Cap. 3
Liste înlănțuite

1. Obiectivele lucrării

În această lucrare vom prezenta listele simplu şi dublu înlănțuite.

2. Breviar teoretic

O listă este o colecție de elemente de informație (noduri) aranjate într-o anumită ordine. Cea mai simplă listă este lista liniară. O listă circulară este o listă în care, după ultimul nod, urmează primul, deci fiecare nod are succesor şi predecesor.

Operațiile curente care se fac în liste sunt: inserarea unui nod, extragerea unui nod, concatenarea unor liste, etc. Implementarea unei liste se poate face în principal în două moduri:

- *secvențial*, în locații succesive de memorie, conform ordinii nodurilor în listă. În acest caz, accesul la un nod se face rapid, însă inserarea/ştergerea unui nod durează mai mult timp (presupune mutarea mai multor noduri).

- *înlănțuit*. În acest caz, fiecare nod conține două părți: informația propriu-zisă şi adresa nodului succesor. Alocarea memoriei fiecărui nod se poate face dinamic, în timpul rulării programului. Accesul la un nod necesită parcurgerea tuturor predecesorilor săi, ceea ce ia mai mult timp. Inserarea/ştergerea unui nod este însă foarte rapidă.

Dacă în nod memorăm şi adresa nodului predecesor, obținem o listă *dublu înlănțuită*, ce poate fi traversată în ambele direcții.

3. Probleme rezolvate

Se vor edita şi apoi executa programele descrise în continuare.

1. Implementarea structurii de listă simplu înlănțuită, cu variabile globale.

Sursa programului:

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

```
#include <conio.h>
typedef struct nod{
        int nr;
        nod* pNext;} NOD;
NOD* pHead;
void init(); //initializarea listei
void add(int x);//inserarea la sfarsitul listei
void afisare();
void stergeLista();
int estePrezent(int x);//este prezent x in lista?
int getLungime();//returneaza lungimea listei
void stergeNodCapLista();
void main()
{
 clrscr();
 init();
 add(1);
 add(2);
 add(3);
 afisare();
 printf("\nLungime lista = %d",getLungime());
 printf("\n%d",estePrezent(13));
 getch();
 stergeNodCapLista();
 afisare();
 getch();
 stergeLista();
 getch();
}

void init()
{
 pHead=NULL;
}

void add(int x)
{
 if(pHead==NULL){
   pHead=new NOD;
   pHead->nr=x;
   pHead->pNext=NULL;
   return;}
 NOD* pNodCrt;
 //ajung la sf. listei:
```

```
 pNodCrt=pHead;
 //aflu ultimul nod, neNULL:
 for(;;){
  if(pNodCrt->pNext==NULL)break;
  else pNodCrt=pNodCrt->pNext;
 }
 NOD* pNodNou;
 pNodNou=new NOD;
 pNodCrt->pNext=pNodNou;
 pNodNou->nr=x;
 pNodNou->pNext=NULL;
}

void afisare()
{
 if(pHead==NULL){
   printf("\nlista este vida.");
   return;}
 NOD* pNodCrt;
 pNodCrt=pHead;
 for(;;){
  printf("\n%d",pNodCrt->nr);
  pNodCrt=pNodCrt->pNext;
  if(pNodCrt==NULL)break;
 }
}

void stergeLista()
{
 //memoria ocupata de ea devine disponibila.
 if(pHead==NULL)return;
 NOD *pNodCrt;
 NOD *pNodUrm;
 pNodCrt=pHead;
 for(;;){
  pNodUrm=pNodCrt->pNext;
  delete pNodCrt;
  pNodCrt=pNodUrm;
  if(pNodCrt==NULL)break;
 }
 pHead=NULL;
}

int getLungime()
```

```
{
 if(pHead==NULL)return 0;
 int L;
 NOD* pNodCrt;
 pNodCrt=pHead;
 L=1;
 for(;;){
  pNodCrt=pNodCrt->pNext;
  if(pNodCrt==NULL)break;
  else L++;
 }
 return L;
}

int estePrezent(int x)
{
 if(pHead==NULL)return 0;
 NOD* pNodCrt;
 pNodCrt=pHead;
 int este=0;//presupunem ca nu este prezent
 for(;;){
  if(pNodCrt->nr==x){
    este=1;
    break;}
  pNodCrt=pNodCrt->pNext;
  if(pNodCrt==NULL)break;
 }
 return este;
}

void stergeNodCapLista()
{
 if(pHead==NULL){printf("\nLista este
vida.");return;}
 NOD* pNodUrm;
 pNodUrm=pHead->pNext;
 delete pHead;
 pHead=pNodUrm;
}
```

2. Liste dublu înlănțuite: implementarea cu variabile globale.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
```

```
#include <stdlib.h>
typedef struct nod{
        int nr;
        nod* pNext;
        nod* pPrec;} NOD;
NOD* pHead;
NOD* pLast;
void init();
void adauga(int x);
void afisare();
void afisareInversa();

void main()
{
 clrscr();
 init();
 adauga(1);
 adauga(10);
 adauga(2);
 adauga(-1);
 afisare();
 printf("\n");
 afisareInversa();
 getch();
}

void init()
{
 pHead=NULL;
 pLast=NULL;
}

void adauga(int x)
{
 if(pHead==NULL){
   pHead=new NOD;
   pHead->nr=x;
   pHead->pNext=NULL;
   pHead->pPrec=NULL;
   pLast=pHead;
   return;
 }
 //adaugare la sfarsit, f. rapida:
 NOD* pNodNou;
```

```
 pNodNou=new NOD;
 pNodNou->nr=x;
 pNodNou->pPrec=pLast;
 pNodNou->pNext=NULL;
 pLast->pNext=pNodNou;
 pLast=pNodNou;
}

void afisare()
{
 if(pHead==NULL){
   printf("lista este vida.");
   return;}
 NOD* pNodCrt;
 pNodCrt=pHead;
 for(;;){
  printf("\n%d",pNodCrt->nr);
  pNodCrt=pNodCrt->pNext;
  if(pNodCrt==NULL)break;
 }
}
void afisareInversa()
{
 if(pLast==NULL){
   printf("lista este vida.");
   return;}
 NOD* pNodCrt;
 pNodCrt=pLast;
 for(;;){
  printf("\n%d",pNodCrt->nr);
  pNodCrt=pNodCrt->pPrec;
  if(pNodCrt==NULL)break;
 }
}
```

3. Să se scrie o funcție în care se crează o listă simplu înlănțuită. În funcție se citesc numărul de noduri ale listei. Funcția are ca parametru adresa de început a listei. În programul principal, se va apela funcția ce crează lista, şi apoi o altă funcție ce afişează lista.

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
typedef struct nod{
        int nr;
```

```
        nod* pNext;} NOD;

void creare(NOD*& pHead);
void afisare(NOD* pHead);
void main()
{
 NOD* pHead;//adresa de inceput a listei
 clrscr();
 creare(pHead);
// inserare(pHead,7,0);
 afisare(pHead);
 printf("\nsuma=%d",suma(pHead));
 printf("\nmaxim=%d",max(pHead));
 if(suntCresc(pHead))printf("\n sunt crescatoare");
 else printf("\n nu sunt in ordine crescatoare");
 int v[100];//dimensiune acoperitoare
 int nv;//numarul de elemente din v
 copiere(pHead,v,nv);
 //afisare v:
 printf("\nvectorul v:");
 for(int i=0;i<nv;i++)
   printf("\n%d",v[i]);
 getch();
}

void creare(NOD*& pHead)
{
  int n;//numarul de noduri
  int i;
  int x;//numarul dintr-un nod al listei

  NOD * pNodAnt;//adresa nodului anterior
  NOD * pNodCrt;//adresa nodului curent
  printf("numarul de noduri din lista=");
  scanf("%d",&n);
  if(n==0){
    pHead=NULL;
    return;}
  printf("numar=");
  scanf("%d",&x);
  pHead=new NOD;
  pHead->nr=x;
  pNodAnt=pHead;
  for(i=1;i<n;i++){
```

```
   printf("numar=");
   scanf("%d",&x);
   pNodCrt=new NOD;
   pNodCrt->nr=x;
   //legatura de la nodul anterior:
   pNodAnt->pNext=pNodCrt;
   pNodAnt=pNodCrt;
  }
  //sfarsitul listei:
  pNodAnt->pNext=NULL;
}

void afisare(NOD* pHead)
{
 if(pHead==NULL){
   printf("\nlista este vida.");
   return;}
 NOD* pNodCrt;
 pNodCrt=pHead;
 for(;;){
  printf("\n%d",pNodCrt->nr);
  pNodCrt=pNodCrt->pNext;
  if(pNodCrt==NULL)break;
 }
}
```

4. Să se insereze într-o listă simplu înlănțuită un număr, pe o poziție dată. Se va scrie o funcție, ce are următorul prototip:

```
void inserare(NOD*& pHead, int x, int poz);
```

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
typedef struct nod{
        int nr;
        nod* pNext;} NOD;

void creare(NOD*& pHead);
void afisare(NOD* pHead);
void inserare(NOD*& pHead, int x, int poz);
void main()
{
 NOD* pHead;//adresa de inceput a listei
 clrscr();
 creare(pHead);
```

```
 afisare(pHead);
 inserare(pHead,7,0);
 printf("\nNoua lista:\n");
 afisare(pHead);
getch();
}

//consideram ca lista nu este vida
void inserare(NOD*& pHead, int x, int poz)
{
  NOD * pNodCrt;
  NOD * pNodNou;
  int i;
  //daca  pozitia  de  inserare  este  0,  trebuie
//modificata adresa de inceput a listei:
  if(poz==0){
    pNodNou=new NOD;
    //completam campul de informatie:
    pNodNou->nr=x;
    //completam  si  legatura  catre  nodul  urmator
//(cel de pe pozitia poz):
    pNodNou->pNext=pHead;
    //noua adresa de inceput:
    pHead=pNodNou;
    return;
   }
   //Nu se insereaza la inceputul listei:
  //Parcurgem lista si ne oprim pe nodul poz-1:
  pNodCrt=pHead;
  for(i=1;i<=poz-1;i++)
   pNodCrt=pNodCrt->pNext;
  //creem un nou nod:
  pNodNou=new NOD;
  //completam campul de informatie:
  pNodNou->nr=x;
  //completam si legatura catre nodul urmator (cel
//de pe pozitia poz):
  pNodNou->pNext=pNodCrt->pNext;
  //completam  legatura  de  la  nodul  de  pe  pozitia
//poz-1, la nodul nou:
  pNodCrt->pNext=pNodNou;
}
```

5. Să se scrie o funcție ce calculează suma numerelor din nodurile unei liste simplu înlantuită. Se va scrie o funcție, ce are următorul prototip:

```
int suma(NOD* pHead);
```

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
typedef struct nod{
        int nr;
        nod* pNext;} NOD;

void creare(NOD*& pHead);
void afisare(NOD* pHead);
int suma(NOD* pHead);
void main()
{
 NOD* pHead;//adresa de inceput a listei
 clrscr();
 creare(pHead);
 afisare(pHead);
 printf("\nsuma=%d",suma(pHead));
getch();
}

//consideram ca lista nu este vida:
int suma(NOD* pHead)
{
 NOD* pNodCrt;
 int S=0;//suma
 //parcurgem lista:
 pNodCrt=pHead;
 for(;;){
  S=S+pNodCrt->nr;
  pNodCrt=pNodCrt->pNext;
  if(pNodCrt==NULL)break;
 }
 return S;
}
```

6. Să se scrie o funcție ce calculează maximul dintre numerele din nodurile unei liste simplu înlantuită. Se va scrie o funcție, ce are următorul prototip:

```
int max(NOD* pHead);
```

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
typedef struct nod{
        int nr;
        nod* pNext;} NOD;

void creare(NOD*& pHead);
void afisare(NOD* pHead);
int max(NOD* pHead);
void main()
{
 NOD* pHead;//adresa de inceput a listei
 clrscr();
 creare(pHead);
 afisare(pHead);
 printf("\nmaxim=%d",max(pHead));
getch();
}

//consideram ca lista nu este vida:
int max(NOD* pHead)
{
 NOD* pNodCrt;
 int m;//maximul cautat
 //Initializam maximul, cu valoarea primului nod
din lista:
 m=pHead->nr;
 //parcurgem lista plecand de la al doilea nod:
 pNodCrt=pHead->pNext;
 for(;;){
  if(pNodCrt==NULL)break;
  if(pNodCrt->nr > m)m=pNodCrt->nr;
  pNodCrt=pNodCrt->pNext;
  if(pNodCrt==NULL)break;
 }
 return m;
}
```

7. Să se scrie o funcție ce testează dacă toate numerele dintr-o listă simplu înlănțuită sunt în ordine strict crescătoare. Se va scrie o funcție, ce are următorul prototip:

```
int suntCresc(NOD* pHead);
```

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
typedef struct nod{
        int nr;
        nod* pNext;} NOD;

void creare(NOD*& pHead);
void afisare(NOD* pHead);
int suntCresc(NOD* pHead);
void main()
{
 NOD* pHead;//adresa de inceput a listei
 clrscr();
 creare(pHead);
 afisare(pHead);
 if(suntCresc(pHead))printf("\n sunt crescatoare");
 else printf("\n nu sunt in ordine crescatoare");
 getch();
}

//consideram ca lista nu este vida:
int suntCresc(NOD* pHead)
{
 NOD* pNodCrt;
 int sunt=1;//variabila semafor: presupunem ca sunt
 int xCrt;//informatia din nodul curent
 int xAnt;//informatia din nodul anterior
 //parcurgem lista:
 pNodCrt=pHead;
 //initializare:
 xAnt=pHead->nr;
 for(;;){
  pNodCrt=pNodCrt->pNext;
  if(pNodCrt==NULL)break;//s-a terminat lista
  xCrt=pNodCrt->nr;
  if(xCrt<xAnt){
    sunt=0;//nu sunt in ordine crescatoare
    break;
  }
  xAnt=xCrt;
 }
 return sunt;
}
```

8. Să se scrie o funcție în care se copiază toate numerele din nodurile unei liste simplu înlănțuite într-un vector. Se va scrie o funcție, ce are următorul prototip:

```
void copiere(NOD* pHead, int v[], int& dim);
```

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
typedef struct nod{
        int nr;
        nod* pNext;} NOD;

void creare(NOD*& pHead);
void afisare(NOD* pHead);
void copiere(NOD* pHead, int v[], int& dim);
void main()
{
 NOD* pHead;//adresa de inceput a listei
 clrscr();
 creare(pHead);
 afisare(pHead);
 int v[100];//dimensiune acoperitoare
 int nv;//numarul de elemente din v
 copiere(pHead,v,nv);
 //afisare v:
 printf("\nvectorul v:");
 for(int i=0;i<nv;i++)
   printf("\n%d",v[i]);
 getch();
}

//consideram ca lista nu este vida:
void copiere(NOD* pHead, int v[], int& dim)
{
 NOD* pNodCrt;
 int i;//index in vectorul v
 //parcurgem lista plecand de la primul nod:
 pNodCrt=pHead;
 i=0;//index in vectorul v
 for(;;){
  v[i]=pNodCrt->nr;
  i++;
  pNodCrt=pNodCrt->pNext;
  if(pNodCrt==NULL)break;//s-a terminat lista
 }
```

```
 dim=i;//numarul de elemente din vectorul v
}
```

9. Se dau două liste simplu înlănțuite. Să se stabilească dacă cele două liste sunt egale (fiecare listă va fi creată cu ajutorul funcției creare(), folosită şi în programele anterioare).

Sursa programului:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
typedef struct nod{
        int nr;
        nod* pNext;} NOD;
void creare(NOD*& pHead);
void main()
{
 NOD* pHead1;//adresa de inceput a primei liste
 NOD* pHead2;
 //creem prima lista:
 printf("Prima lista:\n");
 creare(pHead1);
 //creem a doua lista:
 printf("A doua lista:\n");
 creare(pHead2);
 //Parcurgem    prima  lista.  Fiecare  element  il
//comparam cu elementul  corespunzator din a doua
//lista:
 NOD* p1Crt;//nodul curent din prima lista
 NOD* p2Crt;//nodul curent din a doua lista
 p1Crt=pHead1;
 p2Crt=pHead2;
 for(;;){
    if((p1Crt==NULL)&&(p2Crt!=NULL)){
      printf("Nu");
      getch();
      exit(0);}
    if((p1Crt!=NULL)&&(p2Crt==NULL)){
      printf("Nu");
      getch();
      exit(0);}
    if((p1Crt==NULL)&&(p2Crt==NULL)){
      printf("Da");
      getch();
      exit(0);}
```

```
    //cazul tipic, comparam numerele din nodurile
//curente ale celor doua liste:
    if(p1Crt->nr != p2Crt->nr){
      printf("Nu");
      getch();
      exit(0);}
    //numerele din nodurile curente sunt egale,
//avansam deci in fiecare lista:
    p1Crt=p1Crt->pNext;
    p2Crt=p2Crt->pNext;
 }
}
```

4. Probleme propuse

1. *Să se implementeze fără variabile globale, o listă simplu înlănțuită.*
1. *Să se implementeze fără variabile globale, o listă dublu înlănțuită.*
2. *Să se şteargă dintr-o listă simplu înlănțuită un număr, de pe o poziție dată. Se va scrie o funcție.*
3. *Concatenarea a două liste simplu înlănțuite.*
4. *Să se scrie o funcție ce returnează numărul de apariții al unui număr dat, într-o listă simplu înlănțuită.*
5. *Să se scrie o funcție ce inserează într-o listă dublu înlănțuită un număr, pe o poziție dată.*
6. *Să se scrie o fiuncție ce şterge dintr-o listă dublu înlănțuită un număr, de pe o poziție dată.*
7. *Să se scrie o funcție ce returnează valoarea numărului din nodul doi al unei liste simplu înlănțuite.*
8. *Să se scrie o funcție ce returnează valoarea numărului din penultimul nod al unei liste dublu înlănțuite.*
9. *Se dă o listă simplu înlănțuită. Să se copieze toate numerele prime din acestă listă, într-o altă listă simplu înlănțuită.*
10. *Se dă o listă simplu înlănțuită. Să se insereze în acestă listă, dupa fiecare nod curent, câte un nou nod, ce are valoarea numerică egală cu cea din nodul curent.*
11. *Folosind structura de stivă, să se construiască un vector B, obținut prin copierea în ordine inversă a unui vector A, dat.*

12. *Să se scrie o funcție ce calculează suma numerelor din nodurile unei liste simplu înlănțuite.*
13. *Să se calculeze maximul dintre numerele din nodurile unei liste simplu înlănțuite. Se va scrie o funcție.*
14. *Să se scrie o funcție ce testează dacă toate numerele dintr-o listă simplu înlănțuită sunt în ordine strict crescătoare.*
15. *Să se realizeze o funcție, împreună cu un program de test, care să sorteze după metoda bubble-sort elementele unei liste simplu înlănțuite.*
16. *Se citeşte de la tastatură un număr natural N. Să se realizeze un program care să copieze toți divizorii numărului într-o listă simplu înlănțuită.*
17. *Să se realizeze un program, care calculează şi afişează cel mai mare divizor comun al tuturor numerelor dintr-o listă simplu înlănțuită.*