

Polimorfizam

Novi automobil Quasar će se možda ponašati drugačije od automobila Star kada pritisnete papučicu gasa. Quasar će možda upotrebiti ubrizgavanje goriva i turbopunjač, dok bi Star samo usmerio gorivo u karburator. Vozač ne mora biti upoznat sa ovim razlikama. On može pritisnuti do poda papučicu gasa i desiće se ono što treba da se desi, što zavisi od modela automobila koji vozi.

C++ podržava ideju da različiti objekti rade "pravu stvar", što se naziva *polimorfizam funkcija* (engl. function polymorphism) i polimorfizam klasa (engl. class polymorphism). Poli znači mnogo, a morfizam znači oblik. Polimorfizam se, dakle, odnosi na nešto što može postojati u mnogo oblika i razmatra se u poglavlju Dan 10, "Napredne funkcije", i poglavlju Dan 13, "Polimorfizam".

Kako se razvijao C++

Pošto su objektno orijentisana analiza, struktura i programiranje postali popularni, Bjarn Stroustrup (Bjarne Stroustrup) je najpopularniji programski jezik za komercijalnu upotrebu, programski jezik C, proširio da bi obuhvatio karakteristike koje su neophodne za objektno orijentisano programiranje.

Mada je tačno da je programski jezik C++ nadskup programskog jezika C i mada je svaki program napisan na programskom jeziku C ujedno i program koji je napisan na programskom jeziku C++, razlike između programskih jezika C i C++ su značajne. Programski jezik C++ je izvukao korist iz veze sa programskim jezikom C jer su programeri koji su koristili programski jezik C lako mogli da pređu na novi programski jezik.

Da bi u potpunosti iskoristila C++, većina programera je shvatila da mora da zaboravi mnogo toga što je naučila i da nauči nov način konceptualizacije i rešavanja programskih problema.

Da li treba prvo naučiti programski jezik C?

Pošto je programski jezik C++ nadskup programskog jezika C, neizbežno se nameće pitanje da li prvo treba da se nauči programski jezik C? Stroustrup i mnogi drugi programeri koji koriste programski jezik C++ slažu se da ne samo što nije potrebno prvo naučiti programski jezik C, već da njegovo nepoznavanje može imati prednosti.

U ovoj knjizi se ne pretpostavlja da imate bilo kakvo predznanje programiranja. Ukoliko ste programer koji koristi programski jezik C, onda je za vas prvih pet poglavlja uglavnom pregled karakteristika programskog jezika. Međutim, u ovim poglavljima ćete naučiti kako da koristite GNU kompajlere. Počevši od poglavlja Dan 6, krenućemo u pravljenje objektno orijentisanih programa.

C++ i Java

Programski jezik C++ je danas sasvim sigurno jezik koji se najčešće koristi za pravljenje komercijalnog softvera. Poslednjih godina programski jezik Java ugrožava njegovu dominaciju, ali se stvari menjaju i mnogi programeri koji su prešli sa programskog jezika C++ na programski jezik Java sada se vraćaju korenima. Bilo kako bilo, ova dva programska jezika su toliko slična da ako naučite jedan od njih onda ste naučili 90 posto drugog programskog jezika.

ANSI standard

Accredited Standards Committee, radeći po instrukcijama American National Standards Instituea (ANSI), napravio je internacionalni standard za programski jezik C++.

C++ standard se sada naziva još i ISO (International Standards Organization) standard, NCITS (National Committee for Information Technology Standards) standard, X3 (staro ime za NCITS) standard i ANSI/ISO standard. U ovoj knjizi će se koristiti termin ANSI standard, jer je to termin koji se obično koristi.

ANSI se obično izgovara kao "antsi" sa mekim "t".

ANSI standard je jedan od načina da se programski jezik C++ učini portabilnim, da se obezbedi da se kod koji poštuje ANSI standarde koji pravite za GNU kompajlere može kompajlirati bez grešaka pomoću bilo kog kompajlera koji napravi neka druga programerska kuća (recimo Microsoft). Pošto kod koji se nalazi u ovoj knjizi poštuje ANSI pravila, trebalo bi da se bez problema kompajlira na Mac računarima, pod Windowsom ili na Alpha računarima.

Za većinu ljudi koji uče C++, ANSI standard neće biti vidljiv. Standard se već neko vreme nije menjao i većina velikih programerskih kuća ga poštuje. Mi smo se potrudili da sav kod u ovom izdanju knjige poštuje ANSI standard.

Priprema za pravljenje programa

Programski jezik C++, možda više nego drugi programski jezici, zahteva da programer prvo struktuira program pre nego što ga napiše. Trivijalni problemi, kakvi su oni koji se razmatraju u nekoliko prvih poglavlja knjige, ne zahtevaju mnogo priprema. Složeni problemi, međutim, kakvi su oni sa kojima se profesionalni programeri svakodnevno susreću, zahtevaju pripremu. Što je priprema temeljnija, to je verovatnije da će program rešiti problem za koji je napravljen u zadatom roku i u okviru sredstava koja su namenjena za pravljenje programa. Dobra priprema čini da program ima malo grešaka i da se lako može održavati. Procenjeno je da 90 posto cene programa odlazi na otklanjanje grešaka i njegovo održavanje. Dobra priprema u velikoj meri može smanjiti troškove i ima značajan uticaj na cenu celog projekta.

Prvo pitanje koje sebi treba da postavite prilikom priprema za pravljenje programa jeste kakav je problem koji pokušavate da rešite. Svaki program treba da ima jasan cilj, a videćete da i najjednostavniji programi u ovoj knjizi imaju takav cilj.

Drugo pitanje koje sebi treba da postavi programer jeste može li doći do cilja, a da ne mora sam da napravi ceo softver. Ponovno korišćenje starih programa, korišćenje olovke i papira (stari, originalni, manuelni način obavljanja posla) ili kupovina softvera često je način na koji ćete bolje rešiti problem nego da sami napravite novi softver. Programer koji može da ponudi ovakve alternative nikada neće ostati bez posla; pronalaženje najjeftinijeg rešenja za svakodnevne probleme uvek će stvoriti nove mogućnosti.

Pod pretpostavkom da razumete problem i da on zahteva da napravite potpuno nov program, onda se može reći da možete početi sa radom.

Postupak pomoću kojeg u potpunosti razumete problem (analiza) i pravljenje rešenja (pravljenje programa) neophodni su koraci za pravljenje komercijalnih aplikacija. Mada ovi koraci treba da prethode pisanju koda – to jest, prvo morate razumeti problem i napraviti rešenje pre nego što ga implementirate – bolje je da osnovnu sintaksu i semantiku programskog jezika C++ naučite na osnovu formalnih analiza i tehnika pravljenja programa.

GNU/Linux okruženje

U ovoj knjizi se pretpostavlja da koristite GNU kompajlere u tekstualnom okruženju ili nekoj sličnoj kombinaciji. Dakle, kompajler koji koristite ima režim rada u kojem rezultate možete da šaljete direktno na ekran a da pri tom ne morate da brinete o grafičkom okruženju kakvi su recimo Windows ili Macintosh. Ovaj režim rada je poznat kao režim *konzole* i standardan je za GNU kompajlere. Ukoliko radite u nekom drugom okruženju, moraćete da potražite, recimo, opciju *console* ili *easy window* ili da pogledate dokumentaciju kompajlera. Ukoliko pod Linuxom koristite grafičko okruženje, najbolje bi bilo da otvorite prozor za emulaciju terminala (recimo KDT) tako da možete raditi u čistom tekstualnom režimu.

Kada koristite GNU, možete koristiti EMACS, vi ili neki drugi editor teksta. Ukoliko koristite neki drugi kompajler, na raspolaganju ćete imati drugačije opcije – možda kompajler koji koristite ima ugrađeni editor teksta ili za pravljenje tekstualnih datoteka možete koristiti neki komercijalni editor teksta. Ono što je važno da zapamtite jeste da bez obzira na koji način unosite program, morate da ga sačuvate kao tekstualnu datoteku tako da se u tekstu ne nalaze komande kojima se uređuje tekst. Neki od editora teksta koje možete koristiti jesu Windows Notepad, DOS-ova komanda `edit`, Brief, Epsilon, EMACS i vi. Mnogi komercijalni programi za obradu teksta, kakvi su WordPerfect, Word i mnogi drugi, u tekst umeću specijalne znake, ali pored toga omogućavaju da sadržaj dokumenta sačuvate kao tekstualnu datoteku – dakle, obratite pažnju na koji način zapisujete datoteku.

Datoteke koje pravite pomoću editora nazivaju se resursne datoteke i za programski jezik C++ imaju ekstenziju .cpp, .cp ili.c U ovoj knjizi sve datoteke sa izvornim kodom imenovali smo tako da im je ekstenzija .cxx, jer na taj način GNU kompajleri prepoznaju da se radi o datotekama koje sadrže C++ kod bez obzira na kojoj platformi radite. Ukoliko koristite neku drugu ekstenziju, proverite kakvi su zahtevi kompajlera sa kojim radite.

Za GNU kompajlere ekstenzija je veoma bitna. Trebalo bi da koristite ekstenziju .cxx ili .c++. Mnogim drugim C++ kompajlerima ekstenzija nije bitna. Ukoliko drugačije ne naznačite, oni će koristiti ekstenziju .cpp. Međutim, budite oprezni; neki kompajleri datoteke sa ekstenzijom .c smatraju za datoteke koje sadrže C kod, a datoteke sa ekstenzijom .cpp za datoteke koje sadrže C++ kod. Ponovićemo, proverite dokumentaciju kompajlera koji koristite.

Uradite	Nemojte
Koristite jednostavne editore teksta (kao što su vi, EMACS ili čak DOS-ova komanda edit) kada pravite izvorni kod ili koristite editor koji je ugrađen u kompajler koji koristite.	Nemojte koristiti programe za obradu teksta koji uz tekst zapisuju specijane znake kojima se tekst uređuje. Ukoliko ipak koristite takav program, datoteku zapišite na disk kao ASCII tekstualnu datoteku.
Datotekama koje pravite dodelite ekstenzije .cxx ili .c++.	
Pročitajte priručnik za korišćenje GNU kompajlera da biste saznali kako on i linker rade i da biste naučili kako da kompajlirate i povežete programe.	

Kompajliranje izvornog koda

Mada izvorni kod koji se nalazi u datoteci može izgledati nejasno i svako ko ne poznaje programski jezik C++ imaće probleme da ga razume, kod se ipak nalazi u obliku koji ljudi mogu da čitaju. Izvorni kod nije program i ne može se izvršiti ili pokrenuti kao što to možete uraditi sa programom.

Da biste izvorni kod pretvorili u program, koristite kompajler. Najjednostavniji način da pozovete kompajler jeste da upotrebite komandu kao što je sledeća:

```
g++ file.c++ -o file
```

Ukoliko radite na platformi na kojoj se u imenu datoteke ne može koristiti znak plus (+), kao što je MS-DOS, onda biste upotrebili sledeću komandu:

```
gxx file.cxx -o file.exe
```

U oba slučaja, reč *datoteka* koja se pojavljuje u komandi treba da zamenite imenom datoteke koju kompajlirate, odnosno imenom koje dodeljujete izvršnoj datoteci. Kompajler će napraviti izvršnu datoteku čije je ime `datoteka` ili `datoteka.exe`. Ukoliko izostavite opciju –o, kompajler će napraviti datoteku a.out (Linux) ili a.exe (MS-DOS) respektivno.

Ukoliko koristite neki drugi kompajler, proverite dokumentaciju da biste saznali kako da pozovete kompajler i kako da mu saopštite gde će pronaći izvorni kod – način na koji to radite razlikuje se za razne kompajlere.

Pošto se izvorni kod kompajlira, pravi se objektna datoteka. Ova datoteka često ima ekstenziju .o (Linux) ili .obj (MS-DOS). Međutim, ta datoteka još uvek nije izvršna (program). Da biste tu datoteku pretvorili u program, morate pokrenuti linker.

Pravljenje izvršne datoteke pomoću linkera

C++ programi se obično prave povezivanjem jedne ili više objektnih datoteka sa jednom ili više biblioteka. *Biblioteka* predstavlja skup datoteka koje se mogu povezati a koje dobijate uz kompajler, koji ste zasebno kupili, ili koje ste napravili i kompajlirali. Uz sve C++ kompajlere dobijate biblioteke sa korisnim funkcijama (ili procedurama) i klasama koje možete koristiti u programima. *Funkcija* je blok koda koji obavlja neku operaciju nad podacima, recimo sabira dva broja ili prikazuje rezultat na ekranu. Klasa je kolekcija podataka i odgovarjućih funkcija; klasama ćemo se pozabaviti u poglavlju Dan 5, "Funkcije".

Koraci koje treba obaviti kako bi se napravila izvršna datoteka su:

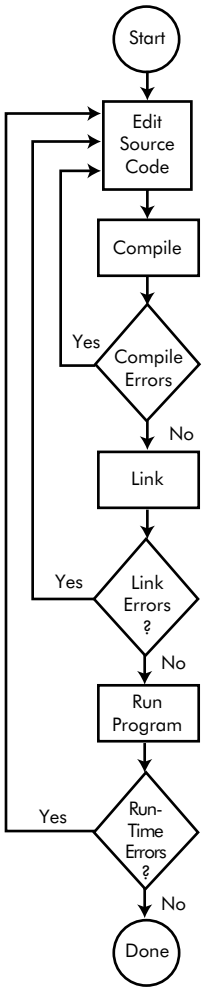
1. Pravljenje datoteke sa izvornim kodom čija je ekstenzija .c++ ili .cxx.
2. Kompajliranje izvršnog koda u datoteku koja ima objektni oblik.
3. Povezivanje objektne datoteke sa bibliotekama kako bi se napravio program.

GNU kompajleri automatski pozivaju linker (njegovo ime je ld) kako bi napravili izvršnu datoteku.

Postupak pravljenja programa

Ukoliko bi se svaki program koji napravite odmah ispravno izvršavao, onda bi to značili da ste završili postupak pravljenja programa: pisanje programa, kompajliranje izvornog koda, povezivanje programa i njegovo pokretanje. Na nesreću, skoro svaki program, bez obzira na to koliko je jednostavan, može i ima trivijalne greške, odnosno bagove. Neke greške će sprečiti kompajliranje, neke će prekinuti postupak povezivanja, dok će se neke otkriti prilikom izvršavanja programa.

Bez obzira na vrstu grešaka na koje naiđete, morate ih ispraviti, što zahteva izmenu izvornog koda, ponovno kompajliranje i povezivanje i ponovno pokretanje programa. Ovaj postupak je predstavljen na slici 1.1. Koraci postupka pravljenja programa su predstavljeni dijagramom.



Slika 1.1
Koraci postupka pravljenja C++ programa.

Hello.cxx – vaš prvi C++ program

Tradicionalne knjige o programiranju počinju tako što vas uče da na ekranu napišete reči "Pozdrav celom svetu" ili neku sličnu rečenicu. U ovoj knjizi se poštuje takva tradicija.

Unesite prvi program u editor, onako kako je pokazano na slici 1.1. Pošto se uverite da je sve u redu, sačuvajte datoteku, povežite je i pokrenite. Na ekranu će se prikazati rečenica "Pozdrav celom svetu". Za sada se nemojte previše brinuti kao program radi; ovaj primer služi samo da biste se upoznali sa postupkom pravljenja programa. Svaki aspekt ovog programa će biti objašnjen u nekoliko narednih poglavlja.

NAPOMENA Svi listinzi koji se koriste u ovoj knjizi nalaze se na propratnom CD ROM-u kako bi vam olakšali rad. Imena datoteka u kojima su listinzi imaju oblik lstCC-NN.cxx. CC označava broj poglavlja (01, 02 i tako dalje), a NN označava broj listinga (01 za prvi listing poglavlja, 02 drugi listing poglavlja i tako dalje). Prvi listing knjige (listing 1.1) nalazi se u datoteci lst01-01.cxx. Nećete moći da kompajlirate sve listinge. Takvi listinzi će biti obeleženi u tekstu.

ULAZ **Listing1.1: hello.cxx, program koji prikazuje "Pozdrav celom svetu"**

```
1: #include <iostream.h>
2:
3: int main()
4: {
5:    cout << "Pozdrav celom svetu!\n";
6:    return 0;
7: }
```

Upotrebite datoteku sa listingom koja se nalazi na propratnom CD ROM-u; ukoliko odlučite da sami unesete program, potrudite se da ga unesete onako kako je ovde prikazan. Obratite pažnju na znake interpunkcije. Znak << u redu 5 je simbol preusmeravanja koji se pomoću većine tastatura dobija kada držite pritisnut taster Shift i dva puta pritisnete taster na kome je znak zarez. Red 5 završava znakom tačka i zarez (;). Nemojte ga izostaviti!

Da biste kompajlirali i povezali program koristeći GNU kompajler pod Linuxom, treba da unesete sledeće:

```
g++ lst01-01.cxx -o lst01-01
```

Ukoliko odaberete neki drugi kompajler, pročitajte instrukcije za njegovo korišćenje. Većina kompajlera automatski povezuje program, ali za svaki slučaj proverite dokumentaciju.

Ukoliko se prilikom kompajliranja prijave greške, pažljivo proverite kod i utvrdite gde se razlikuje od koda koji je dat listingom 1.1. Ukoliko se prijavi greška u redu 1, recimo `cannot find file iostream.h` (ne mogu da pronađem datoteku iostream.h), u dokumentaciji komapajlera o njegovom podešavanju pronađite kako da uključite putanju ili promenljive okruženja.

Ukoliko se prijavi greška da ne postoji prototip funkcije `main` (kada koristite neki drugi kompajler a ne GNU), onda ispred reda 3 unesite red `int main();`. Ovaj red koda ćete morati da unesete pre početka funkcije `main` u svakom programu koji se nalazi u ovoj knjizi. Za većinu kompajlera nećete morati da unosite ovaj red, mada će, ipak, to neki zahtevati.

Program bi trebalo ovako da izgleda:

```
1: #include <iostream.h>
2: int main(); // most compilers don't need this line
3: int main()
4: {
5:     cout <<"Hello World!\n";
6:     return 0;
7: }
```

Pokušajte da izvršite program `lst01-01`; na ekranu bi trebalo da se prikaže

Pozdrav celom svetu!

Ukoliko se prikaže, čestitamo! Upravo ste uneli, kompajlirali i izvršili vaš prvi C++ program. Možda program i nije nešto naročito, ali je gotovo svaki profesionalni C++ programer počeo programiranje istim ovakvim programom.

Korišćenje standardnih biblioteka

Kako bismo bili sugurni da čitaoci koji koriste starije verzije kompajlera neće imati problema sa kodom koji se nalazi u ovoj knjizi mi koristimo raniji oblik datoteka zaglavlja:

```
#include <iostream.h>
```

umesto novih standardnih biblioteka zaglavlja:

```
#include <iostream>
```

Ovakav način zapisivanja bi trebalo da prihvati većina kompajlera, ali ima nekoliko nedostataka. Ukoliko više volite da koristite nove standardne biblioteke, onda vaš kod promenite u

```
#include <iostream>
```

i dodajte red

```
using namespace std;
```

odmah ispod spiska datoteka koje uključujete u program. Napredne teme o korišćenju ključne reči namespace detaljno su objašnjene u poglavlju Dan 17, "Imenovani prostori".

Bilo da koristite ili ne koristite standardne datoteke zaglavlja, kod koji se nalazi u knjizi izvršava se bez menjanja. Osnovna razlika između starijih biblioteka i novih standardnih je u biblioteci iostream (koja je objašnjena u pogavlju Dan 16, "Tokovi"). Čak i takve izmene neće uticati na kod koji se nalazi u knjizi; izmene su jedva primetne, ezoterične i prevazilaze opseg uvodnog primera.

Korišćenje kompajlera g++

Svi programi koji se nalaze u ovoj knjizi testirani su pomoću GNU kompajlera g++ verzija 2.9.5; mnogi primeri koji se nalaze u knjizi testirani su i pomoću verzije 2.7.2. Teorijski gledano, pošto kod poštuje ANSI pravila, onda bi svi programi koji se nalaze u knjizi trebalo da bez problema rade sa svakim kompajlerom koji poštuje ANSI pravila.

Teorijski gledano, teorija i praksa su jedno te isto. U praksi to nikada nije tako.

Da bismo vas upoznali sa postupkom pravljenja programa, u ovom odeljku ćemo vas naučiti kako da unesete, kompajlirate, povežete i pokrenete program pomoću GNU kompajlera. Ukoliko koristite neki drugi kompajler, onda se pojedini koraci mogu razlikovati. Čak i kada koristite GNU kompajler, verziju 2.7.2 ili verziju 2.9.5, pročitajte dokumentaciju kako biste naučili kako da nastavite sa radom.

Izrada projekta "Pozdrav celom svetu"

Da biste napravili i testirali program "Pozdrav celom svetu" sledite naredne korake:

1. Odaberite editor u kome ćete raditi i napravite datoteku.
2. Unesite kod listinga 1.1 (ili upotrebite editor kako biste otvorili datoteku `lst01-01.cxx`).
3. Sačuvajte datoteku i zatvorite editor.
4. Otkucajte komandu `compile`.
5. Otkucajte ime izvršne datoteke kako biste pokrenuli program.

Greške prilikom kompajliranja

Greške prilikom kompajliranja mogu se javiti iz raznih razloga. Te greške su obično posledica pogrešno unetih komandi ili manjih nenamernih grešaka. Dobro napravljeni kompajleri (kakav je GNU) ne samo da vas obaveštavaju šta ste pogrešno uradili već vas obaveštavaju gde ste u kodu napravili grešku. Najbolji kompajleri će vas čak posavetovati kako da ispravite grešku!

Način na koji se prijavljuje greška možete videti ako u programu namerno napravite grešku. Ukoliko se program "Pozdrav celom svetu" pravilno izvršava, onda ga izmenite tako što ćete iz reda 7 ukloniti zatvorenu zagradu. Program bi trebalo da izgleda ovako.

ULAZ **Listing 1.2: Demonstracija načina prijavljivanja grešaka prilikom kompajliranja**

```
1: #include <iostream.h>
2:
3: int main()
4: {
5:    cout << "Pozdrav celom svetu!\n";
6:    return 0;
```

Prilikom ponovnog kompajliranja programa GNU kompajler će prikazati sledeću grešku:

```
./lst01-02.cxx: In function `int main()':
./lst01-02.cxx:7: parse error at end of input
```

Drugi kompajleri bi prijavili grešku na sledeći način:

```
Hello.cpp, line 7: Compound statement missing terminating; in
➥function main().
```

Ili bi to uradili ovako:

```
F:\Mcp\Tycpp21d\Testing\List0101.cpp(8) : fatal error C1004:
unexpected end of file found
Error executing cl.exe.
 }
```

Ova poruka vas obaveštava u kojoj datoteci i u kom redu te datoteke je otkriven problem i kakve je vrste (priznajem, poruka nije baš najjasnija). Primetićete da vas poruka upućuje na red 7. Kompajler nije siguran da li ste zatvorenu zagradu hteli da otkucate pre ili posle komande cout u redu 6. Ponekad vas poruka o grešci vodi do mesta koje je najbliže stvarnom problemu. Kada bi kompajler sa sigurnošću mogao da identifikuje svaki problem onda bi ga sam rešio.

Zaključak

Pošto završite ovo poglavlje, trebalo bi da razumete kako je nastao programski jezik C++ i kakve probleme možete rešiti koristeći ga. Trebalo bi da budete sigurni da je učenje programskog jezika C++ pravi izbor za svakog ko želi da nauči programiranje u narednoj deceniji. Programski jezik C++ sadrži alate za objektno orijentisano programiranje i performanse programskog jezika niskog nivoa, što čini da se C++ najčešće bira za programski jezik pomoću koga se prave programi.

U ovom poglavlju ste naučili kako da unesete, kompajlirate, povežete i pokrenete vaš prvi C++ program i kako izgleda uobičajeni postupak za njegovo pravljenje. Takođe, naučili ste ponešto o objektno orijentisanom programiranju. Ovim temama ćemo se pozabaviti tokom naredne tri nedelje. U dodatku ćete naučiti napredne tehnike rada sa GNU skupom alata i programiranja pod Linuxom.

Pitanja i odgovori

P **Kakva je razlika između editora teksta i programa za obradu teksta?**

O Editor teksta pravi datoteku koja sadrži samo tekst. Nisu neophodne komande za formatiranje teksta ili specijalni simboli kakve zahteva program za obradu teksta. Takođe, tekst datoteke ne smešta automatski na određenu širinu, a nema ni masnih slova i kurziva.

P **Ako kompajler koji koristim ima ugrađen editor teksta, moram li ja da koristim taj editor?**

O GNU kompajleri nemaju ugrađen editor teksta. Uz Linux dobijate dva editora teksta. To su vi i EMACS. Gotovo svi kompajleri će moći da kompajliraju kod iz datoteke koju napravite pomoću bilo kog editora teksta. Prednost korišćenja ugrađenog editora može biti mogućnost naizmeničnog korišćenja koraka editovanja i kompajliranja. Sofisticirani kompajleri imaju potpuno integrisano razvojno okruženje, što programerima omogućava da koriste help datoteke, da odmah edituju i kompajliraju kod i da u okviru tog okruženja reše probleme koji su nastali prilikom kompajliranja ili povezivanja programa.

P **Mogu li da zanemarim poruke upozorenja koje mi upućuje kompajler?**

O U mnogim knjigama nećete naći precizan odgovor na ovo pitanje, ali moje mišljenje je sledeće: Ne! Od prvog dana naučite da poruke upozorenja tretirate kao poruke o greškama. Programski jezik C++ koristi kompajler kako bi vas upozorio da radite nešto što inače niste nameravali da uradite. Ozbiljno shvatite ta upozorenja i učinite sve što je potrebno kako se ne bi ponovo javljala.

Kada dobijete poruku upozorenja, znači da kompajler može da napravi izvršnu datoteku na osnovu izvornog koda koji ste uneli, ali da ne može da poveruje da vi zaista želite da uradite ono što ste napisali u kodu!

P **Šta je to vreme kompajliranja?**

O Vreme kompajliranja je vreme u koje ste pokrenuli kompajler, nasuprot vremenu povezivanja (kada ste pokrenuli linker) ili vremenu izvršavanja (kada se program izvršava). Ovo su samo skraćenice koje programeri koriste kako bi naznačili tri mesta na kojima se najčešće uoče greške. Pošto GNU kompajler poziva linker samo kada je kompajliranje uspešno obavljeno, lako se može predvideti vreme rada likera (ali, verujte mi na reč, ono zaista postoji).

Radionica

U radionici se nalaze pitanja pomoću kojih možete utvrditi znanje i vežbe pomoću kojih bi trebalo da steknete iskustvo praktične primene tog znanja. Pokušajte samostalno da odgovorite na pitanja i uradite vežbe pre nego što pogledate odgovore i rešenja koja se nalaze u Dodatku D, "Odgovori na pitanja i rešenja vežbi". Takođe, potrudite se da razumete sve odgovore pre nego što pređete na sledeće poglavlje.

Kviz pitanja

1. Kakva je razlika između interpretatora i kompajlera?
2. Kako kompajlirate izvorni kod pomoću kompajlera?
3. Šta radi linker?
4. Nabrojte korake uobičajenog postupka pravljenja programa?

Vežbe

1. Proučite sledeći program i pokušajte da bez pokretanja programa pogodite šta on radi.

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

2. Unesite program iz prethodne vežbe, kompajlirajte ga i povežite. Šta on radi? Da li radi ono što ste pretpostavljali?
3. Unesite sledeći program i kompajlirajte ga. Kakve se greške prijavljuju?

```
1: include <iostream.h>
2: int main()
3: {
4:     cout << "Pozdrav celom svetu\n";
5:     return 0;
6: }
```

4. Ispravite greške programa iz prethodne vežbe, ponovo ga kompajlirajte, povežite i pokrenite. Šta on radi?