Cursul 12
STRUCTURI (II)

OBIECTIVE

- structuri cu autoreferire, campuri de biti si uniuni

PREZENTARE GENERALA

Structuri cu autoreferire

Structurile cu autoreferire prezinta avantajul ca se poate memora o cantitate variabila de valori, lucru care nu se poate realiza prin folosirea masivelor care fie sunt prea mari si ocupa mai multa memorie decat este necesar, fie sunt prea mici si nu pot memora toate valorile necesare.

O structura cu autoreferire arata astfel :

```
struct nod {
    Persoana pers;
    nod *next;
};
```

In general, este ilegal ca o structura sa contina o parte a ei insasi, dar **nod *next;** declara `next` ca fiind un pointer catre un nod si nu nodul insusi.

```
struct descendent;
struct nod {
    int eticheta;
    struct nod *rest;
    struct descendent *desc;
};

struct descendent {
    struct nod *id;
    struct descendent *urm;
};
```

Prima declaratie din exemplu este o declaratie "incompleta", care anunta ca ulterior se va defini o structura numita **descendent.** Cea de-a doua declaratie precizeaza ca structura cu numele **nod** are trei campuri: campul **eticheta,** de tip **int**, campul **rest**, de tip pointer catre **nod** si campul **desc**, de tip pointer catre o structura de tip **descendent**, cu doua campuri de tip pointer – unul catre un **nod** si celalalt catre un **descendent.**

Campurile unei structuri se declara dupa sintaxa obisnuita a declaratiilor de variabile, dar nu pot avea valori initiale. In plus, sintaxa este extinsa, permitand specificarea, prin intermediul unei expresii constante, a numarului de biti alocat pentru un anumit camp, sub forma:

```
tip_camp selector:numar_biti;
```

O astfel de componenta, pentru care se specifica numarul de biti alocat, se numeste camp de biti.

Campuri de biti

Se pot aloca unor campuri de structuri sau uniuni biti dintr-un octet.In felul acesta se definesc campuri de biti (bit-field) care pot fi accesate fiecare, separat de restul octetului, pentru evaluare si/sau modificare.

Pentru aplicatii de control al unor dispozitive fizice (interfete), criptare, etc, care solicita structura informatiei dintr-o locatie de memorie, utilizarea campurilor de biti simplifica substantial redactarea programelor.

Campurile de biti se pot declara ca membri ai unei structuri cu sintaxa :

```
struct identificator_tip_structura {
    tip_element1 <identificator_element1>:lungime
    ..............................
    tip_elementN <identificator_elementN>:lungime
} lista_identificatori_variabile_structura ;
```

Exista urmatoarele restrictii pentru campurile de biti :

- tipul poate fi **int**, **signed** sau **unsigned;**
- lungimea este o constanta intreaga cu valoarea in domeniul 0-15;
- nu se poate evalua adresa unui camp (deci operatorul & nu poate avea ca operand un camp de biti);
- nu se pot organiza tablouri de campuri de biti.

Referirea unui element camp de biti dintr-o structura se face ca pentru orice alt tip de element, folosind operatorii de selectare directa sau indirecta (punct sau sageata).

Numele campului poate sa lipseasca. In acest caz, se face alocarea bitilor specificati de valoarea "lungime", dar campul nu poate fi accesat. Pot fi alocati astfel biti care nu sunt utilizati dintr-un octet.

Vom considera, ca exemplu, structura informatiei transferate intr-o comunicatie seriala sincrona dintre calculatoare, pe baza protocolului HDLC. In acest caz, blocurile de date transmise sunt completate cu anumite informatii de control, rezultand in final un "cadru" de date organizat ca o secventa de biti.

Structura din exemplul urmator descrie un cadru de date HDLC in reprezentarea interna, utilizata de un program de comunicatie. Ea contine adresa si campul de control necesare protocolului si dimensiunea si adresa blocului de date :

```
struc cadru_i {
    unsigned char adresa;// adresa pe 8 biti
    /* camp de control pe 8 biti */
    /* bit identificare cadru I*/
    unsigned c_i:1;
    /*contor transmisie N (S) */
    unsigned ns:3;
    /* bit PIF */
    unsigned pf:1;
    /*contor receptie N (R) */
    unsigned nr:3;
    /* dimansiunea si adresa blocului de date */
    unsigned lng;
    char *data;
};
```

Alocarea bitilor se face de la bitul 0 al octetului, in ordinea declararii. Astfel, daca se declara mai multe campuri care insumeaza cel mult 8 biti se aloca spatiul in cadrul aceluiasi octet.

Pentru campurile de tip **int** (**signed int**) valorile sunt memorate in reprezentarea in complement fata de doi, pentru lungimea specificata. Campurile **unsigned int** contin valoarea binara inscrisa.La inscrierea unui camp de biti, programatorul trebuie sa urmareasca respectarea domeniului de valori corespunzator lungimii, altfel alocarea este alterata prin trunchiere. Campurile de biti se pot initializa la fel ca orice alt membru al structurii.

Uniuni

In unele situatii, apare necesitatea ca o aceeasi zona de memorie sa fie accesata la momente diferite de timp, in contexte diferite. De exemplu, consideram ca la inceputul unui program dorim ca o anumita zona de memorie sa fie accesata prin intermediul unei variabilede tip interg, urmand ca ulterior aceeasi zona de memorie sa fie accesata prin intermediul unei variabile de tip real. O zona de memorie poate fi alocata mai multor obiecte de tipuri diferite prin declarea unei uniuni.

Sintaxa declaratiei este similara cu cea a structurii, dar identificatorii declarati ca membri reprezinta numele cu care sunt referite diferitele tipuri de obiectecare utilizeaza in comun zona de memorie.

Declarea unei uniuni se face astfel :

```
union identificator_tip_uniune {
    tip_element1 identificator_element1;
    ..............................
    tip_elementN identificator_elementN;
} lista_identificatori_variabile_uniune;
```

Spatiul alocat in memorie corespunde tipului cu dimensiune maxima. O variabila uniune poate fi initializata numai cu valoarea corespunzatoare primului membru. Pentru refolosirea unui membru al unui uniuni se folosesc tot operatorii de selectie "punct" si "sageata".

```
int k=5;
float r=12.3;
union tipifl nr,*pu=&nr;

/* variabila initializata */
union tipifl alt_nr={10};

nr.i=k; /* nr contine valoarea intreaga 5 */
nr.f=r; /* nr contine valoarea float 12.3 */
pu->f=k; /* nr contine valoarea float 5.00 */
```

Pentru exemplificare putem relua structura unui cadru HDLC. In afara cadrelor de date de care ne-am ocupat in paragraful precedent, protocolul HDLC mai foloseste doua categorii de cadre de control al comunicatiei, cu format diferit : cadre de supervizare (S) si cadre nenumerotate(U). Aceste cadre se pot declara ca structuri cu campuri de biti in mod similar cu cadrele de date din exemplul 1 de la campuri de biti.

In exemplul urmator se declara o uniune care poate fi utilizata pentru memorarea oricarui cadru HDLC:

```
union cadru_hdlc {
```

```
    struc {
        /* cadru informatic */
        unsigned char adresa;
        unsigned c_i:1;
        unsigned ns:3;
        unsigned pf:1;
        unsigned nr:3;
        unsigned lng;char *data
    } cadru_i;

    struct {
        /* cadru supervizare */
        ......................
    } cadru_s;

    struct {
        /* cadru nenumerotat */
        ......................
    } cadru_u;
};
```

In aceasta prima varianta, declaratiile tipurilor membrilor uniunii sunt interne. Daca obtinerea acestui rezultat nu este dorita, declaratiile tipurilor de cadre se fac separat. De exemplu, prin utilizarea unor declaratii **typedef**, tipul uniune se poate declara mai simplu:

```
typedef struc {...} tip_c_i;
typedef struc {...} tip_c_s;
typedef struc {...} tip_c_u;
typedef union
{
    tip_c_i cadru_i;
    tip_c_s cadru_s;
    tip_c_u cadu_i;
} cadru_hdlc;
```

Folosirea tipului uniune de cadre permite simplificarea programului de comunicatie, de exemplu in privinta manevrarii diferitelor cadre (transferuri de parametrii, liste, etc.) si a efectuarii prelucrarilor, care sunt similare pentru toate tipurile.