Програмирање
Материјал - 1. недеља

Београд, 2016.

Садржај

1

Увод

1.1 Писање једноставног програма

Уобичајен први пример је: написати програм који на екрану штампа једноставну поруку "Здраво свима!".

```
#include <stdio.h>

int main()
{
   printf("Zdravo svima!\n");
   return 0;
}
```

Прво је неопходно да направимо датотеку са одговарајућим именом (једна реч) и екстензијом *c*. Екстензија говори компајлеру да је у питању C програм, име није битно (образложење на вежбама). Затим је неопходно да се програм преведе у облик који је машини препознатљив. Ова процедура обично укључује 3 корака:

- Препроцесирање. Програм прво иде препроцесору, који ослушкује препроцерске директиве (почињу тарабом). Препроцесор је сличан едитору, он може додати и мењати програм.
- Компајлирање. Модификовани програм сада иде у компајлер, који га трансформише у машинске инструкције (објектни код). Програм још није спреман за извршавање.
- Повезивање. У последњем кораку линкер комбинује објектни програм са додатним кодом неопходним да се програм изврши. Додатни код у овом

примеру укључује функцију из библиотеке printf.

Овај процес је обично аутоматизован, препроцесор је обично интегрисан у компајлер па се ови кораци изводе као један. Команде за компајлирање и линковање зависе од компајлера и оперативног система. После ових корака је направљен извршни програм са екстензијом *.out* на UNIX оперативним системима, односно *.exe* на Windows оперативним системима.

1.1.1 Интегрисано развојно окружење

На вежбама користимо интегрисано развојно окружење $Dev-C++$ доступно на локацији http://www.bloodshed.net/dev/. $Dev-C++$ је софтверски пакет који омогућава мењање, компајлирање, повезивање, извршавање и дебаговање програма у самом окружењу. Компоненте овог окружења су дизајниране тако да раде заједно. На пример, ако компајлер детектује грешку у програму у едитору се маркира одговарајућа линија која садржи грешку. Дозвољено је коришћење других развојиних окружења као и оперативних система. У лабораторијским учионицама је студентима на располагању ОС Windows.

1.1.2 Генерална форма једноставног програма

Једноставан програм има форму
direktive

```
int main(void)
{
```

naredbe

```
}
```

У овом шаблону, као и у осталим примерима италик словима је наглашен текст који треба да се допуни у шаблону. Може се приметити да заграде у примеру означавају почетак и крај главне функције, што је врло слично кључним речима begin и end у другим програмским језицима. Пример илуструје и главне особине овог програмског језика, који се умногоме ослоња на скраћенице и специјалне симболе. Овај једноставан програм се ослања на 3 особине:

- директиве (команде које мењају програм пре компајлирања),
- фунцкије (именовани блокови извршног кода, `main` је само један пример),
- и наредби (команде које се извршавају у тренутку извршавања програма).

1.1.3 Директиве

Пре него што се програм компајлира, прво се мења у фази предпроцесирања. Команде упућене препроцесору се називају директиве. Биће детаљно објашњене касније, за сада треба објасници `#include` директиву.

```
#include <stdio.h>
```

Ова директива наглашава да библиотека `<stdio.h>` треба да буде укључена пре компајлирања. `<stdio.h>` садржи информације о стандардној C I/O библиотеци. C има велики број заглавља као што је `<stdio.h>`, сваки садржи информације о неком делу стандардне библиотеке. Разлог укључивања заглавља `<stdio.h>` је што C, за разлику од других програмских језика, нема уграђене команде за читање и писање. Уместо уграђених команди, читање и писање је омогућено коришћењем функција из стандардне библиотеке. Директиве увек почињу симболом #, што их разликује од других команди програмског језика. Уобичајено је да се директиве налазе у једном реду, не ставља се ; или било који други симбол на крају директиве.

1.1.4 Функције

Функције су као процедуре или подпрограми у другим програмским језицима, оне су извршни блокови од којих је програм састављен. Уствари у C-у су програми нешто више од колекције функција. Функције спадају у две категорије: оне које сами корисници пишу и оне које су обезбеђене као део C-имплементације (обезбеђене као библиотеке функција и долазе са компајлером). Термин функција долази из математике, где је функција правило за израчунавање вредности када је дат један или више аргумената:

$$f(x) = x + 1g(y, z) = y^2 - z^2$$

C користи термин функција мало ослабљеније. У C-у се под функцијом подразумева серија наредби које се групишу заједно и именују. Неке функције израчунавају вредност али неке не. Функције које израчунавају (враћају вредност) користе `return` наредбу да би вратиле одговарајућу вредност. На пример, функција која увећава свој аргумент за 1 може имати следећу наредбу:

```
return  x + 1;
```

Иако се програм може састојати од много функција само једна је обавезна за сваки C програм - `main`. Ова функција је специјална, она се позива аутоматски када се извршава програм. Кључан је назив `main` који не сме да се мења. Такође

функција `main` има повратну вредност, враћа статус код оперативном систему када програм заврши са радом. Реч `int` пре речи `main` говори да гункција враћа целобројну вредност. Реч `void` унутар заграда говори да `main` функција нема аргумената. Наредба

```
return 0;
```

има два ефекта: она говори `main` функцији да заврши са радом (а тиме и програм завршава са радом), и наглашава да `main` функција враћа 0 по завршетку рада, што се подразумева као исправно извршен програм. Чак иако се изостави ова нараедба, програм ће исправно завршити са радом али ће већина компајлера пријавити упозорење.

1.1.5 Наредбе

Наредбе су команде које се извршавају када се покрене програм на извршавање. Приказани пример користи само две наредбе једна је `return` а друга је позив функције. Позивање функције да одради посао који јој је придружен. Програми писани у C-у захтевају да се свака команда завршава са симболом ; изузетак су здружене наредбе. Симбол ; говори компајлеру где се наредба завршава јер наредба може да се простире у више редова. Директиве су по правилу у једном реду и зато не захтевају симбол за крај наредбе.

1.1.6 Штампање стрингова

`printf` је веома моћна функција. За сада је коришћена само да прикаже стринг (ниску карактера), који се налазе између " ". У приказу стринга се не наводе наводници. Ова функција не прелази аутоматски у следећи ред по испису из тог разлога за наглашавање када треба да се настави у новом реду се користи симбол $\backslash n$.

1.1.7 Коментари

Сваки написани програми би требало документовати нрп. да садржи основне информације шта програм ради, ко је аутор програма, када је програм написан и сл. Ове информације се могу написати у виду коментара.

```
/*************************************************************
* Naziv : primer1.c                                          *
* Svrha: demonstrira stampanje stringa                       *
```

```
* Autor: D. Jandrlic                                          *
*************************************************************/

/* Neophodna informacija o standardnoj C I/O biblioteci */
#include <stdio.h>


/* Izvrsni kod C-a ide u okviru main funkcije, koja predstavlja glavni
    program */
int main(void)
{
   /* Jedina linija koda koristi printf funkciju (komandu) za stampanje,
      ciji je rezultat izvrsavanja formatirana poruka */
   /* '\n' kod naynacava printf komandi da predje u novi red posle
       ispisivanja poruke */
   // Jednoredni komentari

   printf("Zdravo ");
   printf("svima!\n");

   return 0; // Ova komanda inicira da program vrati 0 operativnom sistemu
       kada zavrsi sa radom
}
```

Коментари могу да се налазе било где у програму, било у новој линији или истој у којој се налази команда или да се простиру у више линија. Треба бити пажљив са вишередним коментарима, ако се изостави симбол за крај коментара може се десити да компајлер третира део наредби као коментар и не изврши их.

1.1.8 Промењиве и додељивање вредности

За разлику од приказаног једноставног програма, већина програма изводи серију рачунања пре него што заврши са радом, и зато је неопходно да се подаци привремено чувају током извршавања програма. Ове меморијске локације се називају промењиве.

1.1.9 Типови података

Свака променљива мора да има тип (eng. *type*), који специфира коју врсту података ће променљива да чува. C програмски језик подржава велику колекцију типова података, за сада ћемо се ограничити само на два типа `int` и `float`. Избор типа података је веома важан, јер он одређује како ће се променљива чувати и које операције могу да се спроводе над променљивом. Нумерички типови података условљавају највећу и најмању вредност коју променљива може да прихвати, као и да ли су или не цифре после децималне тачке дозвољене. Променљиве типа `int` могу да чувају целе бројеве: 0, 1 495, -2553. Ипак распон вредности које променљива може да прихвати је ограничен. Променљиве типа `float` могу да чувају много већи распон вредности. Још важније, могу да чувају вредности из скупа реалних бројева. Ипак постоји и друго ограничење, операције над овим типом података могу да буду спорије. И оно што је још значајније вредност `float` променљиве је најчешће апроксимација вредности која је првобитно сачувана у променљивој (грешка заокруживања).

1.1.10 Декларација променљивих

Променљиве морају бити декларисане (описане компајлеру) пре него што се користе. Да би се декларисала променљива прво се наводи тип податка а затим назив променљиве (биће објашњено у секцији Именовање променљивих).

```
int main(void)
{
   int visina;        // Deklaracija celobrojne promenljive
   float prosek;    // Deklaracija realne promenljive

   /* Ako vise promenljivih ima isti tip. */
   int visina, tezina, sirina, zapremina;

   /* Obratiti paznju samo na kraju se stavlja ; dok se promenljive
      razdvajaju zarezom */
   float prosek, profit;
   . . .
}
```

Напомена: добра пракса је оставити један слободан ред између декларације променљивих и наредби. У C99 стандарду није неопходно декларисати све про-

менљиве на почетку програма (али наравно обавезно пре употребе), тако да је могуће да у програму постоји одељак за ддекларацију па наредбе па декларација... Иначе пракса у објектно оријентисаним језицима је да се променљива не декларише пре него што је потребно да се први пут користи.

1.1.11 Додељивање вредности променљивим

```
   int visina;        // Deklaracija celobrojne promenljive
   visina = 172;    // Ddodeljivanje vrednosti

   float profit;
   profit = 2150.48;  // ili bolje
   profit = 2150.48f;
/* izostavljanje f oznake za tip moze izazvati upozorenje od kompajlera */
```

Могуће је додељивање целобројних вредности `float` типу података, и обрнуто али није увек сигурно!

```
int i, j = 5;
float x = 3.82;
i = x;
x = j;
```

Резултат извршења наведене секвенце наредби је да променљива i добије вредност 3, а не 3.82, док променљива x има вредност 5.0. Једном када је променљивој додељена вредност она се може користити у изразима.

1.1.12 Штампање вредности променљивих

`printf` функција се може користити и за штампање вредности променљивих.

```
   printf("Visina: %d\n", visina); /* %d cuva mesto za vrednost
       promenljive "visina" i koristi se za celobrojne tipove */
   printf("Profit: %f\n", profit); /* za stampanje float vrednosti koristi
       se %f, koja podrazuvena prikazuje 6 decimala posle decimalne tacke */

   /* Da bi sami odredili koliko cifara treba da bude prikayano npr. 2
       %.2f */
   printf("Profit: %.2f\n", profit);
```

Све промењљиве могу бити приказане и само једним позивом `printf` функције. Не постоји ограничење у броју промењљивих које могу да се наводе у овој функцији. (Дељење и целобројно дељење).

1.1.13 Иницијализација

Неке промењљиве подразумевано добијају вредност 0 када програм започне извршавање, али већина не. Промењљиве које не добијају подрзумевану вредност и није им додељена вредност, се зову неиницијализоване промењљиве. Покушај приступа и употребе неиницијализоване промењљиве може довести до неочекиваног резултата или програм престаје са радом. Из тог разлога је неопходно или доделити вредност промењљивој пре употребе или је иницијализовати (доделити јој неку почетну вредност при декларацији).

```
int visina = 150;           //Deklaracija sa inicijaliyacijom
int visina = 150, tezina = 50, sirina = 48; //inicijalizacija vise
    promenljivih
```

1.1.14 Штампање вредности израза

Функција за штампање није ограничена само на штампање вредности промењљивих, може приказати вреднот било ког нумеричког израза. Ова могућност се може искористити да се поједностави код, смањи број линија кода и увођење нових промењљивих.

```
int visina = 172, duzina = 100, sirina = 50, zapremina; // zapremina je
    neinicijalizovana promenljiva

   zapremina = visina*duzina*sirina;
   printf("Zapremina: %d\n", zapremina);

   // ili bez uvodjenja promenljive zapremina
   printf("Zapremina: %d\n", visina*duzina*sirina);
```

1.1.15 Читање са улаза

За прихватање улазних информација користимо `scanf` функцију. Суфикс f у обе функције (за читање и писање) означава да је у питању форматирано читање/писање. То значи да обе функције захтевају формат у коме се очекује улаз

односно треба приказати излаз. Да би се учитао само један цео број довољно је написати следећу наредбу:

```
scanf("%d", &i);
```

Овај стринг $\%d$ говори функцији за читање да чита са улаза целобројну вредност. i је int променљива у коју желимо да функција смести улаз. Симбол & у овом моменту не објашњавамо (биће објашњен касније) за сада треба прихватити да је тај симбол неопходан за учитавање.

1.1.16 Дефинисање константи

Када је потребно у програму користити константе уобичајено је да им се задају имена. Константе се могу дефинисати коришћењем макро дефиниција

```
#define MAX 165

/* #define je preprocesorska direktiva, kao i #include na kraju se ne
    stavlja ;
 * Kada se program kompajlira preprocesor zamenjuje svaki definisani makro
    sa predefinisanom vrednoscu
 * Vrednosti makroa mogu biti dodeljenje i iyrazima */

#define RECIPROC_OF_PI (1.0f / 3.14159f)
/* U tom slucaju izraz mora biti u zagradama. Konvencija je pisati imena
    makroa velikim slovima */
```

Свака директива препроцесору захтева засебан ред!

1.1.17 Идентификатори

Писање програма захтева именовање променљивих, функција, макроа... Ова имена се зову идентификатори. У C-у идентификатори могу да се састоје од слова, цифара, подвлаке, АЛИ МОРАЈУ ДА ПОЧНУ СЛОВОМ ИЛИ ПОДВЛАКОМ. Примери неких валидних идентификатора:

```
skola, x, y, moja_promenljiva, ____
```

Примери невалидних идентификатора:

```
int //to je ključna reč
5nedelja //počinje cifrom
```

```
x-y //sadrži znak -
for //ključna reč
```

C је осетљив на мала и велика слова, разликује мала и велика слова стога промењиве: ана, Ана, аНа, АНа ... нису једнаке и могу да се користе симултано у програму за различите сврхе. Уобичајена конвенција у C-у је да се иденфикатори именују малим словима где се подвлака користи за спајање речи, или се спајају речи а свака нова реч у имену идентификатора почиње великим словом. Не постоји ограничење у броју слова у имену (напомена: 31 према старом стандарду и 63 према новом, објашњење на вежбама).

1.1.18 Кључне речи

Кључне речи имају специјално значење за компајлер и не смеју се користити за именовање идентификатора. Кључне речи се пишу малим словима.

1.1.19 Изглед програма

Свака команда може да се наставља у новом реду. Број бланкова, табова - празнина није битан. Празни редови и увлачење је дозвољено и препоручује се да програм буде што читљивији.

1.2 Форматирани улаз и излаз

`scanf` и `printf` функције, које подржавају форматирано читање и писање су две најчешће коришћене функције у C-у. Обе функције су веома моћне али морају се користити пажљиво.

1.2.1 `printf` функција

Ова функција је дизајнирана да прикаже садржај стринга, са вредностима уметнутим на посебним местима у стрингу. У тренутку позива ове функције мора бити обезбеђен стринг за форматирање, који прате одговарајуће вредности које треба уметнути у стринг приликом штампања.

```
printf(string, expr1, expr2,...);
```

Вредности које се приказују могу бити константе, промењиве, или сложенији изрази. Не постоји ограничење у броју ових вредности које могу бити приказане позивом једне `printf` функције. Стринг за форматирање може садржати

уобичајене карактере и спецификацију за конверзију која мора почињати са % карактером. Спецификација за конверзију је место које се резервише за смештање вредности у току штампања. Информација која следи карактер % специфицира како се вредност конвертује из своје примарне (бинарне) форме у коначном запису. Уобичајени карактери се штампају дословно онако како су написани, конверзионе спецификације се мењају одговарајућим вредностима. Напомена: C компајлер не проверава исправност конверзионе ниске. Неисправно уношење већег броја вредности него што је наведено у ниски ће резултовати неприказивањем свих вредности, навођење мањег броја вредности ће резултовати несмисленим приказом, исто се дешава ако конверзиона ниска није исправно наведена за одговарајуће типове података.

1.2.2 Конверзиона ниска

Конверзиона ниска има главну улогу у контроли приказивања резултата, али може бити компликована и тешка за читање. Конверзиона ниска може да буде облика: $\%m.pX$ или $\%\text{-}m.pX$ где су m и p целобројне константе а X је слово. Обе целобројне константе су опционе, ако је p специфицирано тачка која раздваја p и m се изоставља. Нпр. $\%10.2f$, $m = 10$ $p = 2$ $X = f$. m специфицира минимални број карактера за приказивање. Ако је вредност која се приказује мање ширине него што је прецизирано вредности су десно поравнате (празна поља се штампају са леве стране). Ако је вредност која се штампа шира од специфицираног броја карактера, ширина поља за приказ се аутоматски проширује до неопходне величине. Уметање знака - означава лево поравнање (празнине ако их има се налазе после записа - десно). Значење прецизности p није тако једноставно за објаснити јер зависи од избора X спецификатора конверзије. X говори која конверзија треба да буде примењена на вредност пре штампања.

- d - се користи за приказ децималних (у основи 10) бројева. p говори који је најмањи број цифара за приказ (додатне нуле се исписују на почетку ако је неопходно) ако се изостави подразумена вредност је 1.

- e - приказује бројеве у покретном зарезу у експоненацијалном запису. p говори колико се цифара приказује после децималне тачке (подразумевано је 6). Ако је p 0 децимална тачка се не приказује.

- f - приказује броје у покретном зарезу са фиксним децималама без експонента. (p исто значање као и пре).

- g - приказује реалне бројеве или експоненцијалном запису или са фиксним децималним записом, у зависности од величине броја. *p* говори које је ограничење за значајне цифре (не цифре иза децималне тачке). За разлику од претходне конверзије ова конверзија неће приказати водеће нуле.

Последња спецификација је посебно значајна када је потребно приказати број за који се унапред не може предвидети величина и може узимати значајно другачије вредности.

```
int main(void)
{
   int i;
   float x;

   i = 40;
   x = 839.21f;

   // Konverziona niska za celobrojne promenljive
   printf(" |%d|%5d|%-5d|%5.3d|\n", i, i, i, i);

   // Konverziona niska za realne promenljive
   printf("|%10.3f|%10.3e|%-10g|\n sa minusom\n |%9.3f|%8.3e|%-10g|\n", x,
       x, x, -x, -x, -x);

   return 0;
}
```

Објашњење:

- %5.3d приказује целобројну вредност користећи најмање 5 карактера и најмање 3 цифре, како цео број има само две цифре додаје се водећа нула на почетку и 2 празна места испред.
- %10.3f Приказује реална број у фиксном децималном запису користећи 10 карактера укупно и 3 цифре после децималне тачке. Како x захтева само 7 карактера (3 пре децималне тачке, 3 после децималне тачке и децималну тачку) 3 празна места се умећу на почетку.
- %10.3e приказује x у експоненцијалном запису користећи свеукопно 10 карактера, са три цифре иза децималне тачке. x захтева девет места (ту је укључен и симбол за експонент) значи једно место се умеће лево.

- %-10g -приказује x или y фиксном запису или експоненту користећи свеукупно 10 карактера (лево поравнање).
-

Специјални знаци

Овде се подразумевају знаци који имају специјално значење за компајлер:

- Alert (bell) \a
- Backspace \b
- New line \n
- Horizontal tab \t
- \\\" ...

Scanf функција

Баш као што `printf` функција штампа према задатом формату тако и `scanf` функција чита према одређеном формату. И ова функција као аргумент има конверзиону ниску која у суштини има исто значење као до сада, може садржати регуларне карактере и конверзионе спецификације. Ипак, ова функција најчешће садржи само конверзионе спецификације. Уствари ова функција може много више, она покушава да групише улазне карактере са задатом спецификацијом тако што започиње читање са леве стране покушава да лоцира садржај који јој одговара прескачући празне симболе ако је неопходно, заустављa читање када наиђе на карактер који не може припадати одређеном формату. Ако је садржај алоциран и прочитан функција наставља са читањем преосталог садржаја конверзионе ниске. Ако неки садржај није прочитан према коректно функција завршава са радом без читања остатка конверзионе ниске.

```
1-20.3-4.0e3
scanf("%d%d%f%f", &i, &j, &x, &y)
```

%i се у `printf` функцији не разликују, док у `scanf` функцији %d се користи само за читање децималних бројева (основа 10), %i се може користити за учитавање октални, децималних и хексадецималних бројева, нпр. ако улаз започиње 0 (нпр. 056) са %i се учитава октални запис, са 0x и 0X се учитава хексадецимални број.

2

Изрази

Најједноставнији изрази су промењљиве и константе. Сложенији изрази се добијају комбиновањем оператора и операнада (који су сами по себи изрази). Оператори су основни елементи за формирање израза и C има богату колекцију оператора. Елементарни оператори који су присутни и у другим програмским језицима укључују:

- Аритметичке операторе: + , - , * , \, %
- Релациони оператори за поређење.
- Логички оператори за конструкцију услова.

C подржава јако велики скуп оператора који ће се уводити постепено (због обима) кроз сва поглавља. Овде ће бити описани основни оператори, аритметички, оператори доделем и оператори инкрементирања и декрементирања.

2.1 Аритметички оператори

Унарни и бинарни +, - . Опрез са операторима \и % када је други аргумент 0 могу изазвати недефинисано понашање. Такође када је један од операнада негативан у различитм имплементацијама се добијају различити резултати. Ипак у C99 стандарду заокруживање при дељењу увек иде ка 0, а резултата примене оператора % је знака као први аргумент.

2.1.1 Приоритет оператора и асоцијативност

Приоритет операција увек може да се регулише навођењем заграда. Уколико заграде нису наведене приоритет аритметичких оператора је следећи:

Највећи:	+	-	(унарни)
	*	/	%
Најнижи:	+	-	(бинарни)

Приоритет оператора није довољан као правило када се у истом реду нађу оператори истог приоритета. У таквим случајевима се гледа асоцијативност оператора. Лево асоцијативни оператори се групишу са лева на десно, такви су сви аритметички оператори (*,/,%, + и -). Десно асоцијативни оператори се групишу са десна на лево. Такви су унарни аритметички оператори (+ и -).

2.1.2 Оператори инкрементирања и декрементирања

Две од најчешћи операција над променљивим су увећење променљиве за 1 (инкрементирање) и умањење променљиве за 1 (декрементирање).

```
y++;
--x;
```

Наизгле једноставни оператори могу бити прави изазов за коришћење. Разлог томе је што ови оператори могу бити префиксни у постфиксни. Други разлог је бочни ефекат ових оператора, као и оператори доделе и ови оператори мењају свој операнд. Префиксни оператор подразумева да се операнд ажурира одмах, док постфиксни захтева да се операнд ажурира касније. У случају постфиксног оператора операнд се ажурира пре следеће наредбе. Постификсни оператори инкрементирања и декрементирања имају јачи приоритет у односу на унарне аритметичке операторе и лево су асоцијативни. Префиксни оператори инкрементирања и декрементирања имају исти приоритет као унарни + и - и десно су асоцијативни.

2.2 Оператори доделе

Јендом када је израз израчунат његова вредност треба негде да се смести. Најједноставнији оператор доделе се користи у том случају (=). За ажурирање већ сачуване вредности у променљивој C подржава здружене операторе доделе.

```
i = 5;  /* i dobija vrednost 5 */
j = i;  /* j dobija vrednost 5 */
l = 6 * i - j; /* l dobija vrednost 25 */
...
int i;
```

```
float f;
i = 65.32; /* i dobija vrednost 65 */
f = 345; /* f je sada 345.0 */
...
```

Већина оператора не мења своје операнде, што није случај са оператором доделе где се израчунава вредност и придружује једном операнду. Овај ефекат је познат као бочни ефекат оператора доделе.

```
i = 5;  /* i dobija vrednost 5 */
f = 8.2 /* f dobija vrednost 8.2 */


j = i = f; /* prvo i dobija vrednost 8 yatim i j dobija vrednost 8 */
l = 6 * i - j; /* l dobija vrednost 25 */
...
/* zdruzeni operatori dodele */
i += 2; // isto sto i: i = i + 2;

i += j += k; // isto sto i: i += (j += k);

...
```

Лева страна израза

Већина оператора дозвољава да операнди буду промењљиве, константе, или изрази који и сами садрже операторе. Ипак, оператор доделе захтева да лева страна израза буде објекат у меморији, не константа нити резултат израчунавања. Промењљиве могу да буду лева страна израза.

Здружени оператори доделе

Додељивања у којима се користи стара вредност промењљиве како би се израчунала нова вредност су уобичајена у програмском језику C. Здружени оператори додељивања су +=, -=, *=, /=, %= . Има их укупно 9. Ови оператори су десно асоцијативни.

primer

2.2.1 Оценивање израза

Следећа табела приказује приоритет до сада наведених оператора:

Приоритет	Назиив	Симбол	Асоцијативност
1	инкрементирање (постфиксни)	++	лево
	декрементирање (постфиксни)	–	лево
2	инкрементирање (префиксни)	++	десно
	декрементирање (префиксни)	–	
	унарни +	+	
	унарни -	-	
3	множење/дељење..	*,/,%,	лево
4	адитивни	+, -	лево
5	додељивање	=, *=, /=, %=, +=, -=,	десно

Редослед извршавања подизраза

```
a = 5;
c = (b = a + 2) - (a = 1);
```

Ефекат извршавања друге наредбе у изразу је недефинисан јер није познато који подизраз ће се први извршити $(b = a + 2)$ или $(a = 1)$. Уколико се први израз израчунава прво b добија вредност 7, међутим ако се прво израчуна други израз b добија вредност 3. Неки компајлери чак могу да пријаве упозорење. Да би се овакви проблеми избегли треба избегавати коришћење оператора доделе у подизразима, увек је боље направити серију одвојених додела.

2.3 Примери - Задаци са вежби

Задатак 2.1 *Написати програм који на стандардном излазу исписује поруку: 'Zdravo svete!'.*

```
#include <stdio.h>

int main()
{
    printf("Zdravo svete!\n");
    return 0;
```

```
}
```

Задатак 2.2 *Шта се исписује извршењем следећег програма?*

```
#include <stdio.h>

int main() {
    printf("\"Zdravo, svima\"\n");
    printf("\\n\tprelazak u novi red\n");
    printf("\\t\ttabulator\n");
    printf("\\\\\tkosa crta\n");
    printf("%%%%\tprocenat\n");

    return 0;
}
```

Задатак 2.3 *Написати програм који илуструје декларацију и иницијализацију целобројне променљиве и испис њене вредности.*

```
#include <stdio.h>

int main()
{
    int i = 2;
    printf("Vrednost promenjive i je %d\n", i);

    return 0;
}
```

Задатак 2.4 *Написати програм у којем се са стандардног улаза уноси вредност целобројне променљиве и исписује њена вредност на стандардни излаз.*

```
#include <stdio.h>
```

```c
int main()
{
    int x;
    printf("Unesite ceo broj: ");
    /* Obratiti paznju na znak & pre imena promenljive
       u funkciji scanf. */
    scanf("%d",&x);

    /* U funkciji printf ne treba stavljati &. */
    printf("Uneli ste broj %d\n", x);

    return 0;
}
```

Задатак 2.5 *Написати програм у којем се са стандардног улаза уноси вредност реалне промењиве и исписује њена вредност на стандардни излаз заокружена на две децимале.*

```c
#include <stdio.h>

int main()
{
    float x;
    printf("Unesite realan broj: ");
    scanf("%f", &x);
    printf("Broj zaokruzen na dve decimale je: %.2f\n", x);

    return 0;
}
```

Задатак 2.6 *Написати програм који сабира два цела броја задата са стандардног улаза.*

```c
#include <stdio.h>

int main()
```

```
{
    int a, b, c;
    printf("Unesi prvi broj : ");
    scanf("%d", &a);
    printf("Unesi drugi broj : ");
    scanf("%d", &b);
    c = a + b;
    printf("%d + %d = %d\n", a, b, c);

   return 0;
}
```

Задатак 2.7 *Написати програм којим се размењују вредности два цела броја унета са стандардног улаза.*

```
#include <stdio.h>

int main()
{
    int a, b;
    int pom;
    printf("Unesi prvi broj : ");
    scanf("%d", &a);
    printf("Unesi drugi broj : ");
    scanf("%d", &b);

    printf("Pre razmene: a = %d, b = %d\n", a, b);

    pom = a;
    a = b;
    b = pom;

    printf("Posle razmene: a = %d, b = %d\n", a, b);

    return 0;
}
```

Задатак 2.8 *Програм демонстрира аритметичке операције над два унета цела броја.*

```c
#include <stdio.h>

int main()
{
    int a, b;
    printf("Unesi prvi broj: ");
    scanf("%d",&a);

    printf("Unesi drugi broj: ");
    scanf("%d",&b);

    printf("%d + %d = %d\n", a, b, a + b);
    printf("%d - %d = %d\n", a, b, a - b);
    printf("%d * %d = %d\n", a, b, a * b);
    printf("Celobrojni kolicnik a / b je: %d\n", a/b);
    printf("Pogresan pokusaj racunanja realnog kolicnika a/b \
           je: %f\n", a/b);
    printf("Realni kolicnik a/b je: %f\n", (float)a/(float)b);
    printf("Ostatak pri deljenju a/b je: %d\n", a%b);

    return 0;
}
```

Задатак 2.9 *Napisati program koji izračunava obim i površinu kruga poluprečnika* r*, koji se unosi sa standardnog ulaza.*

```c
#include <stdio.h>

#define PI 3.14159

int main() {
    int r;

    printf("Unesi poluprecnik kruga: ");
```

```
    scanf("%d",&r);

    printf("Obim: %g\n", 2 * r * PI);
    printf("Povrsina: %g\n", r * r * PI);

    system("pause");
    return 0;
}
```

Задатак 2.10 *Šta se ispisuje sledećim programom?*

```
/*
   Operator ++ uvecava vrednost promenjive za 1.
   Operator -- je potpuno analogan operatoru ++ osim sto
               umanjuje vrednost za 1
*/
#include <stdio.h>
int main() {
    int x, y;
    int a = 0, b = 0;

    printf("Na pocetku : \na = %d\nb = %d\n", a, b);

    /* Ukoliko se vrednost izraza ne koristi, prefiksni i
      postfiksni operator se ne razlikuju */
    a++;
    ++b;
    printf("Posle : a++; ++b; \na = %d\nb = %d\n", a, b);

    /* Prefiksni operator uvecava promenjivu, i rezultat
       je uvecana vrednost */
    x = ++a;

    /* Postfiksni operator uvecava promenjivu, i rezultat je
       stara (neuvecana) vrednost */
    y = b++;

    printf("Posle : x = ++a; \na = %d\nx = %d\n", a, x);
```

```
    printf("Posle : y = b++; \nb = %d\ny = %d\n", b, y);

    system("pause");
    return 0;
}
```