Curs 1. *25.02.2003*
Aplicatii: Tablouri si pointeri; Pointeri la functii

Tipul pointer (indicator) reprezinta adrese ale unor zone de memorie. Din punct de vedere al continutului zonei de memorie indicate, exista urmatoarele categorii de pointeri:

- ***pointeri de date***, care contin adresa unei variabile
- ***pointeri de functii***, care contin adresa codului executabil a unei functii. In C, o functie nu este o variabila, dar este posibil a se defini pointeri la functii care pot fi atribuiti, plasati in tablouri, transmisi ca parametrii altor functii, etc.
- ***pointeri generici***, pointeri `void *`, pot contine adresa unui obiect oarecare, de orice tip. Se utilizeaza atunci cand nu se cunoaste precis tipul de date care va fi instantiat in mod dinamic la executie.

Relatia intre pointeri si tablouri in C.

Numele unui tablou este echivalent cu un pointer constant catre primul element al tabloului.

Presupunand ca avem declaratia:

```
tip_elem T[nr_elem];
```

Numele tabloului, T este echivalent cu expresia &T si cu expresia &T[0]. Tipul oricarei expresii dintre acestea este pointer catre tipul de baza al tabloului (tip_elem).

Accesul la primul element al tabloului poate fi facut sub forma: T[0] sau *T.

Reciproc, un pointer poate fi folosit ca si cum ar fi numele unui tablou, cu exceptia pointerilor void si a pointerilor catre functii.

```
int t[10], *p;
p=t;
t[0]=p[5];
*t=p[5];
```

Datorita faptului ca numele unui tablou este un pointer constant (ca si cum ar fi fost declarat cu const), valoarea sa nu poate fi modificata.

```
int t1[10], t2[10], *p;

t1=t2; /* eroare de compilare, se incearca modificarea valorii
pointerului constant t1 */
```

```
p=t2;  /* expresie corecta, echivalenta cu p=&t2[0]. Nu se
copiaza tabloul, ci pointerul p va indica catre zona de memorie
incare se gaseste t2  */
```

Inrudirea dintre pointeri si tablouri influenteaza puternic regulile dupa care se fac inC operatiile aritmetice cu pointeri. Rezultatul adunarii sau scaderii dintre un pointer si un intreg are sens daca valoarea pointerului reprezinta adresa unui element de tablou.

Aceste notiuni legate de tipurile de pointeri vor fi ilustrate in rezolvarea urmatoarei aplicatii.

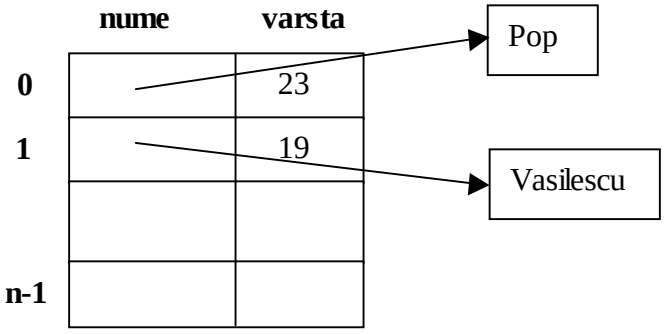
Aplicatie. *Se citesc de pe mediul de intrare date(numele si varsta) despre un numar oarecare de personae si se doreste generarea listei persoanelor, ordonata intai in ordine alfabetica si apoi lista ordonata in ordinea crescatoare a varstei.*

Aplicatia va fi rezolvata in 3 pasi:

1. determinarea structurii de date necesare memorarii informatiilor despre persoane, citirea datelor si initializarea structurii
2. sortarea listei de persoane dupa cele 2 criterii
3. generalizarea functiei de sortare

Pasul 1. *Se citesc date (nume si varsta) despre un numar oarecare de persoane si se memoreaza intr-o structura de date. Intai se citeste n, numarul de persoane, apoi se citesc datele despre toate cele n persoane. Sa se gaseasca o structura de date adecvata astfel incat consumul de memorie sa fie optim.*

Pentru memorarea informatiilor referitoare la o persoana, se defineste tipul structurat persoana, avand campurile nume, de tip char *, si varsta de tip intreg. Implementarea tabloului se face dinamic: dupe ce se stie numarul de persoane, se poate aloca spatial corespunzator pentru n structuri de tip persoana. O structura de tip persoana nu contine numele persoanei, ci un pointer la sirul de caractere pentru nume. In felul acesta, se poate aloca dinamic un spatiu de memorare cu dimensiunea egala cu lungimea numelui fiecarei persoane in parte. Numele este citit intai intr-o variabila tampon de tip sir de caractere cu lungime fixa (40 decaractere), apoi se aloca dinamic spatiu pentru campul nume.



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

typedef struct {
            char *  nume;
            int varsta;
} persoana;



/* citeste date despre n persoane si construieste un tablou
alocat dinamic */
/* parametrii functiei sunt pointeri pentru a forta transferul
lor prin referinta
   n= numarul de persoane - se citeste
   tab = tabloul celor n persoane - se aloca dinamic in cadrul
functiei de  citire,
   deci se modifica insasi adresa de inceput a tabloului -> de
aceea se pune parametru
   pointer la tablou */.


void citeste (int * n, persoana ** tab) {
      persoana * t;

      char sir[40];
      int i;

      printf("Introduceti numarul de persoane\n");
      scanf("%d", n);

      /* aloca spatiu pentru n structuri de tip persoana */
      if (!(t=(persoana *) malloc ((*n) * sizeof(persoana)))) {
            printf("Eroare alocare dinamica memorie \n");
            exit(1);
      }

      /* atribuie spatiul alocat tabloului de persoane */
      *tab=t;

      /* citeste datele despre fiecare persoana */
      for (i=0; i<*n; i++, t++) {

            printf("\n nume: ");
            scanf("%s", sir);

            /* aloca memorie pentru nume, in functie de lungimea
numelui */
            if (!(t->nume=(char *)malloc (strlen(sir)+1))) {
                  printf("Eroare alocare dinamica memorie \n");
                  exit(1);
            }

            strcpy(t->nume, sir);
            printf("\n varsta: ");
            scanf("%d", &t->varsta);
```

```
      }
}


void afiseaza (int n, persoana * tab) {
      int i;
      for (i=0; i<n; i++, tab++)
            printf("\n %-30s %4d", tab->nume, tab->varsta);
}


void main(void) {
      int n;
      persoana *tabel=NULL;

      citeste(&n, &tabel);

      afiseaza(n, tabel);

      }
```

Observatii.

1. *Utilizarea pointerilor pentru transmiterea parametrilor prin referinta.*

Pentru ca o functie C sa poata modifica valoarea unei variabile care ii este parametru, parametrul trebuie declarat de tip pointer iar la apelare trebuie sa i se dea adresa variabilei. Functia `citeste(int * n, persoana ** tab)` realizeaza introducerea valorilor pentru n,numarul de persoane, si tab, tabloul in care sunt memorate datele despre acestea. N si tab trebuie sa fie parametrii transmisi prin referinta, deci pointeri. Trebuie observat faptul ca in acest caz si tabloul este transmis ca si pointer la pointer ! desi se stie ca parametrii de tip tablou constituie in C exceptia de la regula transferului prin valoare. In cazul de fata insa nu este vorba doar de modificarea valorilor elementelor tabloului,ci de insasi alocarea spatiului de memorie pentru tablou, deci de modificarea adresei de inceput a tabloului.

2. *Referirea elementelor de tablou prin adresare indexata sau pointeri*.

Instructiunea `for` in care se face citirea datelor despre cele `n` persoane utilizeaza adresarea persoanelor prin pointerul `t` care este incrementat la fiecare iteratie. Se putea utiliza si adresarea indexata, in forma:

```
for (i=0; i< *n; i++) {
            ....
            ...  t[i].nume ...
            ...
          }
```

De asemenea, se putea utiliza adresarea indexata si pentru functia de afisare:

```
void afiseaza(int n, persoana tab[]) {
/* alta varianta de afisare */
      int i;
      for (i=0; i<n; i++)
            printf("\n %-30s %4d", tab[i].nume, tab[i].varsta);
}
```

Pasul 2. *Sa se completeze aplicatia anterioara, cu sortarea persoanelor odata in ordine alfabetica a numelor si odata in ordine crescatoare a varstei.*

Pentru cele 2 sortari cerute ale listei de persoane, ceea ce difera este criteriul de sortare. O sortare se descompune in 3 tipuri de operatii :

- o comparatie care determina ordinea relativa a 2 elemente
- o interschimbare a unei perechi de elemente
- un algoritm care efectueaza operatii de comparare si interschimbare pana cand elementele sunt in ordinea dorita.

Algoritmul de sortare este independent de operatiile de comparare si interschimbare. Utilizand functii diferite de comparare si interschimbare, se pot realiza sortari dupa diverse criterii (numeric, alfabetic) sau a diferite tipuri de date.

Pentru compararea a 2 persoane in functie de diferite criterii, se vor utiliza functii separate ce primesc ca si parametrii 2 persoane p1 si p2, si returneaza o valoare intreaga pozitiva daca p1 > p2, o valoare intreaga negativa daca p1< p2, respective zero daca intre cele 2 persoane nu se poate face o ordonare pe baza criteriului de comparatie dat.

```
typedef int (*fct_cmp) (persoana , persoana );

void sortare (persoana *sir, int n, fct_cmp f) {
      int i,j;

      persoana temp;

      for (i=0; i<n-1; i++)
            for (j=i+1; j<n; j++)
                  if ((*f)(sir[i], sir[j])<0) {
                        temp=sir[i];
                        sir[i]=sir[j];
                        sir[j]=temp;
                  }
}


int fc1(persoana p1, persoana p2) {
      return strcmp(p1.nume, p2.nume);
}

int fc2(persoana p1, persoana p2) {
      return p1.varsta-p2.varsta;
}
```

```
void main(void) {
      int n;
      persoana *tabel=NULL;

      citeste(&n, &tabel);

      afiseaza(n, tabel);

      sortare(tabel, n, fc1);
      printf("\n Lista in ordine alfabetica:");
      afiseaza(n, tabel);

      sortare(tabel, n, fc2);
      printf("\n Lista in ordinea varstei:");
      afiseaza(n, tabel);

}
```

Observatii.

1. *Transferul ca parametru al unei functii.*

In primul rand se declara `fct_cmp` ca pointer cu tipul "functie cu rezultatul de tip intreg si 2 parametrii de tip persoana". Functia `sortare` are un parametru `f` de acest tip `fct_cmp`. Functia f este apelata in interiorul corpului functiei sortare, in forma `(*f)(sir[i], sir[j]).` La apelul functiei `sortare`, locul acestui parametru formal f este luat de functiile `fc1` si `fc2`;

Pasul 3*. Sa se generalizeze functia de sortare din problema anterioara, astfel incat codul sa poata fi utilizat pentru sortarea tablourilor cu elemente de orice tip.*

Functia sortare definite anterior poate fi utilizata numai pentru sortarea unor tablouri cu elemente de tip persoana, in functie de diferite criterii.
Pentru a putea sorta tablouri cu elemente de orice tip, functia de sortare trebuie scrisa ca avand ca parametru tabloul de sortat dat sub forma `void *`. Este necesara utilizarea tipului de pointer generic void * pentru ca tipul elementelor tabloului de sortat va fi cunoscut doar in momentul executiei.
Functia de comparare a doua elemente de tablou va avea ca parametrii 2 pointeri de tip void, reprezentand adresele a 2 elemente din tablou.
De asemenea, va ridica probleme si generalizarea codului care realizeaza interschimbarea a 2 elemente de tablou care nu sunt in ordinea buna, datorita tipului necunoscut al elementelor, despre care se poate cunoaste doar dimensiunea. Interschimbarea elementelor nu se mai face prin secventa de 3 atribuiri, ci utilizand o functie care muta blocuri de memorie de dimensiuni date (memmove). Fiecare atribuire intre 2 variabile de tipul de baza al tabloului este inlocuita cu mutarea unui bloc de memorie de dimensiunea tipului de baza al tabloului. Tipul de baza al tabloului fiind necunoscut, dimensiunea sa trebuietransmisa ca si parametru explicit functiei de sortare.

```
typedef int (*fct_cmpg) (const void *, const void * );

/* functie generica de sortare. Sorteaza orice tablou sir cu n
elemente, pentru care se cunoaste dimensiunea unui element
(sizeof), si pentru care e data o functie de comparatie intre 2
elemente
*/
void sortareg(void *sir, int n, int dim_elem, fct_cmpg f) {
      int i,j;

      void * temp;
      void * elemi, *elemj;
      temp=malloc(dim_elem);

      for (i=0; i<n-1; i++)
            for (j=i+1; j<n; j++) {
                  elemi=(char *)sir+dim_elem*i;
                  elemj=(char *)sir+dim_elem*j;
                  if ((*f)(elemi, elemj)>0) {
                        memmove(temp, elemi, dim_elem);
                        memmove(elemi, elemj, dim_elem);
                        memmove(elemj, temp,dim_elem);
                  }
            }
}

/* compara 2 elemente de tip persoana, in functie de nume */
int fcg1(const void * p1, const void * p2) {
      return strcmp(((persoana *)p1)->nume, ((persoana *)p2)-
>nume);
}

/* compara 2 elemente de tip persoana, in functie de varsta */
int fcg2(const void * p1, const void * p2) {
      return ((persoana *)p1)->varsta - ((persoana *)p2)->varsta;
}

/* compara 2 intregi */
int fcg_int(const void *a, const void *b) {
      return (*(int*)a - *(int *)b);
}


void main(void) {
      int n;
      persoana *tabel=NULL;

            int tab_int[]={6,3,9,1,8,2};

      citeste(&n, &tabel);

      afiseaza(n, tabel);

      sortareg(tabel, n, sizeof(persoana), fcg1);
      printf("\n Lista in ordine alfabetica:");
```

```
        afiseaza(n, tabel);

        sortareg(tabel, n, sizeof(persoana), fcg2);
        printf("\n Lista in ordinea varstei:");
        afiseaza(n, tabel);

        sortareg(tab_int, 6, sizeof(int), fcg_int);
    }
```

Observatii.

1. *Conversiile de tip de la void **

In primul rand se declara `fct_cmpg` ca pointer cu tipul “functie cu rezultatul de tip intreg si 2 parametrii de tip `void *`”. Functiile care realizeaza acest prototip (fcg1, fcg2, fcg_int) cunosc fiecare dintre ele ce tip au intr-adevar parametrii lor si in corpul lor realizeaza conversiile de tip necesare. De exemplu, in cazul functiei fcg_int, aceasta realizeaza comparatia a 2 intregi, prin diferenta valorilor lor. Valorile parametrilor lui fcg_int se obtin print expresii de forma `*(int *)a`. Intai se face conversia tipului pointerului a de la `void *` la `(int *)`, apoi se poate referi valoarea (intreaga) de la aceasta adresa.

2. *Functia de biblioteca qsort*

Biblioteca C stdlib.h ofera de exemplu functia generica de sortare qsort(), care este de tipul:

```
void qsort(void *base, size_t nelem, size_t width,
           int (fcmp)(const void *, const void *));
```

Se poate utiliza in program, in forma:

```
qsort(tabel, n, sizeof(persoana), fcg1);
```