|  |  |  |
| --- | --- | --- |
| unitbv.emf | **Universitatea *Transilvania* din Brașov**  **Facultatea de Inginerie Electrică și Știința Calculatoarelor**  **Departamentul Automatică şi Tehnologia Informaţiei** | iesc.emf |

**Tema PCLP3**

|  |  |
| --- | --- |
| Andrei Dumitru Vodă  *Gr. 4432, AIA II* | andrei1.jpg |

**BRAȘOV, 2014**

**Tema 1**

Sa se scrie un program care sa genereze, pe baza unui fisier bitmap de intrare, trei fisiere bitmap, care prezinta separat cele trei componente de culoare.

**Codul:**

**package** Tema;

**import** java.io.BufferedInputStream;

**import** java.io.BufferedOutputStream;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**public** **class** Exercitiul1 {

**public** **static** **void** main(String[] args)

{

**try**

{

FileInputStream fis=**new** FileInputStream("D:\\pclp\\colgate.bmp");

BufferedInputStream dis=**new** BufferedInputStream(fis);

FileOutputStream sif=**new** FileOutputStream("D:\\pclp\\colgateR.bmp");

BufferedOutputStream bos = **new** BufferedOutputStream(sif);

FileOutputStream sif2=**new** FileOutputStream("D:\\pclp\\colgateB.bmp");

BufferedOutputStream bos2 = **new** BufferedOutputStream(sif2);

FileOutputStream sif3=**new** FileOutputStream("D:\\pclp\\colgateG.bmp");

BufferedOutputStream bos3 = **new** BufferedOutputStream(sif3);

**byte**[] sti=**new** **byte**[54];

dis.read(sti,0,54);

bos.write(sti);

bos2.write(sti);

bos3.write(sti);

**while**(dis.available()>0) {

**int** b,g,r;

b=dis.read();

g=dis.read();

r=dis.read();

**int** gri=(**int**)(0.587\*g+0\*r+0\*b);

**int** rosu=(**int**)(0.299\*r+0\*g+0\*b);

**int** albastru=(**int**)(0.144\*b+0\*r+0\*g);

bos.write(rosu);

bos.write(rosu);

bos.write(rosu);

bos2.write(albastru);

bos2.write(albastru);

bos2.write(albastru);

bos3.write(gri);

bos3.write(gri);

bos3.write(gri);

}

dis.close();

bos.close();

bos2.close();

bos3.close();

}

**catch** (Exception e)

{

System.*out*.println("ERROR");

}

}

}

Imaginea originala:



Obţinerea versiunii în tonalităţi de gri:



**Tema 2**

Sa se modifice programul de mai sus prin implementarea optiunii de eliminare a elementelor negative din lista (optiunea 4).

**Codul:**

#include<iostream>

#include<conio.h>

#include<stdio.h>

struct Nod

{

int inf;

Nod\* urm;

};

int NumarDeElemente(Nod\*lista)

{

int numar=0;

Nod\*aux=lista;

while(aux!=NULL)

{

numar++;

aux=(\*aux).urm;

}

return numar;

}

void Adauga(Nod \*&lista, int val, int N)

{

if (N==1 || lista == NULL)

{

Nod\* aux=new Nod;

aux->inf=val;

aux->urm=lista;

lista=aux;

}

else

{

int nr=NumarDeElemente(lista);

if(N>nr+1)

{

N=nr+1;

}

Nod \*temp=new Nod;

temp->inf = val;

Nod \*aux = lista;

for(int i=1;i<N-1;++i)

{

aux = aux->urm;

}

temp->urm = aux->urm;

aux->urm=temp;

}

}

void Afisare(Nod \*lista)

{

Nod \*aux = lista;

std::cout<<"Lista este: ";

while(aux != NULL)

{

std::cout<<aux->inf<<" ";

aux=aux->urm;

}

if (lista == NULL)

std::cout<<"vida.";

else std::cout<<"\n";

}

void EliberareMem(Nod \*&lista)

{

Nod\* aux = lista;

while(aux != NULL)

{

lista = lista->urm;

delete aux;

aux = lista;

}

}

void Elimina(Nod \*&lista, int poz)

{

Nod\* aux = lista;

if (lista == NULL)

return;

int nr=NumarDeElemente(lista);

if(poz>nr)

poz=nr;

if(poz==1)

{

lista=lista->urm;

delete aux;

}

else

{

for(int i=1;i<poz-1;++i)

{

aux=aux->urm;

}

Nod\*temp=aux->urm;

aux->urm=temp->urm;

delete temp;

}

}

void EliminaNegative(Nod\*lista)

{

int numar=NumarDeElemente(lista);

Nod \*aux=lista;

for(int i=0;i<numar;++i)

{

if(aux->inf<0)

{

Elimina(lista,i+1);

}

aux=aux->urm; //se trece la urm el.

}

}

int main()

{

Nod \*L = nullptr;

int opt=0;

do

{

std::cout<<"Operatii"<<"\n------\n"<<"1 - Adaugare element\n";

std::cout<<"2 - Eliminare element\n";

std::cout<<"3 - Afisare lista\n";

std::cout<<"4 - Eliminare elemente negative\n";

std::cout<<"0 - Iesire\n";

std::cin>>opt;

switch (opt)

{

case 1:

{

int v,p;

std::cout<<"\nIntroduceti valoarea: ";

std::cin>>v;

std::cout<<"Introduceti pozitia (lista are "<<NumarDeElemente(L)<<" el): ";

std::cin>>p;

Adauga(L,v,p);

Afisare(L);

break;

}

case 2:

{

int p;

std::cout<<"\nPozitia (lista are "<<NumarDeElemente(L)<<" el): ";

std::cin>>p;

Elimina(L,p);

Afisare(L);

break;

}

case 3:

{

std::cout<<"\n";

Afisare(L);

break;

}

case 4:

{

EliminaNegative(L);

break;

}

}

fflush(stdin);

\_getch();

}

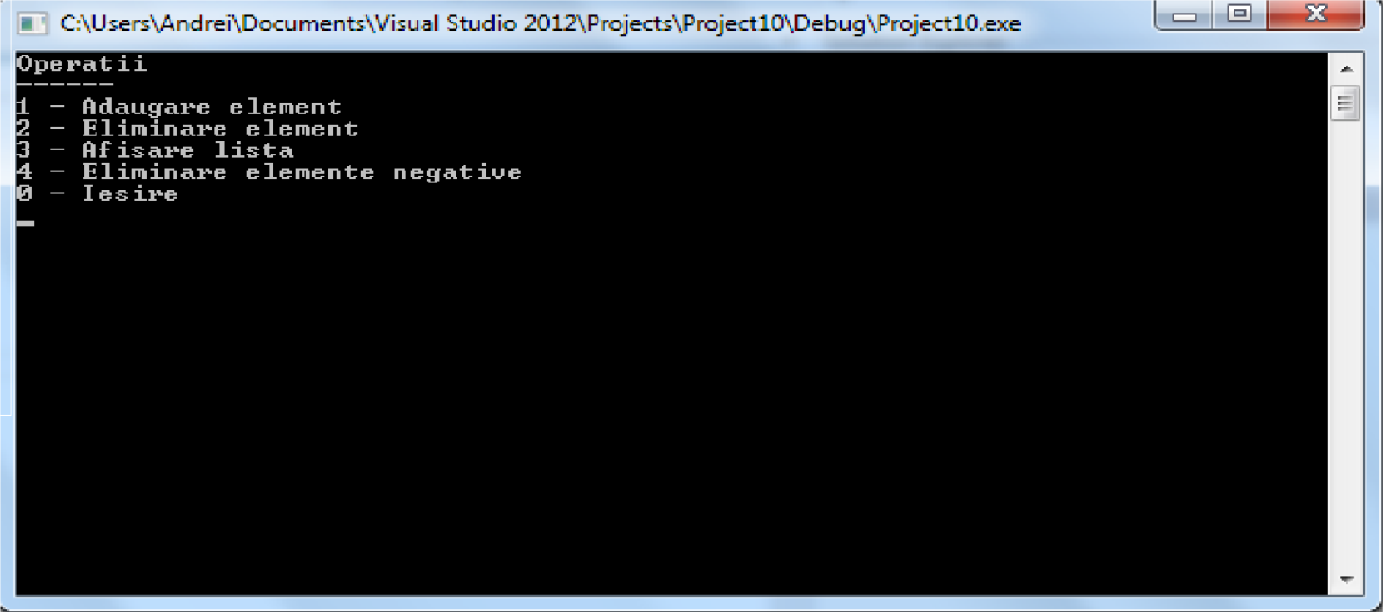
while(opt!=0);

EliberareMem(L);

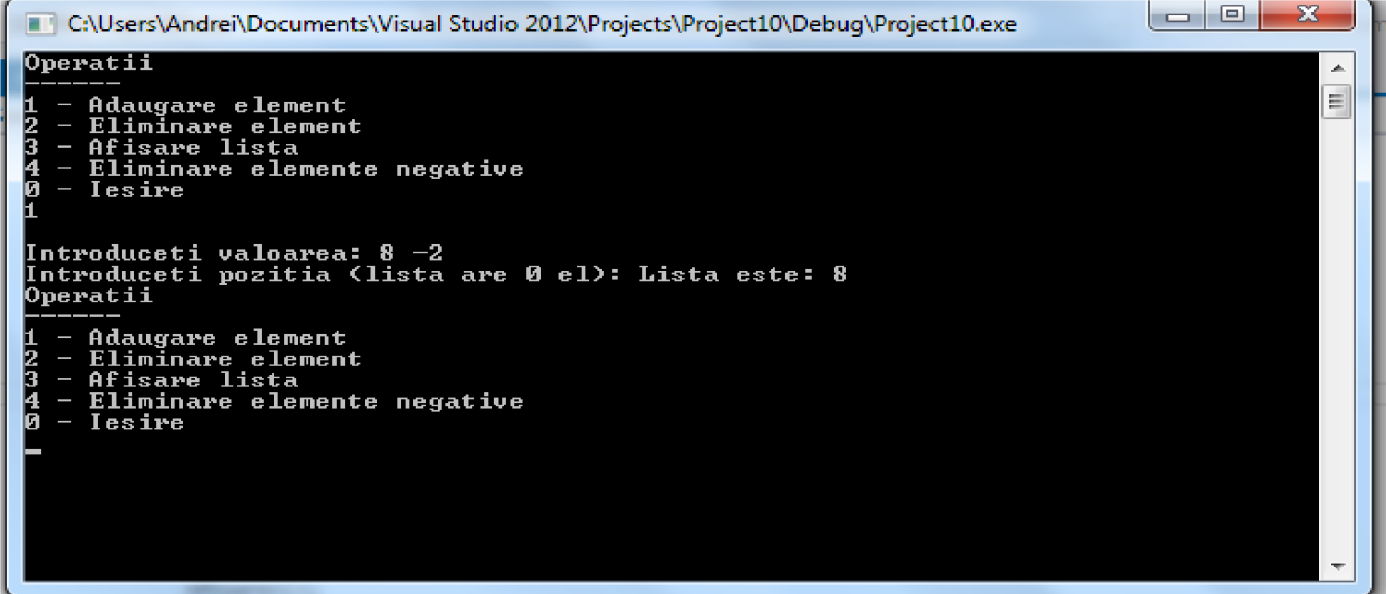
return 0;

}

**Meniul de operatii:**

****

**Eliminarea elementului negativ:**

****

**Tema 3**

Realizati o comparatie între numarul de interschimbari realizate de fiecare algoritm prezentat pentru un sir de numere sortat crescator.

**Codul:**

#include<conio.h>

#include<iostream>

static int quick=0;

static int buble=0; //prin astea se stie nr de interschimbari.

int getFixedPos(int \*T, int startpos, int stoppos)

{

int i=startpos, j=stoppos, mod=0;

while (i<j)

{

if (T[i]>T[j])

{

quick++;

int aux = T[i];

T[i] = T[j];

T[j] = aux;

mod = !mod;

}

if (mod)i++;

else

j--;

}

return i;

}

void quicksort(int \*T, int startpos, int stoppos)

{

if (startpos<stoppos)

{

int k = getFixedPos(T,startpos,stoppos);

quicksort(T,startpos,k-1);

quicksort(T,k+1,stoppos); //treaba ii recursiva, se reapeleaza singura pana rezolva treaba

}

}

void BubleSort(int \*TAB,int numar\_elem)

{

bool sortat=false;

int i=0,aux;

while(!sortat)

{

sortat=true;

for(i=0;i<numar\_elem-1;++i)

{

if(TAB[i]>TAB[i+1])

{

aux = TAB[i];

TAB[i] = TAB[i+1];

TAB[i+1] = aux;

++buble;

sortat=false;

}

}

}

}

int main()

{

int TABq[100],TABb[100],n,i=0;

std::cout<<"Numarul de elemente ale vectorului: ";

std::cin>>n;

for(;i<n;i++)

{

std::cout<<"Elementul "<<i+1<<" este: ";

std::cin>>TABq[i];

TABb[i]=TABq[i];

}

quicksort(TABq,0,n-1);

BubleSort(TABb,n);

std::cout<<"Vectorul sortat este: ";

for (i=0;i<n;i++)

{

std::cout<<TABq[i]<<" ";

}

std::cout<<"\nVectorul sortat de buble este: ";

for (i=0;i<n;i++)

{

std::cout<<TABb[i]<<" ";

}

std::cout<<"\nS-au realizat "<<quick<<" interschimbari quick\n si "<<buble<<" interschimbari buble";

if(buble==quick)

{

std::cout<<"ambele metode au realizat la fel de multe interschimari";

}

else

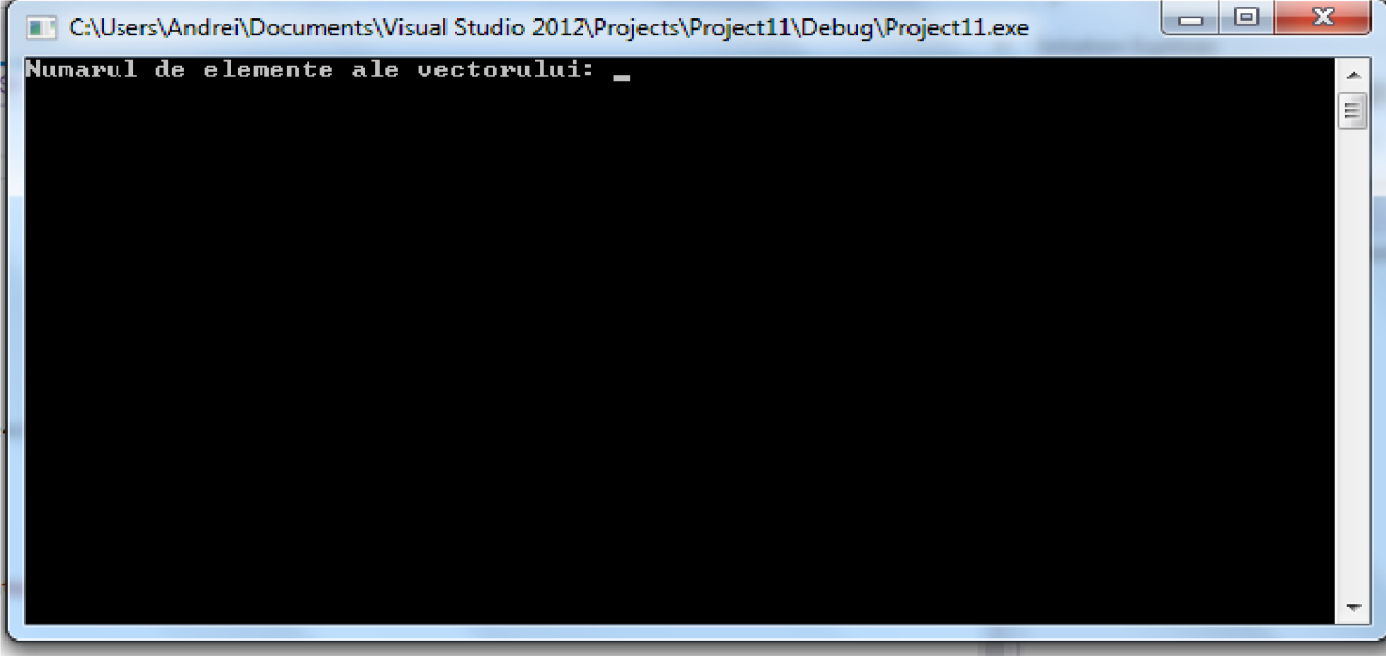
std::cout<<"\n Buble sort a realizat cu "<<buble-quick<<" mai multe interschimbari";

\_getch();

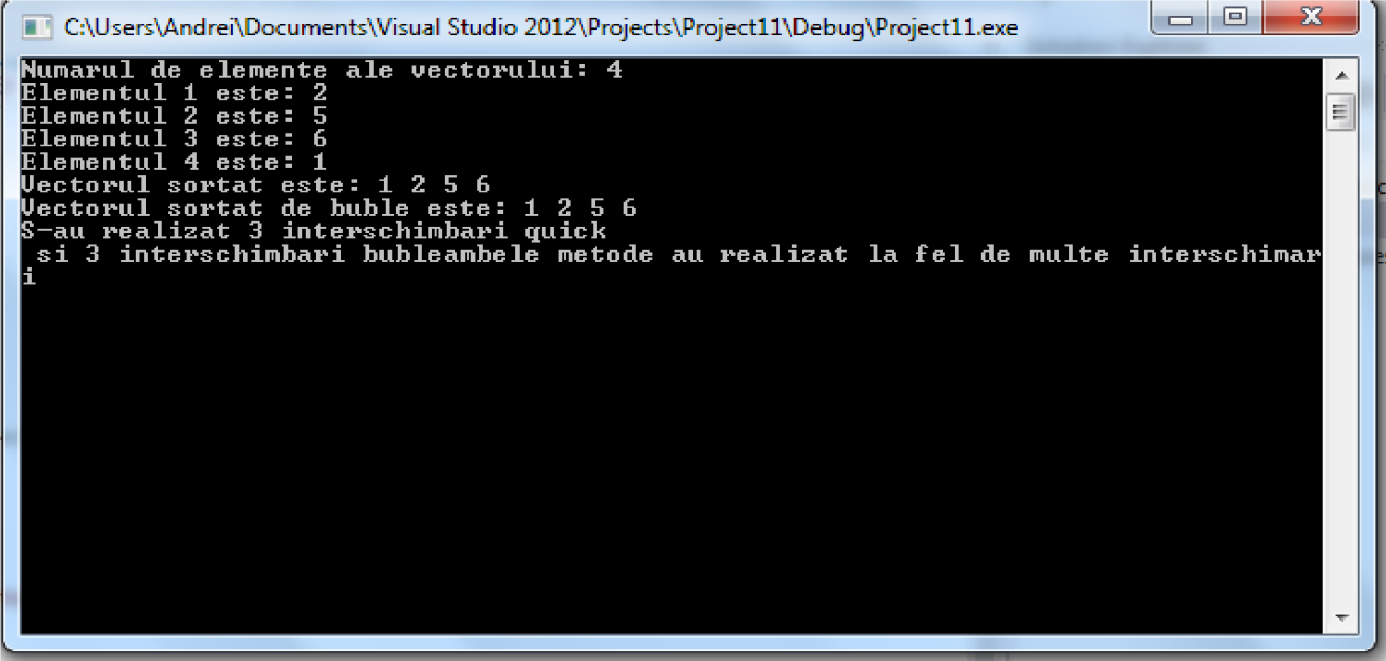
return 0;

}

**Se introduc elementele sirului:**

****

**Sortarea elementelor+diferenta dintre bubble si quick:**

****