Структуре

Структуре у језику *C* су сложени типови података који се састоје од одређеног броја елемената. Елементи структура називају се *поља* структура, могу да буду међусобно различитих типова и обележавају се идентификаторима. Да би се приступило пољу структуре, потребно је навести и идентификатор структуре чијем пољу жели да се приступи. Структуре се у другим програмским језицима називају и *записи* или *слогови.*

Поља структура могу бити произвољног простог или сложеног типа. То значи да поља структура могу бити друге структуре, али и низови. Наравно, могу да се образују и низови чији су елементи структуре.

Треба јасно истаћи разлику између структура и низова. *Елементи* низ*о*ва су међусобно једнаких типова, обележавају се редним бројевима унутар низа (индексима) и приступа им се индексирањем или применом адресне аритметике.

Структуре се обично користе за описивање сложених, апстрактних објеката, на пример:

- тачка у равни се описује координатама *x и y*;
- комплексни број се описује реалним и имагинарним делом;
- аутомобил се описује регистарским бројем, ознаком модела, годином производње, називом произвођача, именом власника,
- књига се описује именом аутора, насловом, називом издавача, годином издавања, бројем страница,...;
- итд., итд....

6.1 Дефинисање структура

Структура се сматра типом података који дефинише програмер. Општи облик дефиниције структуре је:

```
struct име_структуре { низ_декларација } ;
```

име_структуре је идентификатор за идентификацију структуре која се дефинише. Тај идентификатор може касније да се користи у другим декларативним наредбама, или у описима параметара функција за означавање ове структуре. Име структуре нема статус идентификатора типа, па у каснијим позивањима на ову структуру увек мора да се пише **struc**t *име_структуре,* само *име_структуре* није довољно. У неким случајевима *име_структуре* може да се изостави из горњег општег облика.

Низом декларација набрајају се поља структуре. Свака декларација у низу има облик наредбе за дефинисање података:

```
тип  поље,  поље,  …  поље ;
```

Тип представља основни тип свих поља у декларацији. *Поља* су идентификатори поља са евентуалним модификаторима, којима се означава да је неко поље низ или показивач датог основног *типа*.

Тип поља структуре не може да буде структура која се управо дефинише, али може да буде показивач на такву структуру.

Досег идентификатора поља структуре се протеже само на структуру којој припадају та поља. Због тога идентификатори поља не могу да се користе самостално, већ само уз неки податак типа дате структуре (видети одељак 6.2).

Пошто свака структура чини одвојен досег идентификатора, исти идентификатор поља може да се користи у више структура. Та поља у свакој структури могу да буду другачијег типа. Такође, идентификатор поља структуре сме да се поклапа са именом неког од података који се користе у програму.

Подаци структурног типа дефинишу се уобичајеним наредбама за дефинисање података. Као ознаку типа потребно је навести **struct** *име_структуре*. Скреће се пажња да *име_структуре* нема статус идентификатора типа, па се испред ње увек мора написати и службена реч **struct**. Уз свако име податка могу се наводити и уобичајени модификатори [дужина] за дефинисање низа чији су елементи структуре, или * за дефинисање показивача на структуре.

Као и у случају других типова података, и за структуре може да се наведе почетна вредност (иницијализатор) приликом дефинисања. Општи облик наредбе за дефинисање структуре са почетном вредношћу је:

struct *тип име = { вредност , вредност, ... , вредност } ;*

Наведене *вредности* се додељују узастопним пољима структуре. Број и типови *вредности* треба да се слажу са бројем и типовима поља структуре. Навођење више *вредности* од броја поља није дозвољено. Уколико је број *вредности* мањи, почетне вредности преосталих поља постављају се на нуле одговарајућих типова.

У случају да је неко од поља структуре такође структура или низ, одговарајућа *вредност* треба да се састоји од низа вредности унутар заграда {}. Ово је слично задавању почетних вредности вишедимензионалним низовима.

Пример 6.1 -Дефинисање структура

```
struct tacka { int x, y; };
struct tacka a, b, c, p = {35, -12}, q;
struct pravougaonik {
  struct tacka dole_levo, gore_desno;
   };
struct pravougaonik w, x, y = {{1, 1), (3, 3}} ;
struct krug { double r;
              struct tacka centar;
              };
```

Првом наредбом дефинише се тип **struct** tacka за представљање тачака у равни као структура од два целобројна поља x и y која чине координате тачака.

Другом наредбом дефинише се пет променљивих `a, b, c, p и q,` типа **`struct`** `tacka`. Као почетна вредност, променљивој `p` додељују се координате (35,-12).

Трећом наредбом дефинише се тип **`struct`** `pravougaonik` са два поља. Та поља су типа **`struct`** `tacka` које представљају доњи леви и горњи десни угао правоугаоника. Уз претпоставку да су ивице правоугаоника паралелне са координатним осама, те две тачке једнозначно одређују правоугаоник.

Четвртом наредбом дефинишу се три правоугаоника `w, x и y.` Поред тога, за правоугаоник `y,` у облику почетне вредности, речено је да има доњи леви угао са координатама (1,1) и горњи десни угао са координатама (3,3). Скреће се пажња да су имена x и y коришћена и као идентификатори поља структуре `tacka.`

Последњом наредбом дефинише се структура за представљање кругова помоћу полупречника и координата центра. Поља ове структуре су међусобно различитих типова.

Наредба **`typedef`** може да се користи и за додељивање идентификатора структурним типовима:

```
typedef struct ИмеСтруктуре { низ_декларација } ИмеТипа ;
```

Идентификатор `ИмеТипа` може касније да се користи као идентификатор типа без додавања службене речи **`struct.`** *`ИмеСтруктуре`* у наредбама **`typedef`** се најчешће изоставља.

Пример 6.2 - Придруживање имена структурним типовима

```
typedef struct { char licno_ime[16], prezime[16]; } Ime;
typedef struct { char dan, mesec; short int godina; } Datum;
typedef struct {
    char licni_broj [13];
    Ime osoba;
    Datum datum_rodjenja;
    char adresa [30];
    } Gradjanin;

  Gradjanin marko, pero, stevo;
```

Прво се каже да се тип `Ime` састоји од личног имена и презимена од по 16 знакова. Тип `Datum` се, с друге стране, састоји од дана, месеца и године, одговарајућих целобројних типова. После ових помоћних дефиниција, типом `Gradjanin` дефинишу се потребни подаци за представљање података о људима. На крају се дефинишу три променљиве типа `Gradjanin.`

Посебно се скреће пажња на повећану читљивост наредби у овом примеру у односу на наредбе у примеру 6.1. Ово је, сигурно, довољан разлог за интензивно коришћење наредбе **`typedef.`**

Поново се напомиње да употреба великих почетних слова у именима типова који се уводе наредбама **`typedef`** није обавеза коју намеће језик **C**, већ стил који се усталио у пракси.

6.2 Употреба структура

Структуре представљају типове које дефинише програмер. Прецизно је одређено на којим местима могу да се користе структуре и какве операције могу да се примене на променљиве структурних типова.

Подаци структурних типова се сматрају појединачним подацима, без обзира на њихову сложеност. То другим речима значи да се на одређеним местима користе на исти начин као и прости скаларни подаци, што није случај са низовима.

Пре свега, подаци међусобно једнаких структурних типова могу међусобно да се додељују помоћу оператора за доделу вредности =. Тиме се целокупни садржај изворишног податка прекопира у одредишну променљиву, поље по поље. Не смета ако се међу пољима налазе и низови. Наглашава се да „обичне" низове (који нису део структуре) треба копирати елемент по елемент.

Адреса структурне променљиве може се добити оператором &. Такође могу да се дефинишу и подаци који су показивачи на структуре. За приступ показиваним структурама користи се оператор *.

Структуре могу да буду параметри функција. Пошто се структура, без обзира на њену сложеност, сматра једним податком, подразумевани начин преношења структура у функције је помоћу вредности. Из истог разлога, структуре могу да буду и вредности функција.

С обзиром да структуре могу да буду врло велике, препоручује се да се оне преносе у функције посредством показивача. По потреби, као заштита од мењања садржаја податка који се налази у позивајућем програму, параметар треба дефинисати као показивач на непроменљиву (const) структуру.

Чињеница да се структуре могу међусобно додељивати и да могу бити вредности функција, омогућује, поред облика показаног у одељку 6.1, и следећи облик задавања почетне вредности приликом дефинисања структуре:

struct тип име = *израз* ;

где *израз* може да буде адресни израз чији је резултат податак структурног типа, или позив функције чија је вредност функције истог типа као и посматрана структура.

Од структура могу да се образују и низови. Адресна аритметика је дозвољена и за такве низове. Јединична промена показивача представља промену адресе за онолико бајтова колика је величина структурног податка.

Величина структуре у оперативној меморији може да се добије оператором **sizeof.** Величина структуре је приближно једнака збиру величина поља те структуре. Због могућих ограничења ускладиштавања података различитих типова на неким рачунарима, понекад се дешава да је величина структуре нешто већа од збира величина поља. Због тога, величину структуре никад не треба рачунати на основу величина појединачних поља, већ треба користити оператор **sizeof.**

Пољима структуре приступа се помоћу бинарног оператора тачка (.). Први операнд тог оператора је податак структурног типа ***s***, а други операнд је идентификатор поља ***p*** такве структуре:

```
s.p
```

Ако је ***ps*** показивач на структуру, пољу ***p*** те структуре приступа се помоћу:

```
(*ps).p
```

Заграде су неопходне. У одсуству заграда, због вишег приоритета оператора . од оператора *, израз `*s.pp` би се тумачио као `*(s.pp)`, а то је податак на који показује показивачко поље `pp` структурног податка `s.`

Због врло честе употребе показивача на структуре, у језику ***C*** дефинисан је и бинарни оператор **->** за приступ пољу ***p*** структуре на коју показује показивач ***ps***:

```
ps->p
```

Ово је, у ствари, скраћеница за горњи незграпни израз са заградама `((*ps) .p)`.

Оба бинарна оператора `.` и `->` су приоритета 15 и групишу се слева удесно.

Поља структура су подаци (прости или сложени), којима може да се образује адреса изразом `&s.p`. Ако се та адреса додељује неком показивачу `px` (`px=&s.x`), пољу `x` унутар структуре `s` може да се приступа и изразом `*px`.

Пример 6.3 — Употреба структура дефинисаних у примеру 6.1

```
a.x = 13; a.y = - 22;          /* a = {13, -22}; није дозвољено!   */
x.dole_levo.x =4;          x.dole_levo.y =  - 5;
x.gore__desno.x = 7;              x.gore_desno.y = 0;
w.dole_levo = a;           w.gore__desno = p;
```

Променљива ***a*** је типа **struct** *tacka*. Њена поља се обележавају са *a.x* и *a.y*. Додела константних вредности тим пољима могућа је само појединачно. Нотација *a={13, -22}* је дозвољена само за доделу почетне вредности приликом дефинисања променљиве.

Променљива ***x*** је типа **struct** *pravougaonik*, чија поља се обележавају са *x.dole_levo* и *x.gore_desno*. Пошто је *x.dole_levo* типа **struct** *tacka*, оператор . може још једном да се примени и да се тако дође до скаларних поља *x.dole_levo.x* и *x.dole_levo.y*. Скреће се пажња да се у изразу *x.dole_levo.x* идентификатор ***x*** појављује на два места. Први пут је име променљиве, а други пут име поља структуре. Усвојена нотација у језику **C** гарантује да никада нема двоумљења када неки идентификатор означава променљиву, а када поље структуре.

Променљива `w` је типа **struct** `pravougaonik`, па су *w.dole_levo* и *w.gore_desno* типа **struct** `tacka`. Пошто су u променљиве `a` u `p` типа **struct** *tacka*, доделе вредности у последњем реду су исправне.

Пример 6.4 - Показивачи, низови, функције и структуре

```
/* Дефинисање типа Tacka. */
typedef struct { int x, y; } Tacka;

/* Образовање тачке од задатих координата. */
Tacka pravi_tacku (int x, int y) {
Tacka t;
t.x = x; t.y= y;
return t;
}

/* Израчунавање растојања између две тачке. */
#include <math.h>
double rastojanje (Tacka g, Tacka h) {
  return sqrt (pow(g.x-h.x, 2) + pow(g.y-h.y, 2));
}
```

```
/* Налажење најближе тачке координатном почетку у низу тачака. */
Tacka *najbliza (const Tacka a[], int n) {
Tacka *min = a; double r = rastojanje (a[0] ,NULA); int i;
for (i=1; i<n; i++) {
   double s = rastojanje (a[i], NULA);
   if (s < r) {r = s; min = a + i;}
    }
  return min;
}
/* Константа за означавање координатног почетка. */
const Tacka NULA = {0, 0};
```

Пре свега, наредбом **typedef** уведена је скраћеница *Tacka* за tip који представља структуру са два поља за координате тачака у равни.

Функција *pravi_tacku* служи за састављање податка tipa *Tacka* од два скаларна целобројна податка. Резултат образује у локалном податку t, који враћа као вредност функције. Скреће се пажња да имена параметара се поклапају са именима поља структуре *Tacka*.

Функција *rastojanje* израчунава растојање између две тачке. Параметри ове функције су типа *Tacka*. Пошто се структура *Tacka* састоји само од два реална податка, пренос параметара помоћу вредности (када се аргументи при позивању копирају у параметре) није неприхватљиво неефикасно.

Функција *najbliza* проналази у скупу тачака ону која је најближа координатном почетку. Скуп тачака представља се низом *a* чији су елемеити типа *Tacka*. Пошто се садржај тог низа не мења унутар функције, параметар је дефинисан као непроменљив (const) низ. Вредност функције је показивач на елемент низа *a* који садржи координате те најближе тачке.

На крају, глобални непроменљиви податак *NULA* је уведена за симболичко пред- стављање координатног почетка. Модификатор const при дефинисању обезбеђује непроменљивост податка *NULA*. Упркос томе, *NULA* се не сматра константом, већ само непроменљивим податком и не сме да се користи на местима на којима се изричито траже константе.

6.3 Уније

Уније су сложени типови података који омогућавају да се у исти меморијски простор, у различитим тренуцима, смештају подаци различитих типова.

Уније се дефинишу наредбом union, чији је општи облик истоветан наредби s t r u c t . Једино службена реч struct треба да се замени са union. Исто важи и за дефинисање података типа унија.

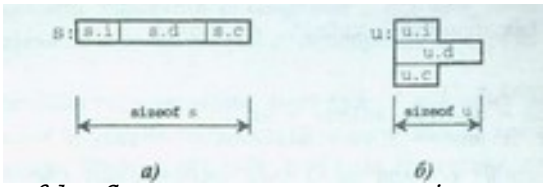
За разлику од структура, код којих сва поља истовремено имају дефинисану вредност, код унија само једно поље, у датом моменту има дефинисану вредност. То је поље којем је последњем додељена вредност. Дужност програмера је да води рачуна о усаглашеном коришћењу поља унија. Последице су непредвидиве ако се додели вредност једном пољу, а потом се користи вредност другог поља.

Почетна вредност за уније може да се наведе на исти начин као и за структуре, али унутар заграда {} сме да буде само једна вредност. Та вредност се увек додељује првом пољу уније.

Ради истицања разлике између структура и унија, нека су структура s и унија u дефинисане истоветним пољима:

```
struct {                    union {
int i;                      int i;
double d;                   double d;
char *c;                    char *c;
} s;                        } u;
```

Начин њиховог смештања у оперативну меморију приказан је на слици 6.1.



Слика 6.1 — Смештање структуре и уније у меморију

У случају структуре (слика 6.1.а), поља се смештају у меморију једно за другим. Сваком од поља може да се додели вредност. Величина структуре је приближно једнака збиру величина поља.

У случају уније (слика 6.1.б), сва поља се смештају почев од исте адресе. Додела вредности једном пољу, очигледно уништава раније додељену вредност неком другом пољу. Величина уније је једнака величини највећег поља.

6.4 Структуре од битова

Структуре од битова су структуре (`struct`) чија су поља дужине неколико битова. Та поља се пакују у машинске речи рачунара. Тачан начин ускладиштавања битова зависи од рачунара и од преводиоца за језик C.

Стандард не прецизира да ли структура од битова сме да буде већа од једне машинске речи, да ли појединачна поља смеју да буду дужа од једне машинске речи и да ли дато поље сме да пређе границу речи (или у целини мора да буде унутар исте машинске речи). Поред тога, од рачунара зависи и да ли се поља унутар машинске речи ређају слева удесно или обрнуто.

Структуре од битова треба да олакшају манипулисање информацијама дужине од неколико битова, које су врло често једнобитни индикатори, тако да програмер не мора да користи операторе померања, маске за издвајање битова итд.

Структуре од битова најчешће се користе за описивање структуре разних хардверских регистара рачунара, као што су регистри придружени улазно-излазним уређајима рачунара.

Због велике зависности од рачунара, програми који користе структуре од битова нису преносиви са једног рачунара на други. Зато се не препоручује употреба структура од битова, а ако баш мора, њихово коришћење треба да се концентрише у што мањи број једноставних функција. На тај начин, у случају преношења са рачунара на рачунар, биће потребно прерадити само незнатан део целокупног програмског система.

Код структура од битова, за све декларације типова и дефинисања променљивих користе се наредбе као и за структурне типове (видети одељак 6.1). Сва поља морају бити неког целобројног типа (најчешће unsigned int) и иза сваког имена поља треба да се дода : d, где d представља дужину поља у битовима, у облику целобројне константе.

Пољима структура од битова приступа се, као и код обичних структура, опера торима . или -> (видети одељак 6.2). Поља структура од битова нису самостални подаци, па не могу да се образују њихове адресе (није дозвољена примена унарног оператора &).

Пример 6.5 – Структура од битова

```
      struct {
    unsigned int crveno:1,  zuto:1, zeleno:1;
} semafor;
semafor.crveno = 1;
semafor.zuto = semafor.zeleno = 0;
if (semafor.crveno == 1 && semafor.zuto ==1) { ... }
```

Семафор се састоји од три бита који представљају светла црвено, жуто и зелено. Вредност један неког поља означава да је то светло укључено.

Паковање битова приликом доделе вредности пољима, односно распакивање приликом узимања вредности поља, обезбеђују се аутоматски. Програмер не треба да се мучи наредбама за манипулисање битовима да би то обезбедио.

6.5 Набрајања

Набројане константе су целобројне симболичке константе којима су вредности додељене експлицитно или имплицитно, набрајањем њихових имена у једном низу.

Скуп одједном набројаних константи чини једно ***набрајање***, које се може сматрати типом података који је дефинисао програмер. Општи облик дефинисања набрајања је:

```
enum име_набрајања {  ИМЕ_КОНСТАНТЕ = вредност ,
                      ИМЕ_КОНСТАНТЕ = вредност , ... } ;
```

Име набрајања је идентификатор који служи за идентификацију датог набрајања. Може да се изостави, али онда касније не могу да се дефинишу подаци типа тог набрајања. Без обзира на то, саме константе могу да се користе као операнди у целобројним изразима.

Константе које се дефинишу наводе се између пара витичастих заграда {}. Свако ИМЕ КОНСТАНТЕ је идентификатор, који представља по једну симболичку константу. Уобичајено је да се имена симболичких константи пишу само великим словима.

Вредности појединих симболичких константи могу да се одреде константним целобројним изразима. Ако иза имена константе нема *=вредност,* константа ће имати вредност која је за један већа од вредности претходне константе у низу. Ако иза првог идентификатора у низу нема вредности, прва константа имаће вредност нула.

Исти идентификатор константе сме да се појави само у једном набрајању. Иденти- фикатор ниједне константе не сме да се поклапа са идентификатором неког од података који се користе у програму. Нема никаквих препрека да више константи има исту вредност.

Симболичке константе дефинисане набрајањима, формално гледано, су типа **int** па могу да се користе на исти начин као и праве целобројне константе

Име набрајања, слично именима структура, нема статус идентификатора типа, па не може самостално да се користи у наредбама за дефинисање података, већ мора да се пише **enum** *име_набрајања*. Уместо тога, као и код структура, препоручује се придруживање идентификатора типа датом набрајању помоћу наредбе **typedef** облика:

```
typedef enum { ИМЕ_КОНСТАНТЕ = вредност ,
               ИМЕ_КОНСТАНТЕ - вредност , ... } Име_типа ;
```

У оваквим случајевима *име набрајања,* непосредно иза речи **enum** обично се изоставља.

Подаци типа набрајања су, у суштини, типа **int**. У изразима могу да се користе равноправно са осталим подацима типа **int.** Приликом доделе вредности променљивима набројаних типова, преводилац није дужан да проверава да ли се додељена вредност налази у скупу могућих вредности за дато набрајање.

Пример **6.6** *- Дефинисање и употреба набрајања*

```
enum { NE, DA } ;
enum { MIN = 10, MAX = 100, POZELJN0=20 };,..
```

```
enum meseci { JAN = 1, FEB, MAR, APR, MAJ, JUN,
              JUL, AVG, SEP, OKT, NOV, DEC };
typedef enum { CRNA, SIVA, SMEDJA, CRVENA,
               ZUTA, SELENA, PLAVA, BELA } Boja;
typedef enum meseci Meseci;
enum meseci mesec = JAN;
Meseci leto = JUN;
Boja boja = CRVENA + 2;                    /* ZELENA */
```

Првом наредбом дефинишу се две константе означене са NE u DA, чије су вред- ности, подразумевано, 0 u 1. Оне могу da се користе за означавање логичких вред- ности.

Друга наредба приказује експлицитно навођење вредности за све наведене симболичке константе.

Трећа наредба показује како могу да се дефинишу симболичке константе са узастопним вредностима, а да прва константа има вредност 1. На тај начин, симболичке ознаке месеца представљају уобичајене бројчане вредности, којима се обележавају месеци у свакодневном животу.

Четврта наредба могла би да се користи у неком програму у којем се ради са бојама. На тај начин боје се кодирају са 0 *(CRNA)*, 1 *(SIVA)*, 7 *(BELA)*. Очигледно, програм у којем се променљива која садржи код боје упоређује са симболичком константом *CRVENA* много је разумљивији од програма у којем се иста променљива упоређује са константом 3 (која треба да представља црвену боју). У овој наредби, набрајању је одмах придружен и идентификатор типа (коришћена је наредба **typedef***)*.

Пета наредба приказује како може накнадно да се придружи идентификатор типа раније дефинисаном набрајању. Уместо навођења целокупног набрајања, за означавање типа коришћено је **enum** *meseci*. Идентификатор типа који се овом наредбом уводи је *Meseci*.

Последње три наредбе приказују дефиниције променљивих типа набрајања са иницијализаторима.

6.6 Питања

1. Шта су структуре?
2. За шта се користе структуре?
3. Како се дефинишу структуре?
4. Ког типа могу да буду поља структура?
5. Да ли структуре могу да буду поља структура?
6. Да ли низови могу бити поља структура?
7. Да ли се име структуре може користити као идентификатор типа?
8. Како се дефинишу променљиве структурног типа?
9. Како се наводе почетне вредности за структуре?
10. Како се може додељивати идентификатор структурном типу?
11. Да ли подаци структурних типова могу користити слично простим подацима
12. Како се међусобно додељују подаци структурних типова?
13. Да ли структуре могу бити параметри функција?
14. Да ли структуре могу бити вредности функција?
15. Да ли постоје низови структура?
16. Може ли адресна аритметика да се примени на низове структура?
17. Како се одређује величина структуре?
18. Како се приступа пољима структуре?
19. Како се, помоћу показивача, приступа пољима структуре?
20. Да ли може да се образује адреса поља структуре?
21. Шта су уније?
22. Како се дефинишу уније?
23. Какво је ограничење при додели почетне вредности унији?
24. Шта су структуре од битова?
25. Какви су проблеми при употреби структура од битова?
26. Како се дефинишу структуре од битова?
27. Шта су набрајања?
28. Како се дефинишу набрајања?
29. Како се одређују вредности набројаних константи?
30. Како се идентификатор типа придружује набрајању?