Laborator 11. Rezolvări și explicații

Se consideră următoarele 2 structuri care definesc o persoană:

```
struct persoana {
char nume[100];
char adresa[100];
int varsta;
}
```
(1)

și

```
struct persoana {
char *nume;
char *adresa;
int varsta;
}
```
(2)

Pentru fiecare dintre aceste 2 reprezentări ale unei persoane, să se scrie un program C folosind funcții care realizează următoarele facilități:

- Citirea unui număr de *n* persoane de la tastatură
- Adăugarea unei persoane noi pe o poziție *m* în șirul de persoane
- Regăsirea unei persoane de pe poziția *m* din șir și afișarea persoanei regăsite
- Extragerea unei persoane de pe poziția *m* din șir și afișarea persoanei extrase. Persoana extrasă nu va mai face parte din șir
- Regăsirea unei persoane în șir dacă se furnizează numele persoanei.

Pentru fiecare dintre aceste facilități se va scrie o funcție corespunzătoare. Programul poate conține un meniu care să invite utilizatorul să selecteze una dintre aceste funcționalități – cu excepția citirii persoanelor, care se realizează exclusiv la începutul programului.

Rezolvare: Vom oferi un model de soluționare a problemei descrisă în enunț, folosind structura **(2)**. Mai întâi declarăm structura cu numele *persoana* alcătuită din câmpurile *nume*, *prenume* și *varsta*. Sintaxa de mai jos combină cuvintele cheie *struct* și *typedef* într-o singură declarație pentru a da un nume tipului de dată introdus.

```
typedef struct persoana{
     char *nume;
     char *adresa;
     int varsta;
} TPersoana;
```

Tipul `TPersoana` va reprezenta un nou tip de dată, iar acest tip de dată va fi folosit pentru a defini variabile de structură în mod direct. De exemplu, dacă dorim să definim o variabilă `p` de tipul persoană atunci definiția este `TPersoana *p;`

1. **Citirea unui număr de *n* persoane de la tastatură**

   Pentru a citi informațiile despre *n* persoane, vom crea o funcție numită *citeste_persoane()* cu două argumente: șirul de persoane și numărul de persoane. Parametrii formali *persoane[]* și *n* din definiția funcției sunt parametri in-out, motiv pentru care trebuie definiți ca și pointeri. Un parametru in-out este acea variabilă care se modifică în interiorul unei funcții, iar modificarea făcută local trebuie să se reflecte și în exterior, când se revine în *main()*.

```
void citeste_persoane(TPersoana* persoane[], int *n)
{
    int i;
    printf("n="); scanf("%d", n); // citim numărul de persoane
    for(i=0; i<*n; i++)
    {
        printf("Introduceti informatiile persoanei %d\n", i);
        persoane[i] = citeste_persoana();
    }
}
```

   În funcția de citire a persoanelor apelăm o funcție auxiliară numită *citeste_persoana()*, ce are rolul de a citi de la tastatură informațiile (nume, adresă și vârstă) legate despre o persoană *i* din sirul de persoane.

```
TPersoana* citeste_persoana()
{
    char nume[100], adresa[100];
    int varsta;
    printf("Nume="); scanf("%s", nume);
    printf("Adresa="); scanf("%s", adresa);
    printf("Varsta="); scanf("%d", &varsta);

    return creeaza_persoana(nume, adresa, varsta);
 }
```

   Această funcție returnează rezultatul apelului unei alte funcții auxiliare numită *creeaza_persoana()* pentru a crea în mod dinamic, în memorie o nouă variabilă de tipul persoană (`TPersoana`) folosind informațiile citite și de a o stoca în șirul de persoane, pe poziția *i*. Pentru a aloca dinamic o memorie inițială pentru membrii structurii folosim funcția predefinită *malloc()* din librăria standard <stdlib.h> a limbajului C.

```
TPersoana* creeaza_persoana(char nume[], char adresa[], int varsta)
{
    TPersoana *p = (TPersoana *)malloc(sizeof(TPersoana));
    p->nume = (char *)malloc(100*sizeof(char));
    p->adresa = (char *)malloc(100*sizeof(char));
    p->varsta = varsta;
    strcpy(p->nume, nume);
    strcpy(p->adresa, adresa);
    return p;}
```

Sintaxa funcției ***malloc()*** este:

```
void *malloc(size_t size);
```

unde paramentrul **`size`** reprezintă dimensiunea blocului de memorie, în octeți.
Funcția *malloc()* returnează un pointer la memoria alocată.

Observație: Membrii structurii `persoana` se manipulează prin `->` deoarece variabila de tipul persoană este un pointer deci `TPersoana *p`

De asemenea, în procesul de creare a unei persoane folosim funcția predefinită *strcpy()* din librăria standard <stdlib.h> pentru a copia valoarea variabilelor nume și adresă în membrii (structurii) corespunzători. Funcția *strcpy()* se folosește pentru a copia șiruri de caractere dintr-o variabila sursă într-o variabilă destinație.

Sintaxa funcției ***strcpy()*** este:

```
char* strcpy (char* destination, const char* source);
```

unde parametrul `dest` este pointerul ce indică înspre șirul destinație unde va fi copiat conținutul, iar parametrul `src` este șirul de caractere ce se va copia.
Funcția *strcpy()* returnează un pointer la șirul destinație rezultat.

2. **Adăugarea unei persoane noi pe o poziție *m* în șirul de persoane**

Pentru a adăuga o persoană pe o anumită poziție *m* în șir, definim o funcție numită *adauga_persoana()* care ia ca și parametri șirul de persoane, numărul *n* de persoane stocate în șir, o persoană *p* ce dorim să o adăugăm în șir, precum și un număr *m* ce semnifică poziția în șir unde dorim să adăugăm persoana *p*. Tipul de return al funcției este *int*, deoarece funcția returnează 0, în cazul în care adăugarea persoanei *p* în șir a eșuat, respectiv 1, în cazul adăugarea s-a realizat cu succes.

```
int adauga_persoana(TPersoana *persoane[], int *n, TPersoana *p, int m)
{
    int i;
    if( m<0 || m>*n ) // m trebuie sa fie o pozitie in sir
        return 0; // encodare pentru false - în caz de eroare
    for(i=(*n)-1; i>=m; i--)
        persoane[i+1]=persoane[i]; //decalare persoane cu o poziție
                                   //la dreapta
    persoane[m] = p; // stocarea persoanei pe pozitia m
       (*n)++; // dimensiunea sirului creste cu o unitate
 return 1; //encodare pentru true - în caz de adăugare cu succes
}
```

3. **Regăsirea unei persoane de pe poziția *m* din șir și afișarea persoanei regăsite**

Pentru a regăsi persoana de pe o anumită poziție *m* din șir, definim o funcție numită *gasestePersoana_dupaPozitie()* care ia ca și parametri șirul de persoane, dimensiunea *n* a acestui șir, precum și poziția *m* de regăsire a persoanei. Dacă indexul *m* are o valoare în afara

limitelor șirului, atunci funcția returnează NULL, altfel returnează un pointer la persoana de pe poziția *m* din șir.

```
TPersoana* gasestePersoana_dupaPozitie(TPersoana *persoane[], int *n, int m)
{
int i;
if( m<0 || m>*n ) // m trebuie sa fie o pozitie in sir
      return NULL;
 else
          return persoane[m];
 }
```

Pentru a afișa persoana regăsită creăm o funcție de afișare numită *afiseaza_persoana()* care o vom apela în *main()* pentru a afișa informațiile (nume, adresă și vârstă) persoanei regăsite.

```
void afiseaza_persoana(TPersoana *p)
{
    if (p==NULL)
        return;
    printf("Nume=%s\n",p->nume);
    printf("Adresa=%s\n",p->adresa);
    printf("Varsta=%d\n",p->varsta);
    printf("\n");
}
```

4. **Extragerea unei persoane de pe poziția *m* din șir și afișarea persoanei extrase. Persoana extrasă nu va mai face parte din șir**

Pentru a extrage persoana de pe o anumită poziție *m* din șir, definim o funcție cu numele *extrage_persoana()* care ia ca și parametri: șirul de persoane, dimensiunea *n* a acestui șir, precum și poziția *m*. Funcția returnează NULL, dacă indexul *m* are o valoare în afara limitelor șirului. Funcția returnează persoana *p* de pe poziția *m* din șir în cazul în care extragerea a decurs cu succes. Pentru ca persoana extrasă să nu mai facă parte din șir, trebuie să parcurgem șirul decalând persoanele cu o poziția spre stânga. Totodată, trebuie să dezalocăm din memorie persoana extrasă.

```
TPersoana* extrage_persoana(TPersoana *persoane[], int *n, int m)
{
    int i ;
    TPersoana *p;
    if( m<0 || m>*n ) // m trebuie sa fie o pozitie in sir
    {
      return NULL;
    }
    p = persoane[m];
    for(i=m; i<*n; i++)
        persoane[i]=persoane[i+1]; //decalare persoane cu o poziție
                                   //la stanga
   (*n)--; //decrementăm n cu o unitate
   return p;
}
```

Pentru a dezaloca din memorie persoana extrasă, creăm o funcție numită *dezaloca_persoana()* ce folosește funcția predefinită *free()* din librăria standard <stdlib.h> a limbajului C. Funcția *free()* dezalocă memoria anterior alocată de *malloc()*.

Sintaxa funcției ***free()*** este:
`void free(void *ptr);`
parametrul `ptr` este un pointer la blocul de memorie (anterior alocat) ce trebuie dezalocat

```
void dezaloca_persoana(TPersoana *p)
{
   free(p->nume); // dezalocăm membrii pe rând
   free(p->adresa);
   free(p); // dezalocăm persoana din memorie
}
```

5. **Regăsirea unei persoane în șir dacă se furnizează numele persoanei**

Pentru a regăsi o persoană după nume, definim funcția *gasestePersoana_dupaNume()* care deține ca parametri șirul de persoane, numărul *n* de persoane din șir, precum și un șir de caractere *nume[]* ce stochează numele persoanei de regăsit. Parcurgem șirul de persoane cu ajutorul unui index *i*. Numele fiecărei persoane din șir este comparat cu valoarea variabilei *nume[]* transmisă ca parametru. Compararea lexicografică a numelor de persoane este realizată cu ajutorul funcției predefinite *strcmp()* din librăria standard <string.h> a limbajului C.

Sintaxa funcției ***strcmp()*** este:
`int strcmp(const char *str1, const char *str2);`
unde parametrii `str1` și `str2` reprezintă șirurile de caractere ce trebuie comparate
Funcția *strcmp()* returnează:

- O valoare < 0 ce indică faptul că primul șir este mai mic decât al doilea
- O valoare > 0 ce indică faptul că al doilea șir este mai mic decât primul
- O valoare = 0 ce indică faptul că cele două șiruri sunt egale.

```
TPersoana* gasestePersoana_dupaNume(TPersoana *persoane[], int *n, char nume[])
{
    int i;
    for(i=0; i<n; i++)
    {
        if(strcmp(persoane[i]->nume, nume)== 0)
            return persoane[i];
    }
    return NULL;
}
```

În cazul în care cele două nume coincid, funcția returnează persoana de pe poziția *i* din șir.

În *main()*, testăm logica programului cu ajutorul unui meniu inteligibil pentru utilizator. Meniul are mai multe opțiuni, în conformitate cu cerințele problemei. Fiecare opțiune înglobează câte una dintre funcționalitățile programului: citirea/afișarea persoanelor, adăugarea, regăsirea, extragerea unui persoane din șir.

```
int main()
{
    int i,n,m,optiune=1;
    char nume[100];
    TPersoana *persoane[100], *p;

    printf("===Meniu principal===\n");
    printf("1. Citirea unui numar de n pers de la tastatura\n");
    printf("2. Adaugarea unei persoane noi pe o pozitie m \n");
    printf("3. Regasirea unei persoane de pe pozitia m din sir si afisarea
            persoanei regasite\n");
    printf("4. Extragerea unei persoane de pe pozitia m din sir si afisarea
            persoanei extrase\n");
    printf("5. Regasirea unei persoane in sir daca se furnizeaza numele
            persoanei\n");
    printf("6. Afisare sir de persoane\n");
    printf("7. Iesire din program\n======================");

    while(optiune!=7)
    {
        printf("\nIntroduceti optiunea dvs=");
        scanf("%d",&optiune);
        switch(optiune)
        {
        case 1:     citeste_persoane(persoane, &n);
                    break;
        case 2:     printf("Introduceti informatiile despre persoana care doriti sa o
                           adaugati\n");
                    p=citeste_persoana();
                    printf("Introduceti pozitia: \n");
                    printf("m="); scanf("%d", &m);
                    if (adauga_persoana(persoane, &n, p, m))
                        printf("Persoana adaugata cu succes pe pozitia (%d) din sir\n", m);
                    else
                        printf("Nu se poate adauga persoana pe pozitia (%d) in sir\n", m);
                    break;
        case 3:     printf("Introduceti pozitia persoanei de gasit\n");
                    printf("m="); scanf("%d", &m);
                    if (gasestePersoana_dupaPozitie(persoane, &n, m))
                    {
                        printf("Persoana gasita cu succes pe pozitia (%d) din sir\n", m);
                        afiseaza_persoana(gasestePersoana_dupaPozitie(persoane, &n, m));
                    }
                    else
                        printf("Nu se poate gasi persoana pe pozitia (%d) in sir\n", m);
                    break;
        case 4:     printf("Introduceti pozitia persoanei de extras\n");
                    printf("m="); scanf("%d", &m);
                    if (p=extrage_persoana(persoane, &n, m))
                    {
                         printf("Persoana extrasa cu succes pe pozitia (%d) din sir\n", m);
```

```
                        afiseaza_persoana(p);
                        dezaloca_persoana(p);
                    }
                    else
                        printf("Nu se poate extrage persoana\n", m);
                    break;
        case 5:     printf("Introduceti numele persoanei de gasit\n");
                    printf("nume="); scanf("%s", nume);
                    if (p=gasestePersoana_dupaNume(persoane, &n, nume))
                    {
                        printf("Persoana cu numele (%s) e gasita in sir\n", nume);
                        afiseaza_persoana(p);
                    }
                    else
                        printf("Nu poate fi gasita persoana cu numele (%s)\n", nume);
                    break;
        case 6:     printf("Sirul de persoane contine:\n");
                    for(i=0;i<n;i++)
                        afiseaza_persoana(persoane[i]);
                    break;
        case 7: break;
        default:    printf("Optiune gresita!\n");
        }//end switch
    }//end while
    return 0;
}
```