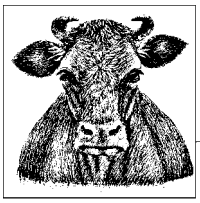
Jezik



1
Osnove jezika

U ovom poglavlju opisujemo osnovne karakteristike i elemente programskog jezika C.

Karakteristike jezika C

C je proceduralni programski jezik opšte namene. Stvorio ga je sedamdesetih godina prošlog veka Dennis Ritchie iz laboratorije AT&T Bell u Murray Hillu (New Jersey) kako bi omogućio implementaciju operativnog sistema Unix i uslužnih programa što manje zavisnih od konkretne hardverske platforme. Sledeće odlike čine jezik C pogodnim za tu svrhu:

- prenosivost izvornog koda
- sposobnost funkcionisanja u „skladu s mašinom“
- efikasnost

To je tvorcima Unixa omogućilo da najveći deo tog operativnog sistema napišu na jeziku C – preostalo je da se na asembleru realizuje tek minimum manipulacija hardverom koje su zavisile od sistema.

Prethodnici jezika C su BCPL (Basic Combined Programming Language), programski jezik bez tipova, autora Martina Richardsa, i naslednik BCPL-a, jezik B, koji je razvio Ken Thompson. U odnosu na njih, C je obogaćen raznim tipovima podataka: znakovima, numeričkim tipovima, nizovima, strukturama itd. Godine 1978, Brian Kernighan i Dennis Ritchie objavili su zvaničan opis programskog jezika C. Taj opis je prvi pravi standard, i često ga kratko nazivaju „K&R“.* Jezik C je izuzetno prenosiv zahvaljujući svom jezgru koje sadrži tek nekoliko elemenata koji zavise od hardvera. Na primer, u izvornom obimu, jezik C nema naredbe za pristup datotekama niti za dinamičko upravljanje memorijom. Nema ni naredbe za ulazne i izlazne operacije za konzole. Umesto toga, obimna standardna biblioteka sadrži funkcije za te zadatke.

* Drugo izdanje, prilagođeno prvom ANSI standardu jezika C, dostupno je kao priručnik *The C Programming Language*, *2nd Ed.*, autora Briana W. Kernighana i Dennisa M. Ritchiea (Englewood Cliffs, N.J.: Prentice Hall, 1988).

Ovakav koncept omogućava kompaktnost prevodioca jezika C i prenosivost na nove sisteme. Povrh toga, kada instalirate prevodilac na nov sistem, moći ćete da bez izmena prevedete većinu funkcija iz standardne biblioteke, jer su napisane na prenosivom jeziku C. Zato su prevodioci jezika C dostupni gotovo svakom računarskom sistemu.

Jezik C je namenjen za sistemsko programiranje, tako da danas najviše služi za programiranje sistema koji se ugrađuju u razne uređaje. Budući da je C prenosiv, strukturalni jezik visokog nivoa, mnogi programeri na njemu pišu aplikacije poput moćnih programa za obradu teksta, baza podataka i grafičkih programa.

Struktura programa na jeziku C

Funkcije (engl. *functions*) predstavljaju proceduralne gradivne elemente jezika C. Funkcije mogu pozivati jedna drugu. Svaka funkcija u dobro osmišljenom programu služi određenoj svrsi. Funkcije sadrže *naredbe* (engl. *statements*) koje određuju sekvencijalno izvršavanje programa; naredbe se mogu grupisati u *blokove* (engl. *blocks*). Programer može upotrebiti gotove funkcije iz standardne biblioteke ili će ih sam napisati ukoliko nijedna standardna funkcija ne obavlja određeni zadatak. Pored standardne biblioteke, dostupne su i mnoge druge, specijalizovane, poput biblioteka grafičkih funkcija. Ukoliko se koriste nestandardne biblioteke, ograničava se prenosivost programa samo na one sisteme na kojima su instalirane.

Svaki program na jeziku C mora imati definisanu barem jednu funkciju – to je funkcija `main()`, koja se prva poziva na početku programa. Funkcija `main()` predstavlja najviši nivo upravljanja programom i može pozivati ostale funkcije kao potprograme (engl. *subroutines*).

Primer 1-1 prikazuje strukturu jednostavnog programa na jeziku C. Deklaracije, pozive funkcija, izlazne tokove i druge konkretne elemente programa, detaljnije ćemo opisati kasnije. Prvo ćemo predstaviti opštu strukturu izvornog koda programa na jeziku C. U programu iz primera 1-1 definisane su dve funkcije, `main()` i `circularArea()`. Prva funkcija poziva drugu da izračuna površinu kruga datog poluprečnika, potom poziva funkciju `printf()` iz standardne biblioteke da prikaže rezultat u obliku formatiranih znakovnih nizova.

Primer 1-1. Jednostavan program na jeziku C

```
// circle.c: Izračunava i ispisuje površinu krugova

#include <stdio.h>                // Pretprocesorska direktiva

double circularArea( double r );  // Deklaracija funkcije (prototip)

int main( )                       // Počinje definicija funkcije main( )
{
  double radius = 1.0, area = 0.0;

  printf( "    Areas of Circles\n\n" );
  printf( "     Radius          Area\n"
          "-------------------------\n" );
```

Primer 1-1. Jednostavan program na jeziku C (nastavak)

```
  area = circularArea( radius );
  printf( "%10.1f     %10.2f\n", radius, area );

  radius = 5.0;
  area = circularArea( radius );
  printf( "%10.1f     %10.2f\n", radius, area );

  return 0;
}

// Funkcija circularArea( ) izračunava površinu kruga
// Parametar: Poluprečnik kruga
// Povratna vrednost: Površina kruga

double circularArea( double r )      // Počinje definicija funkcije
circularArea( )
{
  const double pi = 3.1415926536;    // Pi je konstanta
  return  pi * r * r;
}
```

Rezultat:

```
        Areas of Circles

         Radius          Area
    -------------------------
            1.0          3.14
            5.0         78.54
```

Prevodilac zahteva da se svaka funkcija *deklariše* pre nego što se pozove. Prototip funkcije `circularArea( )` u trećem redu u primeru 1-1, sadrži informacije potrebne da bi se prevela naredba kojom se ta funkcija poziva. Prototipovi funkcija iz standardne biblioteke nalaze se u standardnim datotekama zaglavlja (engl. *header-files*). Pošto datoteka zaglavlja *stdio.h* sadrži prototip funkcije `printf( )`, ta funkcija se indirektno deklariše *pretprocesorskom direktivom* `#include <stdio.h>` koja nalaže pretprocesoru prevodioca da umetne sadržaj navedene datoteke. (Pogledajte i odeljak „Kako radi prevodilac jezika C“, pri kraju ovog poglavlja.)

Funkcije u programu možete definisati proizvoljnim redom. U primeru 1-1, funkcija `circularArea( )` mogla je prethoditi funkciji `main( )`. U tom slučaju, deklaracija prototipa funkcije `circularArea( )` bila bi suvišna, jer je definicija funkcije ujedno i njena deklaracija.

Definicije funkcija ne mogu se ugnezditi jedna u drugu; u bloku funkcije možete definisati lokalnu promenljivu, ali ne i lokalnu funkciju.

Izvorne datoteke

Definicije funkcija, globalne deklaracije i pretprocesorske direktive, sačinjavaju izvorni kôd programa na jeziku C. Izvorni kôd malih programa piše se u jednoj izvornoj datoteci. Pošto definicije funkcija u opštem slučaju zavise od pretprocesorskih

direktiva i globalnih deklaracija, unutrašnja struktura izvorne datoteke obično obuhvata sledeće elemente:

1. pretprocesorske direktive
2. globalne deklaracije
3. definicije funkcija

Jezik C podržava modularno programiranje tako što omogućava da organizujete program u proizvoljno mnogo izvornih datoteka i datoteka zaglavlja, te da ih zasebno menjate i prevodite. Svaka izvorna datoteka sadrži logično povezane funkcije, na primer funkcije programa za korisničko okruženje. Uobičajeno je da se izvorne datoteke na jeziku C označavaju imenom sa sufiksom *.c*.

U primerima 1-2 i 1-3 prikazuje se isti program kao u prvom primeru, samo što je podeljen u dve izvorne datoteke.

Primer 1-2. Prva izvorna datoteka, sa funkcijom main()

```
// circle.c: Ispisuje površine krugova.
// Koristi funkciju circlararea.c u računu

#include <stdio.h>
double circularArea( double r );

int main( )
{
  /* ...Kao u primeru 1-1 ...*/
}
```

Primer 1-3. Druga izvorna datoteka, sa funkcijom circularArea()

```
// circlararea.c: Izračunava površine krugova.
// Poziva je funkcija main( ) u programu circle.c

double circularArea( double r )
{
  /* ... Kao u primeru 1-1 ... */
}
```

Kada je program podeljen u nekoliko izvornih datoteka, moraćete da deklarišete istu funkciju i globalne promenljive, i da definišete iste makroe i konstante u više datoteka. Te deklaracije i definicije sačinjavaju svojevrsno zaglavlje koje je uglavnom nepromenjeno u čitavom programu. Jednostavnosti i jezgrovitosti radi, tu informaciju možete zapisati samo jednom u posebnoj *datoteci zaglavlja*, a potom je pomoću direktive `#include` pozivati u svakoj izvornoj datoteci. Uobičajeno je da se datoteke zaglavlja označavaju imenom sa sufiksom *.h*. Datoteka zaglavlja koju eksplicitno obuhvata datoteka sa izvornim kodom, može i sama da sadrži druge datoteke zaglavlja.

Izvorna datoteka na jeziku C zajedno sa svim datotekama zaglavlja koje obuhvata, sačinjava *jedinicu za prevođenje* (engl. *translation unit*). *Prevodilac* (engl. *compiler*) prevodi njen sadržaj sekvencijalno, razlažući izvorni kôd na tokene – najmanje

semantičke jedinice, poput imena promenljivih i operatora. Više detalja o tome naći ćete u odeljku „Tokeni“ na kraju ovog poglavlja.

Dva susedna tokena može razdvajati proizvoljan broj razmaka, što daje veliku slobodu pri formatiranju izvornog koda. Pošto ne postoje pravila o prekidu reda (engl. *line break*) ili o uvlačenju, slobodno koristite razmake, tabulatore i prazne redove da bi izvorni kôd bio čitljiviji. Pretprocesorske direktive su nešto ograničenije – moraju biti jedine u datom redu, a pre znaka `#` na početku reda mogu se naći samo razmaci i tabulatori.

Izvorni kôd može se formatirati na razne načine. Uobičajena su dva pravila:

- Svaku novu deklaraciju i naredbu počnite u novom redu.
- Uvucite ugnežđene strukture ili blokove naredaba.

Komentari

U izvornom kodu programa na jeziku C, ne štedite komentare. Možete ih umetnuti na dva načina: *blok komentara* počinje znakovima `/*`, a završava se znakovima `*/`, dok *red komentara* počinje dvostrukom kosom crtom `//`, a završava se narednim znakom za novi red.

Graničnicima `/*` i `*/` omeđite jednoredni ili višeredni komentar. Na primer, u narednom prototipu funkcije, tri tačke (...) označavaju da funkcija `open()` ima treći, opcioni parametar. Komentar rasvetljava ulogu opcionog parametra:

```
int open( const char *name, int mode, ... /* dozvoljen celobrojni parametar
                                           */ );
```

Pomoću dvostruke kose crte možete umetnuti komentare koji zauzimaju čitav red, ili organizovati izvorni kôd u dve kolone – u levoj je kôd programa, a u desnoj su komentari:

```
const double pi = 3.1415926536;     // Pi je konstanta
```

Ovakvi komentari zvanično su dodati jeziku C standardom C99, premda ih je i pre toga podržavala većina prevodilaca. Ponekad ih opisuju kao „komentare u stilu jezika C++“, mada vode poreklo iz jezika BCPL, prethodnika jezika C.

Kad su znakovi `/*` i `//` napisani pod navodnicima koji izdvajaju znakovnu konstantu (engl. *character constant*) ili literal znakovnog niza (engl. *string literal*), oni ne određuju početak komentara. Na primer, sledeća naredba ne sadrži komentare:

```
printf( "Komentari na C-u počinju znakovima /* ili //.\n" );
```

Prilikom pregledanja znakova u komentaru, prevodilac traga samo za oznakama kraja komentara; zato je blok komentara nemoguće ugnezditi. Međutim, pomoću graničnika `/*` i `*/` možete umetnuti komentar dela programa koji sadrži komentare obeležene s dve kose crte:

```
/* Privremeno uklanja dva reda:
  const double pi = 3.1415926536;     // Pi je konstanta
  area = pi * r * r                   // Izračunava površinu
   Privremeno uklonjen deo programa  */
```

Da biste komentar dodali programu koji sadrži blok komentara, upotrebite uslovnu pretprocesorsku direktivu (detaljnije u poglavlju 14):

```
#if 0
  const double pi = 3.1415926536;     /* Pi je konstanta    */
  area = pi * r * r                   /* Izračunava površinu */
#endif
```

Pretprocesor zamenjuje svaki komentar razmakom. Na primer, sekvenca znakova `min/*max*/Value`, tumači se kao dva tokena, `min Value`.

Skupovi znakova

Jezik C razlikuje okruženje u kome prevodilac prevodi izvornu datoteku programa – *prevodilačko okruženje* (engl. *translation environment*) – od okruženja u kome se prevedeni program izvršava – *izvršno okruženje* (engl. *execution environment*). U skladu s tim, C definiše dva skupa znakova: *skup izvornih znakova* (engl. *source character set*), koji se može koristiti u izvornom kodu, i *skup izvršnih znakova* (engl. *execution character set*), sa znakovima koji se čitaju prilikom izvršavanja programa. Ovi skupovi znakova identični su u mnogim implementacijama jezika C. Ukoliko se razlikuju, prevodilac će pretvoriti znakovne konstante i literale znakovnih nizova iz izvornog koda u odgovarajuće elemente izvršnog skupa znakova.

Oba skupa sadrže skup osnovnih znakova (engl. *basic character set*) i proširene znakove (engl. *extended characters*). Jezik C ne zadaje proširene znakove – oni obično zavise od lokalnog jezika. Skup osnovnih znakova dopunjen proširenim znakovima, čini prošireni skup znakova (engl. *extended character set*).

U osnovne znakove koje sadrže i izvorni i izvršni skup znakova, spadaju:

Slova latinske abecede

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Decimalne cifre

```
0 1 2 3 4 5 6 7 8 9
```

Narednih 29 interpunkcijskih znakova

```
! " # % & ' ( ) * + , - . / : ; < = > ? [ \ ] ^ _ { | } ~
```

Pet vrsta razmaka

Belina, horizontalni tabulator, vertikalni tabulator, znak za novi red (engl. *new line*) i znak za prelazak na narednu stranu (engl. *form feed*).

Skup osnovnih izvršnih znakova sadrži i četiri znaka koji se ne štampaju (engl. *nonprintable characters*): *prazan znak* (engl. *null*) kojim se označava kraj niza znakova; *upozorenje* (engl. *alert*); *znak za brisanje unazad* (engl. *backspace*); *znak za povratak na početak tekućeg reda* (engl. *carriage return*). Ove znakove predstavljate kao znakovne konstante i literale znakovnih nizova tako što unesete odgovarajuće *izlazne sekvence* (engl. *escape sequences*). Izlazna sekvenca počinje obrnutom kosom crtom: `\0` za prazan znak, `\a` za upozorenje, `\b` za znak za brisanje unazad i `\r` za znak za povratak na početak tekućeg reda. Više detalja naći ćete u poglavlju 3.

Brojevne vrednosti znakova – kodovi znakova – zavise od konkretne implementacije jezika C. Sam jezik nameće samo naredna ograničenja:

- Svaki znak u osnovnom skupu znakova mora biti predstavljen jednim bajtom.
- Svi bitovi u bajtu za prazan znak imaju vrednost 0.
- Vrednost svake decimalne cifre posle 0 za jedan je veća od prethodne.

Široki i višebajtni znakovi

Jezik C je nastao na engleskom govornom području gde je dominantan skup znakova bio sedmobitni ASCII skup kodova. U međuvremenu, uobičajena jedinica za kodiranje znakova postao je osmobitni bajt. Pošto softver za međunarodnu upotrebu mora biti u stanju da predstavi više znakova od onih koji se kodiraju jednim bajtom, za nelatinične alfabete i za nealfabetske znakovne sisteme poput kineskog, japanskog i korejskog pisma, već decenijama se koriste razne višebajtne šeme kodiranja. Godine 1994. usvojena je dopuna „Normative Addendum 1“ kojom je ISO C standardizovao dva načina predstavljanja većih skupova znakova: *širokim znakovima* (engl. *wide characters*) – u kome se ista širina bita koristi za predstavljanje svih znakova iz skupa, i *višebajtnim znakovima* (engl. *multibyte characters*) – sistem u kome se određeni znak može predstaviti jednim bajtom ili sa nekoliko njih, a vrednost znaka date sekvence bajtova zavisi od konteksta u pratećem znakovnom nizu ili toku podataka.

Iako jezik C ima apstraktne mehanizme za upravljanje različitim šemama kodiranja i za njihovu konverziju, on ne definiše i ne određuje nijednu od njih, niti bilo koji drugi skup znakova sem skupova osnovnih izvornih i izvršnih znakova opisanih u prethodnom odeljku. Drugim rečima, šema kodiranja širokih i višebajtnih znakova zavisi od konkretne implementacije.

Od dopune iz 1994. godine, pored tipa `char`, jezik C sadrži tip `wchar_t` – *široke znakove*. Ovaj tip, definisan u datoteci zaglavlja *stddef.h*, dovoljno je velik da predstavi svaki element proširenog skupa znakova date implementacije.

Iako standard jezika C ne zahteva podršku za skup znakova Unicode, mnoge implementacije koriste Unicode formate UTF-16 i UTF-32 za široke znakove (pogledajte Web lokaciju *http://www.unicode.org*). Unicode standard se u velikoj meri poklapa sa standardom ISO/IEC 10646 i nadskup je mnogih skupova znakova, uključujući sedmobitni ASCII kôd. Prilikom primene Unicode standarda, tip `wchar_t` ima najmanje 16 bitova ili 32 bita, a vrednost promenljive predstavlja jedan Unicode znak. Na primer, naredna definicija dodeljuje promenljivoj `wc` kao početnu vrednost grčko slovo α.

```
wchar_t wc = '\x3b1';
```

Izlazna sekvenca koja počinje sa `\x` označava kôd u heksadecimalnom formatu koji postaje vrednost promenljive – u ovom slučaju, kôd za malo grčko slovo alfa.

U skupu višebajtnih znakova, svaki znak se kodira kao sekvenca koju čini jedan ili više bajtova. Skupovi izvornih i izvršnih znakova mogu da sadrže višebajtne znakove. U tom slučaju, svaki znak u skupu osnovnih znakova zauzima samo jedan bajt, a od

višebajtnih znakova, samo prazan znak sme da ima bajt čiji svi bitovi imaju vrednost 0. Višebajtni znakovi mogu se koristiti za znakovne konstate, literale znakovnih nizova, identifikatore, u komentarima i imenima datoteka zaglavlja. Mnogi skupovi višebajtnih znakova podržavaju određeni jezik, poput japanskog industrijskog standarda (Japanese Industrial Standard – JIS). Skup višebajtnih znakova UTF-8, koji je definisao Konzorcijum za Unicode (Unicode Consortium) može da predstavi sve Unicode znakove. Znakovi u tom skupu zauzimaju od jednog do četiri bajta.

Ključna razlika između višebajtnih znakova i širokih znakova (tipa `wchar_t`) jeste u tome što su svi široki znakovi iste veličine, dok višebajtni znakovi mogu zauzimati različit broj bajtova. Zbog toga je znakovne nizove koji se sastoje od višebajtnih znakova teže obrađivati nego znakovne nizove širokih znakova. Na primer, iako se znak `'A'` može predstaviti jednim bajtom, u višebajtnom znakovnom nizu ne možete ga naći poredeći bajt po bajt, jer ista vrednost bajta na određenoj lokaciji može biti deo drugog znaka. Ipak, višebajtni znakovi su pogodni za čuvanje teksta u datotekama (pogledajte poglavlje 13).

Jezik C ima standardne funkcije za čitanje vrednosti tipa `wchar_t` bilo kog višebajtnog znaka, i za pretvaranje širokog znaka u njegovu višebajtnu varijantu. Na primer, ako prevodilac jezika C koristi Unicode standarde UTF-16 i UTF-8, funkcija `wctomb()` (skraćeno od „wide character to multibyte“ – široki znak u višebajtni) u narednom pozivu vraća višebajtni prikaz grčkog slova α:

```
wchar_t wc = L'\x3B1';     // Malo grčko slovo alfa, α
char mbStr[10] = "";
int nBytes = 0;
nBytes = wctomb( mbStr, wc );
```

Posle poziva funkcije, niz `mbStr` sadrži višebajtne znakove – u ovom primeru, to je sekvenca `"\xCE\xB1"`. Rezultat funkcije `wctomb()` koja se ovde dodeljuje promenljivoj `nBytes`, jeste broj bajtova potrebnih da se predstavi višebajtni znak (dva bajta).

Univerzalna imena znakova

Jezik C podržava univerzalna imena znakova kao način da se upotrebljava prošireni skup znakova, bez obzira na kodiranje koje koristi određena implementacija. Svaki znak iz proširenog skupa možete zadati pomoću *univerzalnog imena znaka* (engl. *universal character name*), a to je Unicode vrednost u obliku:

`\u`*`XXXX`*

ili:

`\U`*`XXXXXXXX`*

gde je *`XXXX`* ili *`XXXXXXXX`* Unicode kôd u heksadecimalnom formatu. Možete napisati malo slovo `u`, potom četiri heksadecimalne cifre, ili kao prefiks upotrebite veliko slovo `U` i iza njega napišite osam heksadecimalnih cifara. Ako su četiri heksadecimalne cifre nula, onda univerzalno ime znaka napišite kao `\u`*`XXXX`* ili kao `\U0000`*`XXXX`*.

Univerzalna imena znakova mogu se naći u identifikatorima, znakovnim konstantama i literalima znakovnih nizova. Ne smete ih koristiti za znakove iz skupa osnovnih znakova.

Znak zadat pomoću univerzalnog imena, prevodilac smešta u skup znakova koji koristi određena implementacija. Na primer, ukoliko se u datom programu koristi izvršni skup znakova ISO 8859-7 (osmobitna grčka slova), u narednoj definiciji promenljivoj `alpha` dodeljuje se kao početna vrednost kôd `\xE1`:

```
char alpha = '\u03B1';
```

Međutim, ako je izvršni skup znakova UTF-16, promenljivu morate definisati kao širok znak:

```
wchar_t alpha = '\u03B1';
```

U ovom slučaju, vrednost koda znaka dodeljenog promenljivoj `alpha` jeste heksadecimalno 3B1, što je i univerzalno ime znaka.

Ne podržavaju svi prevodioci univerzalna imena znakova.

Digrafi i trigrafi

Jezik C podržava alternativno predstavljanje nekih interpunkcijskih znakova kojih nema na svim tastaturama. Šest takvih znakova su *digrafi*, ili dvoznakovne leksičke jedinice – tokeni; prikazani su u tabeli 1-1.

Tabela 1-1. Digrafi

Digraf	Ekvivalent
<:	[
:>	]
<%	{
%>	}
%:	#
%:%:	##

Ako se pojave u znakovnoj konstanti ili literalu znakovnog niza, ove sekvence se ne tumače kao digrafi. U svim drugim slučajevima, ponašaju se kao jednoznakovni tokeni koje predstavljaju. Na primer, naredni delovi programa su identični i daju isti rezultat. Sa digrafima:

```
int arr<::> = <% 10, 20, 30 %>;
printf( "Drugi element niza je <%d>.\n", arr<:1:> );
```

Bez digrafa:

```
int arr[] = { 10, 20, 30 };
printf( "Drugi element niza je <%d>.\n", arr[1] );
```

Rezultat:

```
Drugi element niza je <20>.
```

C sadrži i *trigrafe*, troznakovne sekvence koje počinju duplim znakom pitanja. Treći znak određuje koji interpunkcijski znak trigraf predstavlja (tabela 1-2).

Tabela 1-2. Trigrafi

Trigraf	Ekvivalent
??(	[
??)	]
??<	{
??>	}
??=	#
??/	\
??!	|
??'	^
??-	~

Trigrafi omogućavaju pisanje programa na jeziku C uz upotrebu samo znakova definisanih standardom ISO/IEC 646 koji odgovara sedmobitnom ASCII skupu. U prvoj fazi prevođenja, pretprocesor prevodioca menja trigrafe u njihove jednoznakovne ekvivalente. To znači da se, za razliku od digrafa, trigrafi prevode u jednoznakovne ekvivalente gde god da su, čak i u znakovnim konstantama, literalima znakovnih nizova, komentarima i pretprocesorskim naredbama. Na primer, pretprocesor tumači drugi i treći znak pitanja u sledećoj naredbi kao početak trigrafa:

```
printf("Zanemariti???(y/n) ");
```

Pretprocesor će čitavu naredbu protumačiti na sledeći način:

```
printf("Zanemariti?[y/n) ");
```

Ukoliko morate da upotrebite neku od ovih troznakovnih sekvenci, a ne želite da se protumači kao trigraf, navedite znakove pitanja kao izlaznu sekvencu:

```
printf("Zanemariti\?\?\?(y/n) ");
```

Ako se iza dva znaka pitanja ne nalazi nijedan od znakova iz tabele 1-2, sekvenca nije trigraf i ostaje neizmenjena.

Pored digrafa i trigrafa, kao zamena za interpunkcijske znakove koriste se i predstavljanja logičkih operatora jezika C i operatora nad bitovima (na primer `and` za operator `&&` i `xor` za operator ^), definisana makroima u datoteci zaglavlja *iso646.h*. Više detalja naći ćete u poglavlju 15.

Identifikatori

Pojam *identifikator* (engl. *identifier*) odnosi se na imena promenljivih, funkcija, makroa, struktura i drugih objekata definisanih u programu na jeziku C. Identifikator može da sadrži naredne znakove:

- Slova iz osnovnog skupa znakova, (a–z i A–Z). Identifikator razlikuje velika i mala slova.
- Donju crtu, _.

- Decimalne cifre 0-9, premda prvi znak identifikatora ne sme biti cifra.
- Univerzalna imena znakova koja predstavljaju slova i cifre iz drugog jezika.

Prihvatljivi univerzalni znakovi definisani su u aneksu D standarda jezika C i odgovaraju znakovima definisanim u standardu ISO/IEC TR 10176 (bez osnovnog skupa znakova).

Višebajtni znakovi su takođe dozvoljeni u identifikatorima. Međutim, od konkretne implementacije jezika C zavisi koji su višebajtni znakovi dopušteni i kakvim univerzalnim imenima odgovaraju.

Narednih 37 reči su *rezervisane reči* (engl. *keywords*) u jeziku C; one imaju posebna značenja za prevodioca i ne smeju se koristiti kao identifikatori:

```
auto        enum        restrict    unsigned
break       extern      return      void
case        float       short       volatile
char        for         signed      while
const       goto        sizeof      _Bool
continue    if          static      _Complex
default     inline      struct      _Imaginary
do          int         switch
double      long        typedef
else        register    union
```

Evo nekoliko primera valjanih identifikatora:

```
x  dollar  Break  error_handler  scale64
```

Navodimo i četiri neprihvatljiva identifikatora:

```
1st_rank  switch  y/n  x-ray
```

Ako prevodilac podržava univerzalna imena znakova, onda je i slovo α prihvatljivo kao identifikator, tako da možete definisati promenljivu s tim imenom:

```
double α = 0.5;
```

Editor izvornog koda mogao bi da snimi znak α u izvornoj datoteci kao univerzalan znak `\u03B1`.

Kada birate identifikatore za svoje programe, ne zaboravite da se mnogi već koriste u standardnoj biblioteci jezika C. Tu spadaju imena standardnih bibliotečkih funkcija koja ne možete upotrebiti za sopstvene funkcije niti za globalne promenljive. (Opširnije o tome u poglavlju 15.)

Prevodilac jezika C podržava unapred definisan identifikator `__func__`; možete ga upotrebiti u svakoj funkciji da biste pristupili konstantnom znakovnom nizu koji sadrži ime funkcije. To je pogodno za prijavljivanje ili za izveštaje o otklanjanju grešaka (engl. *debugging*); na primer:

```
#include <stdio.h>
int test_func( char *s )
{
  if( s == NULL) {
    fprintf( stderr,
           "%s: primljen pokazivac na vrednost NULL kao argument\n",
           __func__ );
    return -1;
  }
  /* ... */
}
```

U ovom primeru, prosleđivanje pokazivača na vrednost NULL (engl. *null pointer*) funkciji `test_func( )`, dovodi do ispisivanja naredne poruke o grešci:

```
test_func: primljen pokazivac na vrednost NULL kao argument
```

Dužina identifikatora nije ograničena. Većina prevodilaca bitnim smatra samo određen broj znakova u identifikatorima. Drugim rečima, prevodilac možda neće uspeti da razlikuje dva identifikatora koji počinju istom dugom sekvencom znakova. U skladu sa standardom jezika C, prevodilac mora uzeti u obzir najmanje 31 početan znak u imenima funkcija i globalnih promenljivih (tj. identifikatora sa spoljašnjim povezivanjem), odnosno najmanje 63 prva znaka u svim ostalim identifikatorima.

Prostori imena identifikatora

Svi identifikatori pripadaju jednoj od četiri naredne kategorije koje predstavljaju zasebne *prostore imena* (engl. *name spaces*):

- imena labela (engl. *labels*)
- identifikacione oznake (engl. *tags*) za strukture, unije i nabrojive tipove
- imena članova strukture ili unije; svaka struktura ili unija obrazuje poseban prostor imena za svoje članove
- svi ostali identifikatori, koji se zovu *obični identifikatori* (engl. *ordinary identifiers*).

Dva identifikatora koja pripadaju različitim prostorima imena mogu biti identična, a da ne izazovu sukob imena. Drugim rečima, ako različiti objekti nisu iste vrste, možete za njih koristiti isto ime. Na primer, prevodilac može da razlikuje promenljivu od istoimene labele. Slično tome, možete isto ime dati tipu strukture, elementu strukture i promenljivoj, što pokazuje naredni primer:

```
struct pin { char pin[16];  /* ... */ };
_Bool check_pin( struct pin *pin )
{
  int len = strlen( pin->pin );
  /* ... */
}
```

Prvi red primera definiše tip strukture sa labelom `pin` čiji je jedan od članova niz znakova `pin`. U drugom redu, definiše se parametar funkcije `pin` kao pokazivač na strukturu upravo definisanog tipa. Izraz `pin->pin` u četvrtom redu upućuje na člana strukture na koji parametar funkcije pokazuje. Kontekst u kome se identifikator pojavljuje uvek nedvosmisleno određuje njegov prostor imena. Ipak, program će biti čitljiviji ako ne ponavljate identifikatore.

Oblast važenja identifikatora

Oblast važenja (engl. *scope*) identifikatora odnosi se na deo jedinice za prevođenje u kome identifikator ima značenje. Drugim rečima, oblast važenja identifikatora deo je programa u kome ga program može „videti". Tip oblasti važenja uvek određuje mesto na kome se identifikator deklariše (izuzetak su labele čija je oblast važenja uvek funkcija). Postoje četiri vrste oblasti važenja:

Datoteka (engl. *file scope*)
: Ako identifkator definišete van svih blokova i lista parametara, njegova oblast važenja je datoteka. Tada ga možete koristiti svuda, počev od mesta deklaracije do kraja jedinice za prevođenje.

Blok (engl. *block scope*)
: Svi identifikatori deklarisani u bloku, izuzev labela, važeći su u tom bloku. Možete ih koristiti samo od deklaracije do kraja najmanjeg bloka koji sadrži tu deklaraciju. Najmanji sadržani blok je često (ne uvek) telo definicije funkcije. Prema standardu C99, deklaracije se ne moraju nalaziti pre svih naredaba u bloku funkcije. I imena parametara u zaglavlju definicije funkcije takođe su važeća samo u bloku date funkcije.

Prototip funkcije (engl. *function prototype scope*)
: Imena parametara u prototipu funkcije pripadaju oblasti važenja tog prototipa. Pošto imena parametara nemaju značenje van prototipa, korisna su samo kao komentari, a mogu se i izostaviti. Više detalja pročitajte u poglavlju 7.

Funkcija (engl. *function scope*)
: Oblast važenja labele uvek je blok funkcije u kojoj se pojavljuje, čak i kad je blok ugnežđen. Drugim rečima, naredbu `goto` možete upotrebiti da s bilo kog mesta u funkciji pređete do labele koja se u njoj nalazi. (Skakanje u ugnežđene blokove nije preporučljivo, a više o tome pročitajte u poglavlju 6.)

Oblast važenja identifikatora u opštem slučaju počinje *posle* njegove deklaracije. Izuzeci su imena tipova tj. identifikacione oznake struktura, unija i nabrojivih tipova, kao i imena nabrojivih konstanti: njihove oblasti važenja počinju neposredno posle njihovog pojavljivanja *u* deklaraciji, da bi se ponovo mogli pozivati u njoj. (Strukture i unije detaljno opisujemo u poglavlju 10, a nabrojive tipove u poglavlju 2.) Na primer, u narednoj deklaraciji tipa strukture, poslednji član strukture, `next`, predstavlja pokazivač upravo na tip strukture koji se deklariše:

```
struct Node { /* ... */
              struct Node *next; };          // Definiše tip strukture
void printNode( const struct Node *ptrNode); // Deklariše funkciju

int printList( const struct Node *first )    // Početak definicije funkcije
{
  struct Node *ptr = first;

  while( ptr != NULL ) {
    printNode( ptr );
    ptr = ptr->next;
  }
}
```

U ovom delu programa, oblast važenja identifikatora `Node`, `next`, `printNode` i `printList` jeste datoteka. Parametar `ptrNode` ima za oblast važenja prototip funkcije, dok su promenljive `first` i `ptr` važeće u bloku.

Identifikator se može ponovo upotrebiti u novoj deklaraciji ugnežđenoj u njegovu postojeću oblast važenja, čak i ako novi identifikator ima isti prostor imena. U tom slučaju, oblast važenja nove deklaracije mora biti blok ili prototip funkcije, a oni moraju biti pravi podskup spoljašnje oblasti važenja. Tada nova deklaracija istog identifikatora *skriva* spoljašnju deklaraciju, pa su promenljiva ili funkcija deklarisane u spoljašnjem bloku *nevidljivi* u unutrašnjoj oblasti važenja. Na primer, naredne deklaracije su prihvatljive:

```
double x;               // Deklariše promenljivu x čija je oblast važenja
                        // datoteka
long calc( double x );  // Deklariše novu promenljivu x čija je oblast
                        // važenja prototip funkcije

int main( )
{
  long x = calc( 2.5 ); // Deklariše promenljivu x tipa long koja važi
                        // u bloku

  if( x < 0 )           // Ovde x predstavlja promenljivu tipa long
  { float x = 0.0F;     // Deklariše novu promenljivu x tipa float važeću
                        // u bloku
    /*...*/
  }
  x *= 2;               // Ovde x ponovo predstavlja promenljivu tipa long
  /*...*/
}
```

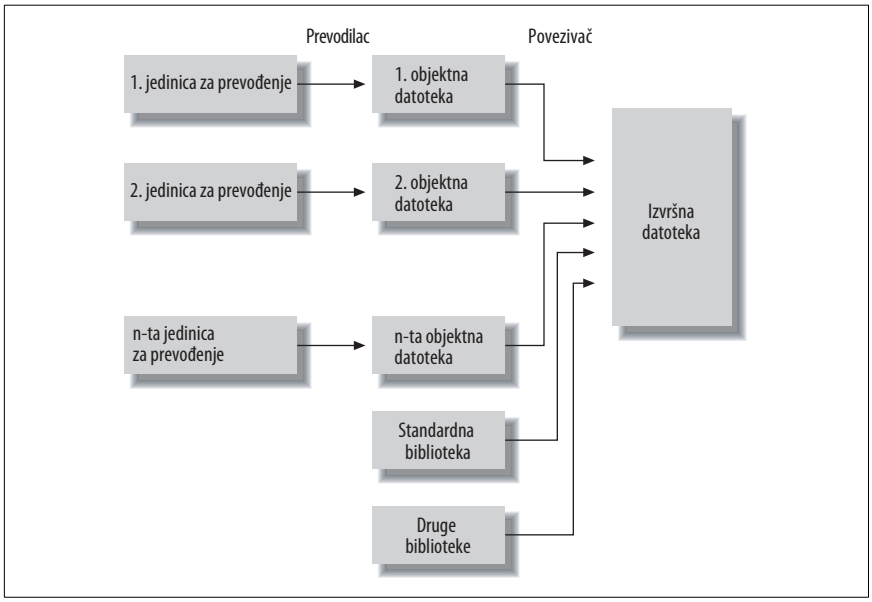
U ovom primeru, promenljiva `x` tipa `long` deklarisana u funkciji `main( )` skriva globalnu promenljivu `x` tipa `double`. Zato ne postoji direktan način pristupa promenljivoj `x` tipa `double` iz funkcije `main()`. Povrh toga, u uslovnom bloku koji zavisi od naredbe `if`, identifikator `x` odnosi se na novodeklarisanu promenljivu tipa `float`, koja skriva istoimenu promenljivu tipa `long`.

Kako radi prevodilac jezika C

Kada u programu za obradu teksta napišete izvornu datoteku, možete pozvati prevodilac jezika C da je prevede na mašinski kôd. Prevodilac obrađuje *jedinicu za prevođenje* (engl. *translation unit*) koja se sastoji od izvorne datoteke i datoteka zaglavlja navedenih pomoću direktiva `#include`. Ako prevodilac ne nađe greške u jedinici za prevođenje, pravi *objektnu datoteku* (engl. *object file*) koja sadrži odgovarajući mašinski kôd. Objektne datoteke obično se označavaju imenom sa sufiksom *.o* ili *.obj*. Pored toga, prevodilac može da napravi i listing na asembleru (o čemu će biti reči u trećem delu knjige).

Objektne datoteke zovu se i *moduli*. Biblioteke, poput standardne biblioteke jezika C, sadrže prevedene module standardnih funkcija kojima se brzo pristupa.

Prevodilac prevodi svaku jedinicu za prevođenje programa – to jest, svaku izvornu datoteku sa svim datotekama zaglavlja koje sadrži – u razdvojene objektne datoteke. Prevodilac onda poziva *povezivač* (engl. *linker*) koji kombinuje objektne datoteke i sve korišćene funkcije iz biblioteke, u *izvršnu datoteku* (engl. *executable file*). Na slici 1-1 predstavljen je proces prevođenja i povezivanja programa koji ima nekoliko izvornih datoteka i koristi nekoliko biblioteka. Izvršna datoteka sadrži i sve informacije potrebne datom operativnom sistemu da je učita i pokrene.



Slika 1-1. Od izvornog koda do izvršne datoteke

Faze prevođenja jezika C

Proces prevođenja izvodi se u osam logičkih koraka. Određeni prevodilac može da kombinuje nekoliko tih faza dok god to ne utiče na rezultat. Prevođenje se odvija u sledećim koracima:

1. Iz izvorne datoteke čitaju se znakovi i po potrebi prevode u znakove iz skupa izvornih znakova. Indikatori kraja reda biće zamenjeni, izuzev znaka za novi red. Takođe, svi trigrafi se pretvaraju u znak koji predstavljaju. (Međutim, digrafi se ne zamenjuju odgovarajućim znakom.)
2. Ako se neposredno iza znaka za novi red nalazi obrnuta kosa crta, pretprocesor briše oba znaka. Pošto se pretprocesorska direktiva završava znakom za kraj reda, ovo brisanje omogućava da stavite obrnutu kosu crtu na kraj reda kako biste naredbu – na primer, definiciju makroa – nastavili u sledećem redu.

Svaka izvorna datoteka, ako nije sasvim prazna, mora da se završi znakom za novi red.

3. Izvorna datoteka se razlaže na pretprocesorske tokene (pogledajte naredni odeljak, „Tokeni“) i na sekvence razmaka. Svaki komentar tumači se kao jedan razmak.
4. Izvršavaju se pretprocesorske direktive i proširuju pozivi makroa.

Koraci 1–4 primenjuju se i za sve datoteke umetnute direktivom `#include`. Kada prevodilac izvrši pretprocesorske direktive, uklanja ih iz svoje radne kopije izvornog koda.

5. Znakovi i izlazne sekvence u znakovnim konstantama i literalima znakovnih nizova, pretvaraju se u odgovarajuće znakove iz izvršnog skupa znakova.
6. Susedni literali se međusobno nadovezuju i spajaju u jedan znakovni niz.
7. Odvija se prevođenje: prevodilac analizira sekvencu tokena i generiše odgovarajući mašinski kôd.
8. Povezivač izdvaja reference na spoljne objekte i funkcije i generiše izvornu datoteku. Ako se u modulu poziva spoljni objekat ili funkcija koji nisu definisani ni u jednoj jedinici za prevođenje, povezivač ih uzima iz standardne biblioteke ili druge navedene biblioteke. Spoljni objekti i funkcije smeju se definisati samo jednom u programu.

Za većinu prevodilaca važi jedno od ova dva pravila: pretprocesor je zaseban program, ili prevodilac može da obavlja samo pretprocesiranje (korake od 1 do 4). To omogućava da proverite jesu li vaše pretprocesorske direktive postigle odgovarajući efekat. Praktičan aspekt procesa prevođenja predstavili smo u poglavlju 18.

Tokeni

Leksička jedinica (leksema) ili token (engl. *token*) može da bude rezervisana reč, identifikator, konstanta, literal znakovnog niza ili simbol. U jeziku C, simboli se sastoje od jednog ili više interpunkcijskih znakova i ponašaju se kao operatori ili digrafi, ili imaju sintaksički značaj, poput tačke i zareza na kraju svake jednostavne naredbe ili vitičastih zagrada, { }, koje ograničavaju blok naredaba. Na primer, sledeća naredba na jeziku C sastoji se od pet tokena:

```
printf("Hello, world.\n");
```

Pojedinačni tokeni su:

```
printf
(
"Hello, world.\n"
)
;
```

Tokeni koje tumači pretprocesor analiziraju se u trećoj fazi prevođenja. Vrlo malo se razlikuju od tokena koje prevodilac analizira u sedmom koraku:

- U direktivi `#include`, pretprocesor prepoznaje dodatne tokene `<`*imedatoteke*`>` i "*imedatoteke*".
- Tokom pretprocesiranja, znakovne konstante i literali znakovnih nizova iz izvornog skupa znakova još uvek nisu konvertovani u izvršni skup znakova.
- Za razliku od prevodioca, pretprocesor ne razlikuje celobrojne konstante od konstanti u formatu s pokretnim zarezom.

Prilikom rastavljanja izvorne datoteke na tokene, prevodilac (ili pretprocesor) uvek prati sledeći princip: svaki naredni znak koji nije razmak mora se dodati tekućem tokenu, sem ako bi takvim dodavanjem valjani token postao neprihvatljiv. Navedeno pravilo otklanja sve nejasnoće u vezi sa sledećim izrazom:

```
a+++b
```

Pošto prvi znak `+` ne može biti deo identifikatora ili rezervisane reči koja počinje slovom `a`, njime počinje nov token. Drugi znak `+` predstavlja prihvatljiv token – operator uvećanja – što ne može da važi i za treći `+`. Zato se izraz razlaže na sledeći način:

```
a ++ + b
```

Više informacija o prevođenju programa na jeziku C naći ćete u poglavlju 18.