

Pointeri

Am utilizat in aplicatiile anterioare declaratii de genul:

```
char* sectia = "Inginerie Electrica";  
String^ nume = "Ionescu"  
System::Drawing::Graphics^ Desen;  
System::Random^ n;
```

In cazurile enumerate mai sus am definit diverse tipuri de variabile. Am definit astfel variabila **sectia** ce reprezinta un sir de caractere, pe care l-am initializat cu valoarea "Inginerie Electrica". In spatiul de nume **System::** am definit variabile de tip sir de caractere folosind de exemplu : **String^ nume = "Ionescu"** pentru a defini variabila **nume** de tip "String", pe care am initializat-o cu valoarea "Ionescu". In WFA, am declarat deseori variabile mai complexe de tipul: Graphics, Random etc. In toate aceste cazuri am folosit defapt **pointeri**

Pointeri

Pointerii sunt variabile ale caror valori sunt adresele altor variabile (obiecte). Valoarea memorata deci într-o variabila pointer este o adresa.

Pointerii sunt utili pentru a usura unele operatii in C++ dar mai ales pentru operatii de alocare dinamica a memoriei, operatie imposibil de realizat fara ajutorul pointerilor.

In functie de continutul zonei de memorie adresate, pointerii pot fi:

- pointeri de date (obiecte) - contin adresa unei variabile din memorie;
- pointeri generici (numiti si pointeri void) - contin adresa unui obiect oarecare, de tip generic de pointer;
- pointeri de functii - contin adresa codului executabil al unei functii.

• Declararea unui pointer

Sintaxa pentru declararea unui pointer:

```
tip* identificador_pointer;
```

Tinand cont ca un pointer reprezinta o adresa, deci ele ar trebui sa fie de tip intreg, totusi tipul unui pointer este de tipul variabilei a carei adresa o reprezinta.

• Initializarea unui pointer

Utilizarea variabilelor pointer se face folosind urmatoorii operatori unari:

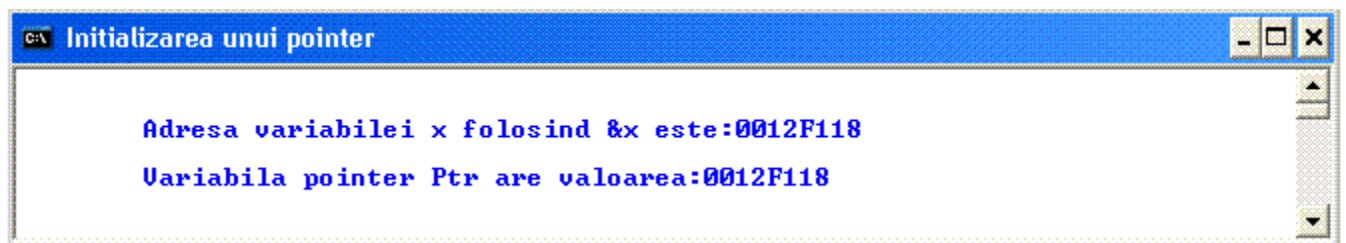
- **&** - operatorul adresa (de referentiere) - pentru aflarea adresei din memorie a unei variabile;

- * - operatorul de indirectare (de deferentiere) - care furnizeaza valoarea din zona de memorie spre care pointeaza pointerul operand.

Sa realizam o aplicatie simpla in care sa evidentiem faptul ca un pointer este adresa unei variabile.

```
// Programul utilizeaza un pointer Ptr
// Se initializeaza o variabila pointer Ptr catre un intreg
// Se defineste o variabila de tip int si se afiseaza adresa acesteia
// Se atribuie o valoare pointer-ului si se afiseaza
#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

int main(void)
{
system("TITLE Initializarea unui pointer "); // Titlul ferestrei consola
system("COLOR F9"); // Fundal alb caractere albastre
int* Ptr ; // Ptr- variabila pointer catre un int
int x;
cout << "\n\n\tAdresa variabilei x folosind &x este:" << &x;
Ptr=&x; // Variabila Ptr contine adresa variabilei x
cout << "\n\n\tVariabila pointer Ptr are valoarea:" << Ptr;
cin.ignore();
cin.get();
}
```



Dupa cum se observa in imaginea de sus, adresa si valoarea pointer-ului este aceeasi.

Urmatoarea aplicatie evidentiaza faptul la o variabila se poate ajunge prin intermediul pointerului ce indica spre aceasta variabila. Se foloseste pentru aceasta operatorul de indirectare care furnizeaza valoarea variabilei indicate de pointer.

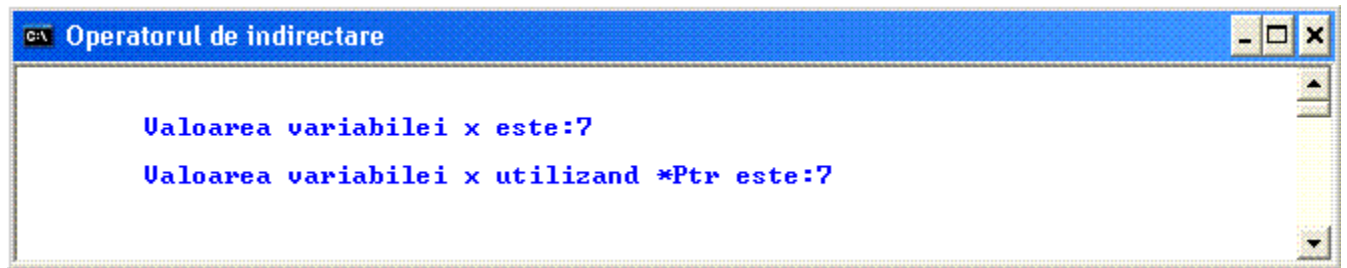
```
// Programul utilizeaza un pointer Ptr
// Se initializeaza o variabila pointer Ptr catre un intreg
```

```

// Se defineste o variabila x de tip int si se initializeaza cu 7 si se
afiseaza
// Se atribuie pointer-ului adresa variabilei x
// Folosind operatorul de indirectare, se afiseaza valoarea variabilei x
#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

int main(void)
{
system("TITLE Operatorul de indirectare "); // Titlul ferestrei consola
system("COLOR F9"); // Fundal alb caractere albastre
int* Ptr ; // Ptr- variabila pointer catre un int
int x=7;
cout << "\n\n\tValoarea variabilei x este:" << x;
Ptr=&x; // Variabila Ptr contine adresa variabilei x
cout << "\n\n\tValoarea variabilei x utilizand *Ptr este:" << *Ptr;
cin.ignore();
cin.get();
}

```



Dupa cum era de asteptat, se afiseaza aceeasi valoare utilizand cele doua metode.

Valoarea unei variabile poate fi modificata direct sau prin intermediul pointerului care indica spre ea. Urmatoarea aplicatie scoate in evidenta acest lucru.

```

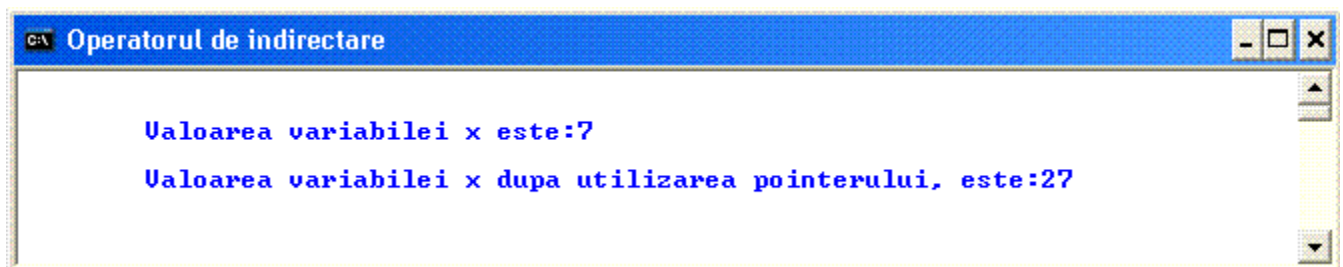
// Programul utilizeaza un pointer Ptr
// Se initializeaza o variabila pointer Ptr catre un intreg
// Se defineste o variabila x de tip int si se initializeaza cu 7 si se
afiseaza
// Se atribuie pointer-ului adresa variabilei x
// Folosind operatorul de indirectare, se modifica valoarea variabilei x
// Se afiseaza valoarea variabilei x
#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

```

```

int main(void)
{
system("TITLE Operatorul de indirectare "); // Titlul ferestrei consola
system("COLOR F9"); // Fundal alb caractere albastre
int* Ptr ; // Ptr- variabila pointer catre un int
int x=7;
cout << "\n\n\tValoarea variabilei x este:" << x;
Ptr=&x; // Variabila Ptr contine adresa variabilei x
*Ptr=27; // Se atribuie valoarea 27 continutului adresei Ptr
cout << "\n\n\tValoarea variabilei x dupa utilizarea pointerului, este:" <<
x;
cin.ignore();
cin.get();
}

```



Valoarea variabilei x a fost modificata prin doua cai.

• Pointeri nuli

Un pointer nu poate fi folosit daca nu i se atribuie o valoare. In cazul in care inca nu se cunoaste inca valoarea ce trebuie sa-i fie atribuita, aceasta i se atribuie valoarea NULL. Nu putem sa atribuim valoarea 0 deoarece acest pointer ar indica spre locatia de la adresa 0, adresa care contine cu siguranta o valoare nenula.

In urmatorul exemplu se utilizeaza un pointer Ptr NULL.

```

// Programul utilizeaza un pointer Ptr NULL
// Se initializeaza o variabila pointer Ptr
// Se atribuie pointer-ului valoarea NULL si se afiseaza

#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

int main(void)
{
system("TITLE Pointerul NULL "); // Titlul ferestrei consola
system("COLOR F9"); // Fundal alb caractere albastre

```

```

int*  Ptr ;                // Ptr- variabila pointer catre un int
Ptr = NULL;               // Se atribuie pointer-ului valoarea NULL
cout << "\n\n\tValoarea pointerului, este:" << Ptr;
cin.ignore();
cin.get();
}

```

• Pointeri de tip void (tip generic de pointer)

Un pointer poate fi declarat de tip void. În aceasta situatie pointerul nu poate fi folosit pentru indirectare (dereferentiere) deoarece compilatorul nu poate cunoaste tipul obiectului pe care pointerul îl indica.

Pointerii generici sînt foarte utilizati de unele functii de biblioteca, atunci cînd este nevoie sa se opereze cu zone de memorie care pot contine informatii de orice natura: operatii de intrare, alocari de memorie, sortare si cautare. Aceste functii nu cunosc tipul datelor cu care opereaza, pentru aceste functii zonele de memorie nu au un tip precizat. Acest fapt este evidentiat prin pointeri de tip void.

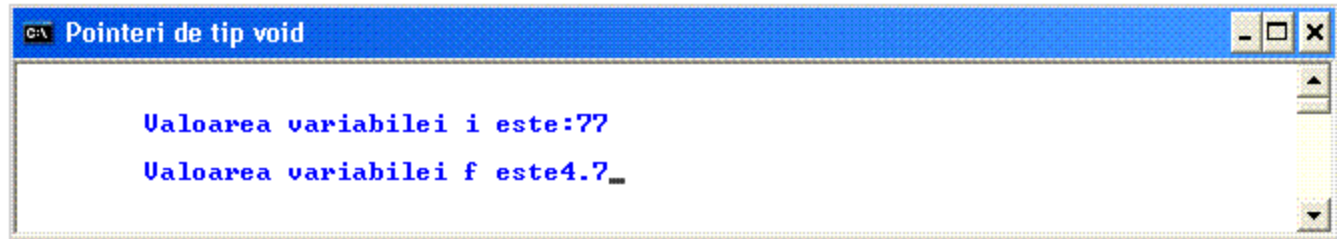
```

// Programul utilizeaza un pointer de tip void
// Pointerul Ptr va fi utilizat pentru a indica spre diverse variabile de
divestre tipuri

#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

int main(void)
{
system("TITLE Pointeri de tip void "); // Titlul ferestrei consola
system("COLOR F9"); // Fundal alb caractere albastre
int i;
float f;
void* Ptr;
Ptr = &i;                                /* p indica spre i */
*(int *)Ptr = 77;
cout << "\n\n\tValoarea variabilei i este:" << i;
Ptr = &f;                                /* p indica spre f */
*(float *)Ptr = 4.7;
cout << "\n\n\tValoarea variabilei f este" << f;
cin.ignore();
cin.get();
}

```



In cazul in care am vrea sa afisam variabila i cu instructiunea :

```
cout << "\n\n\tValoarea variabilei f este" << *Ptr;
```

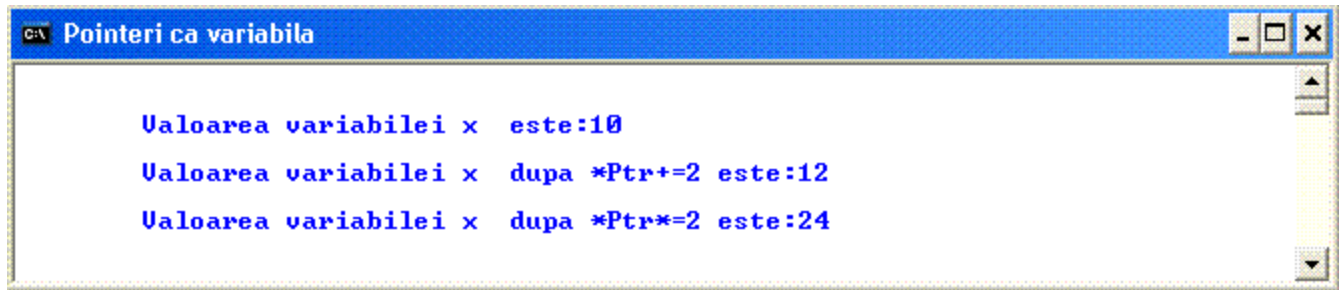
compilatorul ar genera o eroare "illegal indirection", deoarece compilatorul nu poate cunoaste tipul lui Ptr.

- **Pointeri ca variabila sau constanta**

Un Pointer poate fi folosit ca variabila sau constanta. Urmatorul exemplu ilustreaza acest lucru.

```
// Programul utilizeaza un pointer Ptr
// Se initializeaza o variabila pointer Ptr catre un intreg
// Se defineste o variabila x de tip int
// Se atribuie pointer-ului adresa variabilei x
// Se fac operatii utilizand variabila ptr
// Se afiseaza valoarea variabilei x
#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

int main(void)
{
system("TITLE Pointeri ca variabila "); // Titlul ferestrei consola
system("COLOR F9"); // Fundal alb caractere albastre
int x;
int* Ptr ; // Ptr- variabila pointer catre un int
Ptr=&x; // Variabila Ptr contine adresa
variabilei x
*Ptr=10; // Se atribuie valoarea 10 lui x
cout << "\n\n\tValoarea variabilei x este:" << x;
*Ptr+=2;
cout << "\n\n\tValoarea variabilei x dupa *Ptr+=2 este:" << x;
*Ptr*=2;
cout << "\n\n\tValoarea variabilei x dupa *Ptr*=2 este:" << x;
cin.ignore();
cin.get();
}
```



```
C:\> Pointeri ca variabila

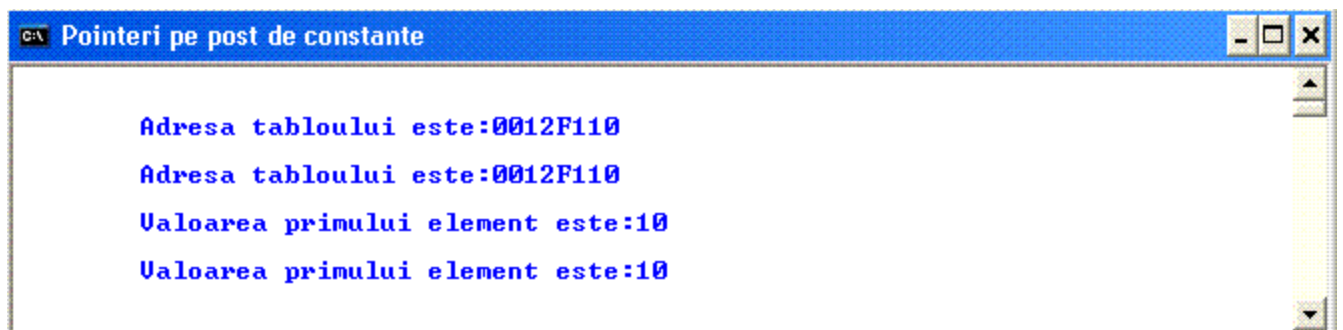
Valoarea variabilei x este:10
Valoarea variabilei x dupa *Ptr+=2 este:12
Valoarea variabilei x dupa *Ptr*=2 este:24
```

Urmatorul exemplu utilizeaza un pointeri pe post de constanta.

```
// Programul utilizeaza numele unui tablou pe post de pointer
// Afiseaza adresa tabloului ca fiind numele tabloului sau adresa primului
// element
// Afiseaza prima valoare a tabloului ca fiind numele tablou[0] sau
// referentierea pointerului

#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

int main(void)
{
    system("TITLE Pointeri pe post de constante "); // Titlul ferestrei consola
    system("COLOR F9"); // Fundal alb caractere albastre
    const int Dmax=4;
    int note[Dmax]={10,9,7,10};
    cout << "\n\n\tAdresa tabloului este:" << note;
    cout << "\n\n\tAdresa tabloului este:" << & note[0];
    cout << "\n\n\tValoarea primului element este:" << note[0];
    cout << "\n\n\tValoarea primului element este:" << *note;
    cin.ignore();
    cin.get();
}
```



```
C:\> Pointeri pe post de constante

Adresa tabloului este:0012F110
Adresa tabloului este:0012F110
Valoarea primului element este:10
Valoarea primului element este:10
```

• Aritmetica pointerilor

Un pointer fiind o adresa, cu el se pot face operatii exact ca si cu o valoare numerica. Aritmetica pointerilor se foloseste in special pentu lucrul cu tablouri.

Sa reluam aplicatia care defineste note intr-un tablou si sa afisam notele prima data fara a folosi pointeri.

```
// Programul afiseaza adresele elementelor si elementele unui tablou

#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

int main(void)
{
system("TITLE Adresele elementelor unui tablou "); // Titlul ferestrei
consola
system("COLOR F9"); // Fundal alb caractere albastre
const int Dmax=4;
int i, note[Dmax]={10,9,7,10};
for (i=0; i < Dmax ; i++){
    cout << "\n\n\tAdresa elementului " << i << " este: " << & note[i];
    cout << "\n\n\tValoarea elementului " << i << " este: " << note[i];
}
cin.ignore();
cin.get();
}
```

Utilizand pointeri, aceeaasi aplicatie poate fi scrisa:

```
// Programul initializeaza un pointer cu numele unui tablou.
// Afiseaza adresa elementelor tabloului ca fiind chiar pointerul
// Afiseaza valoarea elementelor tabloului ca fiind referentierea pointer-
ului
// Pentru a parcurge tabloul s-a utilizat operatiunea de incrementare
pointer

#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

int main(void)
{
system("TITLE aritmetica pointerilor "); // Titlul ferestrei consola
```



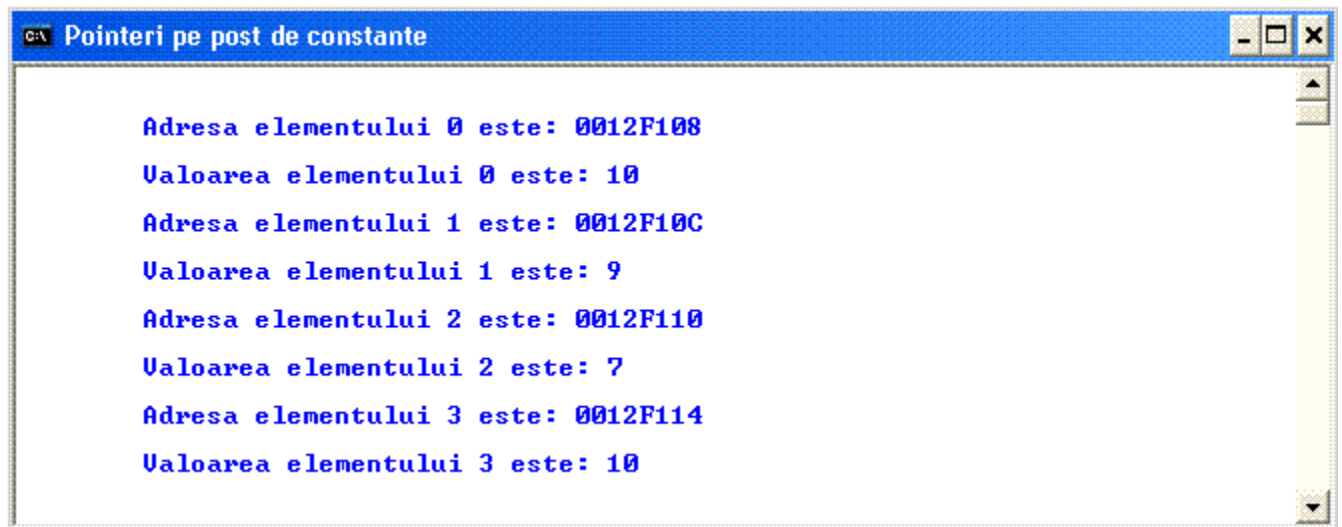
```

system("COLOR F9");          // Fundal alb caractere albastre
const int Dmax=4;
int i, note[Dmax]={10,9,7,10};
int* Ptr=note;
for (i=0; i < Dmax ; i++, Ptr++){
    cout << "\n\n\tAdresa elementului " << i << " este: " << Ptr;
    cout << "\n\n\tValoarea elementului " << i << " este: " << * Ptr;
}

cin.ignore();
cin.get();
}

```

Dupa rularea aplicatiilor de sus, obtinem acelasi rezultat adica:



```

Adresa elementului 0 este: 0012F108
Valoarea elementului 0 este: 10
Adresa elementului 1 este: 0012F10C
Valoarea elementului 1 este: 9
Adresa elementului 2 este: 0012F110
Valoarea elementului 2 este: 7
Adresa elementului 3 este: 0012F114
Valoarea elementului 3 este: 10

```

In exemplul de sus, un pointer a fost incrementat. Cu pointerii se pot face toate operatiile se se pot face si cu numerele asa ca un pointer poate fi si decrementat. Sa afisam acum notele in ordine inversa.

```

// Programul initializeaza un pointer cu numele unui tablou.
// Se afiseaza notele in ordine inversa, prin decrementarea pointerului.
#include "stdafx.h"
#include <iostream>
#include <string>
using namespace std;

int main(void)
{
    system("TITLE Decrementarea unui pointer "); // Titlul ferestrei consola
    system("COLOR F9");          // Fundal alb caractere albastre
    const int Dmax=4;

```

```

int i, note[Dmax]={10,9,7,10};
int* Ptr=note+3;
for (i=Dmax-1; i >=0 ; i--, Ptr--){
    cout << "\n\n\tAdresa elementului " << i << " este: " << Ptr;
    cout << "\n\n\tValoarea elementului " << i << " este: " << * Ptr;
}
cin.ignore();
cin.get();
}

```

Vom utiliza in continuare pointeri si operatii cu point-eri pentru vizualizarea reprezentarii diverselor tipuri de date in memorie. Vom defini un pointer spre un numar de tip int. Tipul int este reprezentat in calculator pe 4 octeti. Deci pointerul indica spre inceputul celor 4 octeti. Daca definim un nou pointer ce indica spre tipul de date char vom putea analiza cei 4 octeti din care e compus tipul int

```

#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

int main(void)
{
system("TITLE Utilizarea pointer-ilor pentru vizualizarea reprezentarii
datelor "); // Titlul ferestrei consola
system("COLOR F9"); // Fundal alb caractere albastre
int x=123456789;
int* Ptr ; // Ptr- variabila pointer catre un int
Ptr=&x;
char* p;
p=(char *)Ptr;
cout << "\n\n\tVariabila x=123456789 este reprezentata pe 4 octeti astfel:";
cout << "\n\n\tOctetul 0:" << hex << static_cast< short int >(*p);
cout << "\n\tOctetul 1:" << hex << static_cast< short int >*(p+1));
cout << "\n\tOctetul 2:" << hex << static_cast< short int >*(p+2));
cout << "\n\tOctetul 3:" << hex << static_cast< short int >*(p+3));
cin.ignore();
cin.get();
}

```

Variabila x=123456789 este reprezentata pe 4 octeti astfel:

Octetul 0:15
Octetul 1:cd
Octetul 2:5b
Octetul 3:7_

Se observa ca cei 4 octeti sunt : 75bcd15 care in zecimal inseamna 123456789:

Putem initializa un pointer cu o adresa absoluta astfel:

```
#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

int main(void)
{
system("TITLE Initializarea unui pointer cu o adresa "); // Titlul ferestrei
consola
system("COLOR F9"); // Fundal alb caractere albastre
int* Ptr ; // Ptr- variabila pointer catre un int

Ptr=(int *)0xb8000000;
*Ptr=100;
cout << "\n\n\tVariabila pointer Ptr are valoarea:" << *Ptr;
cin.ignore();
cin.get();
}
```

Daca rulam acest program se va da un mesaj de eroare de genul :
'System.AccessViolationException' deoarece programul incearca sa acceseze o zona de memorie care nu este alocaata pentru acest program.

Putem realiza un program de genul:

```
#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

int main(void)
```

```

{
system("TITLE Initializarea unui pointer cu o adresa ");// Titlul ferestrei
consola
system("COLOR F9"); // Fundal alb caractere albastre
int x=100;
int* Ptr ; // Ptr- variabila pointer catre un int
cout << "\n\n\t Adresa variabilei x este: "<<&x;
cin.ignore();
cin.get();
}

```

Programul returneaza mesajul:"Adresa variabilei x este: 0012F098". Realizam apoi programul

```

#include "stdafx.h"
#include < iostream >
#include < string >
using namespace std;

int main(void)
{
system("TITLE Initializarea unui pointer cu o adresa ");// Titlul ferestrei
consola
system("COLOR F9"); // Fundal alb caractere albastre
int* Ptr ; // Ptr- variabila pointer catre un int
int x=100;
cout << "\n\n\t Adresa variabilei x este: "<<&x;
Ptr=(int *)0x0012F098;
cout << "\n\n\tContinutul adresei 0012F098 este:" << *Ptr;
cin.ignore();
cin.get();
}

```

• Pointeri ca argumente de functii

In cadrul functiilor s-au transmis argumente spre functii si proceduri, prin referinta pentru ca la intoarvarea din procedura sau functie sa se poata utiliza argumentul modificat de functie sau procedura.

Am realizat atunci o aplicatie care calculeaza cubul unui numar utilizand o procedura careia i se transmite argumentul prin referinta.

```

// Programul foloseste procedura cub careia i se transmite un parametru
prin referinta
// Se cere un numar de la tastatura si se apeleaza procedura cub
// Procedura afiseaza cubul numarului introdus

```

```

// La intoarcerea din procedura se afiseaza noua valoare a parametrului
transmis

#include "stdafx.h"
#include < iostream >

using namespace std;
void cub(int&); // prototipul
int main(void)
{
    system("TITLE Pointeri ca argumente de functii ");
    system("COLOR F9");
    int x;
    cout << " \n\n\tIntroduceti un numar: ";
    cin >> x;
    cub(x);
    cout << "\n\n\tValoarea numarului introdus dupa apelul procedurii cub
este : ";
    cout << x;
    cin.ignore();
    cin.get();
    return 0;
}
// Incepe definirea procedurii cub
void cub(int& nr)
{
    cout << "\n\n\tCubul numarului : ";
    cout << nr;
    cout << " este : ";
    nr=nr*nr*nr;
    cout << nr;
}

```

Utilizand pointeri, putem rescrie aplicatia astfel:

```

// Programul foloseste procedura cub careia i se transmite adresa
argumentului
// Se cere un numar de la tastatura si se apeleaza procedura cub
// Procedurii cub i se transmite adresa argumentului
// Procedura cub, utilizeaza un pointer int* pentru a avea acces la
argumentul transmis
// Procedura cub, face calcule asupra argumentului referentiind pointerul
// Procedura cub afiseaza cubul numarului introdus
// La intoarcerea din procedura se afiseaza noua valoare a argumentului
transmis

#include "stdafx.h"
#include < iostream >

using namespace std;
void cub(int*); // prototipul

```

```

int main(void)
{
    system("TITLE Pointeri ca argumente de functii ");
    system("COLOR F9");
    int x;
    cout << " \n\n\tIntroduceti un numar: ";
    cin >> x;
    cub(&x);
    cout << "\n\n\tValoarea numarului introdus dupa apelul procedurii cub
este : ";
    cout << x;
    cin.ignore();
    cin.get();
    return 0;
}
// Incepe definirea procedurii cub
void cub(int* nr)
{
    cout << "\n\n\tCubul numarului : ";
    cout << *nr;
    cout << " este : ";
    *nr=(*nr)*(*nr)*(*nr);
    cout << *nr;
}

```

Vom transmite un tablou ca argument folosind pointeri. Vom realiza o aplicatie simpla care cere 5 numere, le pastreaza intr-un tablou dupa care apeleaza o functie careia i se transmite ca argument acest tablou pentru a afisa elementele tabloului.

```

// Programul cere 5 numere dupa care le afiseaza ";
// Se defineste si se apeleaza o functie care afiseaza elementele tabloului
// Functia are ca argument un pointer

#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
void afisare(double* v,int imdex_max);
int main(void)
{
    system("TITLE Pointeri ca argumente de functii ");
    system("COLOR F9");
    const int nr_max=5;
    int i;
    double nr[nr_max];
    cout << "\n\tProgramul cere " << nr_max << " numere dupa care le
afiseaza \n\n";
    // introducere numere

    for (i=0;i < nr_max;i++){

```

```

        cout << "\tIntroduceti numarul : [" << i << "]" : ";
        cin >> nr[i];
    }
    afisare(nr, nr_max);
    cin.ignore();
    cin.get();
    return 0;
}

void afisare(double* v, int index_max)
{
    cout << "\n\n\tNumerele introduse sunt :";
    for (int i=0; i < index_max; i++){
        cout << " : " << v[i] ;
    }
}

```

Putem rescrie aplicatia astfel incat sa utilizam pointeri si in programul principal.

```

// Programul cere 5 numere dupa care le afiseaza ";
// Cele cinci elemente sun salvate intr-un tablou utilizand pointeri.
// Se defineste si se apeleaza o functie care afiseaza elementele tabloului
// Functia are ca argument un pointer

#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
void afisare(double* v, int imdex_max);
int main(void)
{
    system("TITLE Pointeri ca argumente de functii ");
    system("COLOR F9");
    const int nr_max=5;
    int i;
    double nr[nr_max];
    double* p;
    p=nr;
    cout << "\n\tProgramul cere " << nr_max << " numere dupa care le
afiseaza \n\n";
    // introducere numere

    for (i=0; i < nr_max; i++, p++){
        cout << "\tIntroduceti numarul : [" << i << "]" : ";
        cin >> *p;
    }
    afisare(nr, nr_max);
    cin.ignore();
    cin.get();
}

```

```

        return 0;
    }

void afisare(double* v, int index_max)
{
    cout << "\n\n\tNumerele introduse sunt :";
    for (int i=0; i < index_max; i++){
        cout << " : " << v[i] ;
    }
}

```

Reluam o aplicatie mai veche pentru afisarea cubului unor numere introduse folosind o functie careia i se transmite tabloul cu numerele citite ca argument folosind pointeri in vederea afisarii cubului elementelor tabloului.

```

// Programul cere 5 numere dupa care le ridica la patrat si le afiseaza ";
// Se defineste si se apeleaza o functie care inlocuieste valorile din
tablou cu patratele acestora
// Functia are ca argument un pointer
// La sfarsit se afiseaza elementele tabloului rezultat rezultat
#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
void patrat(double* v,int imdex_max);
int main(void)
{
    system("TITLE Pointeri ca argumente de functii ");
    system("COLOR F9");
    const int nr_max=5;
    int i;
    double nr[nr_max];
    cout << "\n\tProgramul cere " << nr_max << " numere dupa care le
ridica la patrat si le afiseaza \n\n";
    // introducere numere

    for (i=0;i < nr_max;i++){
        cout << "\tIntroduceti numarul : [" << i << "]" : ";
        cin >> nr[i];
    }
    // afisarea numerelor

    cout << "\n\n\tNumerele introduse sunt: \n\n";
    for (i=0;i < nr_max;i++){
        cout << " : " << nr[i] ;
    }
    // calcularea patratelor numerelor
    patrat(nr,nr_max);

    // afisarea patratelor numerelor

```



```

        cout << "\n\n\tPatratele numerelor sunt: \n\n";
        for (i=0;i < nr_max;i++){
            cout << " : " << nr[i] ;
        }
        cin.ignore();
        cin.get();
        return 0;
    }

void patrat(double*v , int index_max)
{
    for (int i=0; i < index_max; i++){
        v[i]=v[i]*v[i];
    }
}

```

• Alocarea dinamica a memoriei

In spatiul de nume std: nu exista posibilitatea sa definim dimensiunea unui tablou prin intermediul unei variabile. Aplicatia care cere dimensiunea unui vector dupa care cere elementele acestuia cuprindea definirea unui tablou unidimensional la o dimensiune maxima. astfel in aplicatia de jos cerem numarul de elemente dar precizam faptul ca valoarea poate fi maxim 10.

```

// Programul cere n numere dupa care le afiseaza

#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
int main(void)
{
    system("TITLE Alocarea dinamica a memoriei ");
    system("COLOR F9");
    int i,n;
    const int nmax=10;
    cout << "\n\n\tProgramul cere n numere dupa care le afiseaza \n\n";
    cout << " \n\n\tIntroduceti valoarea lui n (maxim 10):";
    cin >> n;
    double numere[nmax];
    for (i=0;i < n;i++){
        cout << "\tIntroduceti numarul : [" << i << "]" : ";
        cin >> numere[i];
    }
    cout << "\n\n\tNumerele introduse sunt: \n";
    for (i=0;i < n;i++){

```

```

        cout << "\n\tNumarul [ " << i << " ] ="<< numere[i] ;
    }
    cin.ignore();
    cin.get();
    return 0;
}

```

Pentru a dimensiona tabloul la exact numarul de elemente precizate , trebuie sa modificam programul astfel:

```

// Programul cere n numere dupa care le afiseaza
// Programul utilizeaza alocarea dinamica a memoriei

#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
int main(void)
{
    system("TITLE Alocarea dinamica a memoriei ");
    system("COLOR F9");
    int i,n;
    cout << "\n\tProgramul cere n numere dupa care le afiseaza \n\n";
    cout << " \n\tIntroduceti valoarea lui n :";
    cin >> n;
    double* ptr = new double[n];
    for (i=0;i < n;i++){
        cout << "\tIntroduceti numarul : [ " << i << " ] : ";
        cin >> ptr[i];
    }
    cout << "\n\n\tNumerele introduse sunt:  \n";
    for (i=0;i < n;i++){
        cout << "\n\tNumarul [ " << i << " ] ="<< ptr[i] ;
    }
    cin.ignore();
    cin.get();
    delete [] ptr;
    return 0;
}

```

Dupa cum se observa trebuie folosit un pointer pentru alocarea dinamica a memoriei. In cadrul programului de sus, alocarea se face cu instructiunea:

double *ptr = new double[n];

Operatorul **new** este cel care face alocarea memoriei in mod dinamic.

La compilarea unui program , acestuia i se alocă memorie statică. Aici este ținut codul și spațiul de memorie pentru diverse variabile și tablouri. Din acest motiv tablourile nu pot fi

redimensionate dupa lansarea programului. In momentul compilarii, compilatorul trebuie sa stie exact cata memorie trebuie rezervata. In momentul lansarii, sistemul de operare rezerva si memorie dinamica (memorie de **heap**), memorie in care se poate face alocare dinamica.

Durata de viata a variabilei alocata dinamic este durata de viata a programului. La iesirea din program nu mai avem acces la memoria alocata deoarece pointerul nu mai este vizibil. Memoria ramne deci ocupata si dupa iesirea din program si se creaza astfel o pierdere de memorie.

Memoria alocata dinamic se sterge cu operatorul "delete nume_pointer" ea fiind restituita astfel sistemului de operare.

Daca variabila dinamica este un tablou, ea se sterge cu: "delete [] nume_pointer "

Sa alocam dinamic memorie pentru un vector cu un singur element pe care il initializam sa zicem cu valoarea 22, sa afisam continutul dupa care sa stergem pointerul si sa vedem continutul memoriei

```
//Alocarea dinamica a memoriei
//Stergerea pointerului

#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;

int main()
{
    int* ip= new int[1];
    *ip=22;
    cout<<"\n\tip = "<< ip <<" \t*ip= "<< *ip <<"\n";
    cout<<"\n\tDupa stergerea pointerului avem:\n";
    delete ip;
    cout<<"\n\n\tip = "<< ip <<" \t*ip= "<< *ip <<"\n";
    cin.get();
    return 0;
}
```

Sa incercam acum sa initializam un vector, sa facem realocare si sa vedem continutul

```
//Programul utilizeaza pointeri pentru realocarea dinamica a memoriei
//La sfarsit se sterge pointerul

#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
int main(void)
```

```

{
    system("TITLE Realocarea dinamica a memoriei ");
    system("COLOR F9");
    int i,n;
    cout << "\n\tProgramul initializeaza un vector cu n elemente \n\n";
    cout << " \n\tIntroduceti valoarea lui n :";
    cin >> n;
    double* ptr = new double[n];
    for (i=0;i < n;i++){
        ptr[i]=rand();
    }
    cout << "\n\n\tNumerele generate sunt:  \n";
    for (i=0;i < n;i++){
        cout << " \n\tNumarul [ " << i << " ] ="<< ptr[i] ;
    }
    realloc(ptr,n+3); // Realocare dinamica
    cout << " \n\n\tDupa redimensionare a vectorului cu 3 elemente
obtinem: \n";
    for (i=0;i < n+3;i++){
        cout << " \n\tNumarul [ " << i << " ] ="<< ptr[i] ;

        delete [] ptr;
        cout << " \n\n\tDupa stergerea pointerului avem: \n";
        for (i=0;i < n+3;i++){
            cout << " \n\tNumarul [ " << i << " ] ="<< ptr[i] ;
        }
        cin.ignore();
        cin.get();

        return 0;
    }
}

```

Dupa realocarea dinamica, continutul locatiilor este afectat, iar dupa stergerea pointerului continutul este initializat.

• Returnarea pointerilor din functii

Sa realizam o aplicatie care defineste in pointer la o tabela de caractere. In cadrul aplicatiei vrem sa scriem o functie care atribuie valori tablei si returneaza adresa tablei astfel incat in programul principal sa putem afisa tabela. Functia trebuie sa returneze deci un pointer.

Pentru inceput sa realizam o aplicatie care initializeaza si afiseaza o tabela de caractere, utilizand pointeri

```

// Returnarea pointerilor din functii

#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;

```

```

int main(void)
{
    system("TITLE Returnarea pointerilor din functii ");
    system("COLOR F9");
    cout << "\n\tProgramul afiseaza un sir initializat cu ajutorul
pointerilor \n\n";
    char* sectia ="Inginerie Electrica";
    cout << "\n\n\tSectia este: " << sectia;
    cin.ignore();
    cin.get();
    return 0;
}

```

Vom modifica acum aplicatia si vom introduce o functie care atribuie valori tabelii si returneaza adresa tabelii, adica un pointer.

```

// Returnarea pointerilor din functii
// Programul cere si afiseaza un sir utilizand un pointer
// Pointerul este returnat de o functie
#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
char* cere_sectia();
int main(void)
{
    system("TITLE Returnarea pointerilor din functii ");
    system("COLOR F9");
    cout << "\n\tProgramul cere si afiseaza un sir utilizand un
pointer.";
    cout << "\n\tPointerul este returnat de o functie \n\n";
    char* sec =cere_sectia();
    cout << "\n\n\tSectia este: " << sec;
    cin.get();
    return 0;
}
char* cere_sectia(){
    static char sectia[55];
    cout << "\n\n\tIntroduceti  sectia: ";
    cin.getline(sectia,80);
    return sectia;
}

```

Se observa ca s-a definit tabloul sectia folosind declaratia **static char sectia[55]** altfel variabila "sectia" nu ar fi vizibila si in programul principal. Cu alte cuvinte un pointer returnat dintr-o functie trebuie sa indice spre o variabila statica sau dinamica, in nici un caz nu trebuie sa indice spre o variabila locala.

Pointerul poate indica si spre o variabila creata dinamic. Urmatoarea aplicatie returneaza un pointer dintr-o functie, pointerul indicand spre o variabila dinamica.

```

// Returnarea pointerilor din functii
// Programul cere si afiseaza un sir utilizand un pointer
// Pointerul este returnat de o functie si indica spre o variabila dinamica.
#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
char* cere_sectia();
int main(void)
{
    system("TITLE Returnarea pointerilor din functii  ");
    system("COLOR F9");
    cout << "\n\t- Programul cere si afiseaza un sir utilizand un
pointer.";
    cout << "\n\t- Pointerul este returnat de o functie \n\n";
    cout << "\n\t- Pointerul indica spre o variabila dinamica \n\n";

    char* sec =cere_sectia();
    cout <<"\n\n\tSectia este: " << sec;
    cin.get();
    delete [] sec;
    return 0;
}
char* cere_sectia(){
    char* sectia;
    sectia= new char[80];
    cout << "\n\n\tIntroduceti  sectia: ";
    cin.getline(sectia,80);
    return sectia;
}

```

In acest caz nu s-a mai eclarat char* sectia de tip static, deoarece durata de viata a unei variabile dinamice este pe toata durata de viata a programului.

• Pointeri in spatiul System

In CLR, spatiul System pointerii sunt tipuri speciale de variabile care indica spre adresa altei variabile sau spre un obiect. Exista trei tipuri de pointeri acceptati in CLR:

- managed pointers (pointeri gestionati)
- unmanaged pointers (pointeri negestionati)
- managed function pointers (pointeri negestionati spre functii)

- Un pointer gestionat cunoscut ca si pointer spre un obiect in memoria heap este un tip nou de pointer, disponibil pentru a gestiona aplicatiile. Pointeri gestionati indica spre blocuri de memorie din memoria heap pentru CLR. In memoria heap CLR ruleaza automat Garbage collection. Toate obiectele lansate aici sunt automat sterse din memorie dupa ce aplicatia se

incheie.

In Visual C++ 2002 si Visual C++ 2003, un pointer gestionat se declara cu `__gc` * In Visual C++ 2005 - 2008, aceasta notatie a fost inlocuita cu `^` astfel la declararea unui tablou unidimensional de exemplu se utilizeaza:

```
array < int >^ numere = gcnew array < int > (10);
```

- Pointerii negestionati sunt pointerii traditionali din C++. Acesti pointeri indica spre memoria statica. Din cauza faptului ca pointerii negestionati nu fac parte din Common Language Specification (CLS), nu exista specificatii pentru a accesa acest tip de pointeri.

- Pointerii negestionati spre functii sunt de asemenea pointeri traditionali din C++ indicand adresa unei functii. CLS furnizeaza "delegati" ca o metoda alternativa spre pointeri negestionati spre functii.

Common Language Specification (CLS) furnizeaza trei operatii aritmetice asupra pointerilor : adunare cu un intreg, scadere cu un intreg si adunarea a doi pointeri.

• Aplicatii CRL cu pointeri

Vom folosi in continuare, pointeri gestionati in spatiul System::. Sa realizam o aplicatie care utilizeaza un vector de anumita dimensiune dupa care il redimensioneaza. Vom constata in urma rularii aplicatiei daca dupa redimensionare se pastreaza valorile vechilor elemente sau nu.

```
// Programul initializeaza un vector de dimensiune 0
// Cere numarul de elemente si redimensioneaza vectorul
// Atribuire valori aleatoare intre 0-100 elementelor vectorului
// Se afiseaza elementele vectorului
// Se redimensioneaza vectorul dupa care se afiseaza din nou
#include "stdafx.h"
#include < iostream >
using namespace std;
using namespace System;

int main(void)
{
    system("TITLE Redimensionare vector "); // Titlul ferestrei consola
    system("COLOR F9"); // Fundal alb caractere albastre
    int i, nrmax ;
    array < int >^ numere = gcnew array < int > (0);
    Console::Write("\n\tDimensiunea vectorului: ");
    nrmax=System::Convert::ToInt16(Console::ReadLine()); // Citirea
    dimensiuni vectorului
    Array::Resize(numere,nrmax); //
    Redimensionare vector
    for (i=0;i < nrmax; i++) // Atribuire
    valori aleatoare intre 0-100
        numere[i]=100*rand()/RAND_MAX;
    Console::WriteLine("\n\tNumerele sunt: \n\n"); //
    Afisare numere
    for (i=0;i < nrmax; i++)
```

```

        Console.WriteLine("\t\t\tNr["+i+"]="+numere[i]);

        Console.Write("\n\tNoua dimensiune: ");
        nrmax=System.Convert.ToInt16(Console.ReadLine()); // Citirea noii
dimensiuni
        Array.Resize(numere,nrmax);
        Console.Write("\n\tDupa redimensionare, elementele vectorului sunt:
\n\n");
        for (i=0;i < nrmax; i++)
            Console.WriteLine("\t\t\tNr["+i+"]="+numere[i]);
        Console.ReadLine();
        delete numere;
        return 0;
    }

```

In urma rularii aplicatiei obtinem:

```

Redimensionare vector

Dimensiunea vectorului: 7
Numerele sunt:

Nr[0]=0
Nr[1]=56
Nr[2]=19
Nr[3]=80
Nr[4]=58
Nr[5]=47
Nr[6]=35

Noua dimensiune: 12
Dupa redimensionare, elementele vectorului sunt:

Nr[0]=0
Nr[1]=56
Nr[2]=19
Nr[3]=80
Nr[4]=58
Nr[5]=47
Nr[6]=35
Nr[7]=0
Nr[8]=0
Nr[9]=0
Nr[10]=0
Nr[11]=0

```

Sa reluam in continuare aplicatia pentru afisarea consumurilor zilnice si a consumului mediu lunar, dar sa utilizam pointeri negestionati pentru a salva consumurile intr-o matrice, in vederea calculului consumului mediu.

In CLR, spatiul System pointerii sunt tipuri speciale de variabile care indica spre adresa altei variabile sau spre un

Deschidem un nou proiect Windows Forms Application intitulat "point_v1" pe care plasam un

obiect de tip button numit button1. Completam procedura deschisa pe evenimentul Klik al obiectului button1 cu :

```
const int nr_z=31;
int poz_x=14; // pozitia curenta pe axa x
float w_r=1,w_a=3; // grosimea(in pixeli) a liniei rosii respectiv
albastre
double consum[nr_z+1]; // vector ce pastreaza consumurile pe nr_z
zile

double* d_ptr=consum; // pointer negestionat
double v_rand; // valoare aleatoare
double c_lun; // consum lunar
double c_med; // consum lunar mediu

// Se definesc pointerii gestionati

System::Drawing::Graphics^ Desen;
System::Drawing::Pen^ Creion_rosu;
System::Drawing::Pen^ Creion_albastru;
System::Random^ n;

// Se instantiaza clasele pentu a crea diferite obiecte folosite pe
parcursul aplicatiei. Obiectele create se mai numesc si instante.

Desen = this->CreateGraphics();
Creion_rosu=gcnnew
System::Drawing::Pen(System::Drawing::Color::Red,w_r);
Creion_albastru=gcnnew
System::Drawing::Pen(System::Drawing::Color::BlueViolet ,w_a);
n = gcnnew System::Random();

// Stergere desen prin invocarea metodei Clear

Desen->Clear(System::Drawing::Color(this->BackColor));

// Trasare axa x si axa y

Desen->DrawLine( Creion_rosu,6,0,6,this->Height-40);
Desen->DrawLine( Creion_rosu,6,this->Height-40,this->Width-20,this-
>Height-40);

// Trasare consumuri si salvare in matricea consum

for ( d_ptr=consum; d_ptr < consum+nr_z; d_ptr++){
    v_rand=n->Next(this->Height-40); // se genereaza o valoare
aleatoare
    Desen->DrawLine( Creion_albastru,poz_x,this->Height-
40,poz_x,Height-40-v_rand);
    poz_x+=14;
    *d_ptr=v_rand;
}
c_lun=0;
```

```
// Calculez consum lunar

for ( d_ptr=consum; d_ptr < consum + nr_z; d_ptr++){
    c_lun=c_lun + *d_ptr;
}

// calculez si afisez consumul mediu

c_med=c_lun/nr_z;
Desen->DrawLine( Creion_rosu,6,this->Height-40-c_med,this->Width-
20,this->Height-40-c_med);
```

Aplicatia ruleaza la fel cu aplicatia descrisa in cadrul capitolului "Tablouri", difera doar codul sursa.

