PROIECTAREA ALGORITMILOR

Lect. univ. dr. Adrian Runceanu

Câteva precizări

Structura cursului

- ✓2 ore curs – titular curs
  Lector dr. Adrian Runceanu
- ✓2 ore laborator – titular aplicaţii practice
  Lector dr. Adrian Runceanu

Câteva precizări

Forme de examinare:

- Examen final – 60%
- Evaluare pe parcursul semestrului a activităţii de laborator – 30%
- Prezenţă curs şi laborator – 10%

Câteva precizări

Bibliografia necesară cursului:

1. Dogaru, O., *Tehnici de programare*, Editura MIRTON, Timişoara, 2002, 2004
2. Creţu, V., *Structuri de date şi algoritmi*, vol.1 – *Structuri de date fundamentale*, Editura Orizonturi Universitare, Timişoara, 2000
3. Livovschi, L.,Georgescu, H., *Sinteza şi Analiza algoritmilor*, Editura Ştiinţifică şi Enciclopedică, Bucureşti, 1986
4. Wirth, N., *Algorithms and Data Structures*, Prentice Hall, Inc.,Englewood, New Jersey, 1986
5. Dr. Kris Jamsa & Lars Klander, *Totul despre C si C++ - Manualul fundamental de programare în C si C++,* ed. Teora, 1999-2006

Câteva precizări

Bibliografia necesară cursului:

6. Liviu Negrescu, *Limbajele C si C++ pentru începători*, vol. II, Limbajul C++, ed. MicroInformatica, 1995
7. A.Runceanu, **Metode si tehnici de programare – indrumar de laborator**, Editura Academica Brancusi Targu-Jiu, 2003
8. *Horia Ciocârlie,* Tehnici fundamentale de programare*, Ed. Orizonturi Universitare, 2002*
9. Pagina web pentru curs:
http://www.runceanu.ro/adrian

Câteva precizări

Referinţele bibliografice nr. 1 şi 7 se pot împrumuta de la Biblioteca Facultăţii de Inginerie, Str. Geneva nr.3, Etaj I – lângă Decanat.

1. Suport curs - varianta electronică disponibilă pe site–ul:

http://www.runceanu.ro/adrian

2. Îndrumar de laborator - varianta electronică disponibilă pe site pentru fiecare lucrare de laborator.

Notă: Actualizarea site-ului se face săptămânal.

Conţinutul cursului

- În cadrul acestui curs se vor studia metode şi tehnici de programare pentru elaborarea eficientă a algoritmilor.

- Exemplificările de la curs şi implementările de la laborator se vor efectua cu ajutorul limbajului de programare - **C++.**

- Mediul de dezvoltare utilizat la aplicatiile practice poate fi **MinGW** sau **CodeBlocks**.

Capitolele cursului

1. **Recursivitate**
2. **Alocarea dinamică de memorie în C++**
3. **Liste simplu şi dublu înlănţuite**
4. **Elemente de teoria grafurilor**
5. **Algoritmi pentru prelucrarea grafurilor**
6. **Arbori. Arbori binari**
7. **Metoda Greedy de elaborare a algoritmilor**
8. **Metoda Divide et Impera de elaborare a algoritmilor**
9. **Metoda Backtracking de elaborare a algoritmilor**
10. **Combinatorică**

Curs 1

Recursivitate

Conţinutul cursului

1. **Conceptul de recursivitate**
2. **Recursivitate directă**
3. **Recursivitate indirectă**
4. **Relaţia dintre recursivitate şi iteraţie**
5. **Exemple de programe recursive**

1. Conceptul de recursivitate

- Un obiect sau un fenomen este definit în mod **recursiv** dacă în definitia sa se face referire la el însusi.
- Conceptul de **recursivitate** oferă posibilitatea definirii unei infinităti de obiecte printr-un număr finit de relatii.
- *O functie este recursivă atunci când executarea ei implică cel putin încă un apel către ea însăsi.*

1. Conceptul de recursivitate

Tipuri de recursivitate:

1. *Recursivitate directă* – apelul recursiv se face chiar din functia invocată.

2. *Recursivitate indirectă (mutuală)* – apelul recursiv se realizează prin intermediul mai multor functii care se apelează circular.

Exemplul 1

Definirea numerelor naturale:

- 1 este număr natural
- succesorul unui număr natural este un număr natural

Se presupune cunoscută *definiţia succesorului unui număr: acel număr obţinut din numărul dat prin adăugarea unei unităţi*.

Exemplul 2

Algoritm de calcul pentru factorialul unui număr n. (notatie n!):

- dacă n = 0 atunci n! = 1
- dacă n > 0 atunci n! = n * (n-1)!

Astfel spus, *factorialul unui număr n > 0 se obţine prin înmulţirea numărului cu factorialul predecesorului*.

Conţinutul cursului

1. **Conceptul de recursivitate**
2. **Recursivitate directă**
3. **Recursivitate indirectă**
4. **Relatia dintre recursivitate si iteratie**
5. **Exemple de programe recursive**

Recursivitate directă

În limbajul C++ functiile se pot apela pe ele însele, adică sunt direct recursive.

Pentru o functionare corectă (din punct de vedere logic), apelul recursiv trebuie să fie conditionat de o decizie care, la un moment dat în cursul executiei, să împiedice continuarea apelurilor recursive si să permită astfel revenirea din sirul de apeluri.

Recursivitate directă

Lipsa acestei conditii sau programarea ei gresită va conduce la executarea unui sir de apeluri a cărui terminare nu mai este controlată prin program si care, la epuizarea resurselor sistemului, va provoca o eroare de executie:

Depăsirea stivei de date.

```
int functie()
{
        return functie();
}
```

Exemplu

void **p (listă de parametri)**{
lista variabile locale
...
p(listă de parametri);
}

if (conditie) **p(listă de parametri)** ...
sau:
while (conditie)
{ ...**p(listă de parametri)**… }
sau:
do { ... **p(listă de parametri)**... } while (conditie)

- Conditia care trebuie testată este specifică problemei de rezolvat.
- *Programatorul trebuie să o identifice în fiecare situatie concretă si, pe baza ei, să redacteze corect apelul recursiv.*
- Revenirea din apeluri se face în ordine inversă.

Conţinutul cursului

1. **Conceptul de recursivitate**
2. **Recursivitate directă**
3. **Recursivitate indirectă**
4. **Relatia dintre recursivitate si iteratie**
5. **Exemple de programe recursive**

Recursivitate indirectă

- Un subprogram S, în corpul căruia apar apeluri la S (la el însuşi) se numeşte subprogram **direct recursiv** iar un subprogram P, pentru care există un subprogram Q, astfel încât P face apeluri la Q, iar Q conţine apelul la P se numeşte subprogram **indirect recursiv**.
- În acest ultim caz, subprogramele P şi Q se mai numesc şi **mutual recursive**.

Recursivitate indirectă

Funcție direct recursivă

```
functia S;
 {
      …
      S; // apel la functia S
      …
 }
```

Funcții mutual recursive

```
functia P;
 {
      …
      Q ; // apel la functia Q
      …
 }
functia Q;
 {
      …
      P ; // apelul functiei P
      …
 }
```

Conţinutul cursului

1. **Conceptul de recursivitate**
2. **Recursivitate directă**
3. **Recursivitate indirectă**
4. **Relatia dintre recursivitate si iteratie**
5. **Exemple de programe recursive**

Relaţia dintre recursivitate şi iteraţie - Comparaţie

Iteraţia

- executia repetată a unei secvente de instructiuni
- o nouă iteratie se execută doar în urma evaluării unei conditii (**la început sau sfârsit**)
- fiecare iteratie se execută până la capăt si apoi se trece, eventual, la o nouă iteratie
- se recomandă atunci când algoritmul de calcul este exprimat printr-o formulă iterativă

Recursivitatea

- executia repetată a unei functii
- un nou apel recursiv se execută tot în urma evaluării unei conditii (**pe parcurs**)
- functia recursivă se apelează din nou, înainte de terminarea apelului precedent
- se recomandă doar atunci când problema este prin definitie recursivă (recursivitatea consumă resurse în exces)

Conţinutul cursului

1. **Conceptul de recursivitate**
2. **Recursivitate directă**
3. **Recursivitate indirectă**
4. **Relatia dintre recursivitate si iteratie**
5. **Exemple de programe recursive**

Probleme rezolvate

1. Se dau doua numere intregi a si b si se cere sa se calculeze cel mai mare divizor comun. (**Algoritmul lui EUCLID – prin împărțiri repetate**).

Formularea recursivă, în cuvinte, a algoritmului:

- *Dacă unul dintre numere este zero, c.m.m.d.c. al lor este celălalt număr.*
- *Dacă nici unul dintre numere nu este zero, atunci c.m.m.d.c. nu se modifică dacă se înlocuieste unul dintre numere cu restul împărtirii sale cu celălalt.*

Probleme rezolvate

Algoritmul poate fi implementat sub forma următoarei functii recursive:

```
int cmmdc (int n, int m)
{
   if (n==0) return m;
   else return cmmdc(n, m % n);
}
```

Probleme rezolvate

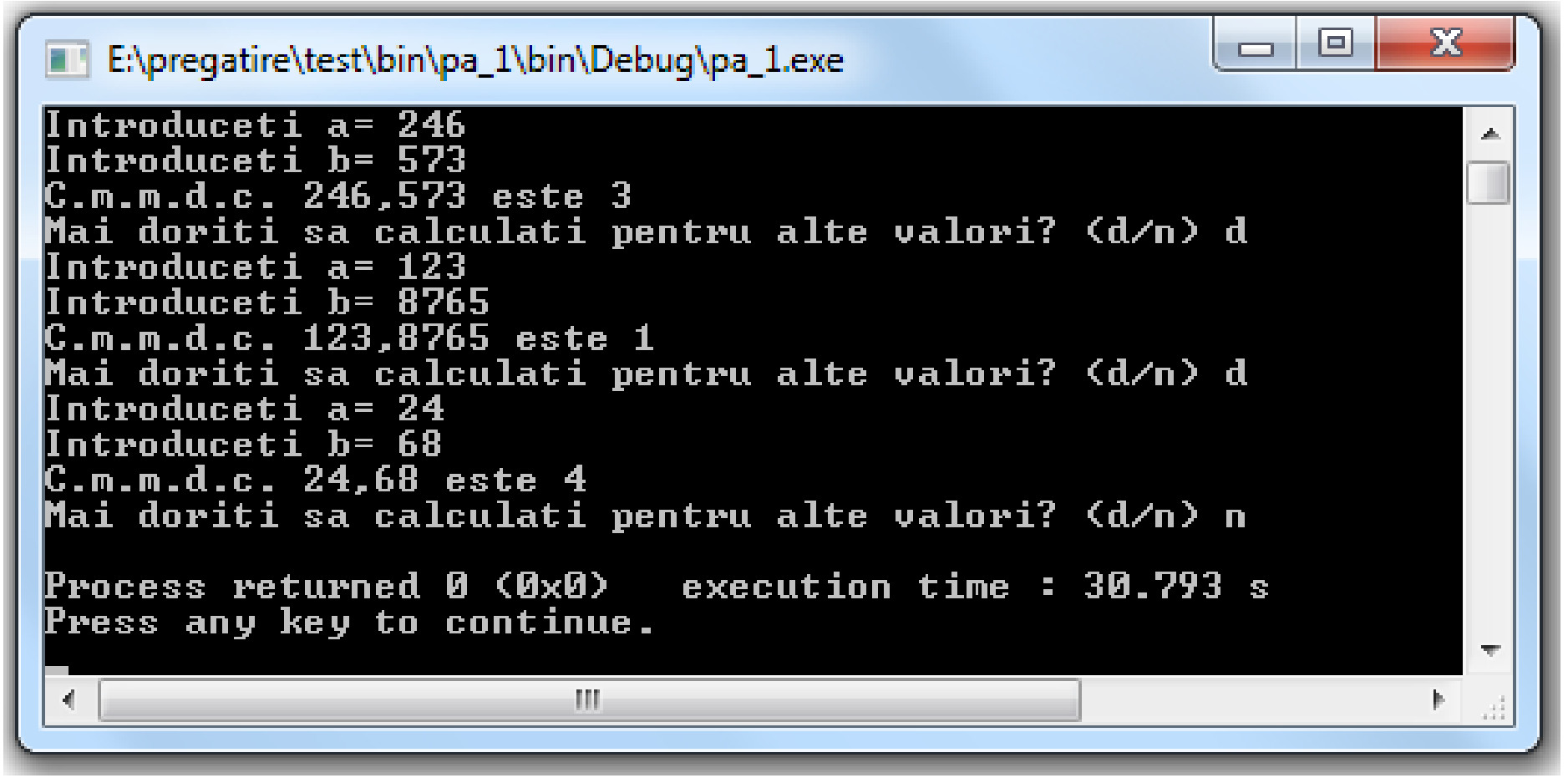
Codul sursa al implementarii (varianta prin scaderi succesive) este urmatorul:

```
#include<iostream.h>

int cmmdc(int a,int b)
{
   if(a==b) return a;
   else if(a>b) return cmmdc(a-b, b);
        else return cmmdc(a, b-a);
}
```

```
int main(void)
{
    int a, b;
    char c;
    do
    {
        cout<<"Introduceti a= "; cin>>a;
        cout<<"Introduceti b= "; cin>>b;
        cout<<"C.m.m.d.c. "<<a<<","<<b<<" este
    "<<cmmdc(a,b)<<endl;
        cout<<"Mai doriti sa calculati pentru alte
    valori? (d/n) ";
        cin>>c;
    }while( toupper(c)!='N' );
}
```

Executia programului pentru cateva date de test:



Probleme rezolvate

2. Să se calculeze suma primelor n numere naturale.

Soluţia este dată de relaţia de recurenţă:

suma(1, 2, . . . , n) =

suma(n, suma(1, 2, . . . , n-1))

```
#include<iostream.h>

long int suma(long int i)
{
   if(i==1) return 1;
   else  return suma(i-1)+i;
}

int main(void)
{
   long int n;
   cout<<"Introduceti n= "; cin>>n;
   cout<<"Suma primelor "<<n<<" numere este
   "<<suma(n)<<endl;
}
```

Executia programului pentru o valoare de test:

Probleme rezolvate

3. Să se afle elementul maxim dintr-un vector dat.

Soluţia este dată de relaţia de recurenţă:

$$maxim(a_1, a_2, \ldots, a_n) =$$

$$maxim(a_n, maxim(a_1, a_2, \ldots, a_{n-1}))$$

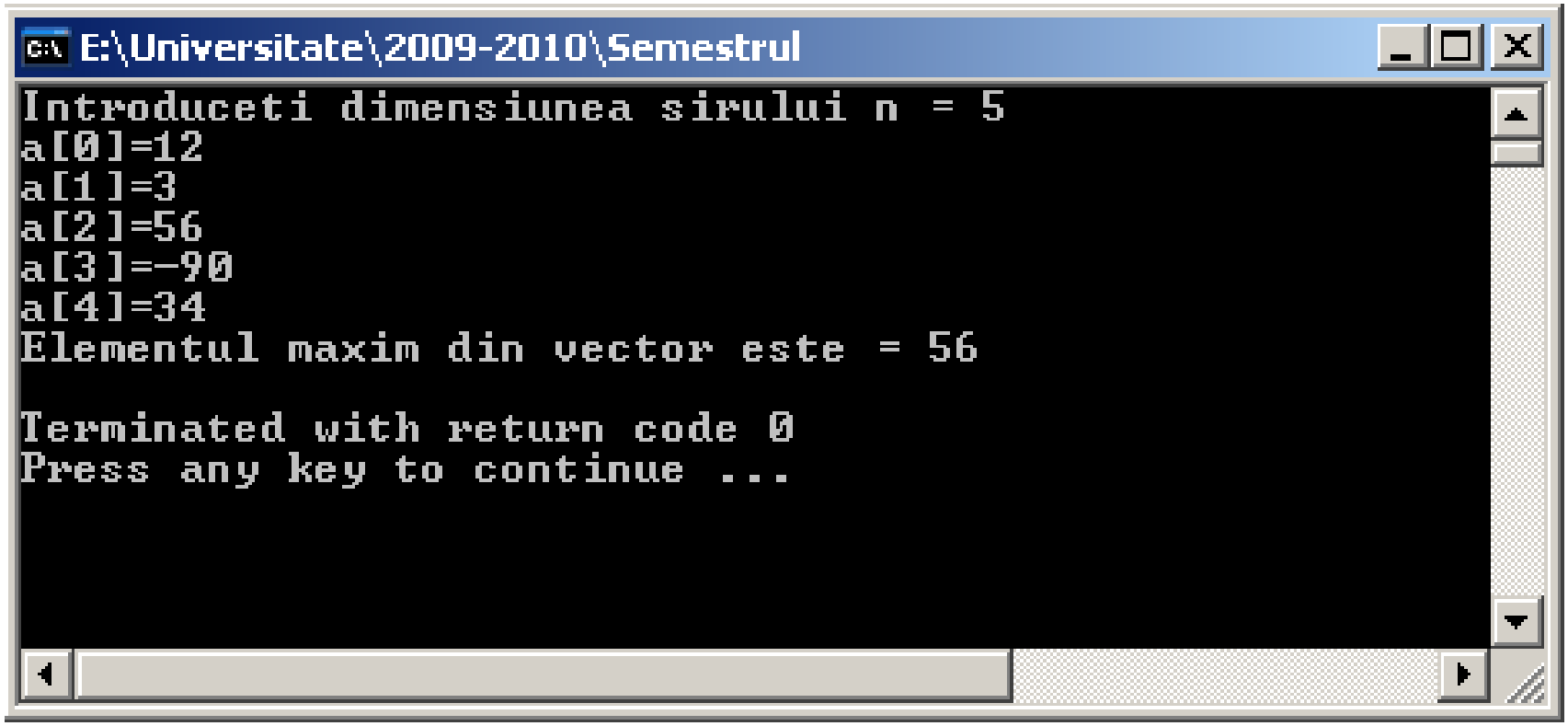
```
#include<iostream.h>
int a[100],n,i;

int max(int x, int y)
{
    if(x > y) return x;
    else return y;
}

int maxim(int a[ ],int n)
{
    if(n==1) return a[1];
        else return
    max(a[n],maxim(a,n-
    1));
}
```

```
int main(void)
{
    cout<<"Introduceti
    dimensiunea sirului n = ";
    cin>>n;
    for(i=1;i<=n;i++)
    {
          cout<<"a["<<i<<"]=";
          cin>>a[i];
    }
    cout<<"Elementul maxim din
    vector este = "<<maxim(a,n);
}
```

Executia programului pentru cateva valori de test:



Probleme rezolvate

4. Sa se transforme un numar n, dat in baza 10, intr-o alta baza b (2<=b<=10).

```
#include<iostream.h>
int n,b;

void baza(int n)
{
    if(n<b) cout<<n;
    else
    {
          baza(n/b);
          cout<<n%b;
    }
}
```

```
int main(void)
{
    cout<<"Dati numarul in baza
    10, n = ";
    cin>>n;
    cout<<"Dati baza in care
    vreti sa se transforme ";
    cin>>b;
    cout<<n<<" in baza "<<b<<"
    este ";
    baza(n);
}
```

Executia programului pentru cateva valori de test:

Probleme rezolvate

5. Se citeste un numar intreg ca un sir de caractere cu cel mult 255 cifre.

Sa se afiseze numarul cu cifrele in ordine inversa.

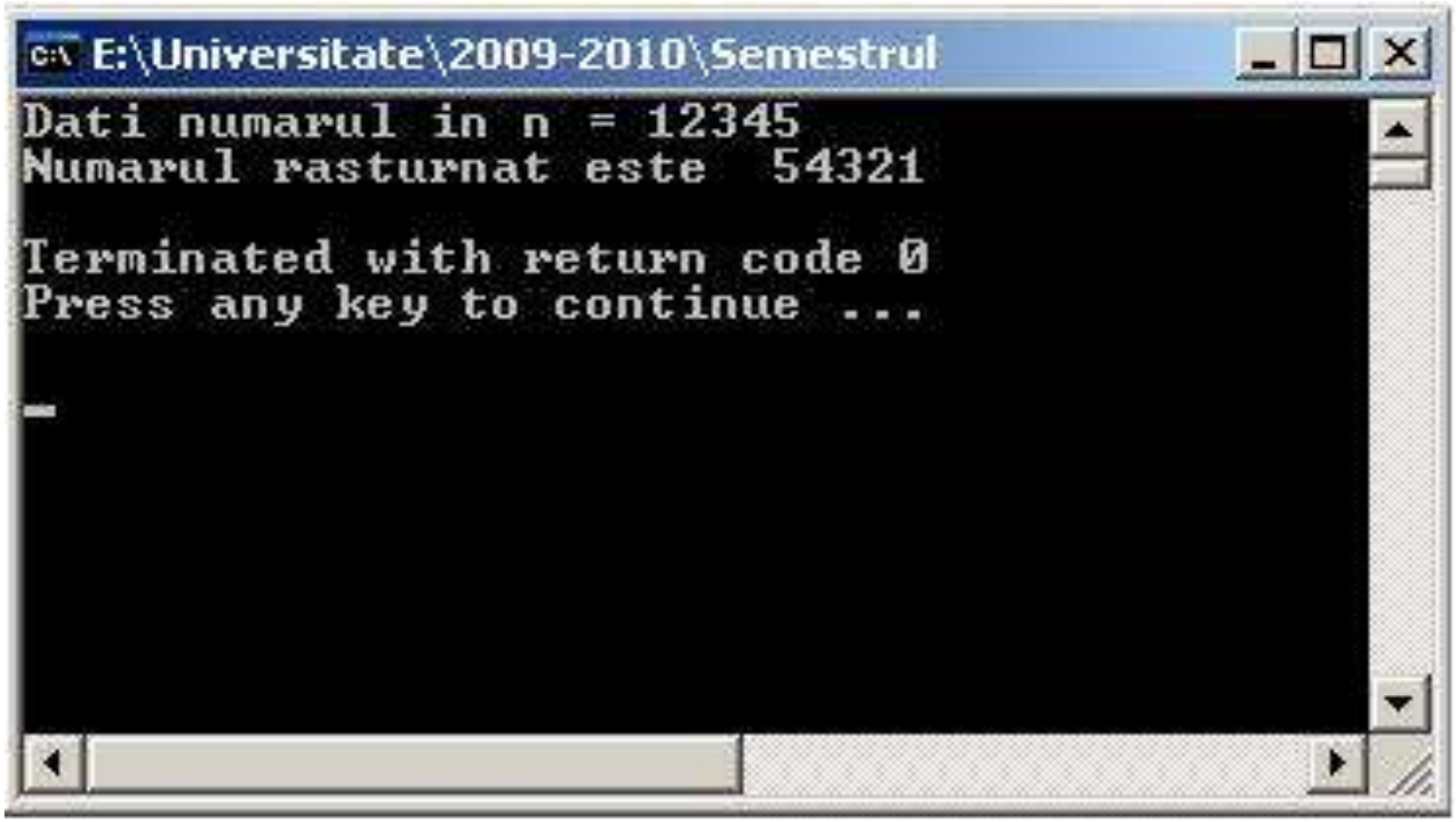
```
#include<iostream.h>
#include<string.h>

char n[255],i,l;

void invers(int i)
{
   if(i<l) invers(i+1);
   cout<<n[i];
}
```

```
int main(void)
{
   cout<<"Dati numarul in n = ";
   cin>>n;
   l=strlen(n);
   cout<<"Numarul rasturnat
   este ";
   invers(0);
}
```

Executia programului pentru o valoare de test:



Probleme rezolvate

6. Suma puterilor rădăcinilor

Fie ecuaţia $x^2 - Sx + P = 0$ cu $S, P \in R$ si x_1, x_2 rădăcinile ecuaţiei.

Să se calculeze $S_n = x_1^n + x_2^n$, $n \in N$.

Căutăm relaţia de recurenţă pentru S_n, ştiind că x_1, respectiv x_2 sunt rădăcinile ecuaţiei date şi deci îndeplinesc relaţiile:

$$x_1^2 - Sx_1 + P = 0 \quad | * x_1^{n-2}$$

$$x_2^2 - Sx_2 + P = 0 \quad | * x_2^{n-2}$$

Înmulţim aceste relaţii cu x_1^{n-2} şi x_2^{n-2} şi adunăm relaţiile obţinute şi rezultă:

$$S_n = x_1^n + x_2^n =$$

$$= S * (x_1^{n-1} + x_2^{n-1}) - P * (x_1^{n-2} + x_2^{n-2}) =$$

$$= S * S_{n-1} - P * S_{n-2}$$

Astfel am obţinut o relaţie de recurenţă:

$S_0 = x_1^1 + x_2^1 = 1 + 1 = 2$, pentru n=0

$S_1 = x_1^1 + x_2^1 = S$, pentru n=1

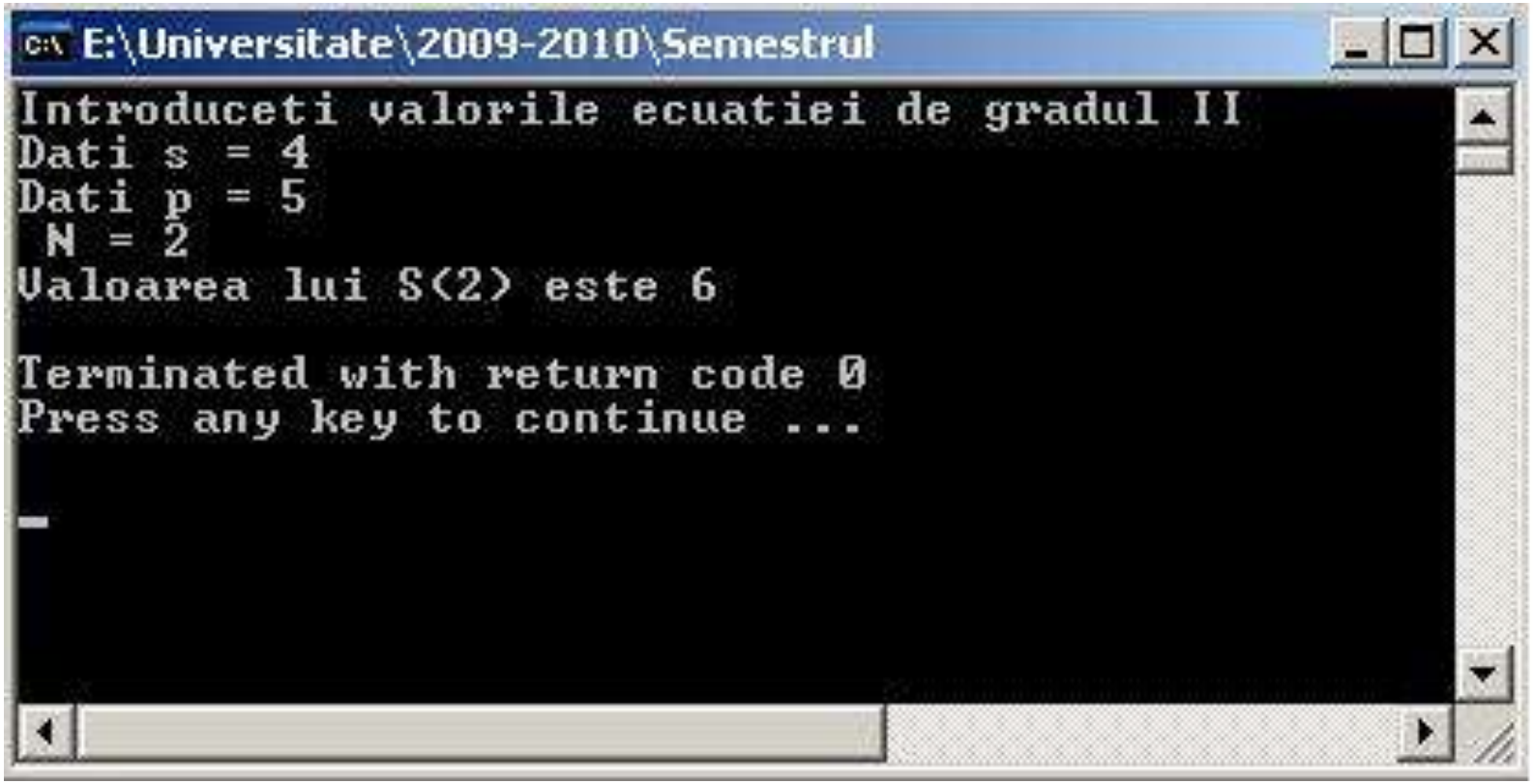
$S_n = S * S_{n-1} - P * S_{n-2}$, pentru $n \geq 2$

```
#include<iostream.h>
int n;
float s,p,r;

float suma(int n)
{
    if(n==0) return 2;
    else if(n==1) return s;
         else
    return(s*suma(n-1)-
    p*suma(n-2));
}
```

```
int main(void)
{
    cout<<"Introduceti valorile
    ecuatiei de gradul II
    "<<endl;
    cout<<"Dati s = ";cin>>s;
    cout<<"Dati p = ";cin>>p;
    cout<<" N = ";cin>>n;
    r=suma(n);
    cout<<"Valoarea lui
    S("<<n<<") este "<<r;
}
```

Executia programului pentru un set de valori de test:



Probleme propuse spre rezolvare

1. Să se scrie un program care să calculeze al n-lea termen din şirul lui Fibonacci, care este definit recursiv astfel:
   - fib[1]=0
   - fib[2]=1
   - fib[n]=fib[n-1] + fib[n-2], pentru n>2
2. Să se caute o soluţie nerecursivă pentru şirul lui Fibonacci.

Întrebări?