Programarea calculatoarelor și limbaje de programare 1

Laborator 3
Structuri, uniuni și enumerări. Instrucțiuni

Dr. Ing. Liviu-Daniel ȘTEFAN
liviu_daniel.stefan@upb.ro

Dr. Ing. Mihai DOGARIU
mihai.dogariu@upb.ro

Prof. Dr. Ing. Bogdan IONESCU
bogdan.ionescu@upb.ro

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Universitatea POLITEHNICA din Bucureşti

30 Octombrie - 1 Noiembrie, 2023

Cuprins

Structuri de date

Semantică

O structură este un tip de date agregat, care grupează mai multe variabile corelate sub același nume. Variabilele care fac parte din structură sunt denumite membrii (elemente, sau câmpuri) ai structuri.

Sintaxă definire

```
struct nume_generic {
tip nume_membru1;
...
tip nume_membruN;
};
```

- O definire de structură formează un șablon care poate fi folosit pentru a crea o variabilă de tip structură. Astfel, definirea unei structuri nu alocă memorie.

Structuri de date

Sintaxă declarare

```
struct nume_generic listă_de_identificatori;
```

- Numele generic al unei structuri este numele structurii;
- Lista de identificatori este o listă separată prin virgulă de identificatori / nume de variabile.
- ☞ Când este declarată o variabilă de tip structură, compilatorul alocă automat suficientă memorie pentru a stoca toți membrii săi.

Sintaxă accesare membru structură

```
identificator.nume_membru;
identificator->nume_membru;
```

- Accesul la membrii unei structuri se realizează prin operatorul (" . "), în timp ce accesul la membrii unui pointer la structură se realizează prin operatorul ("->").

Exemplu de utilizare a tipului de date `struct`

```c
/* Definește sablonul PERSOANA cu membrii nume, prenume, varsta, gen.
Declara variabila persoana_noua de tip PERSOANA și inițializează
fiecare membru cu numele, prenumele, varsta si genul persoanei. */
#include <stdio.h>
int main(){
    struct PERSOANA{
        char nume[256];
        char prenume[256];
        int varsta;
        char gen[9];
    };
    struct PERSOANA persoana_noua;
    printf("Introduceți numele persoanei: ");
    scanf("%[^\n]%*c", persoana_noua.nume);
    printf("Introduceți prenumele persoanei: ");
    scanf("%[^\n]%*c", persoana_noua.prenume);
    printf("Introduceți varsta persoanei: ");
    scanf("%d", &persoana_noua.varsta);
    printf("Introduceți genul persoanei: ");
    scanf("%s", persoana_noua.gen);
    printf("Ati introdus persoana %s %s, %d, %s\n", persoana_noua.nume,
    persoana_noua.prenume, persoana_noua.varsta, persoana_noua.gen);
  return 0;
}
```

Câmpuri de biți

Semantică

Un câmp de biți este un tip special de membru al unei structuri, care definește cât de lung trebuie să fie câmpul, în biți. Câmpul de biți permite accesul la un singur bit dintr-un octet.

Sintaxă definire

```
struct nume_generic {
tip nume_membru1 : lungime_in_biti;
...
tip nume_membruN : lungime_in_biti;
};
```

- Aici, tip specifică tipul câmpului de biți, care poate să fie de tip `int`, `unsigned` sau `signed`. Câmpul de lungimea 1 poate să fie doar de tip `unsigned` (un singur bit nu poate avea semn). Nu puteți folosi operatorul `&` pentru a obține adresa unui câmp de biți.

Exemplu de utilizare a unui câmp de biți

```c
/* Folosește o structură pentru a reprezenta o data, și un câmp pe biți pentru
 a optimiza amprenta asupra memoriei pentru a reprezenta aceași dată.*/
#include <stdio.h>
struct data_prin_structura{
    unsigned int zi;
    unsigned int luna;
    unsigned int an;
};
struct data_prin_camp_pe_biti{
    unsigned int zi : 5; // (0 - 31), 5 biți sunt suficienți
    unsigned int luna : 4; // (0 - 12), 4 biți sunt suficienți
    unsigned int an;
};
int main(){
    printf("Dimensiunea pentru data folosind o structura este de %lu \
    octeți\n", sizeof(struct data_prin_structura));
    struct data_prin_structura d_struct = { 25, 10, 2022 };
    printf("Data este %d/%d/%d\n", d_struct.zi, d_struct.luna, d_struct.an);
    printf("Dimensiunea pentru data folosind un camp de biti este de %lu \
     octeți\n", sizeof(struct data_prin_camp_pe_biti));
    struct data_prin_camp_pe_biti d_biti = { 25, 10, 2022 };
    printf("Data este %d/%d/%d", d_biti.zi, d_biti.luna, d_biti.an);
    return 0;
}
```

Uniuni de date

Semnatică

O uniune este o locație de memorie care este împărțită în momente diferite între două sau mai multe variabile diferite, în general de tipuri diferite.

Sintaxă definire

```
union nume_generic {
tip nume_membru1;
...
tip nume_membruN;
};
```

- Când este declarată o variabilă de tip `union`, compilatorul alocă automat memorie suficientă pentru a păstra cel mai mare membru al acesteia.

Exemplu de utilizare a tipului de date `union`

```c
/* Definește o uniune UM cu membrii metri, centimetri și inci. Inițializează
membrul metri și apoi face conversia din metri in centimetri și din
centimetri în inci. Exemplifică faptul că membrul metri a fost corupt după
inițiealizarea unui alt membru prin realocarea memoriei membrului nou.*/
#include <stdio.h>
#define _INCH_ 2.54
union UM{
  float metri;
  unsigned long int centimetri;
  double inci;
};
int main(){
    union UM dim;
    printf("Introduceți dimensiunea în metri: ");
    scanf("%f", &dim.metri);
    printf("Ați introdus %f metri\n", dim.metri);
    dim.centimetri = 100 * dim.metri;
    printf("Convertiți in centimetri obținem %lu cm\n", dim.centimetri);
    dim.inci = dim.centimetri /_INCH_;
    printf("Convertiți in inci obținem %f inci\n", dim.inci);
    printf("(Afișare eronată!) Dimensiunea în metri este %f", dim.metri);
    return 0;
}
```

Enumerări de date

Semantică

O enumerare este un set de constante de tip întreg care specifică toate valorile permise pe care le poate avea o variabilă de acel tip.

Sintaxă definire

```
enum nume_generic {lista enumerărilor}
lista_identificatori;
```

- Lista enumerărilor este o listă de identificatori unici, numiți simboluri, separată prin virgulă;
- Fiecare dintre simboluri ține locul unei valori întregi. Fiecărui simbol i se dă o valoare cu o unitate mai mare decât a precedentului;
- Puteți specifica valoarea unuia sau mai multor simboluri prin inițializare, folosind operatorul de atribuire;

Exemplu de utilizare a instrucțiunii `enum`

- Simbolurile care apar după inițializare, li se atribuie valori mai mari decât valoarea de inițializare precedentă.

```
1  /*
2  Definește o enumerare numită zile indexată începând cu 1, conținând
3  simbolurile de la Luni la Duminică. Inițializează o variabilă de tip
4  enum numită azi cu un simbol din enumerare și apoi este folosit pentru
5  a afișa indexul zilei dată de simbol.
6  */
7  #include <stdio.h>
8  enum zile {Luni=1, Marti, Miercuri, Joi, Vineri, Sambata, Duminica};
9
10 int main()
11 {
12     enum zile azi;
13     azi = Marti;
14     printf("Ziua %d",azi);
15     return 0;
16 }
```

Instrucțiuni simple, compuse, nule

Semantică

O instrucțiune, în cel mai general sens, este o porțiune a programului ce poate fi executată. Aceasta inseamnă că o instrucțiune specifică o acțiune. Instrucțiunile pot fi simple, compuse sau nule.

- O instrucțiune simplă este o singură expresie validă urmată de punct și virgulă;
- O instrucțiuni compusă, numită și instrucțiune bloc, este un grup de instrucțiuni unite logic care sunt tratate ca o unitate. O instrucțiune compusă începe cu o acoladă deschisă `{`, și se termină cu corespondentul său, acolada închisă `}`;
- O instrucțiuni nulă, numită și vidă, este marcată doar prin punct și virgulă, și nu specifică nici o acțiune.

Instrucțiuni de tip salt

Semantică

Limbajul C are patru instrucțiuni care execută ramificări necondiționate: `return`, `break` `continue` și `goto`.

- Instrucțiunea `return` este utilizată pentru reîntoarcerea dintr-o funcție, exemplu funcția `main()` astfel, `return` determină execuția programului să se reîntoarcă în punctul în care a fost apelată funcția. Forma generală este `return expresie;`
  - ☞ `return` poate sau nu să aibă asociată o valoare. Dacă `return` are o astfel de valoare, aceasta devine valoarea returnată de funcție;
- instrucțiunea `break` are două utilizări. Determină încheierea imediată a unei instrucțiuni de tip `case` (Slide 20), sau a unei instrucțiuni de iterare (Slide 23);
- instrucțiunea `continue` forțează trecerea la următoarea iterație a unei instrucțiuni iterative (Slide 23);

Instrucțiuni de tip salt

- instrucțiunea `goto` este o instrucțiune de tip salt care folosește o eticheta pentru operație.

Sintaxă

```
goto etichetă;
...
etichetă:
```

- ☞ `etichetă` este un identificator valid plasat înainte sau după `goto`.
- ☞ `eticheta` trebuie să se regăsească în aceeași funcție ca și `goto` care o utilizează. Nu permite saltul între funcții sau fișiere;
- ☞ Nu există situații în programare care să necesite explicit `goto`. Este o facilitate care se face utilă într-un domeniu restrâns de situații;

Exemplu cu instrucțiunea `goto`

- Se recomandă evitarea utilizării instrucțiunii `goto` datorită gradului mare de ilizibilitate pe care îl introduce. `goto` este prezentat doar în scop didactic.

```
1   /*
2   Citește de la tastatură o dată întreagă și afișează pe ecran toate numerele
3   până la valaorea citită de la tastatură, exclusiv. Instrucțiunea goto este
4   folosită pentru a face un salt la etichetă iteratie, atunci când rezultatul
5   testului condițional index < limita este evaluat ca adevarat, lucru ce
6   determină o nouă iterație.
7   */
8   #include <stdio.h>
9   int main(){
10      int limita, index = 1;
11      printf("Afișați toate numerele până la ");
12      scanf("%d", &limita);
13      iteratie:
14      printf("%d ", index++);
15      if (index < limita)
16          goto iteratie;
17    return 0;
18  }
```

Instrucțiuni condiționale

Semantică

Instrucțiunile condiționale se bazează pe o expresie de condiționare care determină cursul acțiunilor următoare. Astfel, o instrucțiune condițională execută un set de instrucțiuni dacă o condiție este adevărată, și eventual, alt set de instrucțiuni daca acea condiție este falsă.

- O expresie condițională este evaluată ca adevărat sau fals. În limbajul C, este adevărată orice valoare diferită de zero, inclusiv numerele negative. O valoare falsă este 0;
- Limbajul C admite două tipuri de instrucțiuni condiționale: `if`, și `switch`. Operatorul ternar condițional `?:` este, în anumite condiții, o alternativă a instrucțiuni `if`.

if, if-else, if-else-if

Sintaxa if

```
if (expresie) instrucțiune;
```

Sintaxa if-else

```
if (expresie)
    instrucțiune;
else
    instrucțiune;
```

- `instrucțiune` poate să fie o instrucțiune simplă, compusă, sau vidă. Clauza `else` este opțională;
- Dacă `expresie` este adevărat, atunci se execută instrucțiunea sau blocul care formează obiectul lui `if`; altfel, este executată instrucțiunea sau blocul care face obiectul lui `else`, dacă există;
- `expresie` trebuie să determine un rezultat scalar (tipuri fundamentale de date și variațiuni).

if, if-else, if-else-if

Sintaxa if-else-if

```
if (expresie)
   instrucțiune;
else if (expresie)
   instrucțiune;
...
else
   instrucțiune;
```

- `If-else-if` este o instrucțiune condițională imbricată (aranjată într-o structură ierarhică) – un `if` care face obiectul unui alt `if`, sau al unui `else`;
- într-o instrucțiune `if` imbricată, o instrucțiune `else` se referă întotdeauna la cea mai apropiată instrucțiune care se află în același bloc cu `else` și care nu este deja asociată unui alt `else`. Condițiile sunt evaluate de sus în jos.

Exemplu cu instrucțiunile `if`, `if-else`, `if-else-if`

```c
/*
Se citesc două valori întregi de la tastatură și se testează relația
dintre x și y folosind scara if-else-if afișând rezultatul în consolă
*/
#include <stdio.h>

int main(){
    int x, y;
    printf("Introduceți două valori întregi de la tastatură: ");
    scanf("%d %d", &x, &y);
    if (x > y)
        printf("%d > %d\n", x, y);
    else if (x < y)
        printf("%d < %d\n", x, y);
    else
        printf("%d == %d\n", x, y);
    return 0;
}
```

switch

Sintaxă

```
switch (expresie) {
    case constanta1:
         secvență de instrucțiuni
         break;
    case constanta2:
         secvență de instrucțiuni
         break;
    case constanta3:
         secvență de instrucțiuni
         break;
    ...
    default
         secvență de instrucțiuni
```

switch

- Într-o instrucțiune `switch`, se testează valoarea din `expresie` față de valorile constantelor specificate prin instrucțiunile `case`. Când se întâlnește o egalitate, se execută secvența de instrucțiuni asociate acelui `case`, până la instrucțiunea `break`, dacă există, altfel până când se ajunge la finalul instrucțiunii `switch`. Instrucțiunea `default` se execută în caz de inegalitatea dintre valoarea din `expresie` și valorile constantelor specificate în instrucțiunile `case`;
- Instrucțiunile `break` și `default` sunt opționale;
- Instrucțiunea `switch` diferă de `if` prin aceea că testează doar egalitatea, în timp ce `if` poate să evalueze orice tip de expresie relațională sau logică;
- În același `switch` nu pot exista două constante `case` cu valori identice. Sunt permise instrucțiuni `switch` imbricate.

Exemplu cu instrucțiunea `switch`

```c
/* Citește o valoare întreagă și afișează ziua asociată valorii întregi.
Cazurile 6 și 7 sunt rulate secvențial, neexistând o instrucțiune break */
#include<stdio.h>
int main() {
  int ziua_curenta;
  printf("Introduceți o valoare de la 1 - 5 reprezentând zile lucrătoare: ");
  scanf("%d", &ziua_curenta);
  printf("Ați introdus ");
  switch (ziua_curenta) {
  case 1: printf("Luni\n");
    break;
  case 2: printf("Marți\n");
    break;
  case 3: printf("Miercuri\n");
    break;
  case 4: printf("Joi\n");
    break;
  case 5: printf("Vineri\n");
    break;
  case 6:
  case 7: printf("Sfârșit de săptămână\n");
  default: printf("o valoare invalidă\n");
  }
  return 0;
}
```

Instrucțiuni de iterare

Semantică

Instrucțiunile de iterare determină o instrucțiune numită corpul buclei, să se execute repetat, până când o expresie de control returnează fals.

- Expresia de control poate să fie predefinită (ca în instrucțiunile `for`–se cunoaște a priori sau se poate calcula numărul de iterații), sau cu sfârșit deschis (ca în instrucțiunile `while` și `do-while`–iterația depinde de un test condițional);
- Expresia de control poate determina execuția unei instrucțiuni inaintea evaluării sale (ca în instrucțiunile `do-while`), sau după evaluarea expresiei de control (ca în instrucțiunile `for` și `while`).

for

Sintaxă

```
for (inițializare; condiție; increment)
    instrucțiune;
```

- `inițializare` este, în general, o instrucțiune de atribuire utilizată pentru a inițializa variabila de control a buclei;
- `condiție` este o expresie relațională care determină ieșirea din buclă;
- `increment` definește modul în care se modifică variabila de control a buclei, de fiecare dată când aceasta se repetă;
- ☞ Cele trei secțiuni principale trebuie separate prin punct și virgulă;
- `instrucțiune` poate să fie o insctrucțiune simplă, compusă, sau nulă;

for

- `for` continuă să se execute atâta timp cât `condiție` este evaluat ca adevărat. Când `condiție` este evaluat ca fals, execuția programului se reia de la instrucțiunea care urmează lui `for`;
- `for` verifică condiția de testare înaintea inceperii iterației, ceea ce înseamnă că nu se va executa corpul buclei dacă acea condiție este falsă;
- `inițializare`, `condiție` și `increment` sunt opționale, ce permite mai multe variațiuni ale instrucțiunii `for`.
  - `inițializare` și `increment` acceptă operatorul virgulă pentru a permite ca bucla să fie controlată de mai multe variabile;
  - Când `condiție` este absent, expresia de condiționare este întotdeauna adevărată, lucru ce permite realizarea unei iterații infinite, dacă nu se specifică altfel în interiorul corpului buclei. Iterații infinite se pot obține și prin controlul secțiunilor `inițializare`, `increment` și `instrucțiune`.

Exemple instrucțiunea `for`

```c
#include<stdio.h>

    int main(){
    int var_control;
    char caracter = '\0';
    /* for cu secţiunile iniţializare, condiţie, increment, şi o instrucţiune
     simplă. Afişează cifrele de la 0 - 9
    */
    for(var_control = 0; var_control < 10; var_control++)
        printf("%d\n", var_control);

    /* for fără secţiunile iniţializare şi increment cu o condiţie compusă
       Citeşte de la tastatură un caracter şi îl afişează pe ecran
       până când este introdus caracterul 'Q' sau 'q'.
    */
    printf("\nIntroduceţi un caracter de la tastatură.\
            Tastaţi Q sau q pentru a reununţa\n");
    for(; caracter != 'Q' && caracter != 'q'; ){
        caracter = getchar();
        putchar(caracter);
    }
    return 0;
}
```

Exemple instrucțiunea `for`

```c
#include <stdio.h>

int main() {
   char c;
   /* instrucţiuni iterativă fără secţiunile de iniţializare, condiţie
    şi increment. Condiţia este evaluată întotdeauna ca adevărat şi
    se încheie printr-un test condiţional care, atunci când este
    evaluat ca adevărat, execută instrucţiunea break.
   */
   for(;;) {
      printf( "\nApăsaţi orice tastă. Apăsaţi Q pentru a renunţa: " );
      scanf("%c%*c", &c);
      if (c == 'Q')
          break;
   } // iteraţia se închieie doar dacă tasta 'Q' a fost apăsată.
   return 0;
}
```

Exemple instrucțiunea `for`

```c
/*
Citește de la tastatură o dată întreagă și afișează pe ecran numerele
impare până la valaorea citită de la tastatură, inclusiv. Instrucțiunea
continue este folosită pentru a forta o noua iterare, atunci când testul
condițional dat de !(index%2) este adevărat, astfel, execuția programului
nu ajunge la instrucțiunea de afișare a indexului curent.
*/
#include <stdio.h>
int main() {
   int index, limita;
   printf("Afișați numerele impare până la: ");
   scanf("%d", &limita);
   for (index = 0; index <= limita; index++) {
      if (!(index % 2))
         continue;
      printf("%d ", index);
   }
   return 0;
}
```

while

Sintaxă

```
while (condiție)
       instrucțiune;
```

- `instrucțiune` poate să fie o instrucțiune simplă, compusă, sau nulă;
- `condiție` poate să fie orice expresie. Iterația se continuă atât timp cât `condiție` determină un rezultat adevărat. Când `condiție` devine fals, controlul programului se reia de la linia de cod imediar următoare instrucțiunii `while`;
- `while` verifică condiția de testare înaintea inceperii iterației, ceea ce înseamnă că nu se va executa corpul buclei dacă acea condiție este falsă.

Exemple instrucțiunea `while`

```c
#include<stdio.h>
int main(){
    int var_control = 0;
    char caracter = '\0';
/* while cu instrucţiune simplă. Afișează cifrele de la 0 - 9
    var_control a fost inițializată la declarare cu 0. increment este
    asigurat de operatorul de incrementare postfix. Atunci când se cunoaște
    sau se poate calcula numărul de iterații, se recomandă folosirea
    instrucţiunii for. Acest exemplu este doar didactic.
*/
    while(var_control++ < 10)
          printf("%d\n", var_control);
/* while cu o instrucţiune compusă. Citește de la tastatură un caracter și
   îl afișează pe ecran până când este introdus caracterul 'Q' sau 'q'. Atunci
   când nu se cunoaște sau nu se poate calcula numărul de iterații ca în acest
   exemplu, se recomandă folosirea instrucţiunii while sau do-while.
*/
    printf("\nIntroduceţi un caracter de la tastatură.\
     Q sau q pentru a reununța\n");
    while(caracter != 'Q' && caracter != 'q'){
        caracter = getchar();
        putchar(caracter);
    }
    return 0;
}
```

do-while

Sintaxă

```
do instrucțiune;
while (condiție);
```

- `instrucțiune` poate să fie o instrucțiune simplă, compusă, sau nulă;
- Spre deosebire de `for` și `while`, care testează condiția de continuare a iterației la începutul său, instrucțiunea `do-while` o verifică la sfârșit. Aceasta inseamnă că instrucțiunea din `do-while` se execută cel puțin o dată;
- ☞ Chiar dacă acoladele nu sunt necesare atunci când este vorba despre o instrucțiune simplă sau nulă, o practică bună implică folosirea lor pentru a îmbunătăți lizibilitatea programului.

Exemple instrucțiunea `do-while`

```c
#include<stdio.h>
int main(){
    int var_control = 0;
    char caracter = '\0';
/* do-while cu instrucţiune simplă
Afişează cifrele de la 0 - 9 var_control a fost iniţializată la declarare
cu 0 increment este asigurat de operatorul de incrementare postfix. Atunci
când se cunoaşte sau se poate calcula numărul de iteraţii se recomandă
folosirea instrucţiunii for. Acest exemplu este doar didactic. */
    do
    printf("%d\n", var_control);
    while(var_control++ < 10);
/* do-while cu o instrucţiune compusă. Citeşte de la tastatură un caracter şi
îl afişează pe ecran până când este introdus caracterul 'Q' sau 'q'. Atunci
când nu se cunoaşte sau nu se poate calcula numărul de iteraţii ca în
acest exemplu se recomandă folosirea instrucţiunii while sau do-while.
Atunci când este necesară execuţia instrucţiunii cel puţin o dată, se
recomandă folosirea instrucţiunii do-while */
    do{ printf("\nIntroduceţi un caracter de la tastatură.\
     Q sau q pentru a reununţa\n");
        scanf("%c%*c", &caracter);
        putchar(caracter);
    } while(caracter != 'Q' && caracter != 'q');
    return 0;
}
```