1. **STRUCTURI DE DATE**

**STIVE, COZI ŞI LISTE**

* 1. **SCOPUL LUCRĂRII**

Lucrarea are ca scop o introducere ***sumară*** în vastul capitol al ***structurilor de date***, cu accent pe ***listele înlănţuite*** – instrumentul principal pentru implementarea dinamică a ***stivelor*** şi ***cozilor.***

* 1. **BREVIAR TEORETIC**

Structurile de date reprezintă modalităţi în care datele sunt dispuse în memoria calculatorului (sau sunt păstrate pe discul magnetic).

Structurile de date sunt utilizate în diferite circumstanţe, ca de exemplu:

* Memorarea unor date din realitate;
* Instrumente ale programatorilor;
* Modelarea unor situaţii din lumea reală.

Cele mai des utilizate structuri de date sunt tablourile, listele, stivele, cozile, arborii, tabelele de dispersie si grafurile.

* + 1. **STIVE**

Stiva este o structură de date abstractă, pentru care atât operaţia de inserare a unui element în structură, cât şi operaţia de extragere a unui element, se realizează la un singur capăt, denumit *vârful* stivei.

Singurul element din stivă la care avem acces direct este cel de la vârf.

***Operaţii caracteristice***

Singurele operaţii ce pot fi executate cu o stivă sunt:

* crearea unei stive vide;
* inserarea unui element în stivă (operaţie denumită în literatura de specialitate **PUSH a)**
* extragerea unui element din stivă (operaţie denumită în termeni de specialitate **POP b**)
* accesarea elementului de la vârf (operaţie denumită **TOP c**).



***Notaţii:***

1. În traducere exactă, **PUSH** înseamnă „a împinge”. Termenul sugerează o imagine plastică: imaginându-ne stiva ca un sac, **PUSH** ar însemna că împingem înăuntru un element, prin capătul superior (nu prin mijloc sau pe la fundul sacului).
2. În traducere, **POP** înseamnă „a scoate cu zgomot (de exemplu dopul, etc), a descărca (pistolul, etc), a ieşi repede sau pe neaşteptate” (Leviţchi, Leon; Bantaş, Andrei – *Dicţionar englez-român*). Imaginea este de asemeni sugestivă: **POP** este operaŢia prin care primul element, cel de deasupra, „sare” din structură (sau din sac, ca să păstrăm analogia).
3. În traducere, **TOP** înseamnă „vârf”.

*Ca să ne imaginăm mai bine funcţionarea unei stive, să ne gândim cum lucrăm cu un teanc de farfurii. Când dorim să punem o farfurie în teanc, o punem deasupra, când dorim să luăm o farfurie din teanc, o luăm tot pe cea de deasupra. Motivul este lesne de înţeles: nu ne-am propus să spargem farfuriile!*

Acest mod de funcţionare face ca ultimul element inserat în stivă să fie primul extras. Din acest motiv, *stiva* este definită şi ca o structură de date care funcţionează după principiul **LIFO** (**L**ast **I**n **F**irst **O**ut – Ultimul Intrat Primul Ieşit).

***Care este utilitatea stivelor?***

În informatică, stiva joacă un rol fundamental. Pentru a înţelege mecanisme fundamentale ale programării (de exemplu, funcţiile sau recursivitatea) este necesară cunoaşterea noţiunii de stivă. Pe scurt, stiva este utilă în situaţii în care este necesară memorarea unor informaţii şi regăsirea acestora într-o anumită ordine, descrisă de principiul LIFO. Stiva este utilizată atunci când programul trebuie să amâne execuţia unor operaţii, pentru a le executa ulterior, în ordinea inversă a apariţiei lor.

Operaţia curentă este cea corespunzătoare vârfului stivei, în stivă fiind reţinute toate informaţiile necesare programului pentru a executa operaţiile respective.

## *Cum implementăm o stivă?*

Stiva este o structură de date abstractă, ce poate fi implementată în diferite moduri. De exemplu, putem implementa o stivă ca un vector în care reţinem elementele stivei. Pentru ca acest vector să funcţioneze ca o stivă, singurele operaţii permise sunt operaţiile caracteristice stivei.

Pentru exemplificare, să prezentăm declaraţiile necesare pentru implementarea unei stive cu elemente de tip int:

#define DimMax 100 //numarul maxim de elemente din stiva

**typedef int** Stiva[DimMax];

// tipul Stiva implementat ca vector

Stiva // stiva

**int** vf; //varful stivei

***Crearea unei stive vide***

Pentru a crea o stivă vidă iniţializăm vârful stivei cu -1 (vârful stivei indică întotdeauna poziţia ultimului element introdus în stivă; elementele sunt memorate în vector începând cu poziţia 0):

vf = -1;

***Inserarea unui element în stivă***

Pentru a insera un element x în stiva S, trebuie să verificăm în primul rând dacă „avem loc”, deci dacă stiva nu este plină. Dacă stiva este plină, inserarea nu se poate face, altfel vom mări vârful stivei şi vom plasa la vârf noul element.

De exemplu, dacă dorim să inserăm elementul x = 3 în stiva din figura următoare, obţinem:



**if** (vf = = DimMax - 1) // stiva este plina

cout << “Eroare – stiva este plina”<<endl;

**else** // inseram elementul x in stiva S

S[++vf] = x;

***Extragerea unui element din stivă***

Pentru a extrage un element dintr-o stivă S, trebuie să verificăm în primul rând dacă există elemente în stivă (deci dacă stiva nu este vidă). Dacă da, reţinem elementul de la vârful stivei într-o variabilă (să o notăm x), după care micşorăm cu o unitate vârful stivei. De exemplu, dacă extragem un element din stiva din figura următoare, obţinem:



Stiva S înainte de extragere Stiva S după de extragere

**if** (vf < 0) // stiva este vida

cout << “Eroare – stiva este vida”<<endl;

**else** // extragem elementul de la varf

x = S[vf--] ;

***Accesarea elementului de la vârf***

Prin modul său restrictiv de funcţionare, stiva permite numai accesarea elementului de la vârf. Dacă dorim să aflăm valoarea unui alt element al stivei, ar trebui să „golim” stiva (deci să extragem succesiv elemente) până la elementul dorit.

Accesarea elementului de la vârf presupune determinarea valorii acestuia, valoare pe care noi o vom reţine într-o variabilă denumită x.

x = s[vf];

***Observaţii:***

1. Dezavantajul implementării unei stive ca vector alocat static constă în faptul că, indiferent de numărul de elemente existente în stivă, dimensiunea zonei de memorie alocată stivei este aceeaşi (DimMax).
2. Pentru a executa operaţii cu stiva alocată static, este suficient să cunoaştem vârful stivei. Ca să reţinem mai uşor modul de funcŢionare a stivei, ne imaginăm că la inserare vârful stivei urcă, iar la extragere vârful coboară.
   * 1. **COZI**

***Coada*** este o structură de date abstractă, pentru care operaţia de inserare a unui element se realizează la un capăt, în timp ce operaţia de extragere a unui element se realizează la celălalt capăt.

Singurul element din coadă la care avem acces direct este cel de la început.



***Operaţii caracteristice***

Singurele operaţii ce pot fi executate cu o coadă sunt:

* 1. crearea unei cozi vide;
  2. inserarea unui element în coadă;
  3. extragerea unui element din coadă;
  4. accesarea unui element.

Executarea acestor operaţii asupra unei cozi presupune cunoaşterea începutului cozii (să-l notăm Inc) şi a sfârşitului acesteia (să-l notăm Sf).

*Modul de funcţionare a unei cozi este foarte uşor de intuit: toată lumea a „stat la coadă”, măcar o dată. Orice situaţie în care sunt mai multe cereri de acces la o resursă unică (de exemplu, mai mulţi clienţi şi o singură vânzătoare; o singură pompă de benzină şi mai multe maşini, un singur pod şi mai multe capre, etc) necesită formarea unei „linii de aşteptare”. Dacă nu apar alte priorităţi, cererile sunt satisfăcute în ordinea sosirii.*

Datorită faptului că întotdeauna este extras („servit”) primul element din coadă, iar inserarea oricărui nou element se face la sfârşit („la coadă”), coada este definită ca o structură de date care funcŢionează după principiul **FIFO** (**F**irst **I**n **F**irst **O**ut – Primul Intrat Primul Ieşit).

***Care este utilitatea unei cozi?***

Utilitatea structurii de tip coadă reiese din modul său de funcţionare – este nece- sară utilizarea unei cozi atunci când informaţiile trebuie prelucrate exact în ordinea în care „au sosit” şi ele sunt reţinute în coadă până când pot fi prelucrate. În informatică, cozile sunt utilizate frecvent. De exemplu, să considerăm că avem o reţea de calculatoare şi o singură imprimantă. Când utilizatorii reţelei vor da comenzi de tipărire, imprimanta nu poate răspunde tuturor comenzilor în acelaşi timp (imaginaţi-vă ce-ar ieşi!). Prin urmare comenzile de tipărire primite sunt înregis- trate într-o coadă (*Print Queue* – Coadă de Tipărire). Imprimanta va tipări documentele pe rând, în ordinea în care au fost înregistrate în coadă. Un alt exemplu: pentru a mări viteza de execuţie, microprocesoarele Intel utilizează o coadă de instrucţiuni în care sunt memorate instrucţiunile care urmează a fi executate.

***Cum implementăm o coadă?***

Coada este o structură de date abstractă, care poate fi implementată în diferite moduri. Ca şi în cazul stivei, coada poate fi implementată static, reţinând elementele sale într-un vector.

Să considerăm următoarele declaraţii care reprezintă o coadă cu elemente de tip int alocată static:

## #define DimMax 100 // numarul maxim de elemente din coada

## typedef int Coada[DimMax];

## // tipul Coada implementat ca vector

## Coada C; //coada

## int Inc, Sf; //inceputul si sfarsitul cozii

## Elementele cozii sunt memorate în vector de la poziţia Inc până la poziţia Sf, deci numărul lor este Sf – Inc + 1.

## *Crearea unei cozi vide*

## Pentru a crea o coadă vidă, trebuie să iniţializăm variabilele Inc şi Sf astfel:

## Inc = 0;

## Sf = -1;

Observaţi că numărul de elemente din coadă este 0, iar poziţia pe care va fi plasat primul element din coadă este 0.

***Inserarea unui element în coadă***

Pentru a insera un element x în coada C trebuie să verificăm în primul rând dacă „avem loc”, deci dacă variabila Sf nu depăşeşte dimensiunea maximă a vectorului. Dacă inserarea se poate face, vom mări valoarea variabilei Sf cu o unitate (coada „creşte”) şi vom plasa la sfârşit noul element.

De exemplu, să inserăm elementul x = 3 în coada din figura următoare:



Coada C înainte de inserare



Coada C după de inserare

**if** (Sf = = DimMax - 1) //nu mai avem loc in coada

cout << “Eroare” << endl;

**else** //inseram elementul x in coada C

C[++Sf] = x;

***Extragerea unui element din coadă***

Pentru a extrage un element dintr-o coadă C, trebuie să verificăm în primul rând dacă numărul de elemente din coadă este diferit de 0 (coada nu este vidă). Dacă da, reţinem elementul de la începutul cozii într-o variabilă (să o notăm x), după care mărim cu o unitate începutul cozii.

De exemplu, să extragem un element din coada din figura următoare:



Coada C ***înainte*** de extragere



Coada C ***după*** de extragere

**if** (Inc > Sf) //coada este vidă

cout << “Eroare” << endl;

**else** //extragem primul element

x = C[Inc++];

***Accesarea unui element***

Singurul element al unei cozi care poate fi accesat direct este primul. Dacă dorim să aflăm valoarea unui alt element, va trebui să extragem succesiv elemente până la cel dorit.

Accesarea primului element are ca scop determinarea valorii acestuia, valoare pe care o vom reţine într-o variabilă denumită x.

x = C[Inc];

***Observaţie***

În acest mod de alocare statică a unei cozi observaţi că, pe măsură ce inserăm şi extragem elemente, atât sfârşitul, cât şi începutul cozii „migrează” către limita maximă a vectorului. Dezavantajul este că în vector rămân poziţii neutilizate (de la 0 până la Inc-1).

De exemplu, ar putea să apară o situaţie în care coada conţine un element, dar operaţia de inserare să nu fie posibilă, pentru că Inc = Sf = DimMax - 1.

O soluţie mai bună ar fi de a reutiliza circular spaţiul de memorie alocat cozii. Pentru aceasta, următoarea poziţie, după poziţia i, poate fi:

* i+1, dacă I < DimMax - 1
* 0, dacă i=DimMax-1

Acest lucru se poate scrie concis: (i+1)%DimMax.

* + 1. **LISTE**

O list˘a liniar˘a este o structur˘a de date omogen˘a, secven¸tial˘a, format˘a din ***elemente*** ale listei. Un ***element*** (numit uneori ¸si ***nod***) din list˘a con¸tine o informa¸tie specific˘a ¸si, eventual, o informa¸tie de leg˘atur˘a. De exemplu, în lista echipelor de fotbal înscrise într-un campionat, un element (ce reprezint˘a o echip˘a) poate con¸tine urm˘atoarele informa¸tiie specifice: nume, num˘ar de puncte, num˘ar de goluri înscrise ¸si num˘ar de goluri primite. Fiecare din aceste informa¸tii poate reprezenta o ***cheie,*** care poate fi utilizat˘a pentru compara¸tii ¸si inspec¸tii. De exemplu, luând drept cheie num˘arul de puncte ¸si golaverajul, se poate face clasificarea echipelor.

Pozi¸tia elementelor din list˘a define¸ste ordinea elementelor (primul element, al doilea, etc). Con¸tinutul listei se poate schimba prin:

* ad˘augarea de noi elemente la sfârşitulitul listei;
* inserarea de noi elemente în orice loc din list˘a;
* ¸stergerea de elemente din orice pozi¸tie a listei;
* modificarea unui element dintr-o pozi¸tie dat˘a.

Printre opera¸tiile care schimb˘a structura listei vom considera ¸si ini¸tializarea unei liste ca o ***list˘a vid˘a***.

Alte opera¸tii sunt ***opera¸tiile de caracterizare***. Ele nu modific˘a structura listelor, dar furnizeaz˘a informa¸tii despre ele. Dintre opera¸tiile de caracterizare vom men¸tiona, în cele ce urmează, următoarele:

* localizarea elementului din list˘a care satisface un anumit criteriu;
* determinarea lungimii listei.

Pot fi luate în considerare ¸si opera¸tii mai complexe, ca de exemplu:

* separarea unei liste în dou˘a sau mai multe ***subliste,*** în func¸tie de îndeplinirea unor condi¸tii;
* selec¸tia elementelor dintr-o list˘a care îndeplinesc unul sau mai multe criterii, într-o list˘a nou˘a;
* crearea unei liste ordonate dup˘a valorile creasc˘atoare sau descresc˘atoare ale unei chei;
* combinarea a dou˘a sau mai multor liste prin concatenare (alipire) sau

interclasare.

Spa¸tiul din memorie ocupat de list˘a poate fi alocat în dou˘a moduri: prin

alocare secven¸tial˘a sau prin alocare înlăn¸tuit˘a.

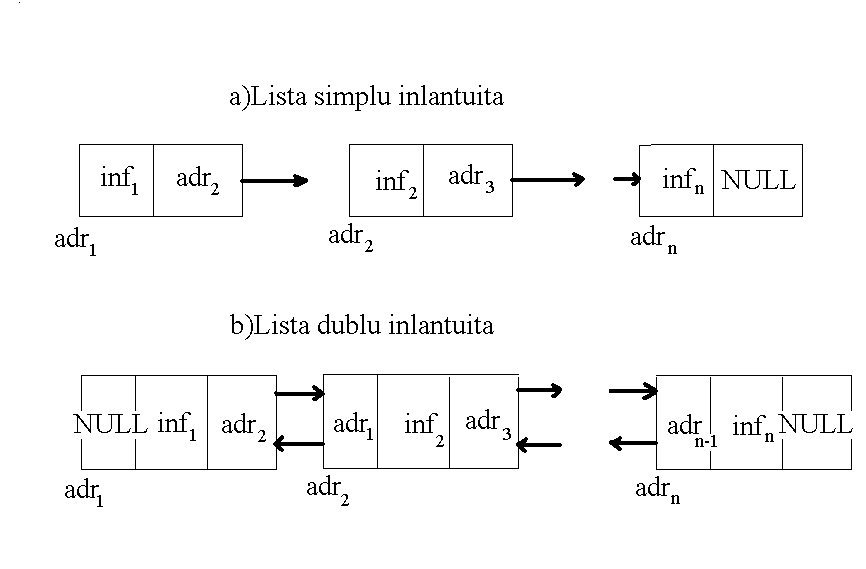
* + - 1. ***Liste alocate secvenţial***

În aceste caz, nodurile ocup˘a pozi¸tii succesive în memorie. Acest tip de alocare este întâlnlit când se lucreaz˘a cu tablouri (vectori). Avantajul aloc˘arii secven¸tiale este dat de faptul c˘a accesul la oricare din nodurile listei este direct.

Dezavantajul constă în faptul c˘a opera¸tiile de ad˘augare, eliminare sau schimbare de pozi¸tie a unui nod necesit˘a un efort mare de calcul.

* + - 1. ***Liste alocate înlănţuit***

Exist˘a dou˘a feluri de alocare înlănţuită: alocare simplu înlănţuită ¸si alocare dublu înlănţuită. Alocarea înlănţuită poate fi efectuat˘a static (utilizând vectori), sau dinamic.



În acest din urm˘a caz (de care ne vom ocupa în cele ce urmeaz˘a), se utilizeaz˘a o zon˘a de memorie numit˘a ***HEAP*** (movil˘a, gr˘amad˘a).

În C++, variabilele p˘astrate în HEAP (cum ar fi de exemplu nodurile listei), se aloc˘a doar atunci când dore¸ste programatorul, cu ajutorul operatorului ***new***, iar zona se elibereaz˘a, tot la dorin¸ta acestuia prin operatorul delete.

În cazul aloc˘arii dinamice, nodul unei liste este o ***structur˘a*** care con¸tine, al˘aturi de informa¸tie, ¸si adresa nodului urm˘ator (pentru liste simplu înlănţuite) sau adresele nodului urm˘ator ¸si a celui precedent (în cazul listelor dublu înlănţuite).

Dup˘a cum se va vedea din exemplele ce urmeaz˘a, principala problem˘a, în cazul opera¸tiilor cu liste, o reprezint˘a redefinirea leg˘aturilor (adreselor) atunci când se ¸sterge sau se insereaz˘a un element.

* 1. **EXEMPLE DE SUBRUTINE PENTRU LISTELE ÎNLĂNŢUITE**

În cele ce urmează, este prezentat un exemplu funcţional (aproape complet) de exploatare a listelor dublu înlănţuite

#include <graphics.h>

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <iostream.h>

#include <fstream.h>

#include <dos.h>

#include <string.h>

struct nod{

nod \* ante;

char nume[30];

float lat, lng;

nod \* post;

};

nod \* curent, \* prim, \* ultim, \* ajut, \* aux;

int n=0, i, poz, pozcautat, gasit;

char oras[30], numefis[25];

void creare(){

clrscr();

cout<<"Pun bazele unei noi liste, prin crearea primului nod"<<endl<<endl;

prim = new nod;

cout << "nume = ? "; gets(prim -> nume);

cout << "latitudinea = ? "; cin >> prim -> lat;

cout << "longitudinea = ? "; cin >> prim -> lng;

prim -> ante = 0;

prim -> post = 0;

ultim = prim;

n=1;

}

void listarestdr(){

clrscr();

cout<<"Lista are "<<n<<" noduri"<<endl;

cout<<"Lista de la stanga la dreapta : "<<endl<<endl;

curent = prim;

while(curent -> post !=0){

cout<<curent->nume<<" "<<curent->lat<<" "<<curent->lng<<endl;

curent = curent -> post;

}

cout<<curent->nume<<" "<<curent->lat<<" "<<curent->lng<<endl;

getch();

}

void listaredrst(){

clrscr();

cout<<"Lista are "<<n<<" noduri"<<endl;

cout<<"Lista de la dreapta la stanga : "<<endl<<endl;

curent = ultim;

while(curent -> ante !=0){

cout<<curent->nume<<" "<<curent->lat<<" "<<curent->lng<<endl;

curent = curent -> ante;

}

cout<<curent->nume<<" "<<curent->lat<<" "<<curent->lng<<endl;

getch();

}

void insmijloc(){

clrscr();

int rasp;

cout<<" Inserarea unui nod dupa un nod cu un nume cautat"<<endl;

cout<<" Introduceti numele orasului cautat: "; gets(oras);

cout<<" cauti orasul "<<oras<<endl<<endl;

poz=0;

i=1; int gasit=0;

curent=prim;

while(i<=n && !gasit){

cout<<"sunt la orasul "<<curent->nume<<endl;

cout<<"compar orasul "<<oras<<" cu "<<curent->nume<<endl;

if(!strcmp(oras,curent->nume)) {poz=i; gasit=1; cout<<"atentie!"<<endl;}

cout<<"gasit="<<gasit<<endl;

curent=curent->post;

i++;

}

if(gasit) {

cout<<"am gasit orasul pe pozitia "<<poz<<endl;

ajut=new nod;

cout<<"Introduceti noul nume de oras "; gets(ajut -> nume);

cout<<"Introduceti latitudinea noului oras "; cin>>ajut -> lat;

cout<<"Introduceti longitudinea noului oras "; cin>>ajut -> lng;

curent = curent ->ante;

ajut -> ante = curent;

ajut -> post = curent -> post;

curent -> post = ajut;

aux = ajut -> post;

aux -> ante = ajut;

n++;

}

getch();

}

void insinaint(){

clrscr();

cout<<"Inserez un nod inainte de prima pozitie"<<endl<<endl;

ajut = new nod;

cout << "nume = ? "; gets(ajut -> nume);

cout << "latitudinea = ? "; cin >> ajut -> lat;

cout << "longitudinea = ? "; cin >> ajut -> lng;

ajut -> post = prim;

prim -> ante = ajut;

ajut -> ante = 0;

prim = ajut;

n++;

}

void inscoada(){

clrscr();

cout<<"Inserez un nod dupa ultima pozitie"<<endl<<endl;

ajut = new nod;

cout << "nume = ? "; gets(ajut -> nume);

cout << "latitudinea = ? "; cin >> ajut -> lat;

cout << "longitudinea = ? "; cin >> ajut -> lng;

ajut -> ante = ultim;

ultim -> post = ajut;

ajut -> post = 0;

ultim = ajut;

n++;

}

void stergint(){

clrscr();

int rasp;

cout<<"In constructie. Voi sterge un nod din interior";

cout<<endl<<endl;

cout<<"Alegeti:"<<endl<<

"1. Stergerea unui nod de pe o pozitie data"<<endl<<

"2. Stergerea unui nod cu un continut dat"<<endl;

cin>>rasp;

if(rasp==1){

cout<<"Introduceti pozitia orasului cautat: "; cin>>pozcautat;

cout<<"cauti orasul de pe pozitia "<<pozcautat<<endl<<endl;

poz=0;

i=1; int gasit=0;

curent=prim;

while(i<=n && !gasit){

if(i==pozcautat){poz=i;gasit=1;break;}

curent=curent->post;

i++;

}

if(gasit){

aux=curent->ante;

aux->post=curent->post;

aux=curent->post;

aux->ante=curent->ante;

n--;

cout<<"Lista are acum "<<n<<" noduri"<<endl;

}

else cout<<"Nu exista pozitia cautata in lista"<<endl;

}

if(rasp==2){

cout<<"Introduceti numele orasului cautat: "; gets(oras);

cout<<"cauti orasul "<<oras<<endl<<endl;

poz=0;

i=1; int gasit=0;

curent=prim;

while(i<=n && !gasit){

cout<<"sunt la orasul "<<curent->nume<<endl;

cout<<"compar orasul "<<oras<<" cu "<<curent->nume<<endl;

if(!strcmp(oras,curent->nume)) {poz=i; gasit=1; cout<<"atentie!"<<endl;}

cout<<"gasit="<<gasit<<endl;

curent=curent->post;

i++;

}

if(gasit) {

cout<<"am gasit orasul pe pozitia "<<poz<<endl;

curent = curent -> ante;

aux = curent ->ante;

aux -> post = curent -> post;

aux = curent -> post;

aux -> ante = curent -> ante;

n--;

cout<<"Lista are acum "<<n<<" noduri"<<endl;

}

else cout<<"n-am gasit orasul"<<endl;

}

getch();

}

void stergprim(){

clrscr();

cout<<"Sterg primul nod"<<endl;

if(ultim->ante==0 && ultim->post==0)

{cout<< "A mai ramas un singur nod" <<endl;}

else{curent=prim->post;

curent->ante=0;

prim=curent;

n--;

}

cout<<"Lista are acum "<<n<<" noduri"<<endl;

getch();

}

void stergult(){

clrscr();

cout<<"Sterg ultimul nod"<<endl;

if(ultim->ante==0 && ultim->post==0)

{cout<< "A mai ramas un singur nod" <<endl;}

else

{curent=ultim->ante;

curent->post=0;

ultim=curent;

n--;}

cout<<"Lista are acum "<<n<<" noduri"<<endl;

getch();

}

void modific(){

clrscr();

int rasp;

cout<<endl<<endl;

cout<<"Alegeti:"<<endl;

cout<<"1. Modificarea continutului unui nod de pe o pozitie data"<<endl;

cout<<"2. Modificarea continutului unui nod cu un nume dat"<<endl;

cin>>rasp;

if(rasp==1){

cout<<"Introduceti pozitia orasului cautat: "; cin>>pozcautat;

cout<<"cauti orasul de pe pozitia "<<pozcautat<<endl<<endl;

poz=0;

i=1; int gasit=0;

curent=prim;

while(i<=n && !gasit){

if(i==pozcautat){poz=i;gasit=1;break;}

curent=curent->post;

i++;

}

if(gasit){

cout<<"am gasit orasul de pe pozitia "<<poz<<endl;

cout<<"Introduceti noul nume de oras "; gets(curent -> nume);

cout<<"Introduceti latitudinea noului oras "; cin>>curent -> lat;

cout<<"Introduceti longitudinea noului oras "; cin>>curent -> lng;

}

else cout<<"nu exista pozitia cautata in lista"<<endl;

}

if(rasp==2){

cout<<"Introduceti numele orasului cautat: "; gets(oras);

cout<<"cauti orasul "<<oras<<endl<<endl;

poz=0;

i=1; int gasit=0;

curent=prim;

while(i<=n && !gasit){

cout<<"sunt la orasul "<<curent->nume<<endl;

cout<<"compar orasul "<<oras<<" cu "<<curent->nume<<endl;

if(!strcmp(oras,curent->nume)) {poz=i; gasit=1; cout<<"atentie!"<<endl;break;}

cout<<"gasit="<<gasit<<endl;

curent=curent->post;

i++;

}

if(gasit) {

cout<<"am gasit orasul pe pozitia "<<poz<<endl;

cout<<"Introduceti noul nume de oras "; gets(curent -> nume);

cout<<"Introduceti latitudinea noului oras "; cin>>curent -> lat;

cout<<"Introduceti longitudinea noului oras "; cin>>curent -> lng;

}

else cout<<"n-am gasit orasul"<<endl;

}

getch();

}

void grafic(){

clrscr();

int n, i;

float x[20], y[20], maxx, minx, maxy, miny, dx, dy, canvmaxx=639, canvmaxy=479, b=20;

char nume[20][20];

cout<<endl<<endl;

ifstream xx(numefis);

xx>>n;

maxx=0; minx=639; maxy=0; miny=439;

for(i=1; i<=n; i++){

xx>>nume[i]; cout<<i<<" "<<nume[i]<<" : ";

xx>>y[i]; cout<<y[i]<<" ";

if(y[i]>maxy) maxy=y[i];

if(y[i]<miny) miny=y[i];

xx>>x[i]; cout<<x[i]<<endl;

if(x[i]>maxx) maxx=x[i];

if(x[i]<minx) minx=x[i];

}

getch();

/\* for(i=1; i<=n; i++) cout<<x[i]<<" "<<y[i]<<endl<<endl<<endl;

cout<<"maxx="<<maxx<<endl;

cout<<"minx="<<minx<<endl;

cout<<"maxy="<<maxy<<endl;

cout<<"miny="<<miny<<endl<<endl; \*/

dx=maxx-minx; //cout<<dx<<endl;

dy=maxy-miny; //cout<<dy<<endl<<endl;

//getch();

// int corectinv = 390;

for(i=1; i<=n; i++) x[i]=(x[i]-minx)\*(canvmaxx-2\*b)/dx+b;

for(i=1; i<=n; i++) y[i]=439-((y[i]-miny)\*(canvmaxy-2\*b)/dy+b)+2\*b;

/\* request auto detection \*/

int gdriver = DETECT, gmode, errorcode;

/\* initialize graphics mode \*/

initgraph(&gdriver, &gmode, "");

/\* read result of initialization \*/

errorcode = graphresult();

if (errorcode != grOk) /\* an error occurred \*/

{

printf("Graphics error: %s\n", grapherrormsg(errorcode));

printf("Press any key to halt:");

getch();

exit(1); /\* return with error code \*/

}

setbkcolor(15);

setcolor(8);

line(b-10,b-10,canvmaxx-b+10,b-10);

line(canvmaxx-b+10,b-10,canvmaxx-b+10,canvmaxy-b+10);

line(canvmaxx-b+10,canvmaxy-b+10,b-10,canvmaxy-b+10);

line(b-10,canvmaxy-b+10,b-10,b-10);

delay(500);

int inalt=420, durata=100, pas=20;

for(i=1; i<=n-1; i++){

inalt=inalt+20;

sound(inalt);

delay(durata);

nosound();

circle(x[i], y[i], 5);

if(x[i]<600) outtextxy(x[i]+10, y[i], nume[i]);

else outtextxy(x[i]-80, y[i], nume[i]);

delay(500);

line(x[i], y[i], x[i+1], y[i+1]);

}

inalt=inalt+10;

sound(inalt);

delay(durata);

nosound();

circle(x[n], y[n], 5);

if(x[n]<600) outtextxy(x[n]+10, y[n], nume[i]);

else outtextxy(x[n]-80, y[n], nume[i]);

int maximumx = getmaxx();

int maximumy = getmaxy();

/\* clean up \*/

getch();

closegraph();

// cout<<maximumx<<" "<<maximumy<<endl;

getch();

}

void salvez(){

clrscr();

cout<<"Salvez lista in fisier"<<endl;

cout<<"Numele fisierului de iesire: "; cin>>numefis;

ofstream fout(numefis);

fout<<n<<" "<<endl;

curent=prim;

fout<<curent->nume<<" "<<curent->lat<<" "<<curent->lng<<" "<<endl;

for(i=2; i<=n; i++){

curent=curent->post;

fout<<curent->nume<<" "<<curent->lat<<" "<<curent->lng<<" "<<endl;

}

getch();

}

void caut(){

clrscr();

cout<<"Caut un nod cu un continut dat"<<endl;

cout<<"In constructie. Voi cauta un nod cu un continut dat"<<endl;

cout<<"Introduceti nume orasului cautat: "; gets(oras);

cout<<"cauti orasul "<<oras<<endl<<endl;

poz=0;

i=1; int gasit=0;

curent=prim;

while(i<=n && !gasit){

cout<<"sunt la orasul "<<curent->nume<<endl;

cout<<"compar orasul "<<oras<<" cu "<<curent->nume<<endl;

if(!strcmp(oras,curent->nume)) {poz=i; gasit=1; cout<<"atentie!"<<endl;break;}

cout<<"gasit="<<gasit<<endl;

curent=curent->post;

i++;

}

if(gasit) cout<<"am gasit orasul pe pozitia "<<poz<<endl;

else cout<<"n-am gasit orasul"<<endl;

getch();

}

void meniu2(){

int rasp;

clrscr();

cout<<"OPERATII CU LISTE :"<<endl<<endl;

cout<<endl;

cout<<" 1: listare de la stanga la dreapta"<<endl;

cout<<" 2: listare de la dreapta la stanga"<<endl;

cout<<" 3: inserare nod in interiorul listei"<<endl;

cout<<" 4: inserare nod pe prima pozitie"<<endl;

cout<<" 5: inserare nod pe ultima pozitie"<<endl;

cout<<" 6: stergerea unui nos din interiorul listei"<<endl;

cout<<" 7: stergerea primului nod"<<endl;

cout<<" 8: stergerea ultimului nod"<<endl;

cout<<" 9: modificarea continutului unui nod"<<endl;

cout<<"10: reprezentare grafica"<<endl;

cout<<"11: salvez lista"<<endl;

cout<<"12: caut dupa continut"<<endl<<endl;

cin>>rasp;

switch (rasp){

case 1: listarestdr();break;

case 2: listaredrst();break;

case 3: insmijloc();break;

case 4: insinaint();break;

case 5: inscoada();break;

case 6: stergint();break;

case 7: stergprim();break;

case 8: stergult();break;

case 9: modific();break;

case 10: grafic();break;

case 11: salvez();break;

case 12: caut();break;

default: exit(1);

}

meniu2();

}

int citire(){

clrscr();

cout<<"Citesc lista din fisier"<<endl;

cout<<"Numele fisierului de intrare: "; cin>>numefis;

ifstream fin(numefis);

fin>>n; cout<<"in fisier sunt "<<n<<" orase"<<endl;

prim = new nod;

fin>>(prim -> nume);

fin >> prim -> lat;

fin >> prim -> lng;

prim -> ante = 0;

prim -> post = 0;

ultim = prim;

cout<<"1"<<" "<<prim -> nume<<" "<<prim -> lat<<" "<<prim -> lng<<" "<<prim -> ante<<" "<<prim -> post<<endl;

for(i=2; i<=n; i++){

ajut = new nod;

fin>>(ajut -> nume);

fin >> ajut -> lat;

fin >> ajut -> lng;

ajut -> ante = ultim;

ultim -> post = ajut;

ajut -> post = 0;

ultim = ajut;

}

getch();

return 0;

}

void meniu1(){

int rasp;

clrscr();

cout<<"EXPLOATAREA LISTELOR INLANTUITE"<<endl<<endl;

cout<<"1 Citire lista din fisier"<<endl;

cout<<"2 Lista noua"<<endl<<endl;

cin>>rasp;

switch (rasp){

case 1: citire();meniu2();break;

case 2: creare();meniu2();break;

default: exit(1);

}

}

void main(){

clrscr();

meniu1();

}

Un exemplu de fişier de intrare este “orase.txt”, având conţinutul:

10

Bucuresti 44.426765 26.102537

Cluj 46.77121 23.623634

Constanta 44.159801 28.634813

Iasi 47.158455 27.601442

Timisoara 45.748871 21.208679

Craiova 44.330177 23.794882

Suceava 47.663452 26.27323

Brasov 45.657974 25.601198

Ploiesti 44.936665 26.012861

Oradea 47.046501 21.918943

* 1. **TEME DE LABORATOR**

Să se rescrie subrutinele de mai sus, pentru cazul listelor liniare simlu înlănţuite.