1 Laborator 1: Recapitulare C

1.1 Obiective

Scopul acestui laborator este de a recapitula principalele elemente de programare în C pe care le vom folosi la implementarea noţiunilor studiate la disciplina Structuri de Date şi Algoritmi.

1.2 Noțiuni teoretice

Principalele elemente de care vom avea nevoie pe parcursul acestui semestru pentru implementarea problemelor de la laborator sunt următoarele:

1. Pointeri în C
2. Definirea şi manipularea elementelor de tip struct
3. Lucrul cu fişiere

1.2.1 Pointeri si functii în C

Tipul pointer

Un pointer este o variabilă care are ca valori adrese. Dacă pointerul p are ca valoare adresa de memorie a variabilei x, se spune că p pointează spre x. Un pointer este legat de un tip. Dacă x este de tipul int, pointerul p este legat de tipul int. Declararea unui pointer se face la fel ca declararea unei variabile, cu deosebirea că numele pointerului este precedat de caracterul *:

```
int *p;
```

Adresa unei variabile se obţine cu ajutorul operatorului unar &, numite operator de referenţiere. De exemplu fiind date declaraţiile:

```
int x;
int *p;
```

Atunci $p = \&x$; are ca efect atribuirea ca valoare pentru p a adresei variabilei x. Furnizarea valorii din zona de memorie a cărei adresă este conţinută în p se face cu ajutorul operatorului unar *****, numit operator de dereferenţiere.

Exemplu:

- instrucţiunea x=y este echivalentă cu una din secvenţele: $p = \&x;$ $*p = y;$ sau $p = \&y;$ $x = *p\ ;$
- instrucţiunea x++ este echivalentă cu secvenţa:

```
p=&x;
(*p)++;
```

Legătura dintre pointeri și tablouri

Numele unui tablou are drept valoare adresa primului său element. Ca urmare, se spune că numele unui tablou este un pointer constant, neputând fi modificat în timpul execuţiei.

Exemplu:

```
int tab[100];
int *p;
int x;
p=t;  /* p primeste ca valoare adresa elementului tab[0] */
```

În acest exemplu, atribuirea x=tab[0] este echivalentă cu x=*p;

1.2.1.1 Operatii asupra pointerilor

Asupra pointerilor sunt permise următoarele operaţii:

1. Incrementare/decrementare cu 1. În acest caz valoarea pointerului este incrementată/decrementată cu numărul de octeţi necesari pentru a păstra o dată de tipul de care este legat pointerul.

```
int tab[100];
int *p;
p=&tab[10];
p++;   /* Valoarea lui p este incrementata cu 1, avand adresa elementului  tab[11]*/
```

2. Adunarea şi scăderea unui întreg dintr-un pointer. Operaţia **p+n** sau **p-n** are drept efect creşterea, respectiv scăderea din valoarea p a **n*numărul de octeţi** necesari pentru a păstra o dată de tipul de care este legat pointerul. Pentru exemplul de mai sus, dacă x este de tipul int, atunci: x=tab[i]; este echivalentă cu: x=*(tab+i);

3. Diferenţa a doi pointeri. Dacă 2 pointeri p şi q pointează spre elementele i şi j ale aceluiaşi tablou (j>i), adică p=&tab[i] şi q=&tab[j], atunci q-p = (j –i).

4. Compararea a doi pointeri Doi pointeri care pointează spre elementele aceluiaşi tablou pot fi comparaţi folosind operatorii de relaţie şi de egalitate: $<; <=; >; >=; ==; !=$

Alocarea dinamica a memoriei

Alocarea memoriei pentru variabilele globale şi statice este statică, adică alocarea rămâne până la terminarea programului. Alocarea memoriei pentru variabilele automatice este dinamică, în sensul că stiva este "curăţată" la terminarea funcţiei. Memoria heap este o zonă de memorie dinamică, specială, distinctă de stivă. Ea poate fi gestionată prin funcţii, care au prototipurile în fişierul alloc.h şi stdlib.h Alocarea unei zone de memorie heap se poate realiza cu ajutorul funcţiilor de prototip:

```
void *malloc (unsigned n);
void *calloc(unsigned nr_elem, unsigned dim);
```

Funcţia malloc alocă în heap o zonă contiguă de n octeţi, iar funcţia calloc o zonă contiguă de nr_elem * dim în octeţi. Funcţiile returnează:

- în caz de succes, adresa de început a zonei alocate (pointerul fiind de tip void, este necesară conversia spre tipul dorit);
- în caz de insucces, returnează zero (pointerul NULL);

Eliberarea unei zone alocate cu malloc sau calloc se face cu ajutorul funcţiei de prototip:

```
void free (void *p);
```

1.2.2 Definirea şi manipularea elementelor de tip struct

O structură conţine mai multe componente de tipuri diferite (predefinite sau definite de utilizator), grupate conform unei ierarhii.

Exemple echivalente pentru declararea unei structuri:

Accesul la componentele unei structuri se poate face prin procedeul de calificare: identificator_variabilă.identificator_câmp; De exemplu: x.re, x.im.

Procedeul de calificare pătrunde din aproape în aproape în ierarhia structură. În limbajul C, transmiterea ca parametru a unei structuri la apelul unei funcţii, se face numai prin adresa variabilei de tip structură. De exemplu:

```
void f (struct NR_COMPLEX *p, ...);
```

Apelul se va face prin:

```
f(&x, ...)
```

Varianta 1:	Varianta 2	Varianta 3	Varianta 4
`struct NR_COMPLEX { float re; float im; } x,y , a, b;`	`struct NR_COMPLEX{ float re; float im; }; struct NR_COMPLEX x,y,a,b;`	`struct { float re; float im; } x,y, a, b;`	`typedef struct { float re; float im; } NR_COMPLEX; NR_COMPLEX x, y;`

Tabela 1.1: Definirea unei structuri

"r" - deschidere în citire (read);	"w" - deschidere în scriere (write);
"a" - deschidere pentru adăugare (append);	"r+" - deschidere în citire/scriere (modificare);
"rb" - citire binară;	"wb" - scriere binară;
"r+b" - citire/scriere binară;	"w+b" - citire/scriere binară;
"ab" – adăugare de înregistrări în modul binar.	

Tabela 1.2: Moduri deschidere fisier

În funcţie, selectarea unui câmp se face astfel:

```
(*p).re
(*p).im
```

sau înlocuind (*p). prin p-> , ca mai jos:

```
p->re
p->im
```

1.2.3 Lucrul cu fișiere

Pentru implementarea problemelor care folosesc citire şi scriere în fişiere text sau binare la laboratorul de SDA vom folosi structuri speciale de tip FILE. Principalele operaţii care se pot efectua asupra fişierelor la acest nivel sunt: crearea, deschiderea, citirea/scrierea unui caracter sau a unui şir de caractere, citirea/scrierea binară a unui număr de articole, poziţionarea într-un fişier, închiderea unui fişier, vidarea zonei tampon a unui fişier

1.2.3.1 Principalele operatii cu structuri de tip FILE

1. Declararea unei variabile de tip fisier:

   ```
   FILE *f; //declaram o variabila f de tip fisier
   ```

   Tipul FILE şi toate prototipurile funcţiilor de prelucrare se găsesc în fişierul **stdio.h**.

2. **Deschiderea unui fişier:** Deschiderea unui fişier existent, precum şi crearea unui fişier nou se face cu ajutorul funcţiei **fopen**, care are următorul prototip:

   ```
   FILE *fopen(const char *cale_nume, const char *mod);
   ```

   unde:

   - cale_nume – este un pointer spre un şir de caractere care defineşte calea de nume a fişierului;
   - mod – este un pointer spre un şir de caractere care defineşte modul de prelucrare a fişierului deschis, după cum este descris in Tabelul 1.2. Conţinutul unui fişier existent deschis în scriere "w" , va fi şters, el considerându-se deschis în creare. Dacă fişierul este deschis în modul "a", se vor putea adăuga noi înregistrări după ultima înregistrare existentă în fişier. Un fişier inexistent deschis în modul "w" sau "a" va fi creat. Funcţia fopen returnează un pointer spre tipul FILE în caz de succes sau pointerul nul în caz de eroare.

3. Citirea / scrierea cu format folosind functiile **fscanf** si **fprintf**: Citirea/scrierea cu format se poate face cu ajutorul funcţiilor fscanf/fprintf, similare cu funcţiile sscanf/sprintf, deosebirea constând în faptul că în cadrul funcţiilor sscanf/sprintf se precizează ca prim parametru pointerul zonei unde se păstrează şirul

de caractere, iar în cadrul funcţiilor fscanf/fprintf se precizează ca prim parametru pointerul spre tipul FILE, aşa cum reiese din prototipurile lor:

```
int fscanf(FILE *pf, const char *format,[adresa, ...]);
int fprintf(FILE *pf, const char *format,[adresa, ...]);
```

Funcţia fscanf returnează numărul de câmpuri citite corect; la întâlnirea sfârşitului de fişier funcţia returnează valoarea EOF. Funcţia fprintf returnează numărul caracterelor scrise în fişier sau –1 în caz de eroare.

4. **Închiderea unui fişier** se realizează cu ajutorul funcţiei fclose, care are prototipul:

   ```
   int fclose(FILE *pf);
   ```

   unde pf este pointerul spre tipul FILE returnat de fopen. Funcţia returnează 0 în caz de succes şi –1 în caz de eroare.

1.3 Exemple de cod

1.3.1 Lucrul cu pointeri

Exemplul 1

```
/* Programul exemplifica folosirea operatiilor asupra pointerilor */
 #include <stdio.h>

 void Max_min1(int n,int a[],int *max,int* min)
 {
        int i;
        *max=a[0];
        *min=a[0];
         for (i=1;i<n;i++)
         {
                 if (a[i]>*max) *max=a[i];
                 else if (a[i]< *min) *min=a[i];
         }
  }

  void Max_min2(int n,int *a,int *max,int *min)
  {
         int i;
         *max=*a;
         *min=*a;
         for (i=1;i<n;i++)
         {
                 if (*(a+i)>*max) *max=*(a+i);
                 else if (*(a+i)< *min) *min=*(a+i);
         }
  }
  void main(void)
  {
          int i,n,maxim,minim;
          int x[100];
          /* Introducerea datelor */
          printf("\nNumarul elementelor tabloului n=");
          scanf("%d",&n);
          for(i=0;i<n;i++)
                {
                  printf("\nx[%d]=",i);
                  scanf("%d",&x[i]);
                }
          /* Apelul primei proceduri */
          Max_min1(n,x,&maxim,&minim);
          printf("\nLa apelul functiei Max_min1 rezulta:  maximul=%d  minimul=%d\n",maxim,minim);
          /* Apelul celei de a doua proceduri */
```

```
        Max_min2(n,x,&maxim,&minim);
        printf("\nLa apelul functiei Max_min2 rezulta:  maximul=%d   minimul=%d\n",maxim,minim);
        printf("\nApasati o tasta!\n");
        getch();
      }
```

Exemplu 2

```
#include <stdio.h>
void main(void)
{
   char *str1,*str2;
   /* Aloca memorie pentru primul sir de caractere */
   if ((str1 = (char *) malloc(100)) == NULL)
   {
      printf("Memorie insuficienta\n");
      exit(1);
   }
   printf("\nIntroduceti primul sir de caractere terminat cu ENTER\n");
   gets(str1);
   printf("\nSirul de caractere introdus este\n  %s\n",str1);

   /* Aloca memorie pentru al doilea sir de caractere */
   if ((str2 = (char *) calloc(100,sizeof(char))) == NULL)
   {
      printf("Memorie insuficienta\n");
      exit(2);
   }
   printf("\nIntroduceti al doilea sir de caractere terminat cu  ENTER\n");
   gets(str2);
   printf("\nSirul de caractere introdus este\n  %s\n",str2);
   printf("\nApasati o tasta\n");
   getch();
   /* Eliberarea memoriei */
   free(str1);
   free(str2);
}
```

1.3.2 Structuri

Exemplul 3

```
#include <stdio.h>
typedef struct {float re,im;}COMPLEX;

void aduna(COMPLEX *a,COMPLEX *b,COMPLEX *c)
 /* transmiterea parametrilor prin pointeri */
{
   c->re=a->re+b->re;
   c->im=a->im+b->im;
};

void scade(COMPLEX *a,COMPLEX *b,COMPLEX *c)
 /* transmiterea parametrilor si a rezultatului prin pointeri */
{
    c->re=a->re-b->re;
    c->im=a->im-b->im;
};

void impartire(COMPLEX *a,COMPLEX *b,COMPLEX *c)
  /*transmiterea parametrilor prin pointeri */
{
   float x;
   x=b->re*b->re+b->im*b->im;
   if (x==0) {
      printf("\nImpartire la zero!\n");
      exit(1);
```

```
   }
   else{
        c->re=(a->re*b->re+a->im*b->im)/x;
          c->im=(a->im*b->re-a->re*b->im)/x;
   }
};

void main(void)
 /* Operatii asupra numerelor complexe */
{
  COMPLEX a,b,c;
  char ch,op;
  ch='D';
  while ((ch=='D')|| (ch=='d'))
  {
     printf("\nIntroduceti primul numar complex\n");
     printf("a.re=");scanf("%f",&a.re);
     printf("a.im=");scanf("%f",&a.im);
     printf("\nIntroduceti al doilea numar complex\n");
     printf("b.re=");scanf("%f",&b.re);
     printf("b.im=");scanf("%f",&b.im);
     aduna(&a,&b,&c);
     printf("\n(%f+j*%f)+(%f+j*%f)=%f+j*%f\n",
               a.re,a.im,b.re,b.im,c.re,c.im);
     scade(&a,&b,&c);
     printf("\n(%f+j*%f)-(%f+j*%f)=%f+j*%f\n",
               a.re,a.im,b.re,b.im,c.re,c.im);

     impartire(&a,&b,&c);
     printf("\n(%f+j*%f)+(%f+j*%f)=%f+j*%f\n",
               a.re,a.im,b.re,b.im,c.re,c.im);
     printf("\nCONTINUATI?DA=D/d,Nu=alt caracter " );
     scanf("%*c%c",&ch);
    }
}
```

1.3.3 Lucrul cu fisiere

Exemplul 4

```
// Exemplu de scriere a unor date intr-un fisier text si de citire a datelor

#include <stdio.h>

int main(int argc, char* argv[])
{
    char nume[256];
    int varsta;
    /* se creeza un fisier text si se deschide pentru scriere*/
    FILE *f = fopen("test.txt", "w");
    if (!f) {
                        printf("Nu se poate deschide fisierul");
                        exit(0);
                }
                //se scriu date in fisier
    fprintf(f, "Ion 25 \n");
    fclose(f);

    /* se creaza un alt fisier si se deschide fisierul pentru citire */
    FILE *f2 = fopen("test.txt", "r");

    if (fscanf(f2, "%s %d", nume, &varsta) !=2)
        printf("Nu s-au putut citi numele si varsta");
        else
      printf("[S-a citit] Nume: %s, Varsta %d\n", nume, varsta);
    fclose(f2);
}
```

1.4 Mersul lucrării

1. Să se parcurgă exemplele de cod, să se implementeze şi să se testeze.
2. Pentru exemplul 3 – operatii cu numere complexe, sa se implementeze functia de produs a doua numere complexe.
3. Să se implementeze problemele propuse mai jos.

1.4.1 Probleme Propuse

1. Sa se citeasca dintr-un fisier text date despre studenti: in fisier se da pe prima linie numarul de studenti (n), apoi pe urmatoarele n linii numele studentului, prenumele, varsta urmate de notele la primele 3 laboratoare la SDA. Exemplu de fisier:

   ```
   5
   Pop Ana 21 10 9 8
   Costea Andrei 20 7 8 5
   Butnaru Dan 21 9 5 7
   Chis Maria 20 8 10 9
   Vlaicu Robert 21 10 9 5
   ```

   - Sa se creeze o structura care sa permita stocarea studentilor.
   - Sa se aloce dinamic un sir de studenti.
   - Sa se afiseze sirul de studenti citit.
   - Sa se parcurga sirul de studenti si sa se calculeze media celor 3 note ale fiecarui student. Sa se afiseze apoi sirul de studenti si media fiecaruia. Sugestie: creati un camp medie in structura Student de la punctul 2.
   - Pentru fiecare student sa se incrementeze toate notele cu 1. Sa se afiseze studentii si notele lor modificate.

2. Se citesc de la tastatura 2 numere, CAPACITY si SIZE. Sa se aloce dinamic un sir care poata sa contina CAPACITY elemente reale. Sa se scrie o functie care adauga un element in sir, pe **ultima** pozitie a sirului. Folosind functia creata sa se insereze in sir SIZE numere. Ce se intampla daca $CAPACITY \geq SIZE$ si daca $CAPACITY < SIZE$. Afisati sirul de numere dupa fiecare operatie de inserare.
3. Se citesc de la tastatura 2 numere, CAPACITY şi SIZE. Sa se aloce dinamic un sir de CAPACITY elemente reale. Sa se scrie o functie care adauga un element in sir, pe **prima** pozitie a sirului (cea cu indicele 0). Folosind functia creata sa se insereze in sir SIZE numere. Ce se intampla daca $CAPACITY \geq SIZE$ si daca $CAPACITY < SIZE$. Afisati sirul de numere dupa fiecare inserare.