LUCRAREA DE LABORATOR NR. 11

Tema: Funcţii în limbajul C/C++

Obiective:

- să definească noțiunea de funcţie ca unitate structurală a programului;
- să aplice noțiunile de definire şi declarare a unei funcţii;
- să aplice diferite tehnici de apel a funcţiilor;
- să descrie funcţiile din biblioteca standard a limbajului;
- să propună programe utilizând funcţii;

11.1. Sarcină pentru soluţionare

De elaborat un program, care soluționează sarcina pentru lucrarea de laborator nr.10 cu astfel de condiţii suplimentare:

- dimensiunea matricei se introduce la executarea programului;
- însăşi soluționarea sarcinii trebuie să fie organizată în formă de funcţie, căreia i se transmite matricea şi dimensiunea ei.

11.2. Exemplu de soluţionare a sarcinii

Rezolvarea se aduce cu trimitere la lucrarea nr.10.

Elaborarea algoritmului de soluţionare

Însuși algoritmul de soluţionare poate fi același, ca şi în lucrarea nr.10. Dar conform condițiilor acestei lucrări dimensiunea matricei din timp nu este cunoscută.

Pentru elementul, care stă pe diagonala principală, indicii satisfac condiția: *i==j*, pe diagonala secundară – *i==D-j-1*. Prin urmare, pentru jumătatea de sus, condiţiile de nimerire în zona nenulă: $i < j < D-i-1$, iar pentru cea de jos: $D-i-1 < j < i$. Sau, în formă generală: $min(i,D-i-1) < j < max(i,D-i-1)$. În schema algoritmului (a se vedea Figura 11.1) este reflectată fragmentarea programului în două funcţii: funcția principală – *main()*, care execută alocarea memoriei pentru matrice şi afișează rezultatele la ecran, cât şi funcţia *umplere()*, care îndeplinește completarea matricei după condiţiile indicate.

În program se verifică mărimea *D*, care este introdusă de utilizator. Hotarul de jos pentru mărimea $D = 1$, întrucât matricea de dimensiunea zero sau număr negativ, pur şi simplu, nu are sens. Hotarul de sus = 20, întrucât o matrice de dimensiune mare nu poate fi prezentată integral la ecran.

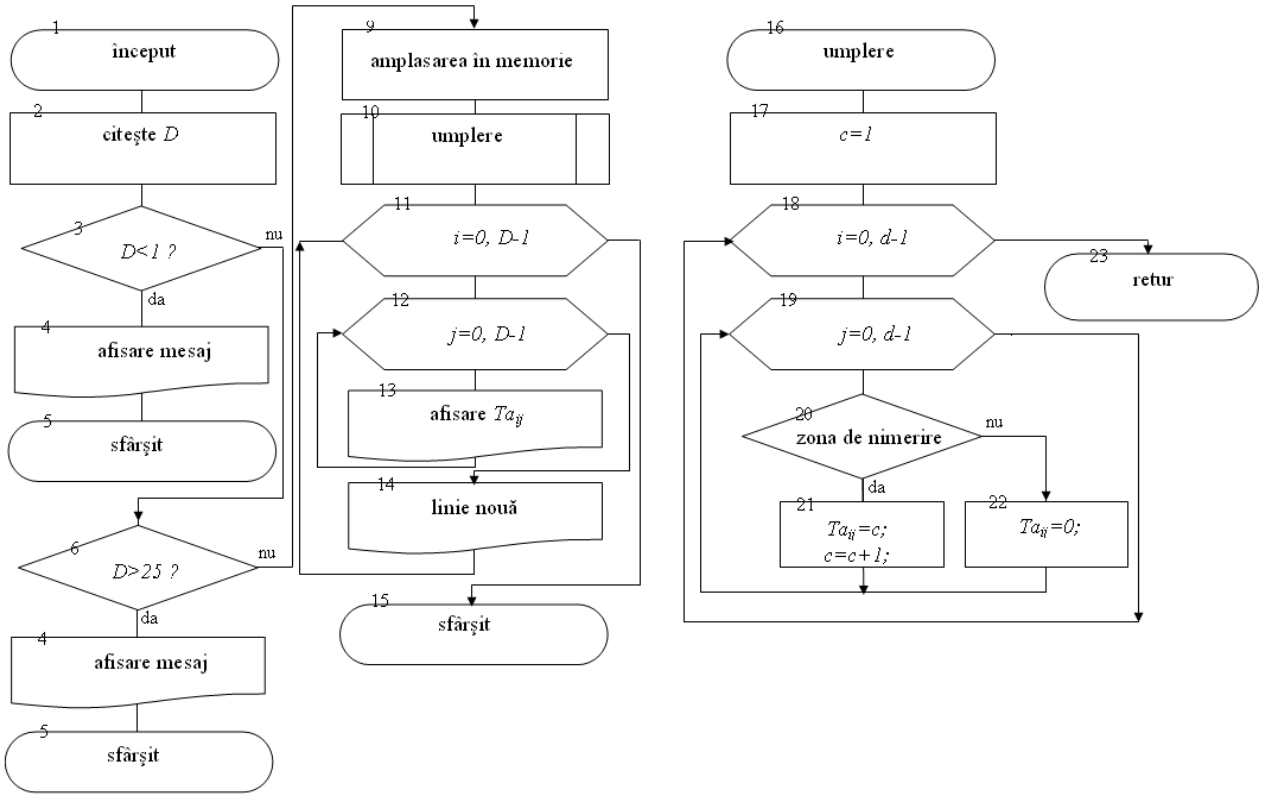

Figura 11.1. Schema bloc

Prezentarea matricei în memorie

Dacă în lucrarea nr.10 a fost posibilă prezentarea matricei în program ca un tablou bidimensional utilizând indicii de rând (*i*) şi coloană (*j*) (prezentarea naturală a matricei), atunci în condiţiile de amplasare dinamică în memorie prezentarea ei nu este atât de simplă. Sunt posibile mai multe variante de amplasare în memorie şi prezentarea matricei în program. În toate variantele posibile este evident că volumul total de memorie pentru amplasarea datelor matricei trebuie să fie D^2 elemente de tipul *int*.

Varianta nr. 1 este prezentată în Figura 11.2.

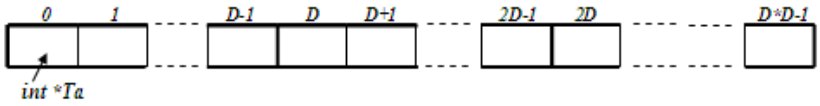

Figura 11.2. Amplasarea în memorie a unui tablou bidimensional (varianta nr.1)

Pentru datele matricei, se alocă volumul necesar de memorie. În program se declară obligator pointeri pentru începutul zonei de memorie. Tipul acestui pointer – *int**. Astfel, matricea este un tablou unidimensional. Pentru a calcula indicele elementului curent (*N*) într-un tablou unidimensional conform numărului de rând (*i*) şi coloană (*j*), se va axecuta calculul: *N=i*D+j*.

Varianta nr. 2 este prezentată în Figura 11.3.

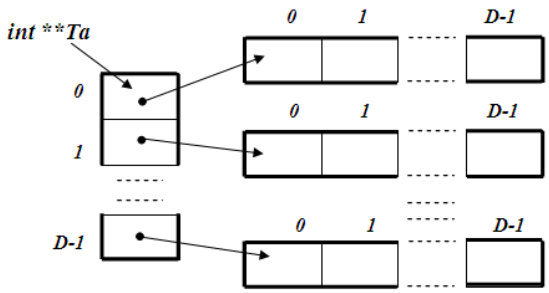

Figura 11.3. Amplasarea în memorie a unui tablou de pointeri (varianta nr.2)

Această variantă se deosebește de cea precedentă prin aceea că, pentru fiecare rând al matricei, memoria se alocă separat. (*D* zone de memorie câte *D* elemente în fiecare). Dispare necesitatea de a aloca zone adiacente de memorie pentru tablouri. Avem posibilitatea să ne adresăm către elementele matricei indicând doi indici. Alocarea memoriei (şi corespunzător – eliberarea) se execută în ciclu. Prima variantă asigură o economie de memorie, însă varianta 2 – posibilitatea de adresare „naturală" către elementele matricei. În continuare, vom prezenta realizarea algoritmului pentru variantele 1 şi 2.

11.3. Soluționarea algoritmului programului (varianta nr. 1)

Determinarea variabilelor

Programul este format din două funcţii – *main()* şi *umplere()*. Toate variabilele, cu care lucrează funcţiile sunt variabile locale sau parametri.

Funcţia *main()* începe cu declararea pointerului care are valoarea – adresa primului element din tablou, în care sunt amplasate datele matricei:

```
int *Ta;
```

Dimensiunea tabloului:

```
int D;
```

Pointer la elementul curent al tabloului:

```
int *Pr;
```

Contorul elementelor afișate:

```
int i;
```

Parametri pentru funcția *umplere()*.

Pointer la începutul tabloului liniar, în care sunt amplasate elementele matricei:

```
int *T;
```

Dimensiunea tabloului:

```
int d;
```

Variabile pentru funcția *umplere()*.

Pointer la elementul curent al tabloului la prelucrarea lui:

```
int *P;
```

Numărul liniei şi al coloanei:

```
int i, j;
```

Membrul curent în consecutivitatea liniară:

```
int c=1;
```

Elaborarea textului programului

Textul programului începe cu includerea fişierelor: *iostream.*

Se include descrierea funcţiilor *umplere()* şi se deschide funcția principală. În funcția principală se declară variabilele locale, apoi se afișează invitația şi se seteze mărimea dimensiunii tabloului *D*. Dimensiunea se compară cu hotarul de jos şi de sus, dacă iese din hotar, se afișează mesajul corespunzător şi programul se finisează (cu funcţia *exit()*). Dacă variabila *D* satisface condiţiile stabilite, valoarea ei este afișată la ecran.

Programul urmează cu alocarea memorie pentru amplasarea matricei. Pentru aceasta se utilizează operatorul *new*. Operatorul *new* returnează un pointer la zona de memorie alocată, valoarea acestui pointer se înscrie în variabila pointer *Ta.* Dacă valoarea returnată de operatorul *new* este un pointer vid, este un criteriu ce indică că memoria nu este alocată, în acest caz, se afișează un mesaj de eroare şi programul se finisează.

Instrucțiunea:

```
umplere(Ta,D);
```

se adresează către funcția de completare a matricei. Funcției i se transmite: pointerul la începutul tabloului şi dimensiunea tabloului.

După finisarea funcției *umplere(),* are loc revenirea în funcția principală şi matricea este afișată la ecran. La afișarea matricei se analizează ca un tablou liniar din D^2 elemente. Contorul *i* se schimbă de la *0* până la D^2*-1*. Adresarea către elementele matricei se face prin pointerul *Pr*. Acest pointer se setează la începutul tabloului, dar după fiecare operație se mărește cu 1. Contorul *i* se utilizează pentru determinarea momentului de ieșire din ciclu, ca şi pentru trecerea cursorului în rândul nou după afișarea a *D* elemente – pentru aceasta se verifică condiția: *i%D==D-1*.

Ultima acţiune în funcţiei *main()* – eliberarea memoriei în prealabil alocată cu ajutorul operatorului *delete*.

Funcţia *umplere()* primește parametrii: pointer la începutul tabloului şi dimensiunile matricei. Corpul funcției începe cu declararea variabilelor ei locale, variabila *c* primește valoarea de iniţiere la declarare.

În continuare, în funcţie se organizează ciclul plasat pentru selectarea liniilor şi coloanelor. Numărul liniei şi al coloanei se utilizează numai pentru controlul elementului, dacă nimerește în zona nulă sau nenulă. Adresarea către elementele matricei se realizează prin pointerul *P.* La ajustarea inițială a ciclului extern, acest pointer se stabilește la începutul tabloului şi se incrementează cu 1 la sfârșitul iterației ciclului intern.

Textul integral al programului este prezentat mai jos.

[*] Lab13_1.cpp

```cpp
#include <iostream>
using namespace std;

//declararea functiei de umplere
void umplere(int*, int);

int main(){
    int *Ta;
    int *Pr;
    int i;
    int D;

    cout<<"\n Seteaza dimeniunea tabloului"<<endl;
    cin>>D;
    //verificarea dimensiunii tabloului
    if(D<=0) {cout<<"\n Dimensiune foarte mica"; exit(0);}
    if(D>20) {cout<<"\n Dimensiune foarte mare"; exit(0);}
    cout<<"\n D="<<D;
    //alocarea memoriei dinamice
    if((Ta=new int[D*D])==NULL)
        {cout<<"\n Neajuns de memorie"; exit(0);        }
    //apelul functiei de umple
    umplere(Ta,D);
    cout<<"\nTabloul bidimensional setat"<<endl;
    for(Pr=Ta, i=0; i<D*D; Pr++, i++){
        cout<<"\t"<<*Pr;
        if(i%D==D-1) cout<<endl;
    }
    //eliberarea zonee de memorie
    delete Ta;
    return 0;
}

//definirea functiei umplere
void umplere(int *T, int d){
    int *P;
    int i,j;
    int c=1;

    for(i=0, P=T; i<d; i++)
        for(j=0; j<d; j++, P++)
            if((j<=max(i,d-i-1))&&(j>=min(i,d-i-1)))  *P=c++;
                else    *P=0;
}
```

Figura 11.4. Textul programului (varianta nr.1)

11.4. Soluționarea algoritmului programului (varianta nr. 2)

Elaborarea textului programului

Din textul programului se exclud toate verificările de corectitudine. Descrierea soluționării se face pentru instrucțiunile ce diferă de varianta 1.

Pointerul *Ta* în funcția *main()* este pointer la începutul tabloului de pointeri de tipul – *int***. Alocarea memoriei se face în două etape: la început se alocă memorie pentru un tablou din *D* pointeri, apoi în ciclu pentru elementele acestui tablou se alocă memorie pentru încă *D* pointeri, adresele tablourilor alocate se înscriu în elementele tabloului pointer. La afișarea matricei se utilizează *i* şi *j* – numărul liniei şi al coloanei. Adresarea către elementele matricei se face ca şi pentru elementele unui tablou bidimensional.

Primul parametru al funcției *umplere()* – pointer la un tablou de pointeri. Aceasta ne permite să utilizăm numărul liniei (*i*) şi al coloanei (*j*) în adresarea către elementele matricei.

Lab13_2.cpp

```
#include <iostream>
using namespace std;

//declararea functiilor de umplere si afisare
void umplere(int**, int);
void afis(int**, int);

int main(){
    int **Ta;
    int i, j;
    int D;

    cout<<"\n Seteaza dimeniunea tabloului"<<endl;
    cin>>D;
//verificarea dimensiunii tabloului
    if(D<=0) {cout<<"\n Dimensiune foarte mica"; exit(0);}
    if(D>20) {cout<<"\n Dimensiune foarte mare"; exit(0);}
    cout<<"\n D="<<D;
//alocarea memoriei dinamice
    Ta=new int* [D];
        for(i=0; i<D; i++)
            Ta[i]=new int[D];
//apelul functiei de umple
    umplere(Ta, D);
//apelul functiei de afisare
    afis(Ta, D);
//eliberarea zonee de memorie
    delete Ta;
    return 0;
}
```

```
32  //definirea functiei umplere
33  void umplere(int **T, int d){
34      int *P;
35      int i,j;
36      int c=1;
37
38      for(i=0; i<d; i++)
39          for(j=0; j<d; j++)
40              if((j<=max(i,d-i-1))&&(j>=min(i,d-i-1)))  T[i][j]=c++;
41                  else    T[i][j]=0;
42  }
```

```
44  //definirea functiei de afisare
45  void afis(int **T, int d){
46      int i, j;
47      cout<<"\n Tabloul setat"<<endl;
48          for(i=0; i<d; i++){
49              for(j=0; j<d; j++)
50                  cout<<"\t"<<T[i][j];
51                  cout<<endl;
52      }
53  }
54
```

Figura 11.5. Textul programului (varianta nr.2)

Ajustarea programului

La ajustarea programului funcționează aceleași considerente, care sunt expuse în lucrarea nr.10: se recomandă efectuarea căutării erorilor pe baza analizei datelor incluse în program.

Rezultatele executării programului

Mai jos sunt aduse două rezultate ale executării programului: pentru valoarea pară şi cea impară ale dimensiunii.

```
C:\Users\User\Desktop\Indrumar C\Lab13_1.exe

 Seteaza dimeniunea tabloului
5

 D=5
Tabloul bidimensional setat
        1       2       3       4       5
        0       6       7       8       0
        0       0       9       0       0
        0       10      11      12      0
        13      14      15      16      17

--------------------------------
Process exited after 2.307 seconds with return value 0
Press any key to continue . . .
```

Figura 11.6. Rezultatele furnizate de program

1. Descrieţi ordinea de creare a tablourilor dinamice.

2. Cum pot fi create tablouri de pointeri?
3. Explicaţi destinaţia prototipului funcţiei.
4. Câte rezultate poate returna o funcţie?
5. În ce mod putem asigura obţinerea mai multor rezultate cu ajutorul funcţiilor?

11.5. Subiecte de evaluare

Exerciții:

1. Să se implementeze variantele programului din exemplul prezentat.
2. Folosind operaţiile cu pointeri, să se citească de la tastatură elementele unui tablou unidimensional. Să se înlocuiască toate valorile pozitive cu zero. Să se afișeze tabloul până la prelucrare şi după prelucrare.
3. Folosind operaţiile cu pointeri, să se citească de la tastatură elementele a două tablouri unidimensionale de aceeași dimensiune. Să se calculeze elementele unui al treilea tablou ca sumă a elementelor de același indice al primelor două tablouri. Să se afișeze tabloul trei.
4. Folosind operaţiile cu pointeri, să se citească de la tastatură două șiruri de caractere. Să se determine un al treilea șir prin concatenarea primelor două şi să se afișeze la ecran.
5. Pentru o matrice cu *i* linii şi *j* coloane, declarată ca variabilă locală şi având elemente întregi, să se scrie următoarele funcţii cu parametri de intrare pointeri:
   a) citire – citește elementele matricei pe coloane;
   b) tipărire – tipărește elementele matricei pe linii;
   c) det_maxim – determină şi afișează valoarea şi poziția elementului maxim din matrice;
   d) constr_tab – construiește un tablou unidimensional declarat ca variabilă globală, ale cărui elemente sunt sumele elementelor de pe câte o linie a matricei;
   e) interschimbare – inter schimbă elementele de pe două coloane ale matricei cu indicii citiți;
   f) caută – caută în matrice o valoare citită, afișându-i poziţia.

Verificarea cunoștințelor:

1. Definiți noțiunea de pointer.
2. Prezentați exemple de adresare către pointeri.
3. Relatați despre metodele de inițializare a pointerilor.
4. Descrieți prioritatea prezentării informaţiei prin intermediul pointerilor.
5. Descrieți modul de amplasare în memorie a pointerilor.
6. Identificați operatorii ce pot fi aplicați asupra pointerilor. Prezentați exemple.
7. Identificați destinația operatorului adresă & în lucru cu pointerii.
8. Definiți noțiunea de tablou dinamic.
9. Relatași despre funcțiile utilizate la declararea/lichidarea tabloului dinamic,

11.6. Bibliografie

1. Kris Jamsa & Lars Klander, *Totul despre C şi C++*, Editura Teora, Bucureşti, 2006.
2. Herbert Schildt, *C++ manual complet*, Editura Teora, Bucureşti, 1999. Disponibil: https://drive.google.com/file/d/1BrQtITgykcWk03xxtl3Q0-nvcRRFAsy7/view?usp=sharing
3. Tutorials C++ Language. Operators. Disponibil: https://m.cplusplus.com/doc/tutorial-ro/functions/