UNIVERSITATEA
Petru Maior
TÂRGU-MURES

TURC TRAIAN

Programarea calculatoarelor si limbaje de programare 1

CURS

2009

Arhitectura calculatoarelor - notiuni introductive

Calculatoarele sunt echipamente complexe pentru prelucrarea datelor si gestionarea informatiei

Pentru a introduce notiunea de informatie ,se presupune ca intr-o situatie oarecare pot avea loc N evenimente egal probabile , probabilitatea unui eveniment fiind p=1/N.Prin realizarea unui eveniment din cele N posibile se obtine informatie.

Prin definitie informatia este i=log1/p=logN

Informatia se transmite prin semnal. Semnalul este o manifestare fizica (unda electromagnetica ,unda sonora etc) capabila de a se propaga printr-un mediu dat.

Semnalele sunt supuse perturbatiilor. Se numeste perturbatie un semnal care modifica semnalul aleator util care transmite informatie, micsorind cantitatea de informatie transmisa.

Mesajul este un semnal ce corespunde unei realizari particulare din ansamblul de idei,imagini,date care trebuie transmise unui corespondent.

Sursa de informatie este mecanismul prin care ,din multimea mesajelor posibile se alege intr-un mod imprevizibil (pentru corespondent) un mesaj particular destinat a fi transmis unui corespondent.

Termeni de baza

Hardware - echipamentele fizice care compun un sistem de calcul
Software - programele pe care le executa calculatorul in general
Aplicatii Software - programele destinate sa rezolve o anumita problema
Program - un set de instructiuni care instruiesc calculatorul pas cu pas ce are de facut
Programator - persoana care scrie programe pentru calculator
Utilizatori - persoanele care cumpara si utilizeaza softul pentru calculator

Notiunea de data si informatie

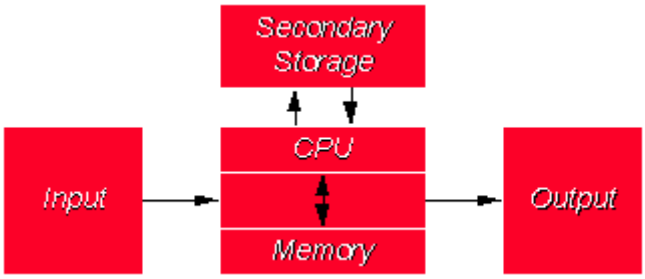
Calculatoarele sunt echipamente complexe pentru prelucrarea si gestionarea informatiei.

Prin definitie informatia se defineste ca fiind i=ln(1/p), unde p este probabilitatea de a avea loc un eveniment din N cazuri posibile. Informatia devine deci i=ln(N). Informatia despre un eveniment este cu atat mai valoroasa deci cu cat probabilitatea unui eveniment de a avea loc este mai mica.

Un calculator este un sistem care accepta date la intrare si le prelucreaza dupa un program (algoritm), rezultind informatii utile pe care apoi le furnizeaza la iesire

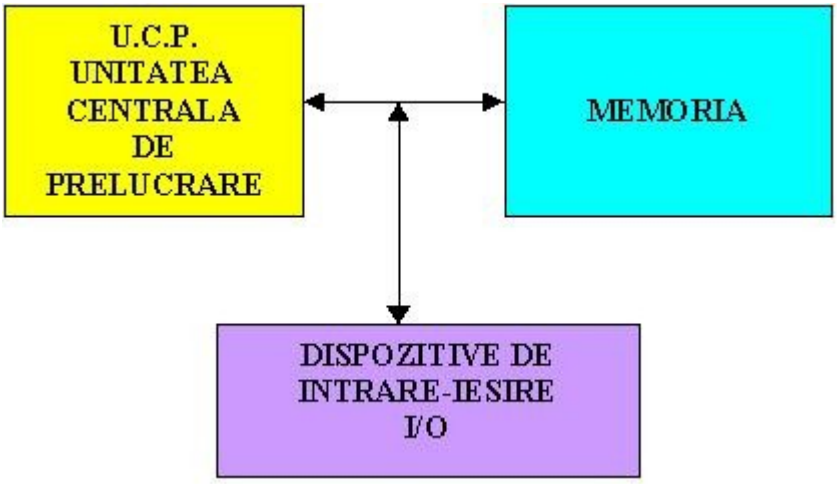
Componentele de baza ale unui calculator

- **Dispozitivele de intrare** primesc datele si le trimit calculatoruluiintr-o forma in care calculatorul le poate intelege si utiliza.
- **Unitatea centrala de prelucrare CPU (Central Processing Unit)** contine circuite electronicepentru prelucrarea datelor de intrare si transformarea lor in informatii care sunt ulterior oferite la iesire
- **Memoria** Tine temporar date si instructiuni necesare CPU
- **Dispozitive de iesire** fac posibila utilizarea datelor prelucrate.
- **Dispozitive de memore externa (secundara)** Stocheaza datele si programele.
- **Magistrala sistem**-reprezinta calea fizica prin care se interconecteaza componentele calculatorului.
  Latimea magistralei sistem depinde de marimea cuvintelor de date



Structura von Neumann

Structura unui calculator se bazeaza schema bloc din figura de jos numita si structura von Neumann

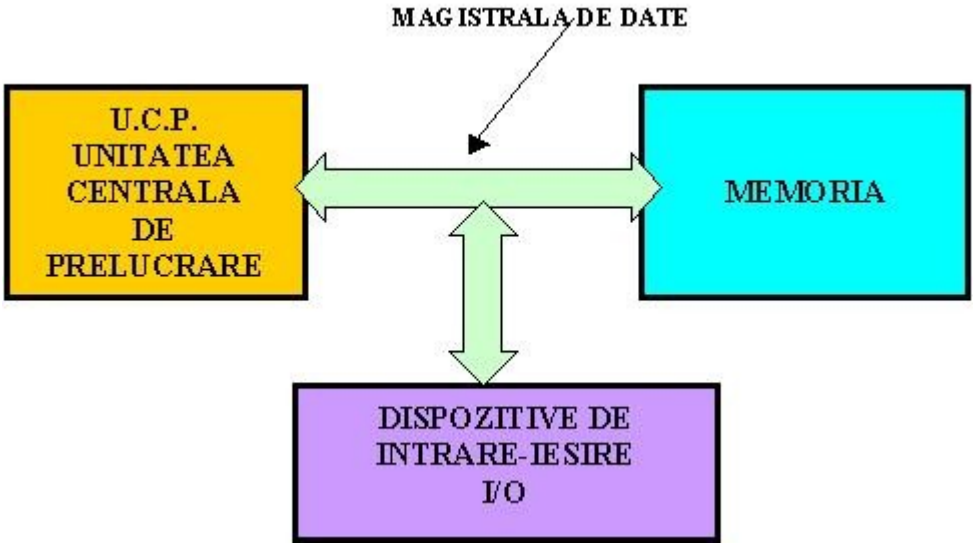


Un calculator este un sistem capabil sa prelucreze, dupa un anumit algoritm(program) datele primite la intrare, si sa furnizeze informatii la iesire.

Datele primite la intrare prin intermediul sistemului de I/O sunt pastrate in memorie de unde sunt transferate in unitatea centrala UC si prelucrate conform unui algoritm (program) aflat tot in memorie, introdus in prealabil. In urma prelucrarii datelor rezulta informatii care se stocheaza in memorie si sunt transmise pentru afisare prin intermediul sistemului I/O.

Reprezentarea informatiilor in calculator

Datele si informatile sunt vehiculate intre partile componente ale calculatorului prin intermadiul unor cai de legatura numite magistrale.



Pentru a putea fi transmisa, informatia are nevoie de un suport fizic numit semnal.

Semnalul este o manifestare fizica (unda electromagnetica, unda sonora, interactiune mecanica ect.) capabila de a se propaga printr-un mediu dat.

Mesajul este un semnal ce corespunde unei realizari particulare din ansamblul de date care trebuie transmis unui corespondent.

In calculatoarele actuale semnalul folosit este semnalul electric.

Magistrala este realizata deci din conductoare electrice prin care se poate propaga semnal electric.

Semnalul electric este supus perturbatiilor. Se numeste perturbatie, un semnal care modifica semnalul aleator util care transporta informatie, micsorand cantitatea de informatie transmisa.

Pentru a reduce la minim posibil influenta zgomotelor asupra semnalului util s-a ales utilizarea semnaluli electric digital.

Semnalul digital are doua stari 1 si 0 adica prezenta semnal si lipsa semnal adica semnal electric cu tensiunea 5v si semnal electric cu tensiunea 0v. Acest tip de semnal poate transporta numai informatie elementara de tipul Fals si Adevarat.

Pentru a utiliza semnalul digital, informatia trebuie deci codificata digital sau binar.

Orce tip de informatie poate fi codificata numeric, dupa care poate fi transformata in numere binare. Este deci posibila realizarea de calculatoare care sa utilizeze semnalul electric digital pentru a vehicula orce tip de informatie sub forma binara.

In functie de numarul de linii electrice se pot transmite simultan mai multe semnale electrice.

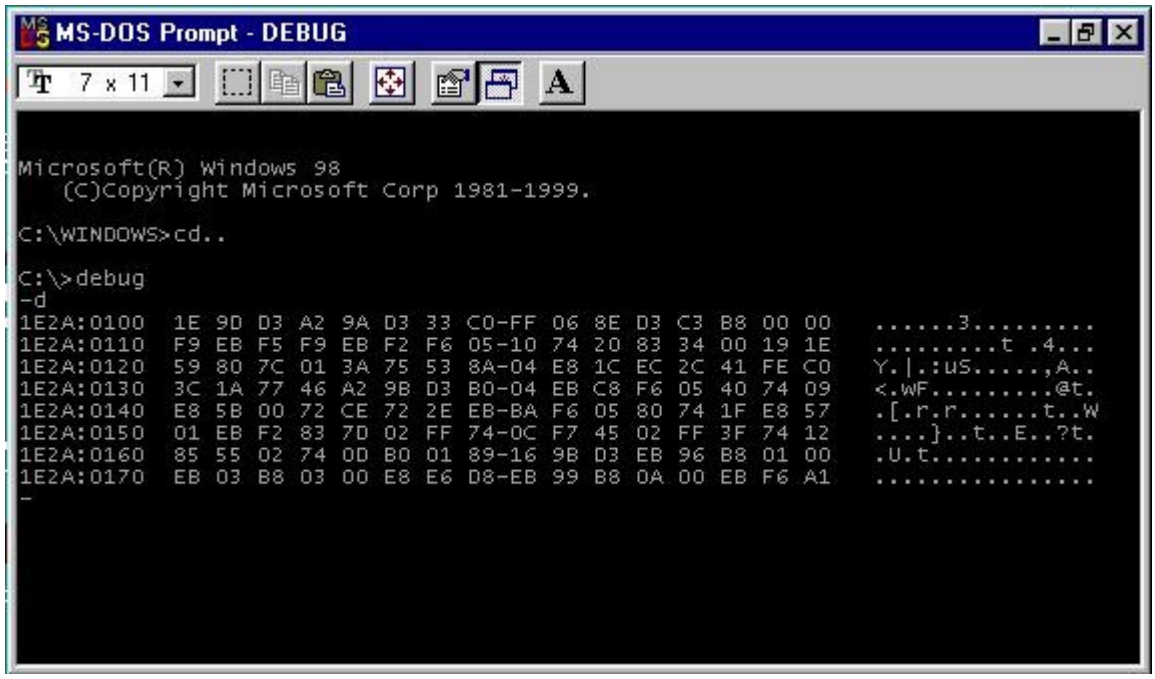
Vom numi bit de informatie informatia elementara (cea mai mica cantitate de informatie,informatia 1-adevarat sau informatia 0-fals.)

Pe o magistrala cu n linii electrice se pot deci transmite n biti. Numarul de linii este impus de nivelul tehnologic. Dupa aparitia unitatilor centrale integrate numite microprocesoare, au fost realizate primele calculatoare personale.

Primele calculatoare personale realizate cu microprocesoare foloseau magistrale de 8 linii, deci pe magistrala se puteau vehicula informatii de maxim 8 biti.Cei opt biti de informatie luati impreuna formeaza un octet sau un byte.

Primele calculatoare PC-IBM compatible au fost realizate cu microprocesoare de 16 biti. Cantitatea de informatie vehiculata era deci compusa din 16 biti(cuvant sau word) Calculatoarele actuale folosesc procesoare de 32 biti (dword-double word) si 64 de biti(qword-quad word)

* Informatia din memoria calculatorului este codificata binar si afisata in cod hexazecimal
* Orice fel de informatie (programe, valori numerice, texte, imagini, sunete, etc.) este codificata numeric in baza 2 (Codificare binara). Se grupeaza apoi cate 4 biti si se reprezinta informatia pe ecran in cod hexazecimal pentru o mai usoara citire.



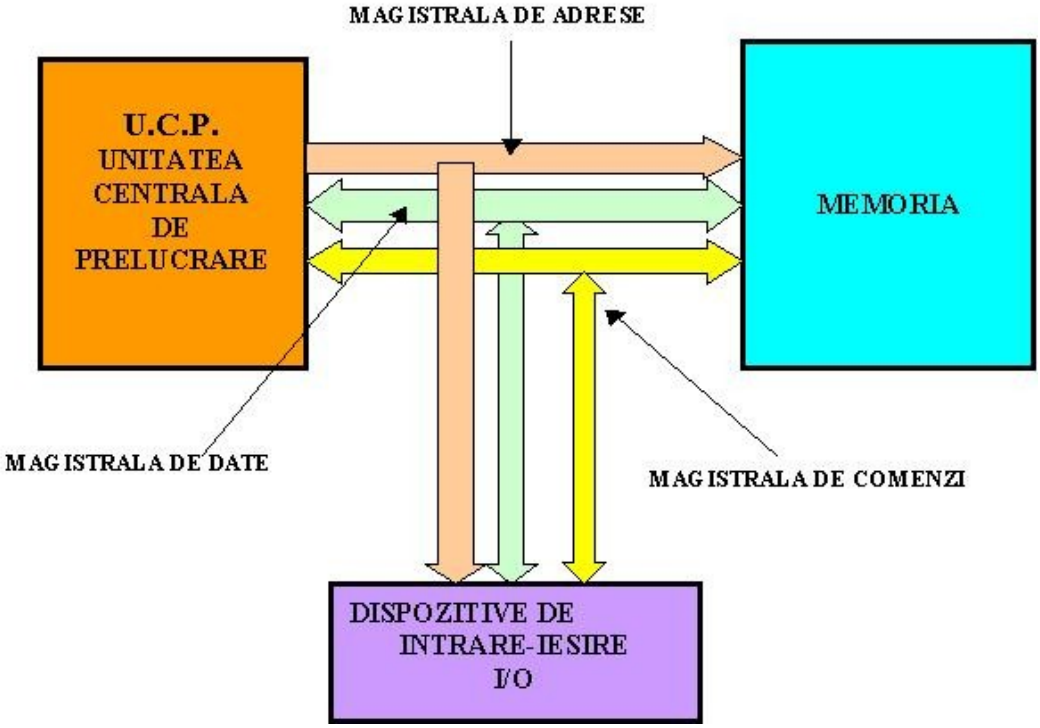
* Tabelul de valori hexa afisat mai sus, reprezinta continutul memoriei de la o anumita adresa.

* Datele exprimate in hexazecimal de la aceasta adresa pot codifica orce fel de informatie. Ele pot reprezenta de exemplu o portiune de imagine, un text etc. Nu putem descifra ce reprezinta aceste valori decat daca incepem decodificarea de la adresa 0000:0000 si decodificam pas cu pas continutul meemoriei functie de pasii facuti de calculator de la pornire pana in momentul afisarii, lucru aproape inposibil.

Memoria este realizata dintr-o sucsesiune de locatii de memorie.

Locatia de memorie este structura fizica ce poate memora 16 biti de informatie(word).

Memoria este astfel realizata incat sa poata prelua de pe magistrala un cuvant pe care sa-l memoreze(scrie) in orce locatie si totodata sa se poata aduce pe magistrala (citi) orce cuvant din orce locatie. Pentru a realiza aceste cerinte, fiecare locatie trebuie sa aiba o "adresa" precisa. Adresele sunt atribuite in ordine fiecarei locatii. Pentru orce operatie (de citire sau scriere in sau din memorie) este deci necesar sa precizam continutul cuvantului, felul operatiei si adresa locatiei la care se refera operatia. Structuara unui calculator se poate deci detalia astfel:



La un moment dat pe magistrala se face o singura operatei (de citire sau scriere)

Coordonarea operatiilor pe magistrala se face de UC prin magistrala de comenzi prin care se traansmit comenzile necesare pentru fiecare operatiune.

Unitatea centrala UC (realizata pe baza MICROPROCESOR-ului) este cea care coordoneaza toate operatiile intr-un calculator.

Programul si datele sunt incarcate in memorie sub controlul UC. Dupa aceasta operatie UC formeaza pe magistrala de adrese adresa primei locatii de unde incepe programul. UC pune si comenzile corespunzatoare pentru operatiile dorite (scriere sau citire) pe magistrala de comenzi.

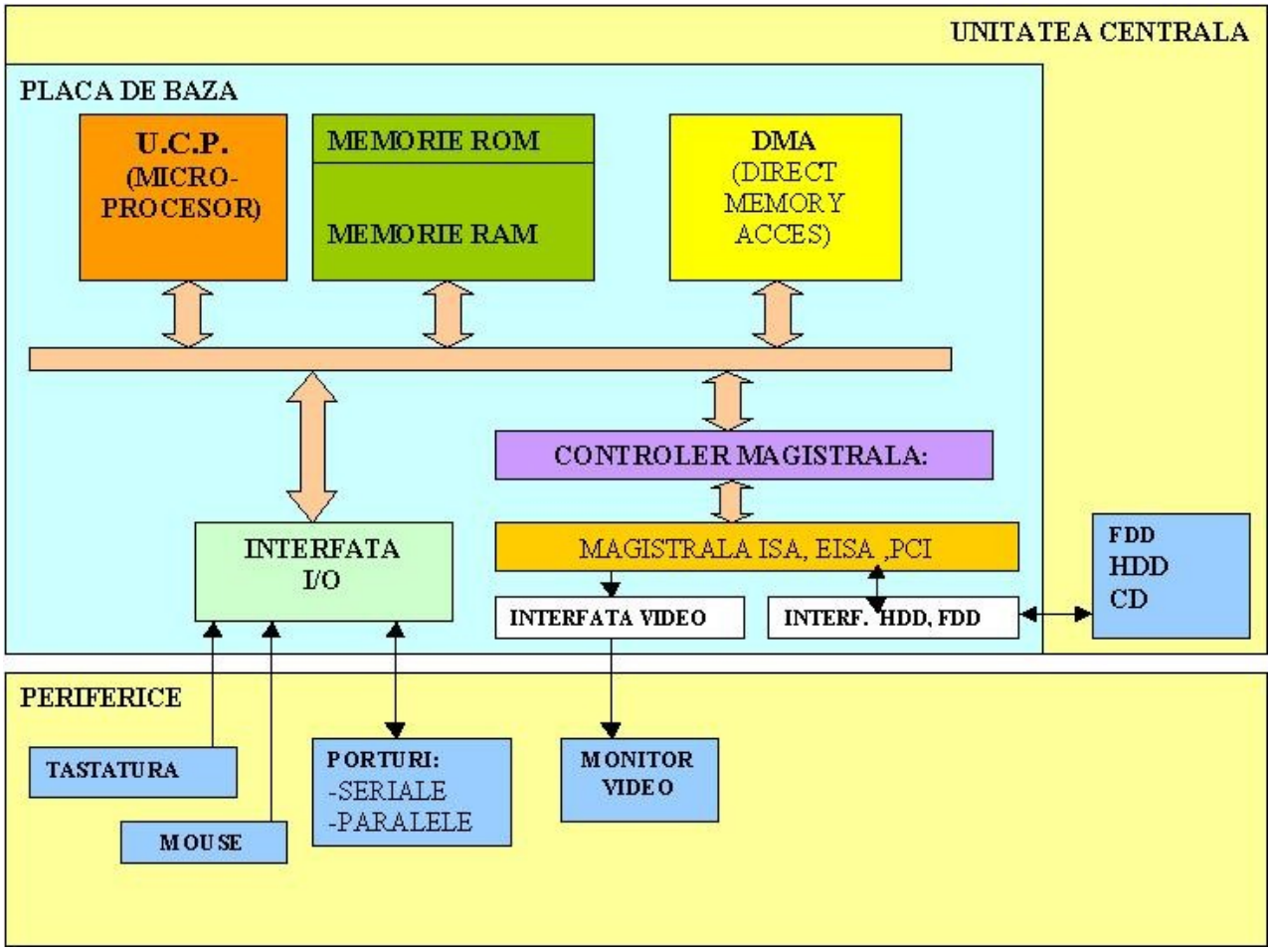
Prima instructiune de la adresa specificata de UC este pusa de memorie pe magistrala de date si UC-u o poate citi si pe urma executa. Ciclul se repeta pentru urmatoarele instructiuni din program pana la terminarea programului.

Rezultatele prelucrarii sunt puse de UC in memorie prin operatii de scriere pe parcursul rularii programului. Din memorie rezultatele sunt trimise la dispozitivele de iesire pentru afisare.

Viteza cu care sunt executate operatiile de UC este data de frecventa de lucru a Microprocesorului.

Frecventa de lucru actuala atinge ordinul Mhz.

Schema bloc functionala a unui calculator

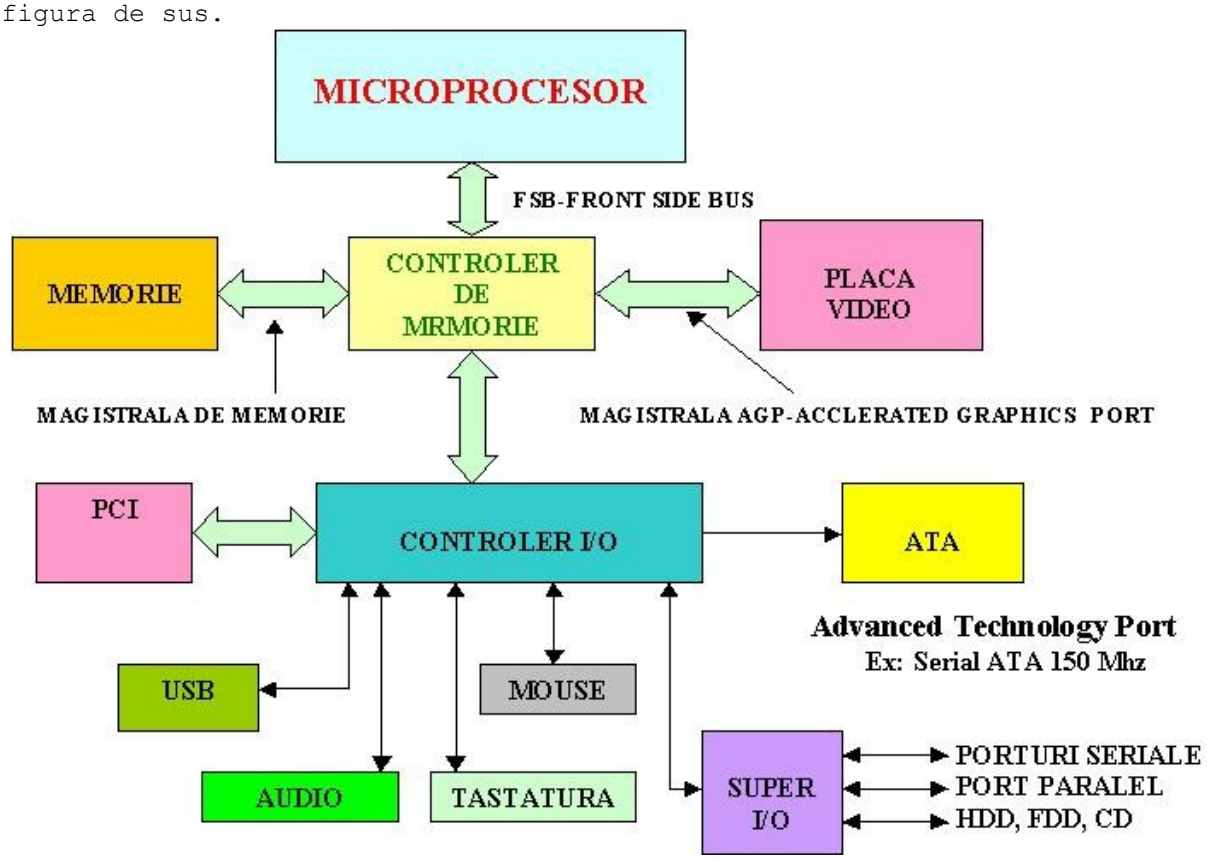


Unitatea centrala
Periferice (Tastatura, Mouse, Monitor video, ..)

Unitatea centrala contine:
PLACA DE BAZA
UCP
Memoria
Diferite interfete
MEMORIA EXTERNA (FDD, HDD, CD)

Exemplu de implementare

Schema bloc a unei variante constructive a placii de baza este prezentata in figura de sus.



- **Imagini placi de baza**

In imaginile de mai sus sunt prezentate cateva placi de baza .

Elemente de programare în limbajul C++.

Elemente de baza

Program de calculator

Set de instructiuni scris de programatori in vederera rularii pe un calculator.

Limbaj de programare

Este un set bine definit de expresii si reguli (sau tehnici) valide de formulare a instructiunilor pentru un computer. Un limbaj de programare are definite un set de reguli sintactice si semantice.

Limbajul de programare C

Limbajul C a fost creat la începutul anilor '70 de catre Brian W Kernigham si Dennis M Ritchie de la Bell Laboratories New Jersey
Caracteristicile distinctive ale limbajului au fost clar definite de la început, ele pastrîndu-se în toate dezvoltarile ulterioare:

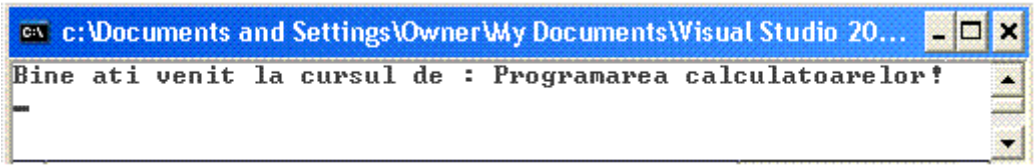
- portabilitate maxima;
- structurare;
- posibilitatea efectuarii operatiilor la nivelul masinii cu pastrarea caracteristicilor unui limbaj evoluat.

Este un limbaj de o importanta cruciala in lumea programarii, drept pentru care exista o serie de variante standardizate. Cel mai important standard ce ofera o variana standardizata a limbajului C este standardul ANSI

Primul program ANSI C

```
// Primul program scris in C++ Visual Studio 2005 de tipul:ANSI C

#include "stdafx.h"
#include < stdio.h >
int main() {
char c[1];
printf("Bine ati venit la cursul de : Programarea calculatoarelor!");
gets(c);
return 0;
}
```

Acest program afiseaza mesajul: "Bine ati venit la cursul de : Programarea calculatoarelor!".



De obicei un program nu "incepe de la 0", in sensul ca un programator trebuie sa se concentreze asupra problemei de rezolvat si nu asupra lucrurilor de rutina de genul cum sa afisez ceva pe ecran. Exista o serie de "programe" numite functii grupate in biblioteci si care rezolva problemele des intalnite. Astfel prima linie indica faptul ca se folosesc functii de intrare / iesire, iar descrierea modului de utilizare (numele, tipul argumentelor, tipul valorii returnate etc) a acestora se afla în fisierul cu numele stdio.h . Programul scris de programator va fi gazduit tot intr-o functie. Fiind cea mai importanta functie, ea se numeste functia main si va contine instructiunile programului. In cazul programului de sus, cea mai importanta instructiune este un apel al functiei printf care afiseaza un mesaj la terminal. Mesajul este dat între ghilimele si se termina cu un caracter special new-line (\n). Daca programul ar contine numai aceasta linie, s-ar afisa mesajul pe consola standard (ecranul) dupa care fereastra in care se afiseaza acest mesaj dispare si practic nu vedem nimic. Se mai introduce si apelul functiei **gets()** care asteapta introducerea unui text de la dispozitivul satandard de intrare adica tastatura. Textul introdus va fi pastrat in variabila c pe care am definit-o in prima instructiune ca fiind de tip char (un sir de caractere de lungime 1). Functia **gets()** asteapta un text care sa se termine cu **Enter**. Cum in cazul de fata nu astept decat apasarea unei taste, este suficient sa apas tasta **Enter**. Din acest motiv am definit constanta c, de tip caracter de lungime 1. Instructiunea return preda controlul sistemului de operare la terminarea programului si comunica acestuia codul 0 pentru terminare. Prin conventie, aceasta valoare semnifica terminarea normala a programului – adica nu au aparut erori în prelucrarea datelor. Corpul functiei main apare între acolade. Orice program C trebuie sa aiba o functie main.

Functia gets() asteapta introducerea unui text, care in prealabil trebuie definit. Avand in vedere ca in programul de sus nu ne intereseaza textul introdus, am putea folosi o functie care asteapta introducerea unui caracter indiferent care. Functia se numeste getch() si se gaseste in biblioteca "conio.h"

```
#include "stdafx.h"
#include < stdio.h >
#include < conio.h >
int main() {
printf("\n\tS-a utilizat functia getch() pentru a astepta apasarea unui
caracter");
printf("\n\n\n\t\tApasati orce tasta pentru iesire!\a");
getch();
return 0;
}
```

Dupa cum se vede, s-au folosit caracterele speciale : \n;\t;\a; Caracterul special \n determina cursorul sa avanseze pe un rand nou, \t determina cursorul sa avanseze un tab iar \a emite un sunet scurt.

Programarea in C++

Programele C care respecta standardul ANSI C pot fi rulate pe orice mediu de programare C si sub orice sistem de operare.

Din pacate nucleul C standard ANSI C nu este acoperitor pentru diferite implementari mai noi de tip Object-Oriented Programming (OOP) si "Visual".

C++ include noi tehnici procedurale de programare. Daca programarea clasica in C este o programare structurata modulara , programarea C++ include si programarea orientata obiect OOP . Obiectele sunt noi tipuri ce integreaza atat datele cat si metodele asociate crearii, prelucrarii si distrugerii acestor date. Un obiect este definit de o clasa. Clasa reprezinta structura care defineste caracteristicile abstracte ale unui obiect.

O clasa contine functii si date numite functii membru respectiv date membru. Functiile membru se mai numesc si metode. Lansarea unei functii membru se mai numeste si invocarea unei functii membru. Un obiect se obtine prin instantierea unei clase. Prin instantierea unei clase, se obtine deci un obiect sau o instanta. Clasa este un concept de baza al programarii orientate obiect.

Vom folosi in continuare mediului de programare: Visual Studio 2005, fiind un mediu OOP Visual.

Va trebui deci sa analizam si extensiile limbajului C implementate in Visual Studio 2005, mediu ce ne va permite sa utilizam facilitatile OOP si Visual.

Sa luam de exemplu aplicatia de mai jos:

```
// Primul program scris in C++ Visual Studio 2005

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

int main(void)
{
	cout <<" Primul program scris in C++ Visual Studio 2005\n\n
Felicitari!!";
	cin.get();
	return 0;
}
```

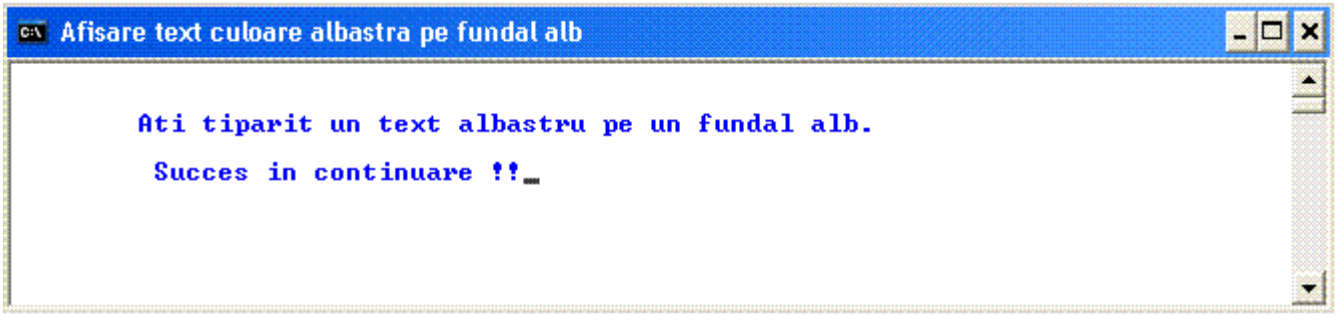
Programul ruleaza in fereastra "Command" pe un fundal negru cu caractere albe. Exista posibilitatea schimbarii atributelor ferestrei "Command" din program. Putem schimba de exemplu culorile, titlul ferestrei, etc. Urmatoarea aplicatie reia aplicatia de sus dar cu cateva modificari de atribute ale ferestrei "Command".

```
// Program scris in C++ Visual Studio 2005
// Se afiseaza un text si se modifica diverse atribute ale ferestrei "Command".
#include "stdafx.h"
#include < iostream >
using namespace std;

int main(void)
{
	system("TITLE Afisare text culoare albastra pe fundal alb ");// Titlul
ferestrei consola
	system("COLOR F9"); // Fundal alb caractere albastre
	cout <<" \n\n\tAti tiparit un text albastru pe un fundal alb. \n\n\t
Succes in continuare !!";
	cin.get();
	return 0;
}
```

In instructiunea: **system("color F9");** , prima cifra reprezinta culoarea fundalului iar cifra a doua reprezinta culoarea textului dupa cum urmeaza:

- 0 = Black
- 1 = Blue
- 2 = Green
- 3 = Aqua
- 4 = Red
- 5 = Purple
- 6 = Yellow
- 7 = White
- 8 = Gray
- 9 = Light Blue
- A = Light Green
- B = Light Aqua
- C = Light Red
- D = Light Purple
- E = Light Yellow
- F = Bright White

Dupa rularea aplicatiei obtinem:



Structura unui program C++

Sa analizam structura programului afisat mai jos:

```
// Primul program scris in C++ Visual Studio 2005 de tipul:CLR (Common Language
Runtime) console application

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

int main(void)
{
        cout <<" Primul program CLR console application\n\n Felicitari!!";
        cin.get();
        return 0;
}
```

Comentarii

Sunt precedate de // si pot aparea oriunde in program. Nu sunt executate de calculator, ele fiind destinate celor care scriu sau citesc programele.
Comentariile sunt plasate in vederea explicitarii programelor si pentru intelegerea mai usoara a programelor de catre alti programatori sau chiar de autorul programului. In cazul de fata linia de comentariu :

// Primul program scris in C++ Visual Studio 2005 de tipul:CLR (Common Language Runtime) console application

da informatii despre mediul de programare in care a fost scris programul.

Directiva #include

C++ contine un numar de fisiere de biblioteca standard unde sunt incluse functii si obiecte utilizate frecvent. Aceste fisiere sunt sunt grupate in biblioteci
#include < iostream > include biblioteca **iostream** ce contine functii si obiecte pe care le vom folosi in functia **main** , de exemplu **cout >>**
In cazul in care avem mai multe biblioteci de inclus sau daca pentru orce program trebuie neaparat sa includem niste biblioteci, in loc de numele bibloteciI se poate indica un fisier ce contine toate numele bibloteciLor. Fisierul are de obicei extensia .h .In cazul primului program scris anterior, #include "stdafx.h" include bibliotecile si directivele de compilare scrise in fisierul stdafx.h

```
// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

#pragma once


#define WIN32_LEAN_AND_MEAN    // Exclude rarely-used stuff from Windows headers
#include < stdio.h>
#include < tchar.h>


// TODO: reference additional headers your program requires here
```

#include < iostream > ar putea fi inclusa in fisierul stdafx.h, in acest caz ne mai fiind nevoie s-o includem in programul principal.

Functia main

Functia este un grup de instructiuni scrise in vederea realizarii unei sarcini. Functia este referita prin nume. In cazul primului program scris mai sus, **main** este numele functiei principale. Un program poate avea mai multe functii si de aceeea functia care se lanseaza prima in cadrul executiei programului trebuie sa poarte denumirea **main**

Spatiul de nume

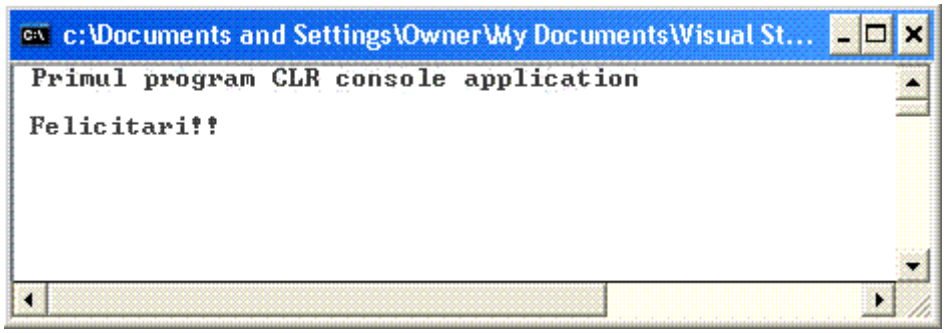
Instructiunea **using namespace std;** este un spatiu de nume. C++ foloseste spatii de nume (namespaces) pentru a organiza clase cu functionalitati inrudite, folosite in cadrul programelor. Si bibliotecile folosesc spatii de nume , astfel clasele din biblioteca sunt referite prin intermediul spatiului de nume.

Corpul functiei

Toate instructiunile functiei sunti incluse in corpul functiei. Un corp de functie incepe cu { si se termina cu } De obicei fiecare instructiune se termina cu ; In cazul primului program scris anterior functia **main** contine cele trei instructiuni afisate mai jos. Instructiunile se executa secventiial de sus in jos.

```
    cout <<" Primul program CLR console application\n\n Felicitari!!";
    cin.get();
    return 0;
```

Instructiunea **cout <<** afiseaza pe consola standard ,Textul : "Primul program" pe randul 1 iar pe randul 3 textul: Felicitari. \n comanda trecerea pe randul urmator. La inceput cursorul se afla pe randul 1 unde va scrie textul : "Primul program" . Dupa secventa \n\n cursorul se muta pe randul 3 unde va scrie textul : "Felicitari !!"



Declaratia **cin.get();** reprezinta invocarea metodei get a obiectului cin. Pentru precizarea metodei get au fost deci necesare precizarea tuturor componentelor adica:

obiect . metoda

In cazul ca spatiul de nume nu este prezentat in antet,atunci forma generala este:

spatiu de nume :: obiect . metoda

Metoda **get();** asteapta apasarea tastei Enter. Invocarea acestei metode este necesara deoarece dupa scrierea textului pe consola, aceasta ar disparea foarte repede fara sa ajungem sa citim textul afisat. Textul afisat ar trebui completat cu mesajul: "Tastati Enter"

Daca nu am fi folosit spatiul de nume **using namespace std;**, instructiunile din corpul functiei **main** ar fi trebuit scrise:

```
        std::cout <<" Primul program CLR console application \n\n Felicitari!!";
        std::cin.get();
        return 0;
```

Practic primul programul, dupa modificarea stadfix.h , poate sa arate astfel:

```
// Primul program scris in C++ Visual Studio 2005 de tipul:CLR console
application

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

int main(void)
{
        std::cout <<" Primul program  CLR console application\n\n
Felicitari!!\n\n Tastati Enter";
        std::cin.get();
        return 0;
}
```

Invocarea metodei **get();** are rolul de a astepta apasarea unei taste altfel fereastra consola in care se afiseaza textul dispare fara a avea timp sa citim

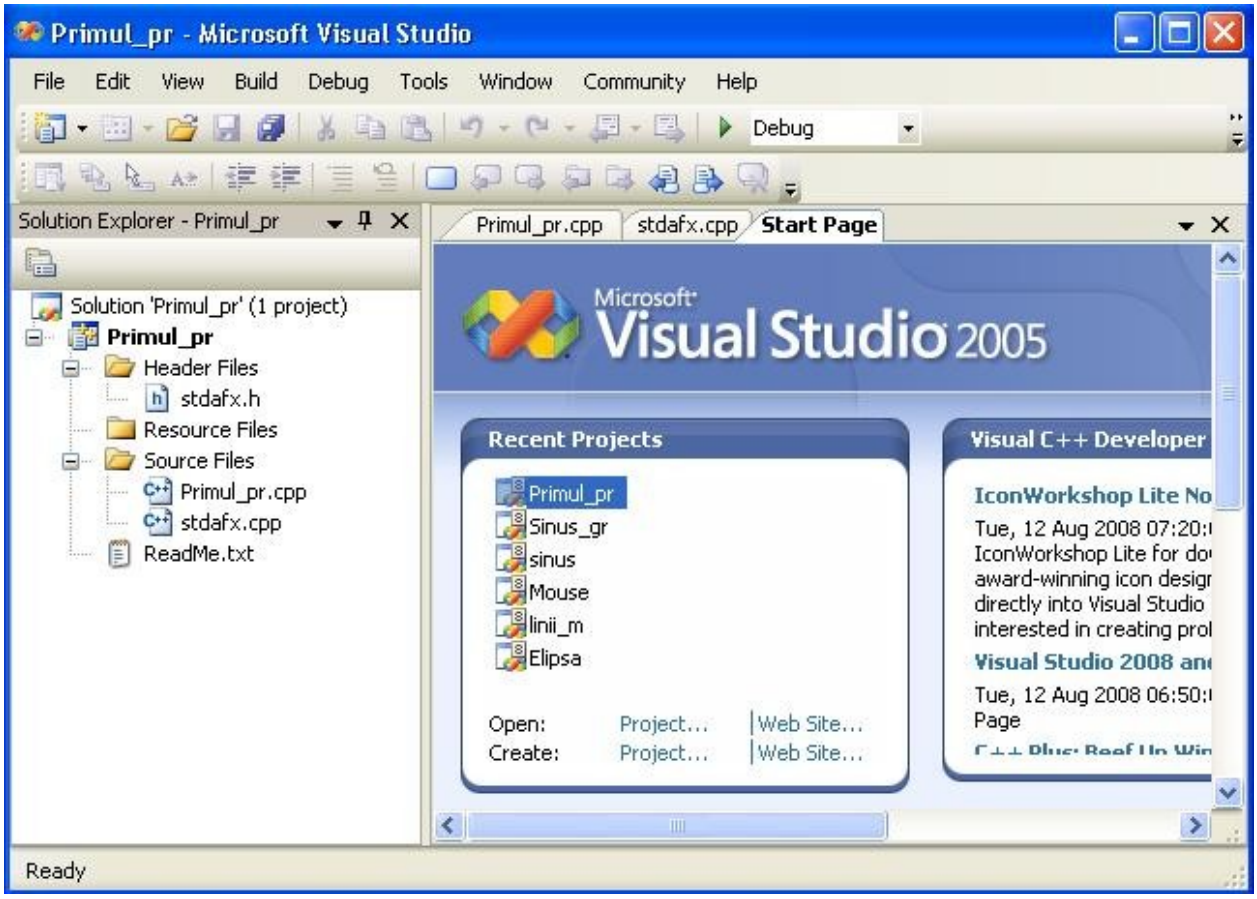
textul afisat.
    O alta metoda care ne permite sa citim textul afisat, este reprezentata de folosirea functiei sleep() care determina "inghetarea" programului pentru un timp determinat, astfel avem timp sa citim cosola.

```
 // Programul  utilizeaza functia sleep din biblioteca

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

int main(void)
{
        cout << " Va urez o zi buna!!";
        Sleep(1000);
        cout << " \n\n\t Salut!!";
        Sleep(500);
        return 0;
}
```

    Un alt avantaj il constituie asiatarea permanenta in momentul scrierii liniilor de program care contin denumirea metodelor si claselor din acest spatiu de nume.

Utilizare Visual Studio 2005 - CLR(Common Language Runtime) Console Application

IDE(Integrated Developement Environment) utilizat :Visual studio 2005



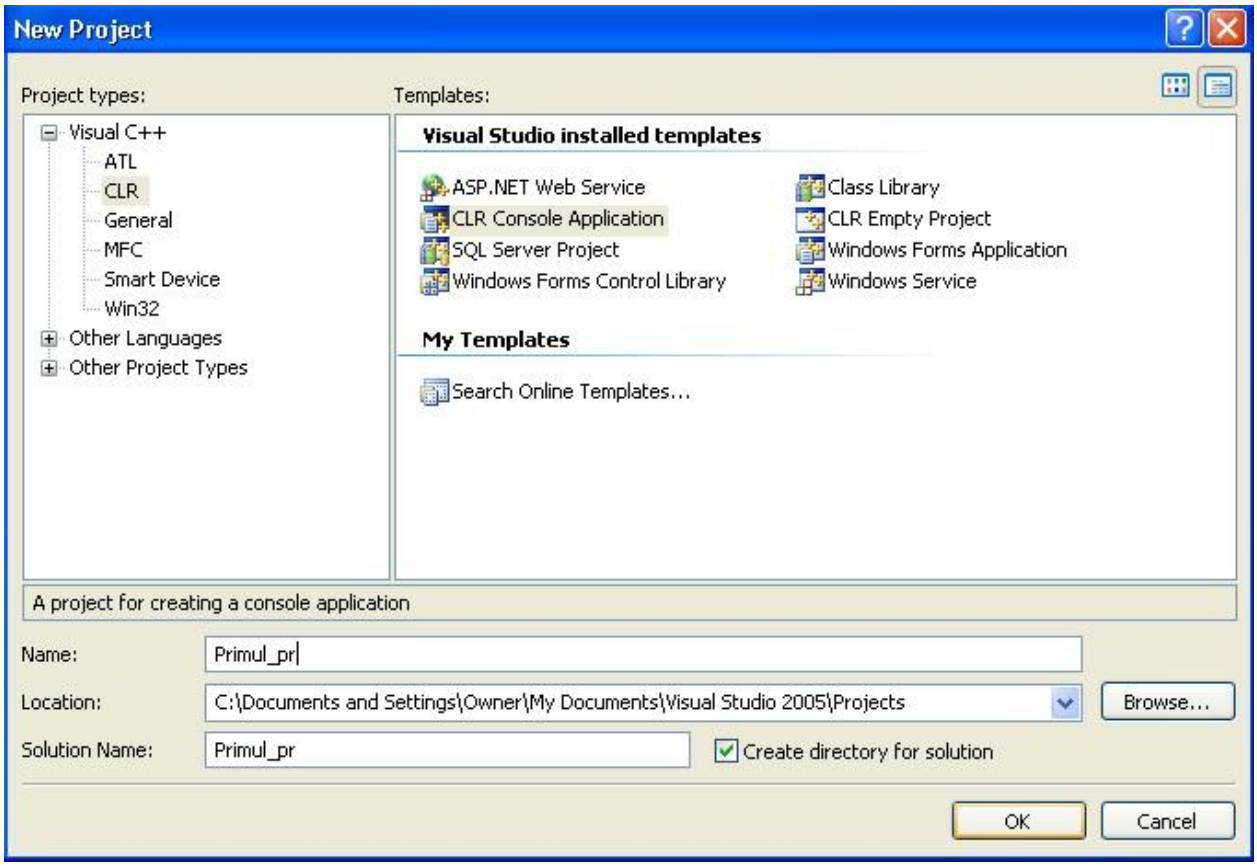
Are o interfata grafica GUI (Graphics User Interface) si contine:

- editor de texte
- preprocesor
- compilator
- link-editor

IDE-uri similare:Borland C++ Builder, IBM VisualAge, etc.

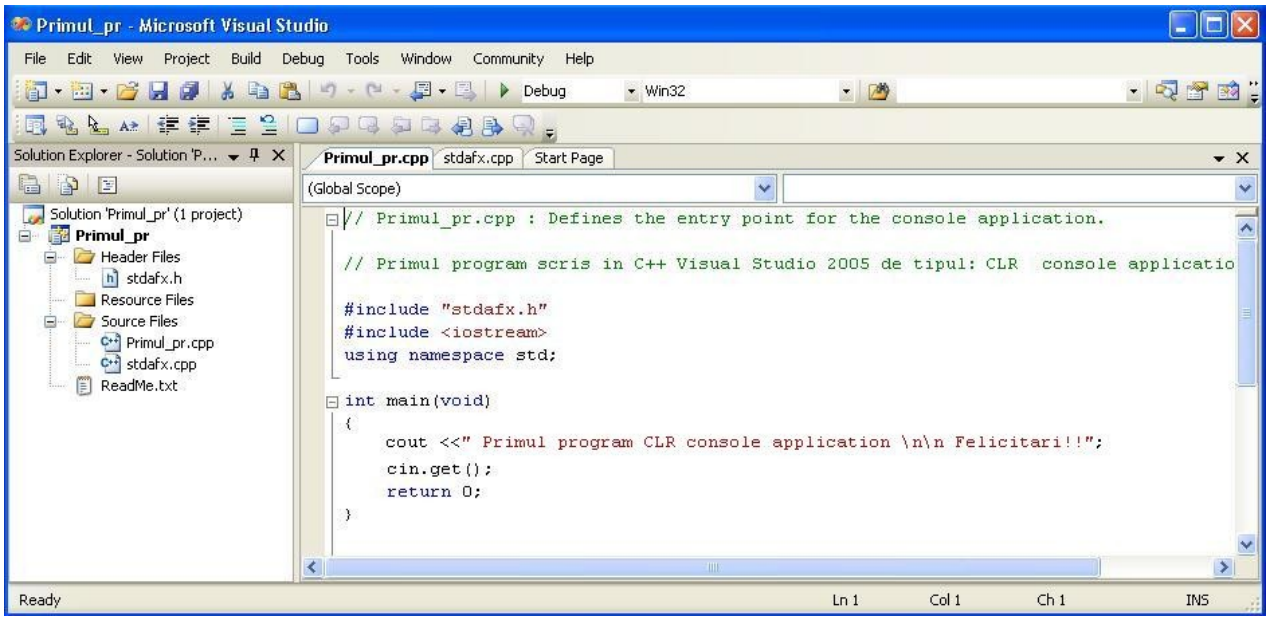
Crearea proiectelor Visual Studio 2005 de tipul CLR Console Application

- Lansare Visual Studio 2005
- File-New Project
- Se alege - Visual C++ CLR (Common Language Runtime)-CLR Console Application si se completeaza numele proiectului, in cazul de fata Primul_pr



Scrierea codului sursa

- Se alege fisierul Primul_pr.cpp
- Se scrie codul sursa al programului primul_pr

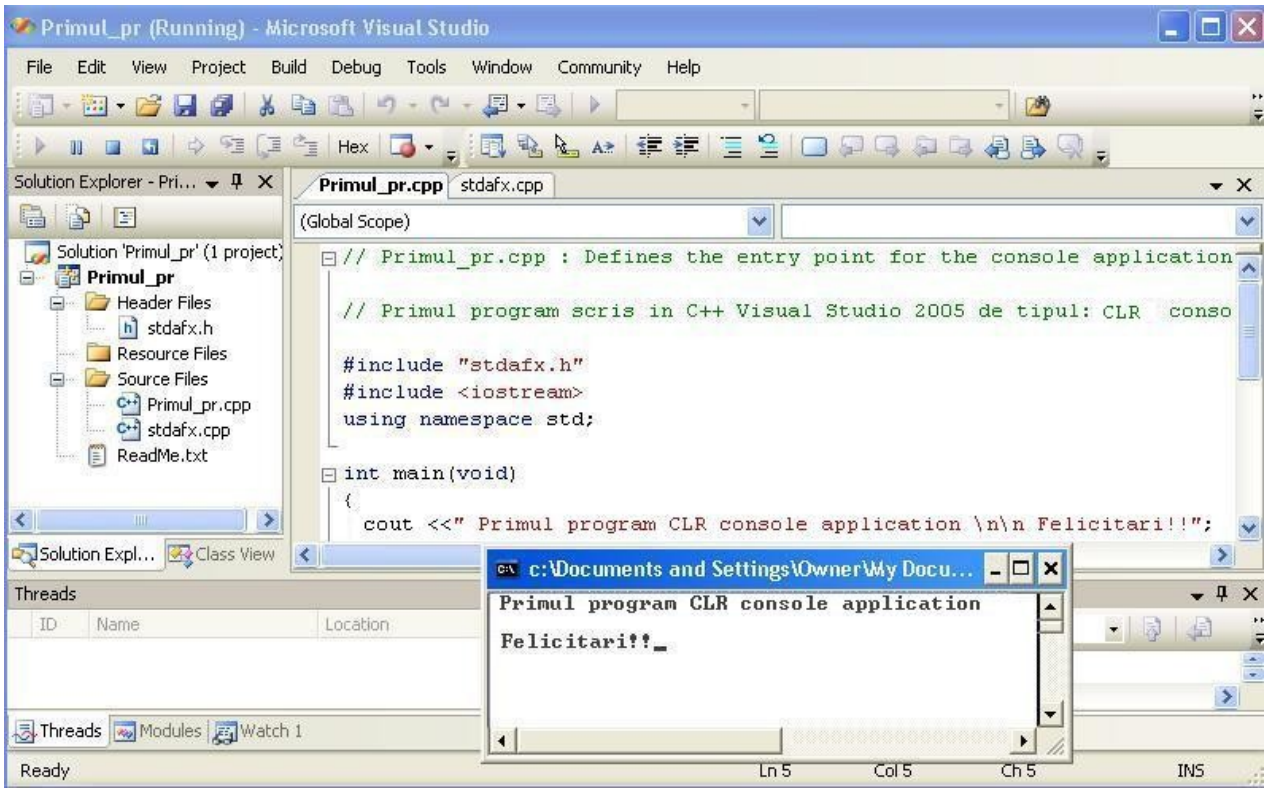


Constructia proiectului

- Build- Build solution

Rularea codului

- Debug-Start Debugging



Spatiul de nume System::

Spatiul de nume **System::** contine clase fundamentale, de baza si clasele frecvent folosite, interfete, metode de manipulare a evenimentelor. Folosind acest spatiu, avem posibilitatea de a utiliza diverse metode pentru a defini si converti diferite tipuri de date necesare in programarea de tip "Visual" pe care o vom utiliza in cadrul modulului Windows Forms Application.

Spatiul de nume **System::** contine si functii pentru consola, putand astfel sa dezvoltam atat aplicatii de tipul CLR Console Application cat si aplicatii de tipul Windows Forms Application. Functiile pentru consola se gasesc in clasa **Console::** Functiile definite in interiorul unei clase poarta numele de metode ale clasei. Apelarea unei metode (functi) definite in cadrul unei clase se face prin precizarea tuturor componentelor si anume:

spatiul de nume:: clasa:: metoda ( )

Pentu a apela functia **ReadLine** de exemplu vom utiliza: **System::Console::ReadLine()**

Declaratia de mai sus poate fi citita si astfel: "Se invoca metoda *ReadLine* a clasei *Console* aflata in spatiul de nume *System* .

Pentru a afisa de exemplu textul :" Succes la examene !! " , vom invoca metoda *WriteLine* a clasei *Console* aflata in spatiul de nume *System* .

Mai jos se prezinta un program de tipul CLR Console Application, program ce invoca metode din clasa *Console* din spatiul de nume *System* .

```
// sp_system.cpp : main project file.
// Primul program ce utilizeaza functii consola din spatiul de nume System::

#include "stdafx.h"

using namespace System;

int main(void)
{
        Console::WriteLine("Program ce utilizeaza spatiul de nume System::");
        Console::ReadLine();
    return 0;
}
```

Utilizare Visual Studio 2005 - Windows Forms Application

Dupa cum am amintit, mediul de programare "Visual Studio 2005" este un mediu visual de programare orientata obiect(OOP), permitand realizarea de aplicatii OOP folosind modul "Visual". Obiectele dorite sunt plasate pe o planseta de design (form) dupa care sunt completate si adaugate diverse secvente de cod pentru a stabili comportarea dorita pentru obiectul plasat.

Notiuni utilizate in OOP

- **Obiecte** - sunt noi tipuri ce integreaza atat datele cat si metodele (functiile) asociate crearii, prelucrarii si distrugerii acestor date.
- **Clasa** - reprezinta structura care defineste caracteristicile abstracte ale unui obiect.
- **Metode** - functiile membru definite in cadrul unei clase
- **Prporietati** - membrii unei clase care permit accesul controlat la datele membru ale unei clase
- **Evenimente** - membrii unei clase care permit clasei sau obiectelor clasei sa faca notificari, adica sa anunte celelalte obiecte asupra unor schimbari petrecute la nivelul starii lor.

Apelarea unei metode (functi) sau setarea unei proprietati, ale unui obiect se face prin precizarea tuturor componentelor si anume:

nume_aplicatie::nume_form::nume_obiect-> metoda sau proprietate

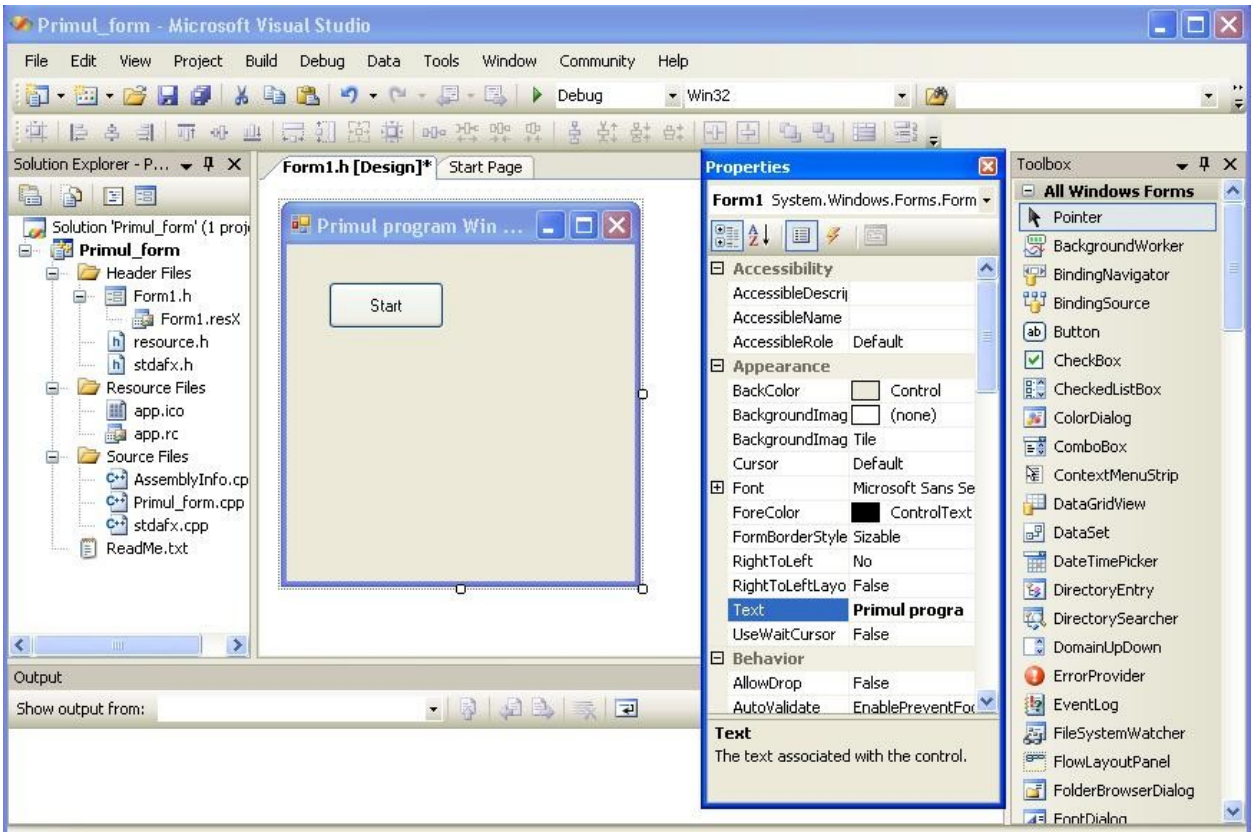
Componentele **nume_aplicatie::nume_form::** pot fi inlocuite cu **this** deci precizarea unei metode sau a unei proprietati, devine:

this->nume_obiect-> metoda sau proprietate

Realizarea proiectelor Visual Studio 2005 de tipul CLR-Windows Form Application

Lansare Visual Studio 2005

- File-New Project
- Se alege - Visual C++ CLR (Common Language Runtime)-Windows Form Application si se completeaza numele proiectului, in cazul de fata " Primul_form"

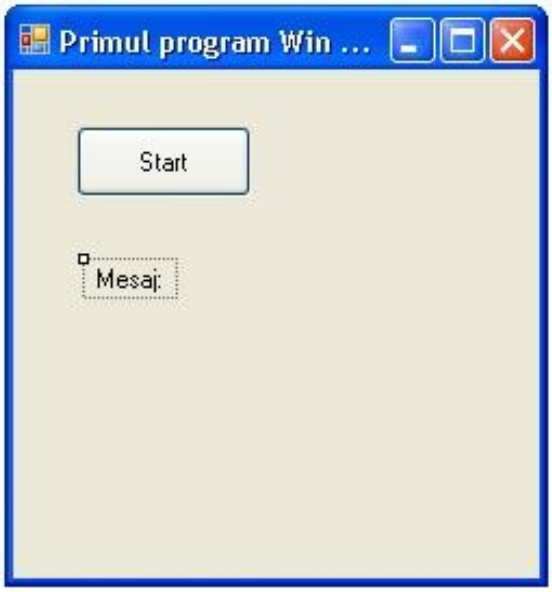


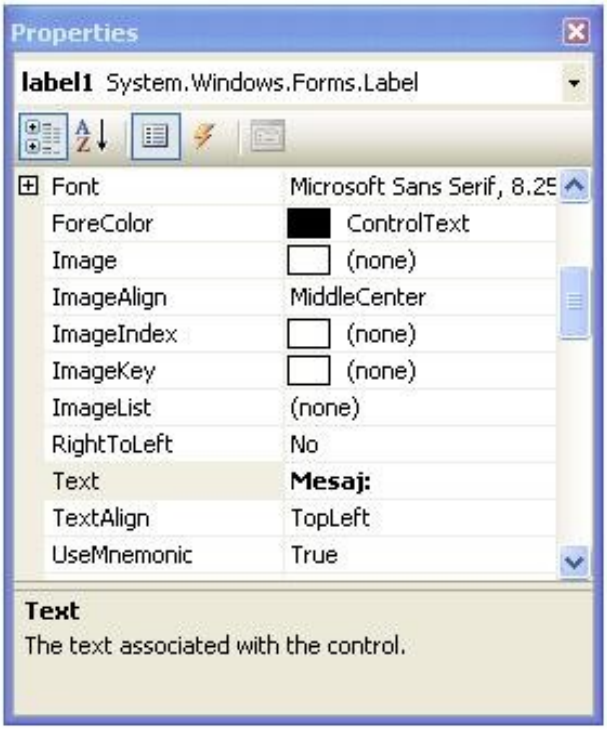
□ Plasare buton Start

```
Din ToolBox se alege All windows Forms--Button si se plaseaza pe Form-ul
deschis.
Se selecteaza butonul plasat pe Form cu click dreapta si se alege din meniul
deschis optiunea "Properties"
Se selecteaza propietatea "Text" si i se va atribui valoarea "Start". In acest
moment pe buton va scrie "Start"
Se selecteaza de aceasta data Formul cu click dreapta, se alege din meniul
deschis optiunea "Properties"
Se selecteaza propietatea "Text" si i se va atribui valoarea "Primul program
Win Form App". In acest moment titlul ferestrei principale va fi:"Primul
program Win Form App"
```

□ Plasare label

```
Din ToolBox se alege All windows Forms--Label si se plaseaza pe Form-ul
deschis.
Se selecteaza label-ul plasat pe Form cu click dreapta si se alege din meniul
deschis optiunea "Properties"
Se selecteaza propietatea "Text" si i se va atribui valoarea "Mesaj". In acest
moment pe buton va scrie "Mesaj"
```





□ Scrierea codului

Daca selectam butonl **start** cu click dreapta si se alegem din meniul deschis optiunea "Properties" gasim la proprietatea Name setat numele **button1**, atribuit automat la plasarea butonului. La fel pentru labelul pe care am setat textul mesaj, vom gasi numele **label1**

Ne propunem sa scriem un program care sa afiseze textul:"Primul program Windows Forms Application" la apasarea butonului "Start".
Se selecteaza butonul "Start" cu click dreapta si se alege din meniul deschis optiunea "Properties". Se apasa butonul "Events".
Se alege Action--Dublu click pe optiunea Click.In acest moment evenimentului "click" al butonului "Start" i se genereaza un schelet de procedura care va trata evenimentul click al butonului button1 numita: **button1_click** pe care trebuie sa-l completam cu liniile de instructiune necesare pentru a trata evenimentul click.In cazull de fata: cu instructiunea:
this->label1->Text="Primul program Windows Forms Application";
adica pe formul curent (this) obiectului label1 sa atribuim proprietatii Text valoarea : "Primul program Windows Forms Application"
Procedura care trateaza evenimentul click va fi deci:

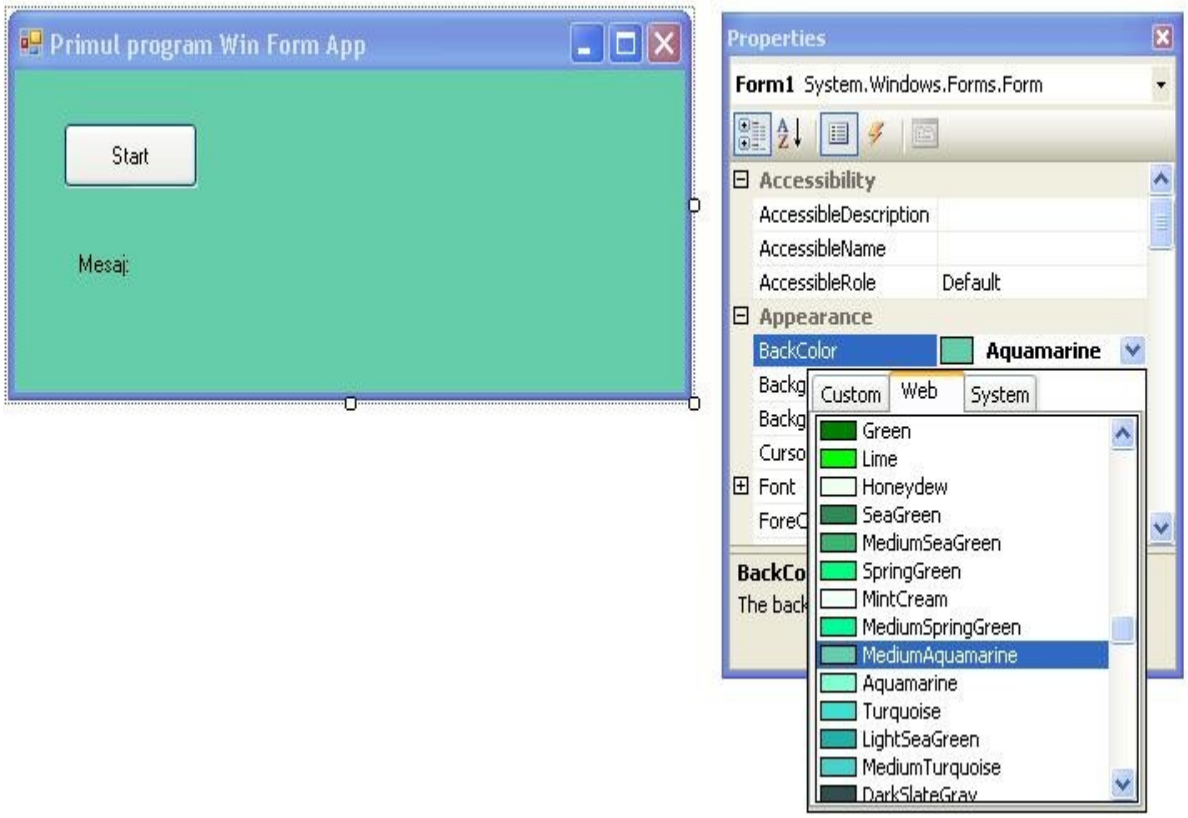
```
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^  e) {

             this->label1->Text="Primul program Windows Forms Application";
         }
```

□ Rulare aplicatie

In acest moment prin apasarea butonului "Start debug" se lanseaza aplicatia si putem apasa butonul "Start" din aplicatie. Dupa aceste operatii, ecranul aplicatiei arata astfel:

□ Modificare atribute

Revenim la Form Design si modificam diferite proprietati (atribute) ale obiectelor amplasate pe form .
Sa modificam de exemplu BackColor pentru fereastra principala:



Putem modifica dupa preferinta proprietatile tuturor obiectelor amplasate pe form. Putem modifica de exemplu Fonturile si culoarea mesajului.

Varianta finala

Dupa ce am efectuat toate modificarile si am testat aplicatia, in Folder-ul project gasim Folderul:Primul_form apoi in Folderul debug: gasim executabilul "primul_form.exe" pe care il putem lansa in executie.

Tipuri de date, date simple, operatori, expresii

Tipuri de date în limbajul C++

Tipul unei date determina lungimea zonei de memorie ocupata de acea data.

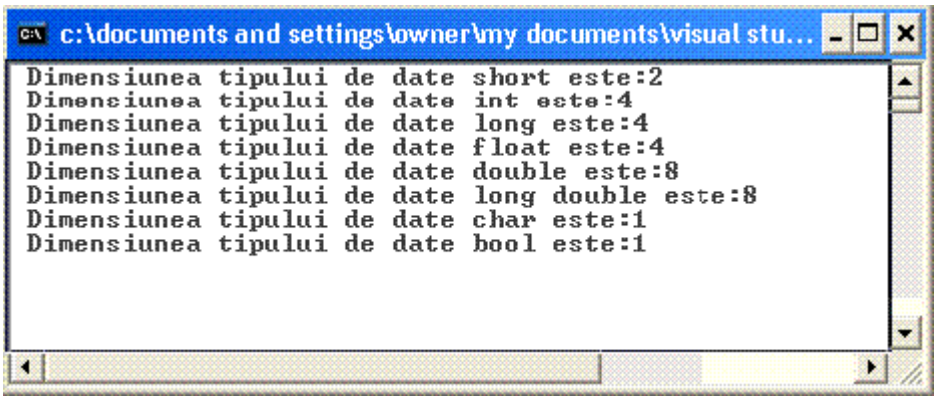
Tipurile de baza sunt:

- **char** un singur octet (1 byte=8 biti), capabil sa contina codul unui caracter din setul local de caractere;
- **int** numar întreg, reflecta în mod tipic marimea naturala din calculatorul utilizat;
- **float** numar real, în virgula mobila, simpla precizie;
- **double** numar real, în virgula mobila, dubla precizie.
-

Program C++ pentru determinarea dimensiunii diferitelor tipuri de date

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// Determinarea dimensiunii diferitelor tipuri de date
#include "stdafx.h"
#include < iostream >
using namespace std;

int main(void)
{
cout <<" Dimensiunea tipului de date short este:" << sizeof(short) <<"\n";
cout <<" Dimensiunea tipului de date int este:" << sizeof(int) <<"\n";
cout <<" Dimensiunea tipului de date long este:" << sizeof(long) <<"\n";
cout <<" Dimensiunea tipului de date float este:" << sizeof(float) <<"\n";
cout <<" Dimensiunea tipului de date double este:" << sizeof(double) <<"\n";
cout <<" Dimensiunea tipului de date long double este:" << sizeof(long double)
<<"\n";
cout <<" Dimensiunea tipului de date char este:" << sizeof(char) <<"\n";
cout <<" Dimensiunea tipului de date bool este:" << sizeof(bool) <<"\n";
cin.ignore();
cin.get();
return 0;
}
```

Dupa rularea programului in "Command Prompt" se afiseaza:



Constante

O constanta este un literal (o forma externa de reprezentare) numeric, caracter sau sir de caractere. Numele si valoarea unei constante sunt identice. Valoarea unei constante nu poate fi schimbata în timpul executiei programului în care a fost utilizata. Tipul si valoarea ei sunt determinate în mod automat, de catre compilator, pe baza caracterelor care compun literalul.

Constante intregi

O constanta întreaga consta dintr-o succesiune de cifre.Exemplu: 36789

Constante numerice reale

Daca o constanta numerica contine punctul zecimal, ea este de tipul double. Exemplu: 3.141 tip double
Daca numarul este urmat de L sau l, este de tip long double. Exemplu 3.7659L--tip long double

Constante numerice flotante

O constanta flotanta consta dintr-o parte întreaga, un punct zecimal, o parte fractionara, litera e sau E, si optional un exponent care este un întreg cu semn

Constante caracter

O constanta caracter consta dintr-un singur caracter scris între apostrofuri, de exemplu 'x'

Constante siruri

Un sir este o succesiune de caractere scrise între ghilimele, de exemplu "Programare".

Exemplu de program ce utilizeaza constante

In cadrul unei instructiuni de scriere pe consola standard vor fi folosite de data aceasta expresii cum ar fi:**"\tValoare\t\t= "+cant*pret** in care functia de scriere nu mai are numai un singur argument ci o intreaga expresie.

Sunt folosite de asemenea si secvente escape cum ar fi \t\n\a etc, semnificatia lor fiind urmatoarea:

\n new-line	\r carriage return	\\ backslash
\t tab orizontal	\f form feed	\' apostrof
\b backspace	\a semnal sonor	\" ghilimele

Program pentru calculul valorii unei facturi (inclusiv TVA**)**

```
// utilizarea diferitelor tiputi de constante

#include "stdafx.h"

using namespace System;

int main(void)
{
        Console::WriteLine(L"\n\n\t\tValoare marfa\n\n\a");
        const double pret=123.25;
        const int cant=23;
        const double tva=0.19;
        String^ linie_o = "\t--------------------------------";
        Console::Write("\tPret\t\t= "+pret);
        Console::WriteLine("\n\tCantitate\t= "+cant);
        Console::WriteLine(linie_o);
        Console::Write("\tValoare\t\t= "+cant*pret);
        Console::WriteLine("\n\n\tTVA 19 % \t= "+pret*cant*tva);
        Console::WriteLine("\n\n\ "+linie_o);
        Console::WriteLine("\tTotal General\t= "+pret*cant*(1+tva));
        Console::ReadLine();
        return 0;
}
```

Variable

Spre deosebire de constante, variabilele sunt date (obiecte informationale) ale caror valori se pot modifica în timpul executiei programului.

Si variabilele sunt caracterizate de atributele nume, tip, valoare si clasa de memorare.

Variabilele sunt nume simbolice utilizate pentru memorarea valorilor introduse pentru datele de intrare sau a rezultatelor.

Daca la o constanta ne puteam referi folosind caracterele componente, la o variabila ne vom referi prin numele ei.

Numele unei variabile ne permite accesul la valoarea ei, sau schimbarea valorii sale, daca este necesar acest lucru.
Numele unei variabile este un identificator ales de programator.

Sintaxa de declarare a variabilelor

Inainte de folosire, orice variabila trebuie declarata, pentru a i se putea aloca memorie in vederea stocarii ei. Variabila este defapt adresa primei locatii in care este rezervata memorie pentru continutul ei. Sintaxa pentru declararea unei variabile este urmatoarea:

[tip date] [nume variabila]

Exemple:
int anul;
int cantitate
string nume
double pi

Tipul de variabila caracter si string

O variabila de tip caracter se declara prin specificatorul de tip char. Zona de memorie alocata unei variabile de tip char este de un octet.
Variabila sir de caractere sau string se declara ca fiind masive de tip char sau in unele spatii de nume se declara string

Exemple de utilizare a variabilelor char si string .

- **Program pentru citirea si afisarea unui text utilizand ANSI C**

```
// Program pentru citirea unui sir de caractere si afisarea unui text

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

int main() {
char c[1];
char nume[40];
printf("\n\tCum va numiti:");
gets(nume);
printf("\n\n\tBine ati venit la cursul de : Programarea Calculatoarelor: ");
puts(nume);
gets(c);
return 0;
}
```

- **Program pentru citirea si afisarea unui text utilizand spatiu de nume std::**

```
// Program pentru citirea unui text (string sau sir de caractere) si afisarea
unui text

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

using namespace std;

int main(void)
{
        string nume;
        cout <<"\n\tIntroduceti numele:";
        cin >> nume;
        cout << "\n\n\tBine ai venit " << nume << "!\n";
        cin.ignore();
        cin.get();
        return 0;
}
```

Asupra sirurilor se pot face si operati. Cea mai cunoscuta este operatia de adunare a sirurilor sau concatenare.

Programul urmator face o astfel de operatie.

```
// concatenarea a doua siruri

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

using namespace std;

int main(void)
{
        string nume;
        string prenume;
        string nume_pr;
        cout <<"\n\tIntroduceti numele:";
        cin >> nume;
        cout <<"\n\tIntroduceti prenumele:";
        cin >> prenume;
        nume_pr=nume+" "+prenume;
        cout <<"\n\n\n\tNumele tau este: " << nume_pr;
        cin.ignore();
        cin.get();
        return 0;
}
```

Pentru a citi numele si prenumele cu o singura instructiune trebuie sa folosim o instructiune noua care nu ignora caracterele introduse dupa caracterul spatiu. Va trebui sa definim o variabila de tip caracter de dimensiune sa zicem 80.

```
// Utilizarea instructiunii cin.getline pentru a citi variabile string ce contin
spatii

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

using namespace std;

int main(void)
{
        char nume[80];
        cout <<"\n\tIntroduceti numele si prenumele:";
        cin.getline(nume,80);
        cout <<"\n\n\n\tNumele tau este: " << nume;
        cin.ignore();
        cin.get();
        return 0;
}
```

- **Program pentru citirea si afisarea unui text utilizand spatiu de nume System::**

Functia ReadLine citeste un text de la dispozitivul de intrare. Pentru a pastra valoarea citita trebuie sa declaram o variabila de tip sir de caractere si sa-i atribuim valoarea Console::ReadLine(). In spatiul de nume System:: o variabila sir se declara astfel:**String^ nume_sir** unde nume_sir este numele variabilei sir de caractere.

```
// Program pentru citirea si afisarea unui text utilizand spatiul de nume
System::

#include "stdafx.h"

using namespace System;

int main(void)
{
        String^ nume ;
        Console::WriteLine( "\n\n\tBuna ziua utilizatorule! " );
        Console::Write( "\n\n\tCum te numesti? :" );
        nume = Console::ReadLine();
        Console::WriteLine("\n\n\n\tSucces la programare "+ nume+" !\n" );
        Console::ReadLine();
    return 0;
}
```

Tipul de variabila numeric

Variabile intregi

Variabilele întregi pozitive sau negative pot fi declarate prin specificatorul de tip int.
Zona de memorie alocata unei variabile intregi poate fi de cel mult trei dimensiuni.
Relatii despre dimensiune sunt furnizate de calificatorii short, long si unsigned, care pot fi aplicati tipului int.
Calificatorul short se refera totdeauna la numarul minim de octeti pe care se reprezinta un întreg, de obicei 2.
Calificatorul long se refera la numarul maxim de octeti pe care poate fi reprezentat un întreg, de obicei 4.

Variabile flotante (virgula mobila)

Variabilele flotante pot fi în simpla precizie si atunci se declara prin specificatorul de tip float sau în dubla precizie si atunci se declara prin specificatorul double.
Majoritatea sistemelor de calcul admit si reprezentarea în precizie extinsa; o variabila în precizie extinsa se declara prin specificatorul long double.

Exemple de programe care folosesc variabile numerice

- **Program pentru citirea unui numar si afisarea patratulu acelui numar utilizand spatiul de nume std::**

```
// program pentru citirea unui numar si afisarea patratulu acelui numar utilizand
spatiul de nume std::
// citesc un numar de tip double si calculez patratul lui

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

int main(void)
{
        double nr;
        float nrp;
        cout <<" \n\tIntroduceti un numar:";
        cin >> nr;
        nrp=nr*nr;
        cout <<" \n\tPatratul numarului :"<< nr << "este : " << nrp;
        cin.ignore();
        cin.get();
        return 0;
}
```

Alte tipuri de variabila

Exista o multitudine de alte tipuri de variabile predefinite cum ar fi data si ora calendaristica.
Spatiul de nume System:: ofera spre exemplu tipul de variabila DateTime

☐ **Program se foloseste tipul de variabila System::DateTime pentru a afisa data si ora**

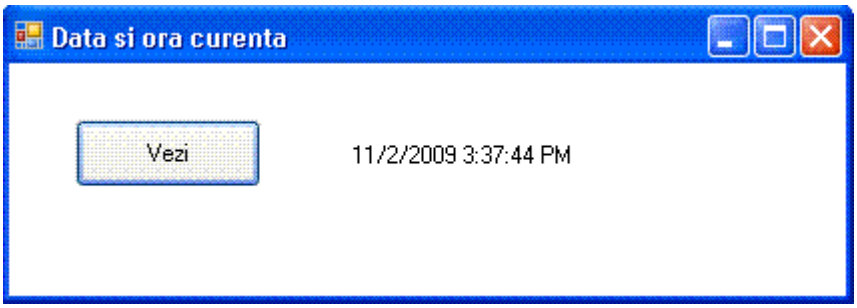
```
// tipul de variabila DateTime.

#include "stdafx.h"
using namespace System;

int main(void)
{
        String^ nume;
        String^ ani_s;
        System::DateTime dt=System::DateTime::Now;
        double ani;
        Console::WriteLine("\n"+dt);
        Console::Write("\n\n\tCum te numesti?:" );
        nume = Console::ReadLine();
        Console::Write( "\n\n\tCiti ani ai?:" );
        ani_s = Console::ReadLine();
        ani = System::Convert::ToDouble( ani_s );
        Console::WriteLine("\n\n\t"+nume+ ", inseamna ca ai, cel putin :"
+ani*360 +" zile");
        Console::ReadLine();
    return 0;
}
```

Folosind WFA vom realiza o aplicatie cafre afiseaza data si ora sistem. Aplicatia este asemanatoare cu cea realizata in mod consola, diferind doar elementele grafice utilizate.

Deschidem un nou proiect numit "data" dupa care:
Plasam un buton cu numele button1
Plasam un obiect label cu numele implicit label1
Selectam button1 si pe evenimentul Click completam procedura cu:

```
this->label1->Text=System::Convert::ToString(System::DateTime::Now);
```

Rulam aplicatia si dupa apasarea butonului obtinem:

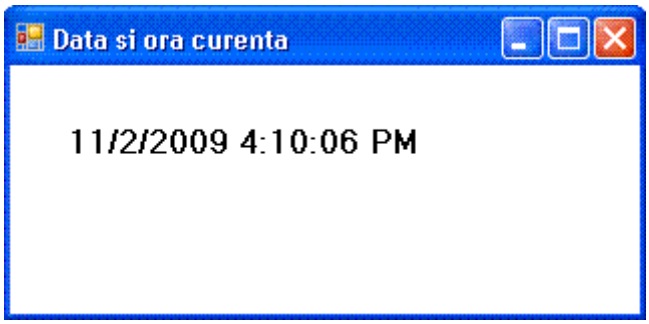


Daca dorim sa afisam in mod continuu ora adica sa reinprospatam valoarea afisata in label1 va trebui sa introducem un nou obiect. De aceasta data obiectul nu mai este vizibil, va rula "in spatele form-ului" si va relansa instructiunea care atribuie valoarea datei curente textului obiectului label1. Abiectul se numeste "timer" si are posibilitatea setarii intervalului intre evenimente numite Tick

Deschidem un nou proiect numit "ceas" dupa care:
Plasam un obiect label cu numele implicit label1
Plasam un obiect timer cu numele implicit timer1 caruia ii setam proprietatea "Interval" la 1000 si proprietatea "Enabled" la True
Selectam timer1 si pe evenimentul Tick completam procedura cu:

```
this->label1->Text=System::Convert::ToString(System::DateTime::Now);
```

Rulam aplicatia si dupa apasarea butonului obtinem:

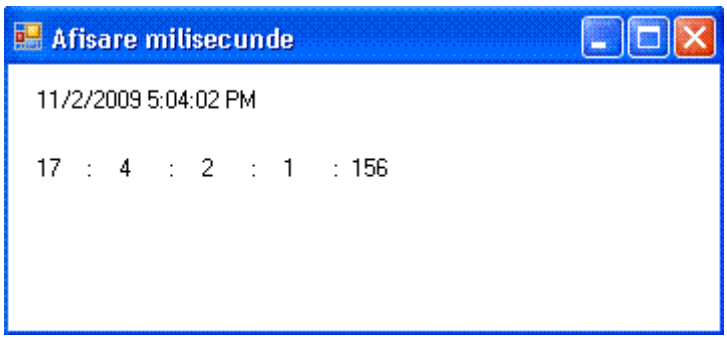


Aplicatia afiseaza in mod continuu data si ora sistem. Intervalul dupa care se face o noua afisare este setat prin intermediul proprietatii "Interval" setata la 1000 milisecunde adica la 1 secunde. Daca am dori afisarea din minut in minut am seta proprietatea "Interval" la 60000.

Reluam aplicatia de sus pentru a afisa atat secundele cat si milisecundele
Deschidem un nou proiect numit "ceas_v1" dupa care:
Plasam sase obiecte label cu numele implicit label1 ... lanel6
Plasam un obiect timer cu numele implicit timer1 caruia ii setam proprietatea "Interval" la 100 si proprietatea "Enabled" la True
Selectam timer1 si pe evenimentul Tick completam procedura cu:

```
this->label1->Text=System::Convert::ToString(System::DateTime::Now );
this->label2->Text=System::Convert::ToString(System::DateTime::Now.Hour );
this->label3->Text=System::Convert::ToString(System::DateTime::Now.Minute );
this->label4->Text=System::Convert::ToString(System::DateTime::Now.Second );
int ms;
ms=System::Convert::ToInt16(System::DateTime::Now.Millisecond );
this->label5->Text=System::Convert::ToString(ms/100);
this->label6->Text=System::Convert::ToString(ms);
```

Rulam aplicatia si dupa apasarea butonului obtinem:



Aplicatia afiseaza in mod continuu data si ora sistem, precum si separat ora:min:sec:zecimi:milisecunde

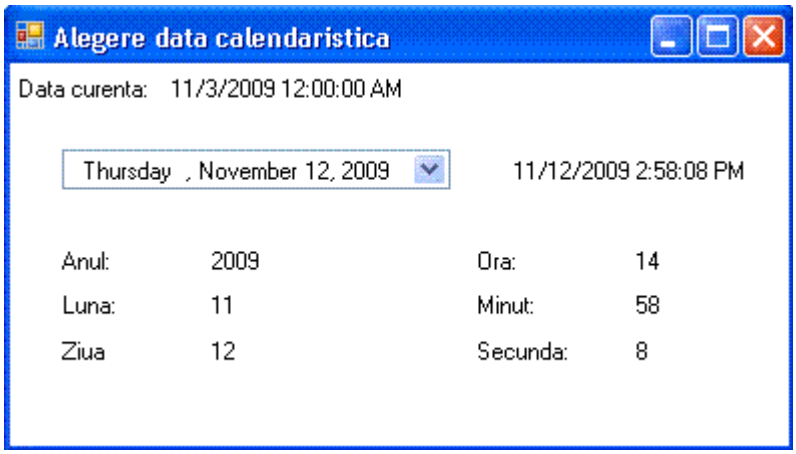
In multe aplicatii este nevoie sa se lucreze cu data calendaristica. In WFA avem posibilitatea sa utilizam obiecte prietenoase care ne ajuta sa introducem date calendaristice prin alegerea datei dintr-un calendar. Obiectul care permite alegerea datei calendaristice se numeste "dateTimerPicker". Vom realiza in continuare o aplicatie care permite alegerea datei calendaristice dupa care acceseaza componentele care formeaza data calendaristica

Deschidem un nou proiect numit "aleg_data" dupa care:

- Plasam opt obiecte label cu numele implicit label1 ... lanel8
- Plasam un obiect "dateTimerPicker" cu numele implicit dateTimerPicker1
- Selectam dateTimerPicker"1 si pe evenimentul ValueChanged completam procedura cu:

```
this->label1->Text=System::Convert::ToString(dateTimePicker1->Value);
this->label2->Text=System::Convert::ToString(dateTimePicker1->Value.Year );
this->label3->Text=System::Convert::ToString(dateTimePicker1->Value.Month );
this->label4->Text=System::Convert::ToString(dateTimePicker1->Value.Day );
this->label5->Text=System::Convert::ToString(dateTimePicker1->Value.Hour );
this->label6->Text=System::Convert::ToString(dateTimePicker1->Value.Minute );
this->label7->Text=System::Convert::ToString(dateTimePicker1->Value.Second );
this->label8->Text=System::Convert::ToString(System::DateTime::Now.Date);
```

- Rulam aplicatia si dupa apasarea alegerea unei date calendaristice folosind obiectul dateTimerPicker1, obtinem:



Aplicatia afiseaza data aleasa precum si separat componentele din care se compune data calendaristica

Conversii de tip

De multe ori e necesar sa convertim variabilele dintr-un tip in altul. Sa scriem un program care citeste raza cercului si afiseaza aria. Citirea se face cu **Console::ReadLine()** care citeste un sir de caractere in cazul de fata un sir de numere, care trebuie convertit intr-un numar. Conversia se face cu **System::Convert::ToDouble()**

Pentru a calcula aria cercului avem nevoie de constanta pi, pe care o apelam din biblioteca System::Math astfel **double pi= System::Math::PI**

☐ **Program pentru calculul ariei cercului utilizand spatiul de nume System::**

```
// program pentru calculul ariei cercului
// programuul cere raza cercului si afiseaza aria acestuia

#include "stdafx.h"

using namespace System;

int main(void)
{
        double raza;
        String^ raza_s;
        double pi= System::Math::PI;
        Console::Write( L"\n\n\tIntroduceti raza cercului:" );
        raza_s= Console::ReadLine();
        raza = System::Convert::ToDouble( raza_s );
        Console::WriteLine( "\n\n\tAria cercului de raza: "+ raza +" este:
"+pi*raza*raza );
        Console::ReadLine();
    return 0;
}
```

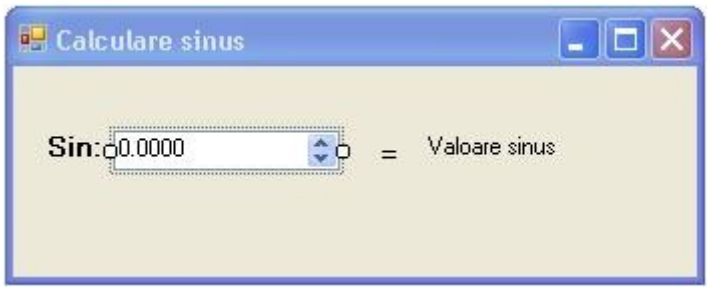
Ne propunem sa afisam sinusul unui unghi exprimat in radiani realizat in Windows Forms Application.

Va trebui sa initiem un nou proiect de tipul CLR Windows Forms Application cu numele "calc_sin_v0".

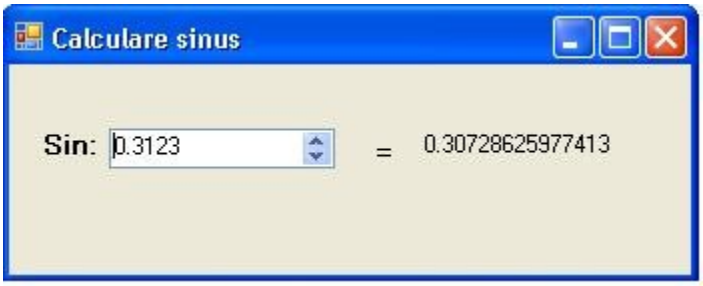
- Redimensionam formul si-i modificam proprietatea text in "Calculare sinus"
- Plasam un label cu numele label1 si-i modificam proprietatea text in "Sin",font bold si size 10
- Plasam un obiect NumericUpDown cu numele implicit numericUpDown1 si-i setam: proprietatea DecimalPlaces la 4 , proprietatea Maximum la 1000
- Plasam un label cu numele label2 si-i modificam proprietatea text in "=",font bold si size 10
- Plasam un label cu numele label3 si-i modificam proprietatea text in "Valoare sinus"
- Selectam NumericUpDown si pe evenimentul ValueChanged completam procedura cu:

```
double rad2;
rad2=System::Convert::ToDouble(this->numericUpDown1->Value);
this->label3->Text =System::Convert::ToString(System::Math::Sin(rad2));
```

Formul trebuie sa arate astfel:



Se lanseaza in executie si formul trebuie sa arate astfel:



In aplicatia anterioara am utilizat un obiect de tip NumericUpDown care ne permite sa introducem valori numerice.

In cazul in care dorim sa utilizam un simplu obiect TextBox in care se pot introduce atat texte cat si cifre si punem aceeasi procedura pe evenimentul TextChanged adica:

```
double rad2;
rad2=System::Convert::ToDouble(this->numericUpDown1->Value);
this->label3->Text =System::Convert::ToString(System::Math::Sin(rad2));
```

In momentul cand tastam numai cifre aplicatia merge corect insa daca se tasteaza si caractere, aplicatia se blocheaza si se afiseaza un mesaj de eroare.

Va trebui sa luam masuri suplimentare si sa ne asiguram ca nu se introduc si caractere in acest camp, in caz contrar aplicatia nu ruleaza corect si la apasarea unui caracter alfanumeric (altul decat cifra) se genereaza un mesaj de eroare care anunta ca formatul nu este corespunzator si aplicatia se opreste.

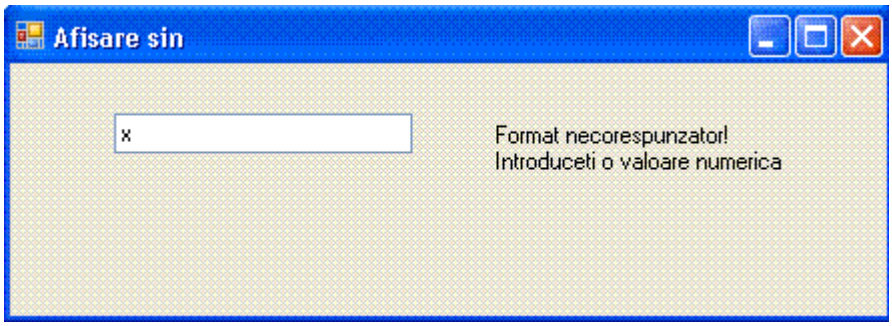
Va trebui sa tratam aceasta eroare, sa afisam un mesaj si sa pastram controlul programului.

Deschidem un nou proiect numit "calc_sin_v2" dupa care:

- Plasam un obiect TextBox cu numele implicit textBox1
- Plasam un label cu numele label1 si-i modificam proprietatea text in "Valoare sinus"
- Selectam textBox1 si pe evenimentul TextChanged completam procedura cu:

```
try{
    double rad ;
    rad=System::Convert::ToDouble(this->textBox1->Text);
    this->label1->Text =System::Convert::ToString(System::Math::Sin(rad));
    }
catch(System::FormatException^ err){
    this->label1->Text="Format necorespunzator!\nIntrod. o valoare numerica";
    }
```

Rulam aplicatia tastam un "x" si obtinem:



Instructiunea try - catch ete cea care sesizeaza o exceptie de format si permite afisarea textului:
Format necorespunzator!
Introduceti o valoare numerica

Operatori

Un operator este un simbol care reprezinta o anumita actiune

Operatorul de adresa

Prin declararea unei variabile se aloca memorie in functie de tipul variabilei.Numele variabilei este chiar adresa de inceput a zonei de memorie rezervata. Aceasta adresa se gaseste folosind operatorul: **&**.
Sintaxa este:**&[nume variabila]**

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

using namespace std;

int main(void)
{
        string nume;
        cout <<"\n\tIntroduceti numele:";
        cin >> nume;
        cout << "\n\n\tBine ai venit " << nume << "!\n";
        cout << "\n\n\tAdresa variabilei nume este :" << &nume;
        cin.ignore();
        cin.get();
        return 0;
}
```

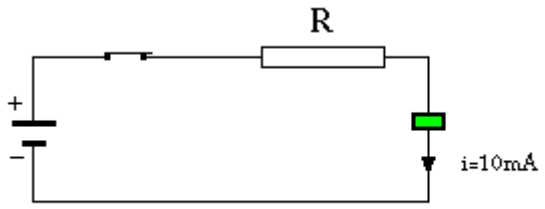
Dupa rularea acestei aplicatii, obtinem:

Operatorul de atribuire

Se foloseste pentru a asocia o valoare unei variabile: **&**.
Sintaxa este:**[nume variabila]=[valoare]**

Valoarea atribuita unei variabile trebuie sa fie compatibila cu tipul de date al variabilei. In caz de incompatibilitate se genereaza erori.

Vom folosi in continuare Windows Forms Application pentru realiza o aplicatie grafica ce va inchide si va deschide circuitul de jos.



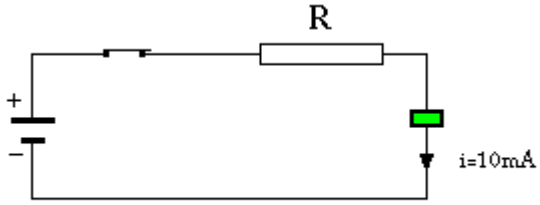
Aplicatia nu face altceva decat atribuie unor proprietati ale unor obiecte niste valori: De exemplu

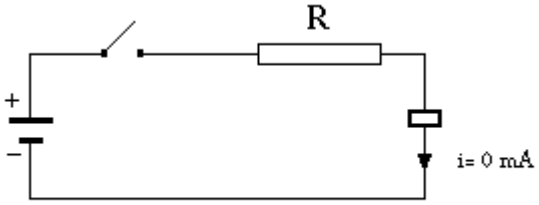
-atribuim proprietatii text a butonului, valoarea "Start"

-atribuim proprietatii Image a obiectului pictureBox1 ,valoarea System::Drawing::Image::FromFile("circ_i.gif")

Sa initiem un nou proiect de tipul CLR Windows Forms Application cu numele "led".

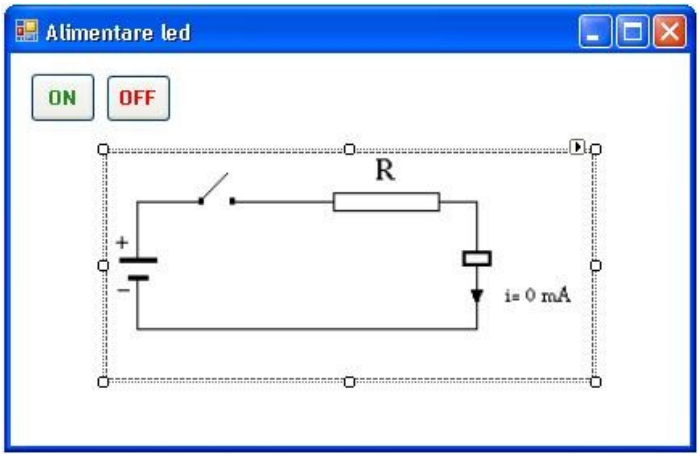
- Redimensionam formul si-i modificam proprietatea text in "Alimentare led"
- Plasam doua butoane cu numele button1 respectiv button2 apoi le modificam proprietatea text atribuindu-i valorile "ON" respectiv "OFF"
- Plasam un obiect PictureBox cu numele pictureBox1
- Pregatim doua imagini cu numele circ_i.gif respectiv circ_d.gif de forma celor doua imagini de jos:

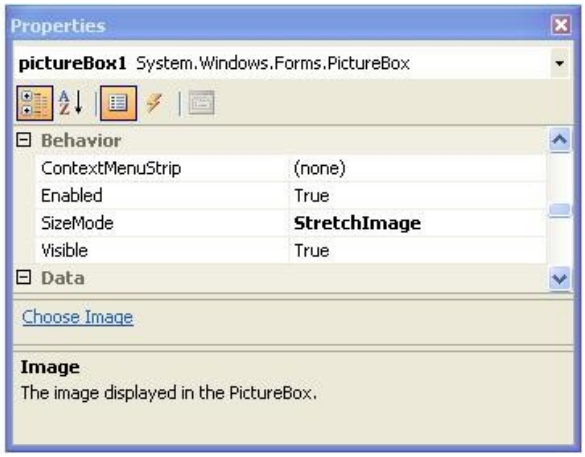




- Copiem cele doua fisire in folder-ul Projects/led/led
- Selectam fisierul "circ_d.gif" la proprietatea Image a obiectului pictureBox1
- Setam proprietatea SizeMode a obiectului pictureBox1 la "StretchImage"

In acest moment Formul arata astfel:





Procedura button1_Click a evenimentul **click** a obiectului **button1** va fi completata cu:

```
this->pictureBox1->Image = System::Drawing::Image::FromFile( "circ_i.gif" );
```

Scheme logice operatori relationali, expresii relationale in C++

Operatori relationali

Programele scrise si analizate pana in acest moment, au fost simple inlantuiri de instructiuni de atribuire, executia programului era una liniara unde instructiunile se inlantuiau unele dupa altele de la ineputul programuli pana la sfarsitul lui.

Problemele pe care dorim sa le rezolvam cu ajutorul calculatorului sunt de cele mai multe ori, mult mai complexe. O simpla insiruire de instructiuni nu poate rezolva probleme complexe. De multe ori e nevoie ca programul sa ia cai diferite in functie de datele introduse de operator , de anumite valori ale diferitelor variabile sau constante. Programul trebuie sa contina toate variantele posibile, si in momentul rularii , in urma unor operatii de comparare sa urmeze calea corespunzatoare. Comparatia se face prin intermediul operatorilor relationali.

Operator	Semnificatie
>	Mai mare
<	Mai mic
>=	Mai mare sau egal
<=	Mai mic sau egal
==	Egal
!=	Diferit

Expresii relationale

O instructiune cu doi operanzi si un operator relational intre ei se numeste expresie relationala.

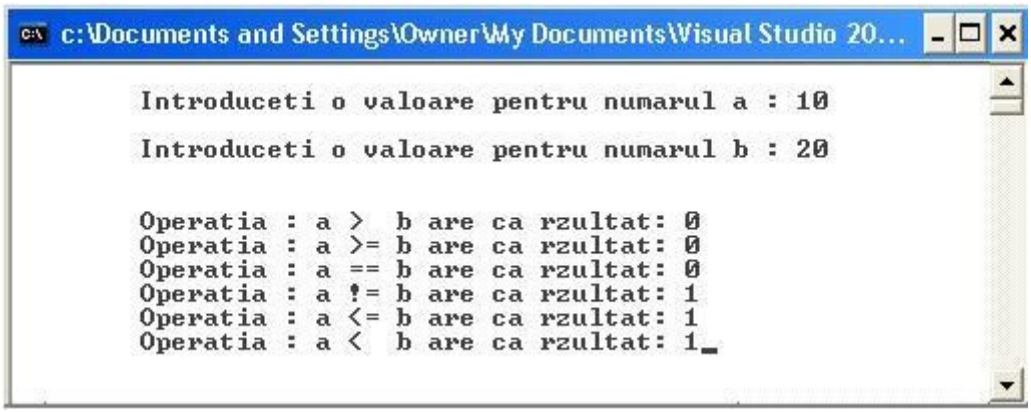
Rezultatul unei expresii relationale este o valoare bool-eana care poate lua deci doua valori: true sau false

- **Program C++ pentru a exemplifica modul de scriere al expresiilor relationale**

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// Expresii relationale

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

int main()
{
  double  a,b;
```

```
  cout << "\n\tIntroduceti o valoare pentru numarul a : ";
  cin >> a ;
  cin.ignore();
  cout << "\n\tIntroduceti o valoare pentru numarul b : ";
  cin  >> b;
  cin.ignore();
  cout <<"\n\n\tOperatia : a >  b are ca rzultat: " << (a>b);
  cout <<"\n\tOperatia : a >= b are ca rezultat: " << (a>=b);
  cout <<"\n\tOperatia : a == b are ca rezultat: " << (a==b);
  cout <<"\n\tOperatia : a != b are ca rezultat: " << (a!=b);
  cout <<"\n\tOperatia : a <= b are ca rezultat: " << (a<=b);
  cout <<"\n\tOperatia : a <  b are ca rezultat: " << (a < b);
  cin.get();
  return 0;
}
```

Dupa rularea programului in "Command Prompt" se afiseaza:



Scheme logice

Schemele logice – au rolul de a reprezenta grafic fluxul general al datelor si a algoritmului de prelucrare.

Sintetizeaza succesiunea etapelor de rezolvare a unei probleme constituind o reprezentare grafica a functiilor algoritmului utlizat. In cadrul acestor scheme se folosesc simboluri standard, carora li se asociaza în principiu anumite instructiuni.

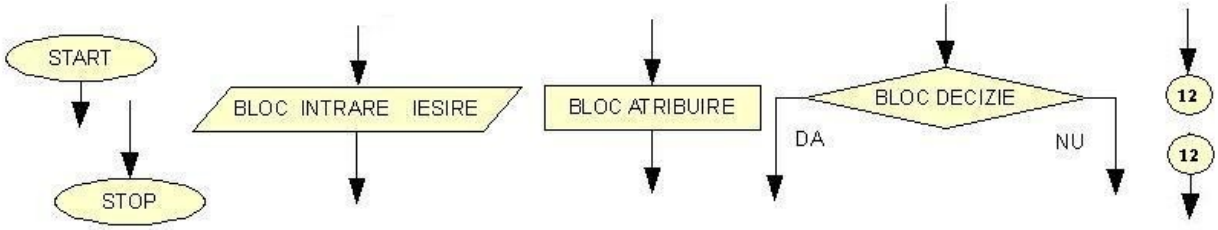
Schema logica reprezinta de fapt un graf orientat în care exista cel putin urmatoarele elemente:

- instructiune de pornire (START) si una de oprire (STOP);
- instructiuni de atribuire;
- instructiuni de citire a datelor;
- instructiuni de decizie;
- instructiuni de scriere/afisare a rezultatelor prelucrarii.
- sageti care indica sensul fluxului de prelucrare, deci ordinea de executie a operatiilor elementare.

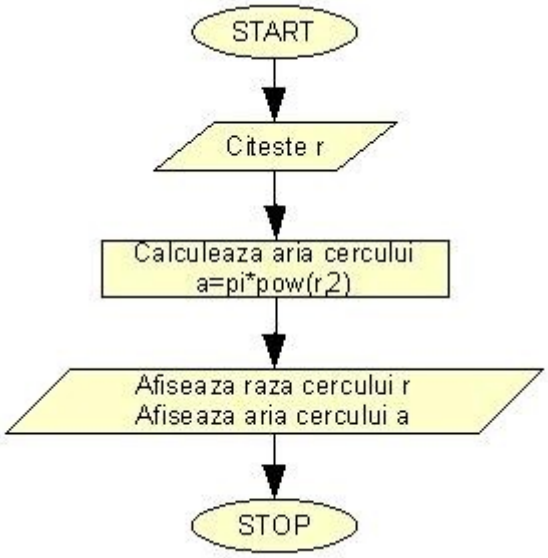
Ordinea în care instructiunile programului sunt executate constituie structura de control a acestuia.

Schema logica este importanta la analiza problemei, la scrierea programuli si la intelegerea programului de alte persoane, schema logica fiind reprezentarea vizuala a functionarii programului.

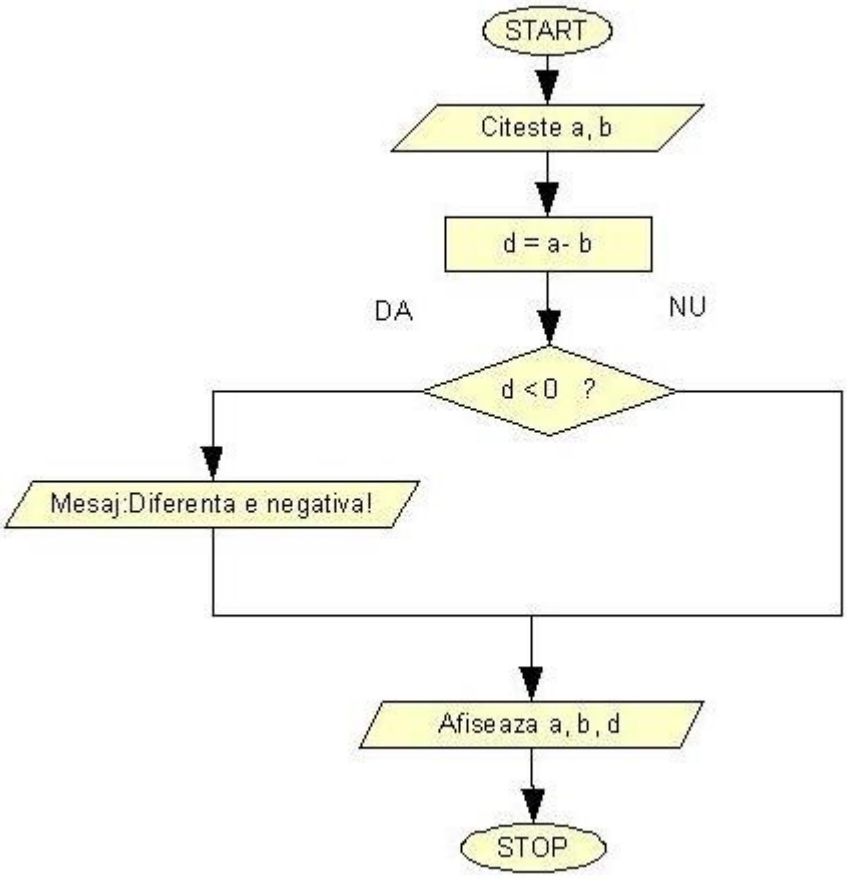
Se folosesc urmatoarele simboluri grafice in cadrul schemelor logice



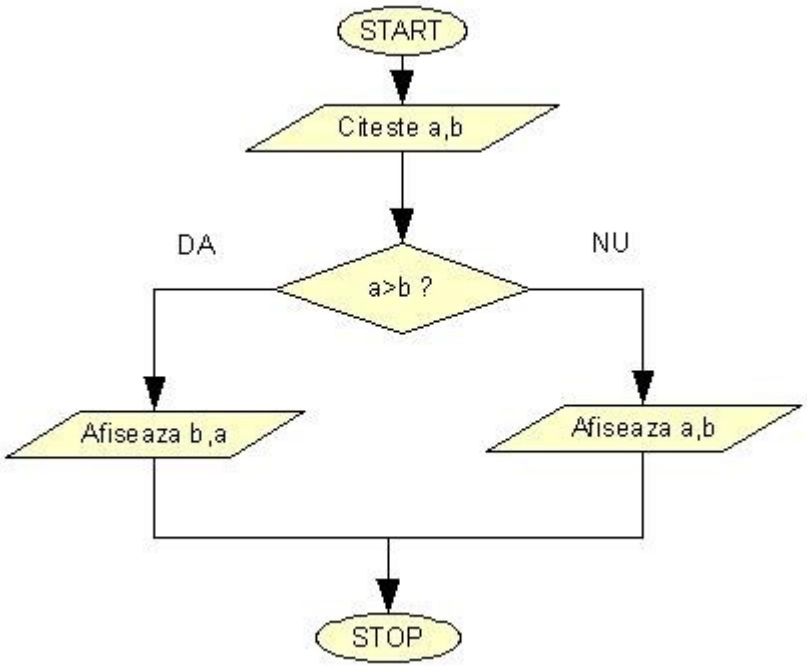
Schema logica a programului care asteapata de la utilizator raza unui cerc si afiseaza aria acestuia , arata astfel:



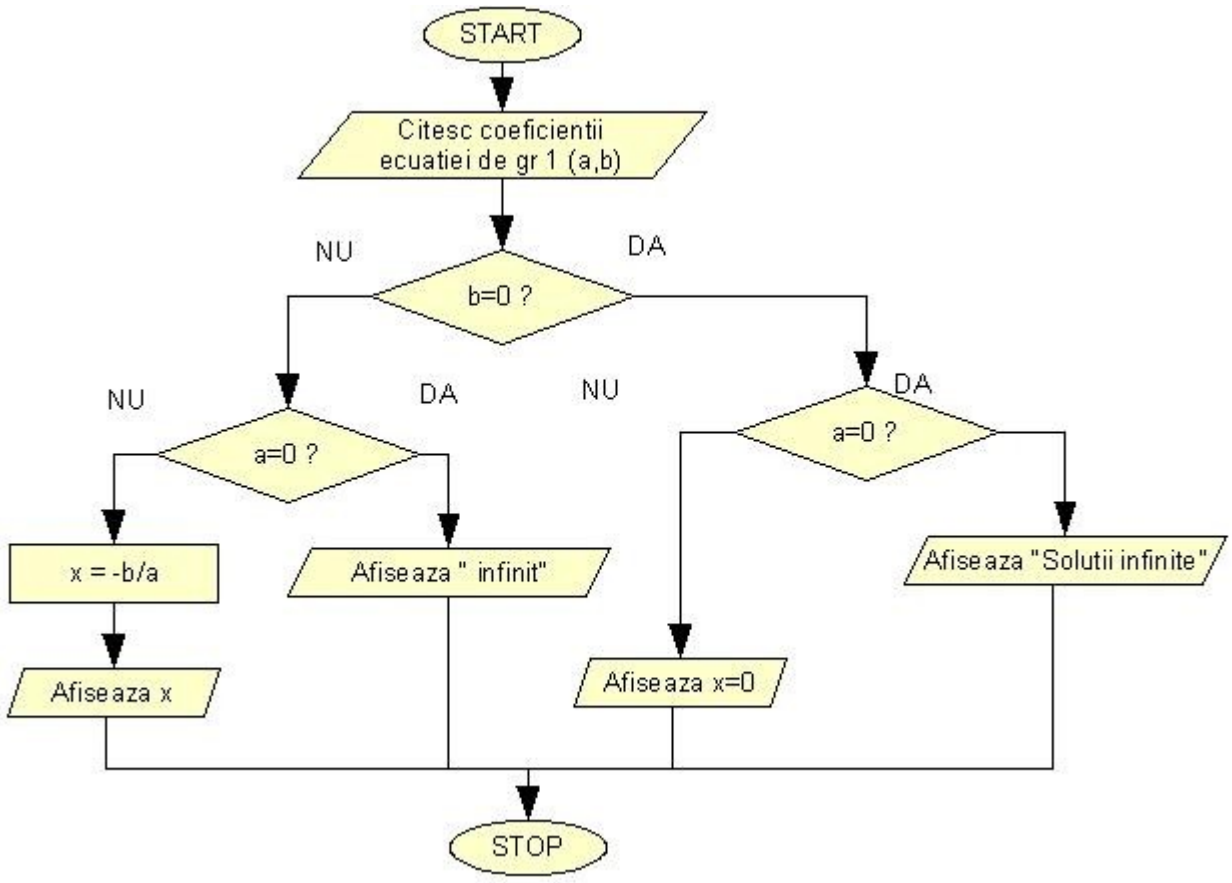
Sunt cazuri in care programul trebuie sa ruleze anumite instructiuni numai in anumite cazuri. Sa presupunem ca un program asteapta doua numere de la utilizator si vrea sa calculeze diferenta lor si s-o afiseze insa inainte sa afiseze diferenta numerelor introduse am vrea sa fim avertizati cu un mesaj ca diferenta este negativa. Deci instructiunea care afiseaza mesajul "Diferenta este negativa", trebuie sa se execute numai daca diferenta dintre numerele introduse este negativa. Schema logica a unui astfel de program arata astfel:



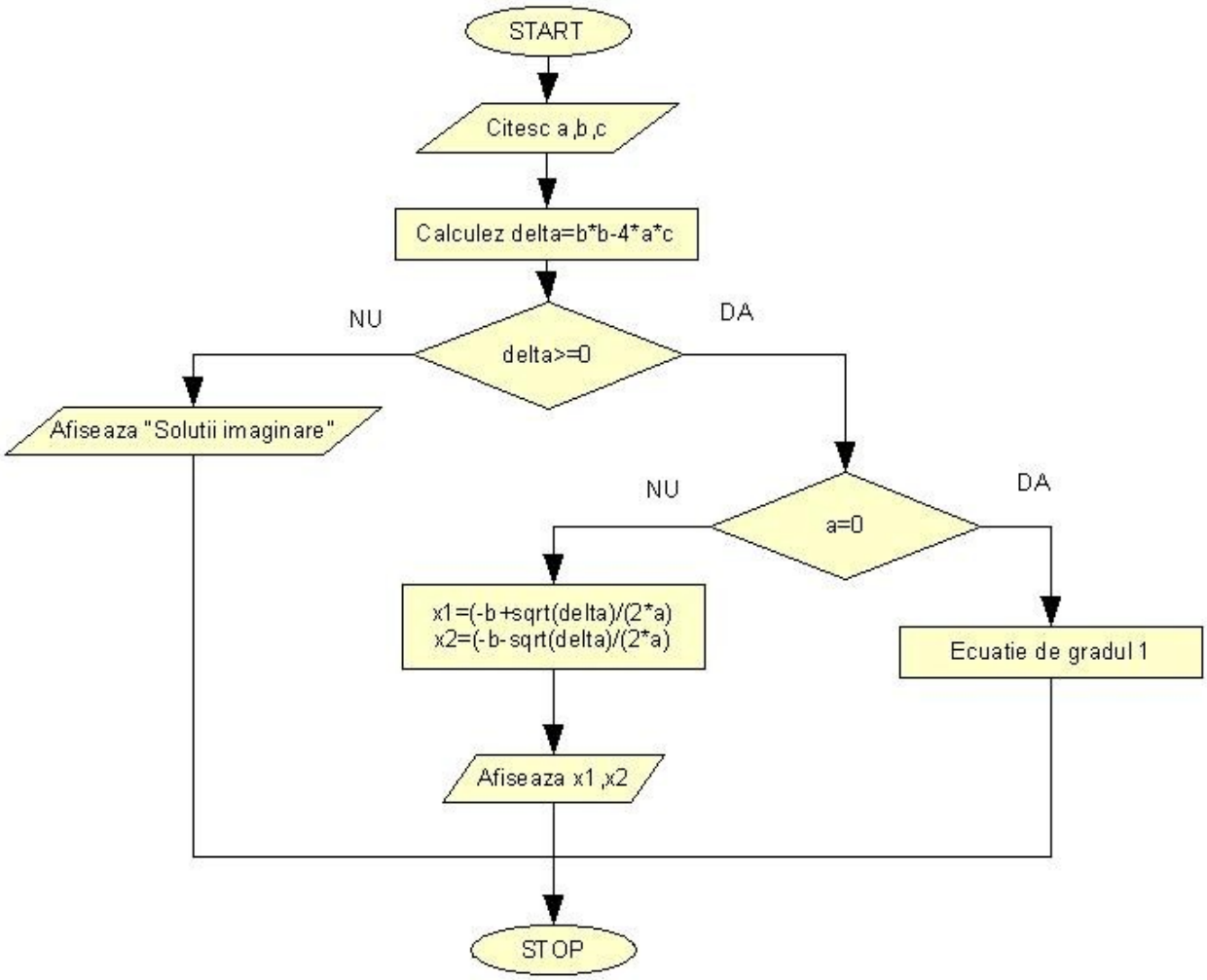
Sa realizam acum schema logica a unui program care asteapata de la utilizator doua numere (a si b) pe care le afiseaza in ordine crescatoare adica in ordinea a,b daca a < b sau b,a daca b < a



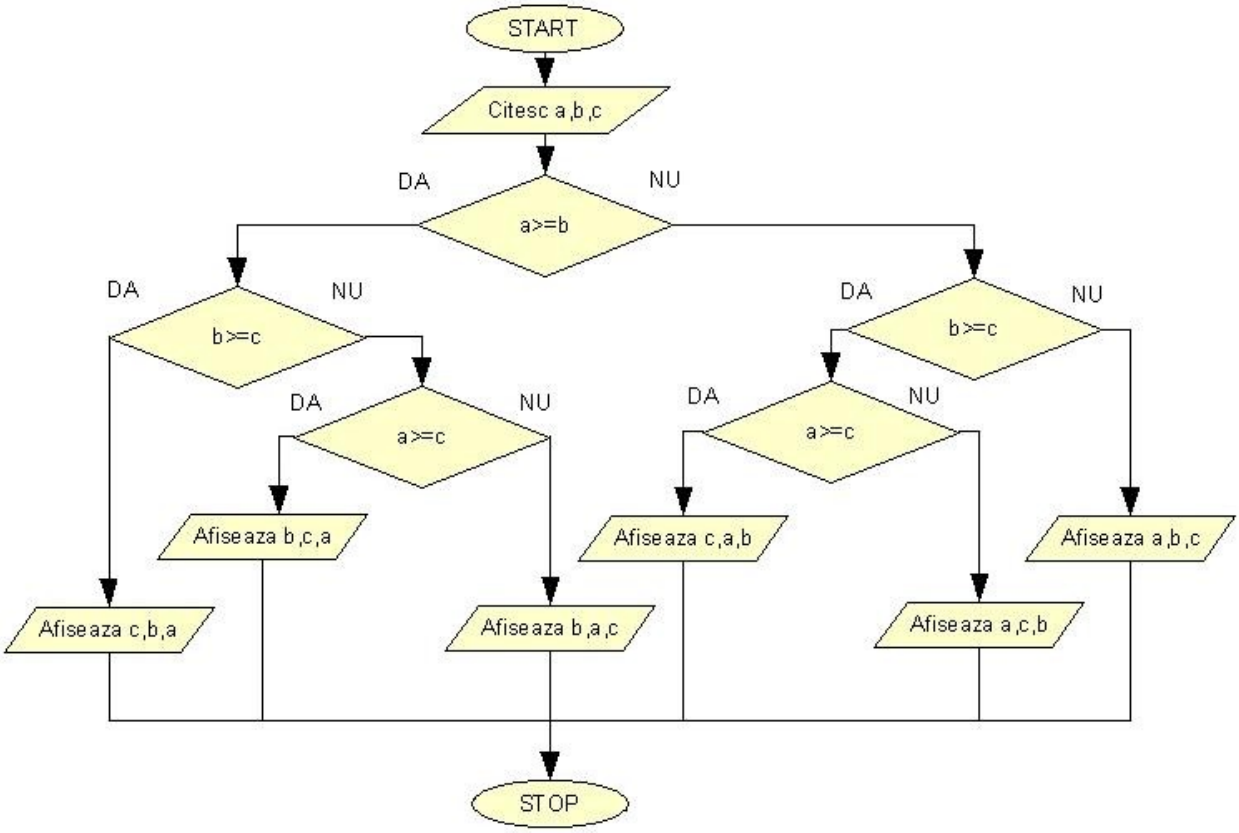
Schema logica a unui program care asteapata de la utilizator coeficientii euatiei de gradul 1 (a si b) si afiseaza solutiile



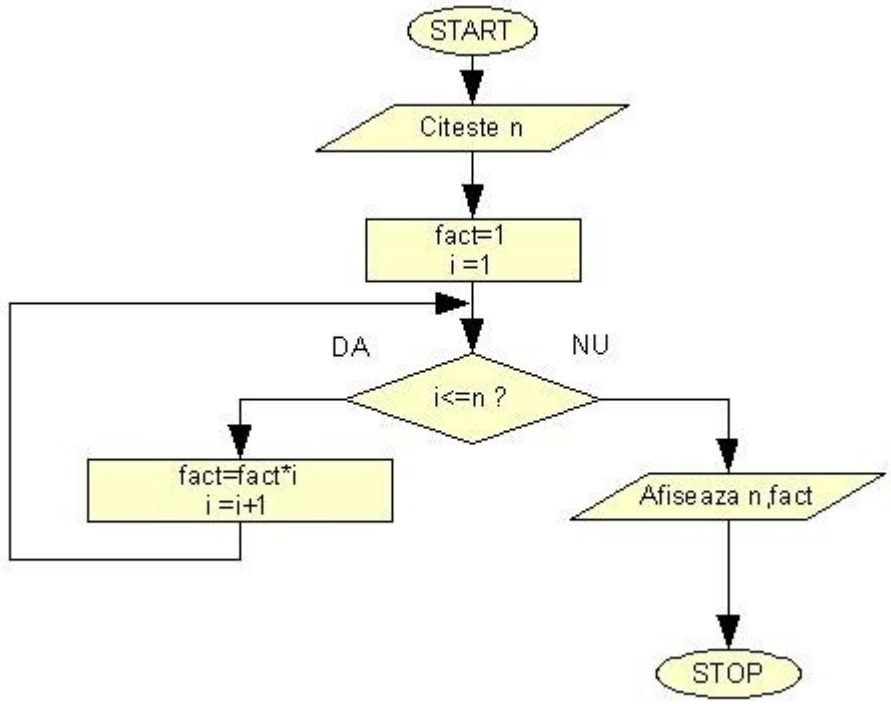
Schema logica a unui program care asteapata de la utilizator coeficientii euatiei de gradul 2 (a, b si c) si afiseaza solutiile



Sa realizam acum schema logica a unui program care asteapata de la utilizator trei numere (a,b si c) pe care le afiseaza in ordine crescatoare.

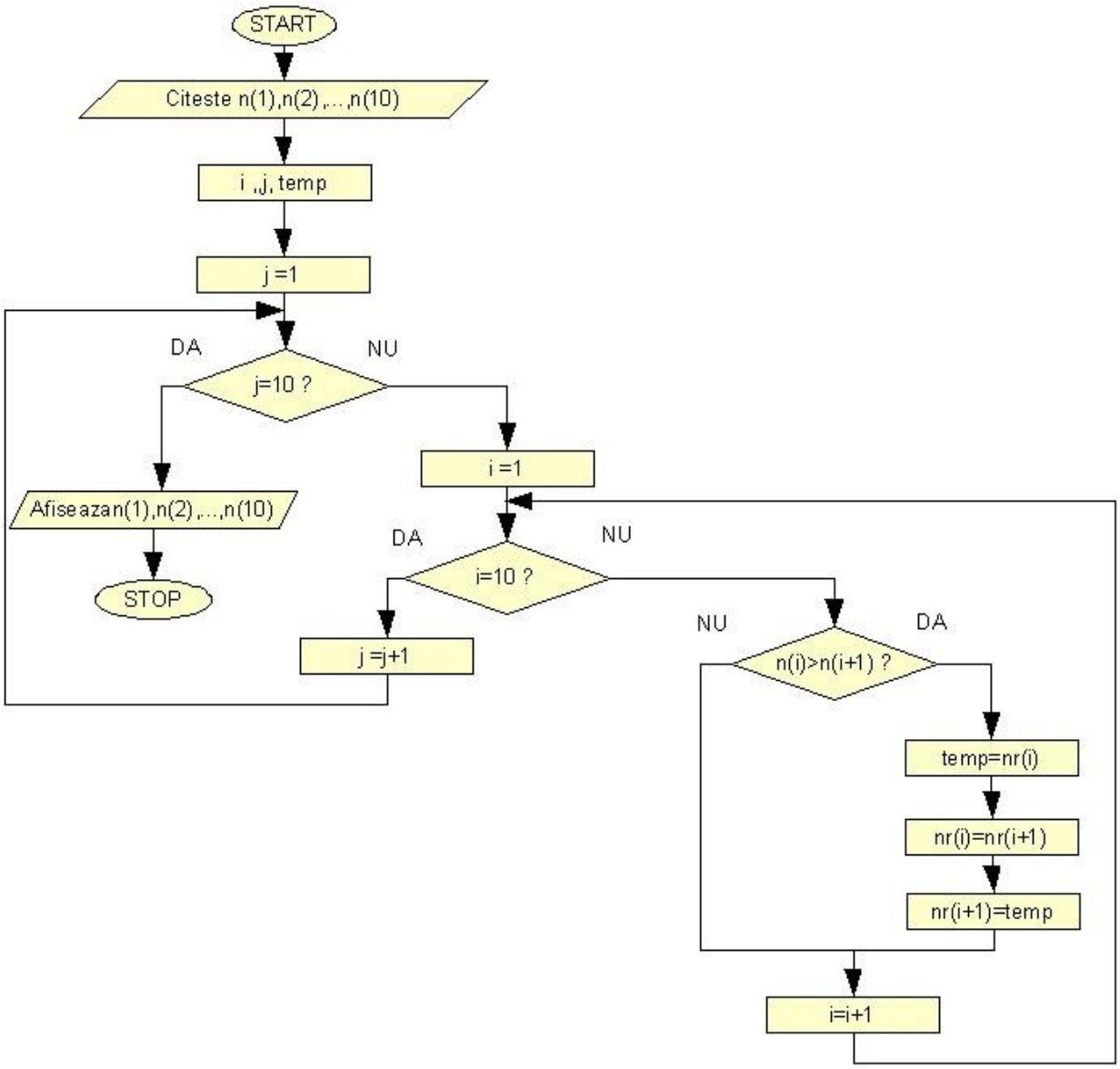


De multe ori e nevoie sa calculam repetitiv ceva. Sa luam de exemplu calculul sumei primelor 100 de numere naturale sau calculul lui n!. Sa realizam schema logica a unui program ce calculeaza n factorial.



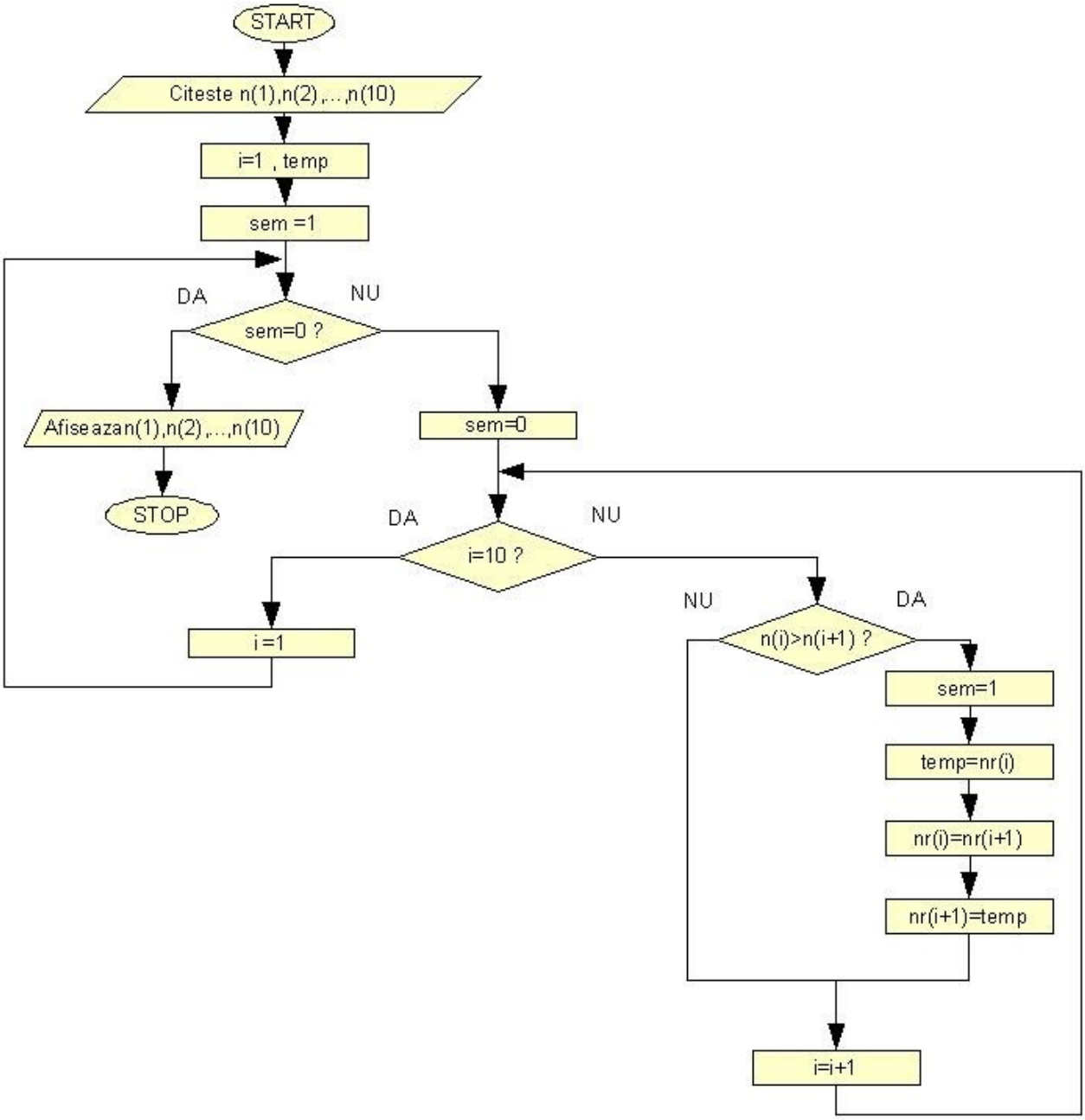
Am realizat anterior schema logica a unui program care astepata de la utilizator trei numere (a,b si c) pe care le afiseaza in ordine crescatoare.

Pentru mai mult de 3 numere situatia se complica foarte mult asa ca trebuie sa concepem alta strategie. Folosind facilitatile de repetitie putem concepe urmatoarea schema logica. Algoritmul schitat grafic mai jos poate ordona n numere . Caz particular 10 numere.



In principiu algoritmul schitat sus ia pe rand doua cate doua numere si le schimba locul daca primul numar este mai mare decat cel de-al doilea. In primul pas se compara nr1 cu nr2, in pasul doi nr2 cu nr3.Dupa n-1 pasi se ajunge la comparatia n-1 cu n. Daca se repeta operatia de cel putin n-1 ori avem certitudinea ca sirul este ordonat crescator. In cazul cel mai defavorabil in care primul numar este cel mai mare, dupa n-1 treceri prin sirul de numere, acesta ajunge pe ultimul loc fiind cel mai mare.

Daca sirul anterior este deja ordonat crescator programul totusi ruleaza 81 de pasi adica (n-1)*(n-1). Am putea realiza o optimizare in sensul ca daca sirul e deja ordonat sa nu mai continue algoritmul de aranjare si sa treaca direct la afisarea sirului de numere. Vom introduce o variabila suplimentara numita si semafor care va fi inscrisa cu o anumita valoare cand in urma trecerii prin sirul de numere nu a avut loc nici o schimbatre intre pozitiile numerelor si deci sirul este deja ordonat.


START
Citeste n(1),n(2),...,n(10)
i=1 , temp
sem =1
DA
NU
sem=0 ?
Afiseazan(1),n(2),...,n(10)
STOP
sem=0
DA
NU
i=10 ?
i=1
NU
DA
n(i)>n(i+1) ?
sem=1
temp=nr(i)
nr(i)=nr(i+1)
nr(i+1)=temp
i=i+1

Instructiuni decizionale: if, swich, if imbricate

Instructiunea if

Instructiunea if se foloseste pentru a executa o instructiune sau o secventa de instructiuni numai cand valoare logica a unei expresii este adevarata

- Formatul instructiunii:

Instructiunea if se compune din urmatoarele linii:

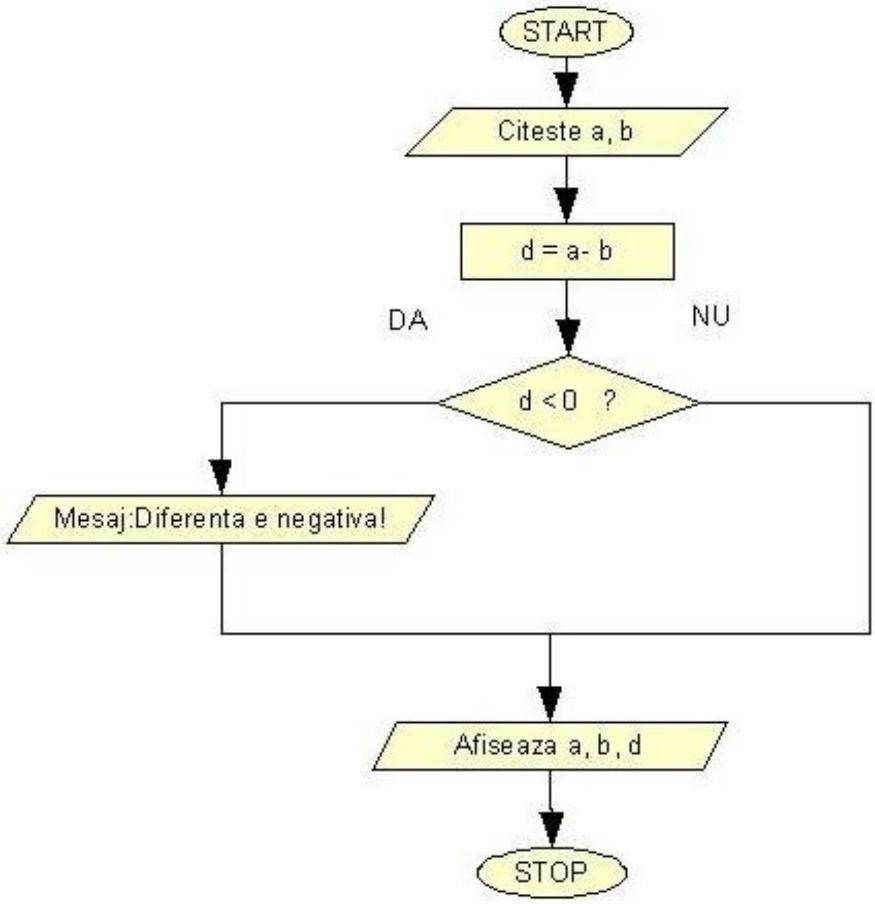
```
if(valoare booleana)
  instructiune;
```

Prima linie contine intre paranteze o expresie relationala si nu se termina cu ;

Urmatoarea linie este o instructiune conditionata care va fi executata numai daca valoarea expresiei relationale este adevarata.

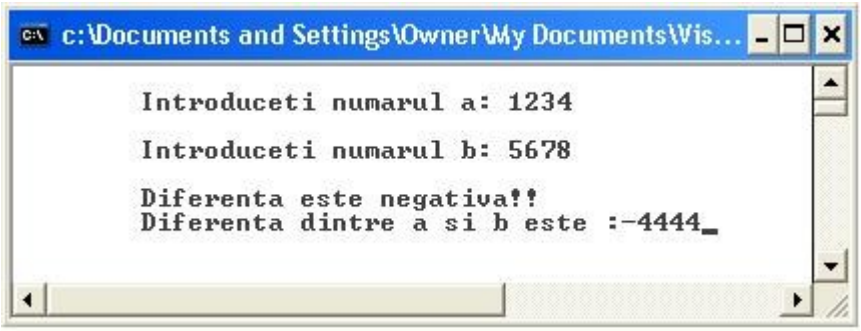
- Exemplu de program C++ ce utilizeaza instructiunea decizionala if

Sa realizam un program care asteapta doua numere de la utilizator apoi calculeaza diferenta lor si o afiseaza, insa inainte sa afiseze diferenta numerelor introduse am vrea sa fim avertizati cu un mesaj ca diferenta este negativa. Instructiunea care afiseaza mesajul "Diferenta este negativa", trebuie sa se execute numai daca diferenta dintre numerele introduse este negativa. Schema logica a unui astfel de program arata astfel:




```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// program care asteapta doua numere de la utilizator apoi calculeaza diferenta
lor si o afiseaza
// inainte sa afiseze diferenta numerelor, avertizeaza cu un mesaj ca diferenta
este negativa.

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

int main(void)
{
        int a,b,d;
        cout <<" \n\tIntroduceti numarul a: ";
        cin >> a;
        cout <<" \n\tIntroduceti numarul b: ";
        cin >> b;
        d=a-b;
        if (d<<" \n\tDiferenta este negativa!! ";
        cout <<" \n\n\tDiferenta dintre a si b este :"<< d ;
        cin.ignore();
        cin.get();
        return 0;
}
```

Dupa rularea programului in "Command Prompt" se afiseaza:



In cazul in care instructiunea conditionala e compusa din mai multe linii trebuiesc folosite acolade.

Spre exemplu daca vrem sa mai adaugam mesajul "Atentie", programul devine:

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// program care asteapta doua numere de la utilizator apoi calculeaza diferenta
lor si o afiseaza
// inainte sa afiseze diferenta numerelor, avertizeaza cu un mesaj ca diferenta
este negativa.

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

int main(void)
{
        int a,b,d;
        cout <<" \n\tIntroduceti numarul a: ";
        cin >> a;
        cout <<" \n\tIntroduceti numarul b: ";
        cin >> b;
        d=a-b;
        if (d<<" \n\t      Atentie!!           ";
                cout <<" \n\tDiferenta este negativa!! ";
                }
        cout <<" \n\n\tDiferenta dintre a si b este :"<< d ;
        cin.ignore();
        cin.get();
        return 0;
}
```

Instructiunea if / else

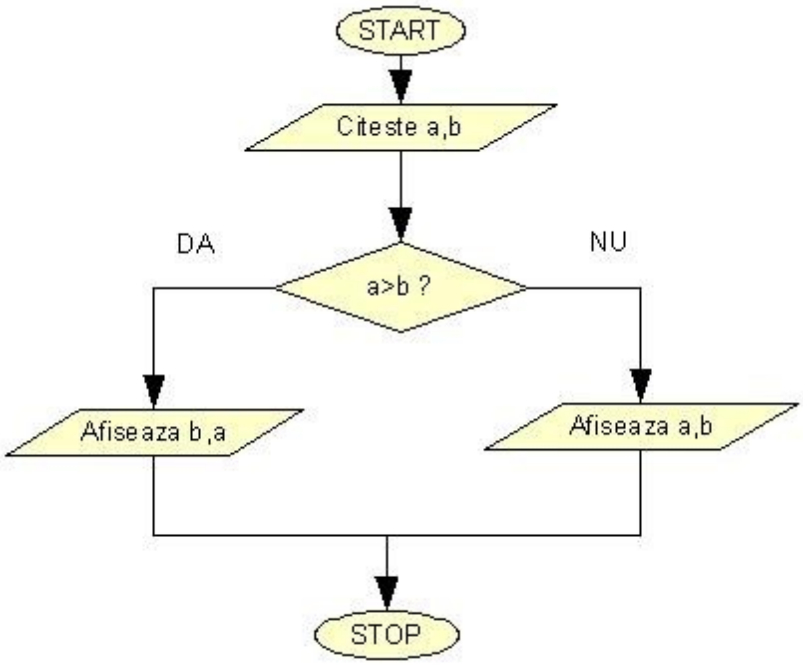
Instructiunea if / else se foloseste pentru a executa o instructiune sau o secventa de instructiuni cand valoare logica a unei expresii este adevarata si alta instructiune sau o secventa de instructiuni pentru cazul cand valoarea logic a aceleiasi expresii nu este adevarata.

- Formatul instructiunii:

Instructiunea if / else se compune din urmatoarele linii:

```
if(valoare booleana)
  instructiune;

else
  instructiune;
```

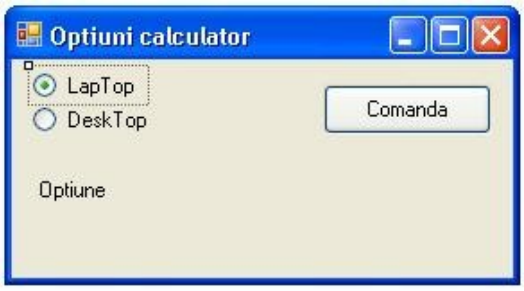
Programul pentru ordonarea a doua numere (vezi schema logica de mai jos), foloseste instructiunea if / else .

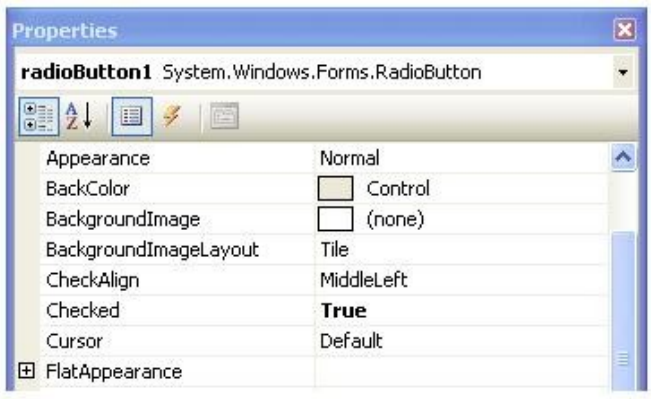


```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// program care asteapta doua numere de la utilizator apoi le ordoneaza in
ordine crescatoare

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

int main(void)
{
        int a,b;
        cout <<" \n\tIntroduceti numarul a: ";
        cin >> a;
        cout <<" \n\tIntroduceti numarul b: ";
        cin >> b;
        if (a>b)
          cout <<"\n\n\tNumerele ordonate crescator sunt b,a adica: " << b <<",
" << a;
        else
          cout <<"\n\n\tNumerele ordonate crescator sunt a,b adica: " << a <<",
" << b;
   cin.ignore();
        cin.get();
        return 0;
}
```

Si in modul grafic Windows Forms Application sunt utile instructiunile decizionale if. Vom realiza o aplicatie grafica in care vom folosi un obiect "radio button", si in functie de selectia facuta de utilizator prin intermediul obiectului radio button plasat, vom afisa optiunea selectata.

Deschidem un nou proiect Windows Forms Application numit if
Plasam doua obiecte de tip Radio Button cu numele radioButton1 respectiv radioButton2.
Plasam un obiect button1 .
Plasam un obiect label1.
Am plasat obiectele de mai sus pentru a schita un formular de comanda pentru un calculator.



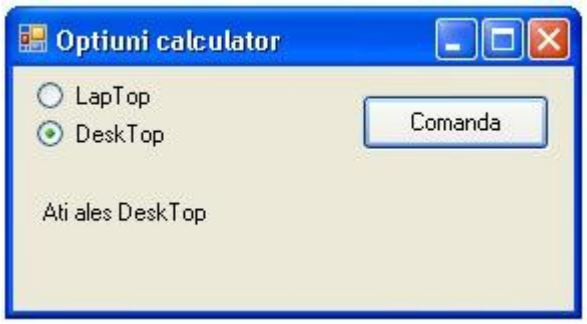


Utilizatorul trebuie sa decida ce fel de calculator vrea sa comande, apoi sa apese button1:"Comanda" pentru a lansa comanda. Label1:"optiune" va scrie

mesajul corespunzator optiunii marcate. Completam procedura creata pe evenimentul click a obiectului button1 cu

```
if (this->radioButton1->Checked)
        this->label1->Text="Ati ales LapTop";
else
        this->label1->Text="Ati ales DeskTop";
```

Dupa lansarea aplicatiei si marcarea optiunii DeskTop ecranul aplicatiei arata astfel:



Operatorul conditional

Operatorul conditional are urmatoara sintaxa:
[expresie relationala] ? [instructiune pentru expresie "adevarata"]: [instructiune pentru expresie "falsa"]
Folosind operatorul relational programul pentru ordonarea a doua numere devine:

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// program care asteapta doua numere de la utilizator apoi le ordoneaza in
ordine crescatoare

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

int main(void)
{
        int a,b;
        cout <<" \n\tIntroduceti numarul a: ";
        cin >> a;
        cout <<" \n\tIntroduceti numarul b: ";
        cin >> b;
        cout <<"\n\n\tNumerele ord. crescator sunt:" << ( a>b ? " b , a " : "
a , b");
        cin.ignore();
        cin.get();
        return 0;
}
```

Instructiunea if / else if else

Instructiunea if / else if /else se foloseste pentru a implementa structuri decizionale cu mai mult de doua alternative.

- Formatul instructiunii:

Instructiunea if / else if /else se compune din urmatoarele linii:

```
if(valoare booleana1)
  instructiune;

else if(valoare booleana2)
  instructiune;

else if(valoare booleana3)
  instructiune;
.
.
.

else
  instructiune;
```

Sa presupunem ca la aplicatia grafica pentru comanda calculatorului vrem sa adaugam optiunea:Server station. Avem de ales intre trei opriuni:LapTop DeskTop si ServerStation. Programul scris in Visual Studio 2005 Console application arata astfel:

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// program care asteapta optiunea d-voastra si o afiseaza.

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

int main(void)
{
        int a;
        cout <<" \nTastati: ";
        cout <<" \n\t1-Pentru LapTop";
        cout <<" \n\t2-Pentru DeskTop";
        cout <<" \n\t3-Pentru ServerStation";
        cout <<" \n\n\tOptiunea d-voastra: ";
        cin >> a;
        if (a==1)
                cout <<"\n\n\tAti ales LapTop: " ;
        else if (a==2)
                cout <<"\n\n\tAti ales DeskTop: " ;
        else
                cout <<"\n\n\tAti ales ServerStation: " ;
        cin.ignore();
        cin.get();
        return 0;
}
```

Instructiunea Switch

Instructiunea switch este asemanatoare cu instructiunea if/ else if/ else Aceasta instructiune evalueaza valoarea unei expresii intregi dupa care compara aceasta valoare cu 2 sau mai multe valori pentru a decide pe care ramura de program sa continue.

- Formatul instructiunii:

Instructiunea switch se compune din urmatoarele linii:

```
switch(expresie){

case V1:
  instructiune;
  break;

case V2:
  instructiune;
  break;
.
.
.
case Vn:
  instructiune;
  break;

default:
  instructiune;
}
```

Aplicatia de mai sus devine:

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// program care asteapta optiunea d-voastra si o afiseaza.

#include "stdafx.h"
#include < iostream >
using namespace std;
int main(void)
{
        int a;
        cout <<" \nTastati: ";
        cout <<" \n\t1-Pentru LapTop";
        cout <<" \n\t2-Pentru DeskTop";
        cout <<" \n\t3-Pentru ServerStation";
        cout <<" \n\n\tOptiunea d-voastra: ";
        cin >> a;
        switch(a){
        case 1:
                cout <<"\n\n\tAti ales LapTop: " ;
                break;
        case 2:
                cout <<"\n\n\tAti ales DeskTop: " ;
                break;
        case 3:
                cout <<"\n\n\tAti ales ServerStation: " ;
                break;
        default:
                cout <<"\n\n\t Nu ati ales nimic !" ;
```

```
        }
        cin.ignore();
        cin.get();
        return 0;
}
```

In cazul in care expresia nu este o valoare numerica ci un caracter de exemplu, aplicatia devine:

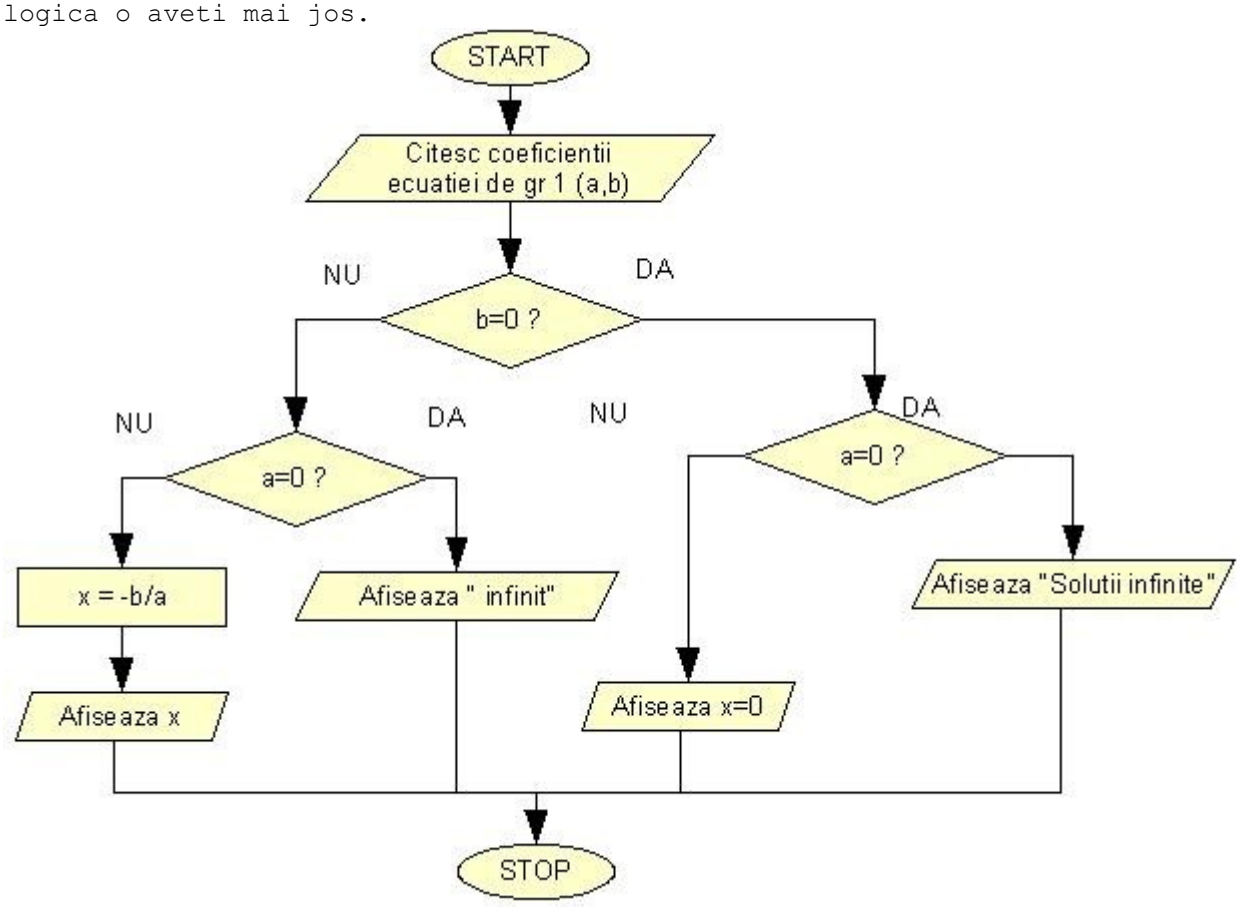
```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// program care asteapta optiunea d-voastra si o afiseaza.

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

int main(void)
{
        char a;
        cout <<" \nTastati: ";
        cout <<" \n\tL-Pentru LapTop";
        cout <<" \n\tD-Pentru DeskTop";
        cout <<" \n\tS-Pentru ServerStation";
        cout <<" \n\n\tOptiunea d-voastra: ";
        cin >> a;
        switch(a){
        case 'L':
                cout <<"\n\n\tAti ales LapTop: " ;
                break;
        case 'D':
                cout <<"\n\n\tAti ales DeskTop: " ;
                break;
        case 'S':
                cout <<"\n\n\tAti ales ServerStation: " ;
                break;
        default:
                cout <<"\n\n\t Nu ati ales nimic!" ;
        }
        cin.ignore();
        cin.get();
        return 0;
}
```

Instructiunea break este apsolut necesara cand nu dorim sa se mai execute urmatoarele ramuri ale instructiunii switch.

Instructiuni if imbricate

Instructiunea if poate sa apara in interiorul altei instructiuni if. In acest caz se spune ca o instructiue if este imbricata in interiorul altei instructiuni if.

Sa realizam programul care rezolva ecuatia de gradul 1 a carei schema logica o aveti mai jos.

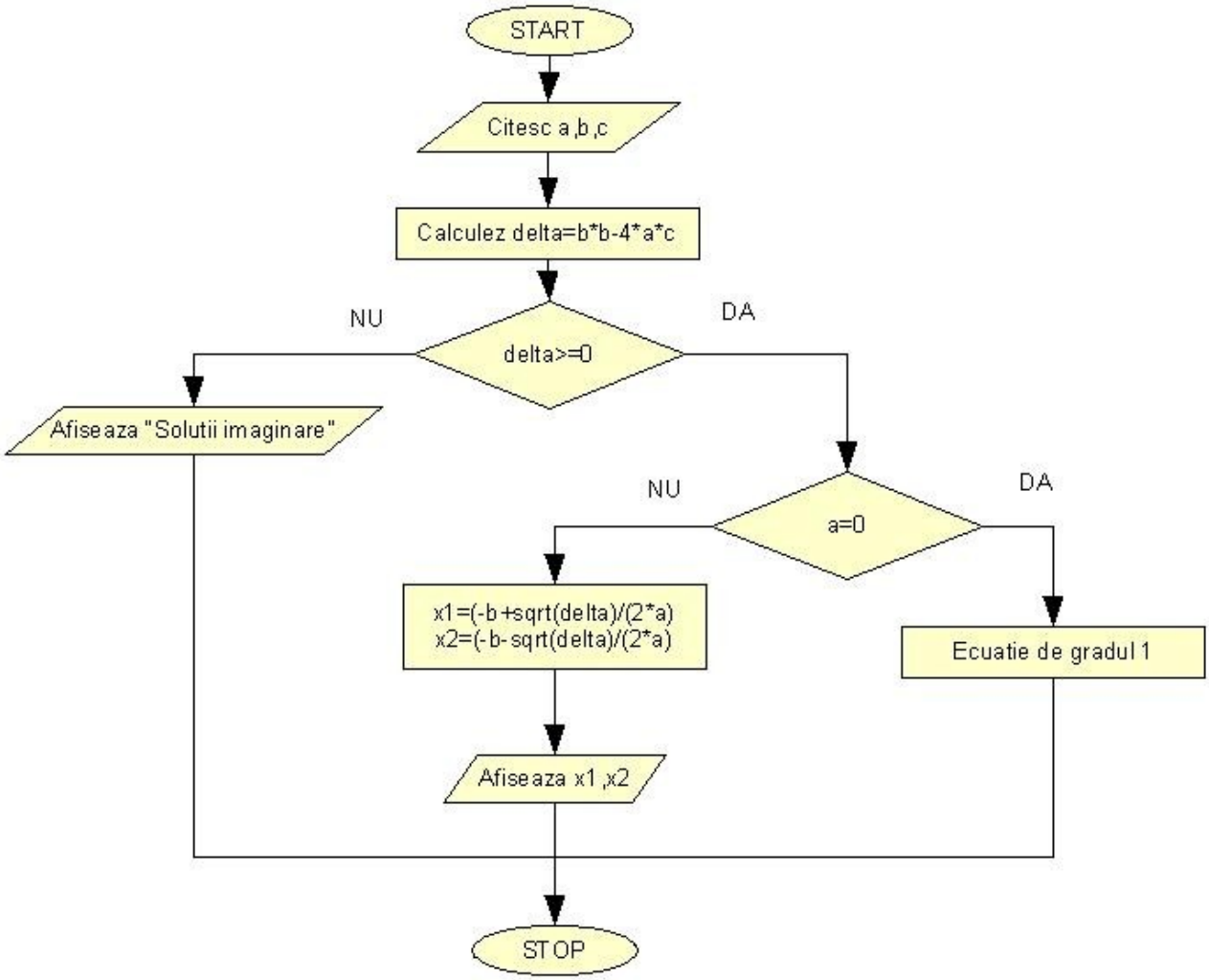


Pentru a implementa schema logica de mai sus e nevoie sa imbricam instructiuni if in alte instructiuni if

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// programul rezolva ecuatia de gradul I ax+b=0

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

int main(void)
{
```

```
        double a,b,x;
        cout <<" \nProgramul rezolva ecuatia de gradul I de forma: ax+b=0";
        cout <<" \n\n\tIntroduceti coeficientul a: ";
        cin >> a;
        cout <<" \n\tIntroduceti coeficientul b: ";
        cin >> b;
        if (b==0){
                if (a==0)
                        cout <<"\n\n\tSolutii infinite!! " ;
                else
                        cout <<"\n\n\tx=0 " ;
        }
        else{
                if (a==0)
                cout <<"\n\n\t x= infinit ";
                else{
                        x=-b/a;
                        cout <<"\n\n\t x=" << x;
                }
        }
        cin.ignore();
        cin.get();
        return 0;
}
```

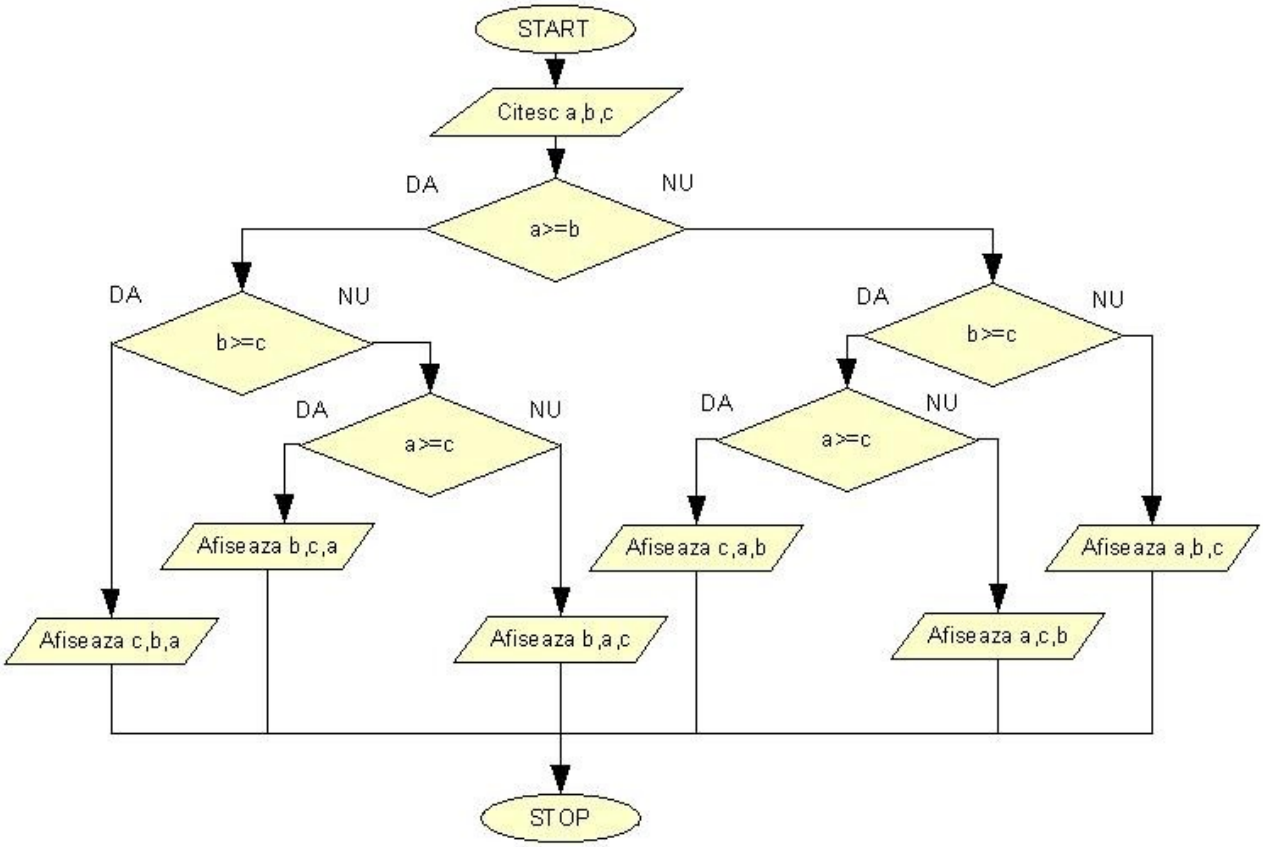
Rezolvarea ecuatiei de gradul doi se gaseste in schema logica de mai jos.



Pentru a implementa schema logica de mai sus se folosesc instructiuni if imbricate.

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// programul rezolva ecuatia de gradul II de forma ax*x+bx+c=0

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

int main(void)
{
        double a,b,c,delta,x1,x2;
        cout <<" \nProgramul rezolva ecuatia de gradul II de forma:
ax*x+bx+c=0";
        cout <<" \n\n\tIntroduceti coeficientul a: ";
        cin >> a;
        cout <<" \n\tIntroduceti coeficientul b: ";
        cin >> b;
        cout <<" \n\tIntroduceti coeficientul c: ";
        cin >> c;
        delta=b*b-4*a*c;

        if (delta>=0){
                if (a==0){
                        // ecuatie de gradul I
                        if (c==0){
                                if (b==0)
                                        cout <<"\n\n\tSolutii infinite!! " ;
                                else
                                        cout <<"\n\n\tx=0 " ;
                        }
                        else{
                        if (b==0)
                                cout <<"\n\n\t x= infinit ";
                                else{
                                        x1=-c/b;
                                        cout <<"\n\n\t x1=" << x1;
                                }
                        }
                }
                else{
                        x1=(-b+System::Math::Sqrt(delta))/(2*a);
                        x2=(-b-System::Math::Sqrt(delta))/(2*a);
                        cout <<"\n\n\tx1= " << x1;
                        cout <<"\n\n\tx2= " << x2;
                }
        }
        else
        cout << "\n\n\tSolutii imaginare!! " ;
        cin.ignore();
        cin.get();
        return 0;
}
```

Sa realizam programul pentru ordonarea a trei numere conform schemei logice de mai jos.

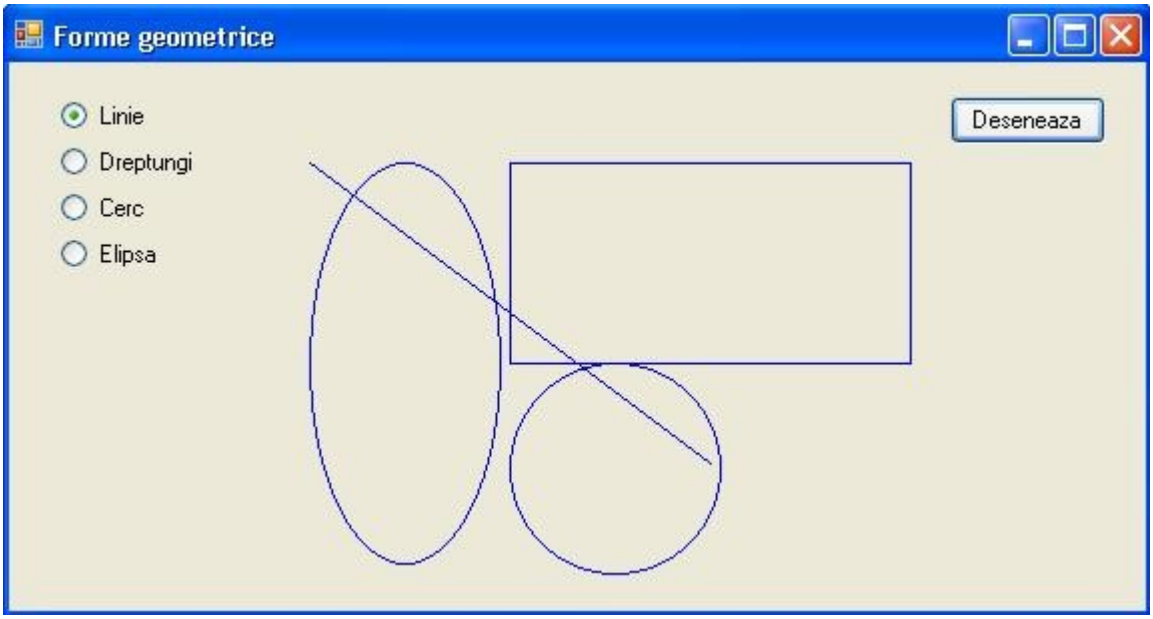



```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// programul ordoneaza crescator trei numere citite de la tastatura

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

int main(void)
{
        double a,b,c;
        cout <<" \nProgramul ordoneaza crescator trei numere :a,b,c" ;
        cout <<" \n\n\tIntroduceti numarul a: ";
        cin >> a;
        cout <<" \n\tIntroduceti numarul b: ";
        cin >> b;
        cout <<" \n\tIntroduceti numarul c: ";
        cin >> c;
        if (a>=b){
                if (b>=c){
                        cout <<"\n\n\tOrdinea este c-b-a " << c <<" : " << b
<<" : " << a  ;
                }else{
                        if (a>=c){
                                cout <<"\n\n\tOrdinea este b-c-a " << b <<" : "
<< c <<" : " << a  ;
                        }else{
```

```
                                cout <<"\n\n\tOrdinea este b-a-c " << b <<" : "
<< a <<" : " << c  ;
                        }
                }
        }else{
                if (b>=c){
                        if (a>=c){
                                cout <<"\n\n\tOrdinea este c-a-b " << c <<" : "
<< a <<" : " << b  ;
                        }else{
                                cout <<"\n\n\tOrdinea este a-c-b " << a <<" : "
<< c <<" : " << b  ;
                        }
                }else{
                        cout <<"\n\n\tOrdinea este a-b-c " << a <<" : " << b
<<" : " << c  ;
                }
        }
        cin.ignore();
        cin.get();
        return 0;
}
```

Vom utiliza in continuare instructiunea if pentru a alege forma geometrica ce urmeaza a fi desenata pe ecran.

Deschidem un nou proiect Windows Forms Application intitulat "forme_geom" si plasam patru obiecte de tip Radio Button cu numele radioButton1...radioButton4. Plasam apoi un Buton cu numele button1 .



Utilizatorul trebuie sa decida ce fel de forma geometrica doreste, apoi sa apese button1:"Deseneaza".

Completam procedura creata pe evenimentul click a obiectului button1 cu

```
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_albastru;
        Creion_albastru =gcnew System::Drawing::Pen
( System::Drawing::Color::Blue);
        if(this->radioButton1->Checked)
                Desen->DrawLine(Creion_albastru,150,50,350,200);
        if(this->radioButton2->Checked)
                Desen->DrawRectangle(Creion_albastru,250,50,200,100);
        if(this->radioButton3->Checked)
                Desen->DrawEllipse(Creion_albastru,250,150,105,105);
        if(this->radioButton4->Checked)
                Desen->DrawEllipse(Creion_albastru,150,50,95,200);
        delete Creion_albastru;
        delete Desen;
```

Pe langa folosirea instructiunii if mai apar niste elemente noi. Pana acum am plasat pe form-ul deschis o serie de obiecte (butoane, etichete..). Aceste obiecte sunt vizibile si pe form .Vom defini un obiect nou pe care sa desenam, numit chiar "Desen".

Definirea lui se face la fel cum definim variabilele (double a=23.55 ; char a ;System:string^ nume; etc) dar de data aceasta sintaxa este mai complicata si anume:

System::Drawing::Graphics^ Desen; adica am definit obiectul Desen de tipul System::Drawing::Graphics^

Creem acest obiect astfel:Desen = this->CreateGraphics();

Vom defini un obiect nou cu care sa desenam cu culoarea albastra , numit chiar "Creion_albastru".

System::Drawing::Pen^ Creion_albastru; adica am definit obiectul Creion_albastru de tipul System::Drawing::Pen^

Creem acest obiect astfel:Creion_albastru =gcnew System::Drawing::Pen(System::Drawing::Color::Blue);

Creerea acestor obiecte inseamna generarea codului sursa si incarcarea lui in memorie pentru a putea fi folosit atunci cind este nevoie. Dupa ce nu mai este nevoie de aceste obiecte, codul din memorie trebuie sters, altfel riscam sa incarcam inutil memoria cu cod ce nu va mai fi utilizat. Acest lucru se face cu instructiunile de la sfarfit :delete Creion_albastru;, delete Desen;

Instructiunile if se folosesc pentru a "gasi" RadioButtton-ul activat si pentru a lansa desenarea figurii geometrice dorite.

In cazul ca s-a optat pentru desenarea unei linii, se lanseaza instructiunea :

Desen->DrawLine(Creion_albastru,150,50,350,200); adica traseaza o linie cu creionul albastru pe obiectul "Dsen". Linia incepe din pozitia X1,Y1 pana in pozitia X2,Y2 exprimate in pixele. In cazul de fata X1=150;Y1=50;X2=350;Y2=200). Coltul stanga sus al form-ului are coordonatele 0,0 iar coltul dreapta jos are coordonatele Xmax=This->form1->Width, Ymax=This->form1->Height

In cazul ca s-a optat pentru desenarea unui dreptunghi, se lanseaza instructiunea :

Desen->DrawRectangle(Creion_albastru,250,50,200,100); adica se deseneaza un dreptungi cu creionul albastru pe obiectul "Dsen". Linia incepe din pozitia X1,Y1 pana in pozitia X2,Y2 exprimate in pixele. In cazul de fata X1=250;Y1=50;X2=200;Y2=100).

In cazul ca s-a optat pentru desenarea unui cerc, se lanseaza instructiunea :

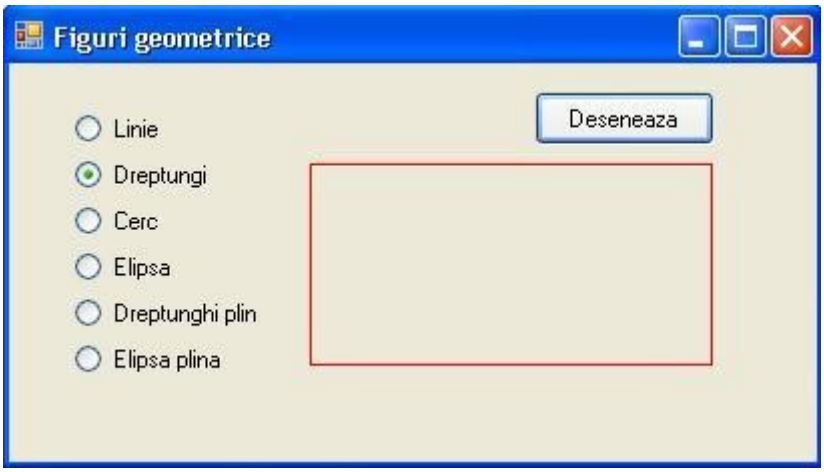
Desen->DrawEllipse(Creion_albastru,250,150,105,105); adica se deseneaza o elipsa cu creionul albastru pe obiectul "Dsen". Linia incepe din pozitia X1,Y1 de inaltime si latime egale exprimate in pixele. In cazul de fata X1=250;Y1=150;H=105;L=105). Desigur in acest car va rezulta un cerc.

In cazul ca s-a optat pentru desenarea unei elipse, se lanseaza instructiunea :

Desen->DrawEllipse(Creion_albastru,150,50,95,200);; adica se deseneaza o elipsa cu creionul albastru pe obiectul "Dsen". Linia incepe din pozitia X1,Y1 de inaltime si latime egale exprimate in pixele. In cazul de fata X1=150;Y1=50;H=95;L=200). Desigur in acest car va rezulta o elipsa.

Modificam aplicatia anterioara si vom desena si forme geometrice pline iar inainte de fiecare desenare vom sterge ecranul. Vom utiliza in continuare instructiunea if pentru a alege forma geometrica ce urmeaza a fi desenata pe ecran.

Deschidem un nou proiect Windows Forms Application intitulat "forme_geomm" si plasam patru obiecte de tip Radio Button cu numele radioButton1...radioButton6. Plasam apoi un Buton cu numele button1 .



Completam procedura creata pe evenimentul click a obiectului button1 cu

```
        System::Drawing::Graphics^ Desen;
                Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_albastru;
                Creion_albastru =gcnew System::Drawing::Pen
(System::Drawing::Color::Blue);
        System::Drawing::Pen^ Creion_rosu;
                Creion_rosu =gcnew System::Drawing::Pen
(System::Drawing::Color::Red);
        System::Drawing::Pen^ Creion_ciocolatiu;
                Creion_ciocolatiu =gcnew System::Drawing::Pen
(System::Drawing::Color::Chocolate);
        System::Drawing::SolidBrush^ Pensula;
                Pensula=gcnew System::Drawing::SolidBrush
(System::Drawing::Color::Aquamarine);
                if(this->radioButton1->Checked){
                        Desen->Clear(System::Drawing::Color(this->BackColor));
                        Desen->DrawLine(Creion_albastru,150,50,250,150);
                }
                if(this->radioButton2->Checked){
```

```
                        Desen->Clear(System::Drawing::Color(this->BackColor));
                        Desen->DrawRectangle(Creion_rosu,150,50,200,100);
                }
                if(this->radioButton3->Checked){
                        Desen->Clear(System::Drawing::Color(this->BackColor));
                        Desen->DrawEllipse(Creion_ciocolatiu,150,50,105,105);
                }
                if(this->radioButton4->Checked){
                        Desen->Clear(System::Drawing::Color(this->BackColor));
                        Desen->DrawEllipse(Creion_albastru,150,50,55,100);
                }
                if(this->radioButton5->Checked){
                        Desen->Clear(System::Drawing::Color(this->BackColor));
                        Desen->FillRectangle(Pensula, 150, 25, 75, 130);
                }
                if(this->radioButton6->Checked){
                        Desen->Clear(System::Drawing::Color(this->BackColor));
                        Desen->FillEllipse(Pensula, 150, 25, 75, 130);
                }
        delete Creion_albastru;
        delete Creion_rosu;
        delete Creion_ciocolatiu;
        delete Desen;
        delete Pensula;
```

Inainte de fiecare desenare se sterge ecranul cu instructiunea:
Desen->Clear(System::Drawing::Color(this->BackColor)); Culoarea folosita la stergere este chiar fundalul form-ului curent

Pentru desenarea formelor pline se defineste o pensula
System::Drawing::SolidBrush^ Pensula;
Se activeaza apoi cu:
Pensula=gcnew System::Drawing::SolidBrush(System::Drawing::Color::Aquamarine);
Formele geometrice pline se deseneaza apoi cu:
Desen->FillRectangle(Pensula, 150, 25, 75, 130); pentru dreptunghi si:
Desen->FillEllipse(Pensula, 150, 25, 75, 130); pentru elipsa.

Operatori logici

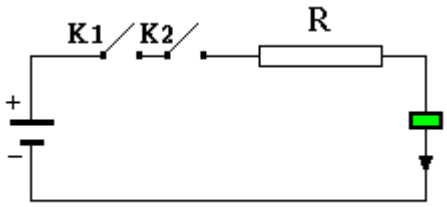
Pentru a combina comparatiile multiple intr-o singura instructiune if, se folosesc operatorii logici care leaga una sau mai multe expresii intr-o expresie finala.

Operator	Nume	Efect
&&	AND (si)	Expresia finala e adevarata daca toate expresiile sunt adevarate
||	OR (sau)	Expresia finala e adevarata daca cel putin una dintre expresii este adevarata
!	NOT (nu)	Inverseaza expresia adica daca expresia e adevarata , !expresia e falsa si invers

Operatorul logic &&

Operatorul logic SI este un operator binar combinand doua sau mai multe expresii binare. Returneaza "adevarat" numai daca toate expresiile combinate au valoarea "adevarat"

Sa presupunem ca avem circuitul de mai jos. Sa realizam un program in care sa interbam care este starea comutatoarelor K1 si K2 si programul sa afiseze daca ledul este aprins sau stins.




```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// Programul analizeaza starea unui circuit cu doua comutatoare serie K1 si K2.
// Afiseaza daca circuitul este inchis sau deschis.

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

int main(void)
{
        char  k1,k2;
        bool  c1,c2;
        cout <<" \nAnaliza stare circuit cu doua comutatoare serie K1 si K2." ;
        cout <<" \n\n\tK1 este inchis sau deschis(I/D): ";
        cin >> k1;
        cout <<" \n\tK2 este inchis sau deschis(I/D): ";
        cin >> k2;
        if (k1=='I')
```

```
                c1=true;
        else
                c1=false;
        if (k2=='I')
                c2=true;
        else
                c2=false;
        if (c1 && c2)
                cout <<" \n\n\tCircuitul este inchis si ledul e aprins: ";
        else
                cout <<" \n\n\tCircuitul este deschis si ledul e stins: ";
        cin.ignore();
        cin.get();
        return 0;
}
```

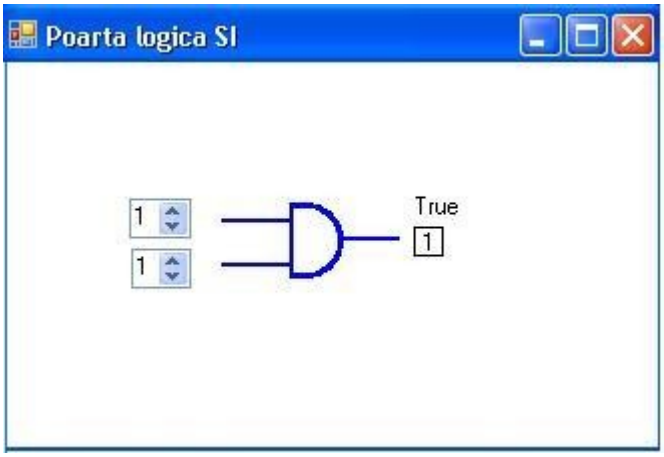
Sa realizam in Windows Forms Application o aplicatie care sa simuleze functionarea portii logice SI

Deschidem un nou proiect Windows Forms Application numit "si"
Plasam un obiect PictureBox cu numele pictureBox1.
Setam proprietatea Image cu imaginea de mai jos:

Plasam apoi doua obiecte de tip NumericUpDown cu numele numericUpDown1 respectiv numericUpDown2 pentru a simula intrarile portii logice.
Se mai plaseaza doua obiecte Label cu numele label1 respectiv label2, pentru a afisa iesirea portii sub forma numerica si sub forma booleana.
Completam procedurile create pe evenimentele ValueChanged a obiectelor NumericUpDown cu acelasi lucru si anume:

```
 this->label1->Text=System::Convert::ToString((this->numericUpDown1-
>Value)*(this->numericUpDown2->Value));
 this->label2->Text=System::Convert::ToString(System::Convert::ToBoolean(this-
>numericUpDown1->Value)&&
 System::Convert::ToBoolean(this->numericUpDown2->Value));
```

Dupa lansarea aplicatiei si stabilirea intrarilor la "1" ecranul aplicatiei arata astfel:



Operatorul logic ||

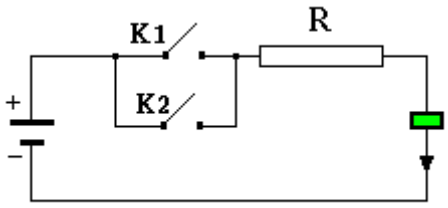
Operatorul logic SAU este un operator binar combinand doua sau mai multe expresii binare. Returneaza "adevarat" daca cel putin una din expresiile combinate au valoarea "adevarat"

Aplicatia care analiza un circuit cu doua comutatoare raspundea corect numai pentru raspunsul "I" nu si pentru raspunsul "i". Sa modificam aplicatia astfel incat sa lucreze curect si pentru raspunsul "I" nu si pentru raspunsul "i"

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// Programul analizeaza starea unui circuit cu doua comutatoare serie K1 si K2.
// Afiseaza daca circuitul este inchis sau deschis.

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

int main(void)
{
        char  k1,k2;
        bool  c1,c2;
        cout <<" \nAnaliza stare circuit cu doua comutatoare serie K1 si K2." ;
        cout <<" \n\n\tK1 este inchis sau deschis(I/D): ";
        cin >> k1;
        cout <<" \n\tK2 este inchis sau deschis(I/D): ";
        cin >> k2;
        if (k1=='I' || k1=='i')
                c1=true;
        else
                c1=false;
        if (k2=='I'|| k2=='i')
                c2=true;
        else
                c2=false;
        if (c1 && c2)
                cout <<" \n\n\tCircuitul este inchis si ledul e aprins: ";
        else
                cout <<" \n\n\tCircuitul este deschis si ledul e stins: ";
        cin.ignore();
```

```
        cin.get();
        return 0;
}
```

Sa presupunem ca avem circuitul de mai jos. Sa realizam un program in care sa interbam care este starea comutatoarelor K1 si K2 si programul sa afiseze daca ledul este aprins sau stins.




```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// Programul analizeaza starea unui circuit cu doua comutatoare in paralel K1
si K2.
// Afiseaza daca circuitul este inchis sau deschis.

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

int main(void)
{
        char  k1,k2;
        bool  c1,c2;
        cout <<" \nAnaliza stare circuit cu doua comutatoare paralel K1 si
K2." ;
        cout <<" \n\n\tK1 este inchis sau deschis(I/D): ";
        cin >> k1;
        cout <<" \n\tK2 este inchis sau deschis(I/D): ";
        cin >> k2;
        if (k1=='I' || k1=='i')
                c1=true;
        else
                c1=false;
        if (k2=='I'|| k2=='i')
                c2=true;
        else
                c2=false;
        if (c1 || c2)
                cout <<" \n\n\tCircuitul este inchis si ledul e aprins: ";
        else
                cout <<" \n\n\tCircuitul este deschis si ledul e stins: ";
        cin.ignore();
        cin.get();
        return 0;
}
```

Operatorul logic !

Operatorul logic NU este un operator binar folosit pentru inversarea valorii unei expresii booleene. Returneaza "adevarat" daca expresia are valoarea "fals" si invers returneaza "fals" daca expresia are valoarea "adevarat"

Programul urmator cere trei valori si calculeaza daca ele pot reprezenta lungimile laturilor unui triunghi. In cadrul acestui program s-au folosit pe langa operatorii $$ || si operatorul !.

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// programul cere trei valori si calculeaza daca ele pot reprezenta lungimile
laturilor unui triunghi

#include "stdafx.h"
#include < iostream >
using namespace std;
int main()
{ float a,b,c;
  bool t;
  cout << "Laturile unui triunghi\n";
  cout << "\n\tIntroduceti 3 numere reale:\n";
  cin >> a >> b >> c;
  cin.ignore();
  t = !( a <= 0 || b <= 0 || c <= 0);
  t = t && a < b + c && b < a + c && c < a + b;
  if( !t)
                cout << "\n\n\tVal. introduse NU pot fi lungimile laturilor unui
triunghi !";
  else
                cout << "\n\n\tValorile introduse pot fi lungimile laturilor
unui triunghi !";

  cin.get();
  return 0;
}
```

Prioritatile operatorilor logiici

Prioritatile intre operatorii logici si operatorii relationali sunt urmatoarele:

!

Operatori relationali <,<=,==,>,=>,!=

&&

||

In cazul in care dorim sa schimbam prioritatile se folosesc paranteze

In exemplul de sus avem expresiat = !(a <= 0 || b <= 0 || c <= 0); pentru a forta executia oeratorilor || mai intai si pe urma operatorul !

la fel se procedeaza si in cazul in care vrem sa schimbam prioritatea intre && si || si vrem sa fortam executia || mai repede

Exemplu: Sa reluam aplicatia de analiza a unui circuit cu doua comutatoare

K1 si K2 legate in paralel, la care adaugam si conditia suplimentara ca tensiunea sa fie peste 1,5V ca sa poata aprinde ledul.

```
// Program scris in C++ Visual Studio 2005 de tipul: CLR console application
// Programul analizeaza starea unui circuit cu doua comutatoare in paralel K1
si K2.
// Analizeaza daca circuitul este inchis sau deschis si afiseaza mesajul "led"
aprins
// numai daca circuitul e inchis si tensiunea aplicata este peste 1,5v
#include "stdafx.h"
#include < iostream >
using namespace std;

int main(void)
{
        char  k1,k2;
        bool  c1,c2;
        double u;
        cout <<" \nAnaliza stare led intr-un circuit cu doua comutatoare montate
paralel K1 si K2." ;
        cout <<" \n\n\tK1 este inchis sau deschis(I/D): ";
        cin >> k1;
        cout <<" \n\tK2 este inchis sau deschis(I/D): ";
        cin >> k2;
        cout <<" \n\tTensiunea U aplicata : ";
        cin >> u;
        if ((k1=='I' || k1=='i')&& (u>1.5))
                c1=true;
        else
                c1=false;
        if ((k2=='I'|| k2=='i')&& (u>1.5))
                c2=true;
        else
                c2=false;
        if (c1 || c2)
                cout <<" \n\n\Ledul e aprins: ";
        else
                cout <<" \n\n\tLedul e stins: ";
        cin.ignore();
        cin.get();
        return 0;
}
```

Pentru a forta executia operatorului || in fata operatorului && s-a utilizat expresia:

```
if ((k1=='I' || k1=='i')&& (u>1.5))
```

Instructiuni repetitive: while, do while, for

Instructiunea while

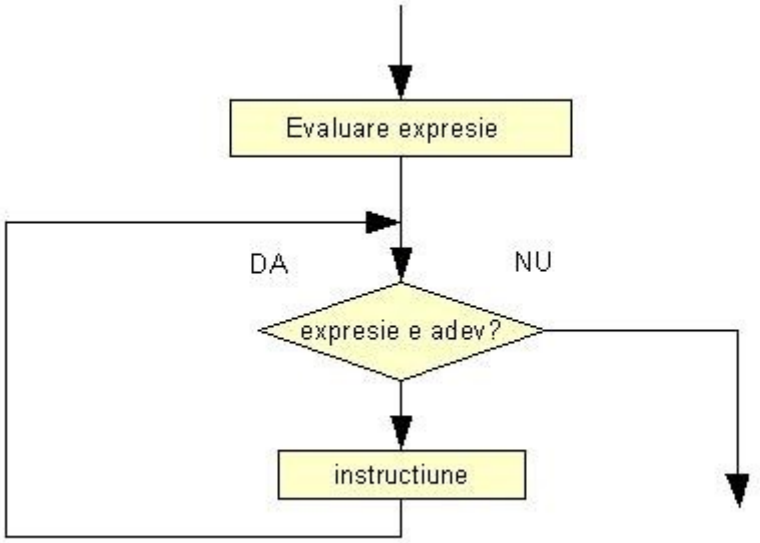
Instructiunea **while** Se foloseste pentru a executa repetitiv o instructiune sau o secventa de instructiuni atata timp cat o expresie este adevarata.

- **Formatul instructiunii:**

Instructiunea **while** are urmatorul format:
while (expresie) instructiune;

Instructiunea se executa repetat atîta timp cît valoarea expresiei este adevarata. Testul are loc înaintea fiecarei executii a instructiunii. Modul de functionare al instructiunii este urmatorul:

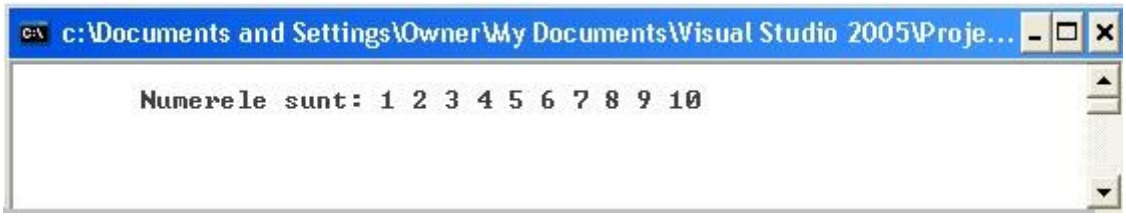
- Se testeaza expresia din paranteze. Daca ea este adevarata (sau expresia din paranteze are o valoare diferita de zero)
- Se executa corpul instructiunii while
- Se reia testarea si executia pana expresia devine falsa (sau valoarea expresiei din paranteze este zero)
- Se continua executia cu instructiunea de dupa corpul instructiunii while, deci instructiunea while se termina.



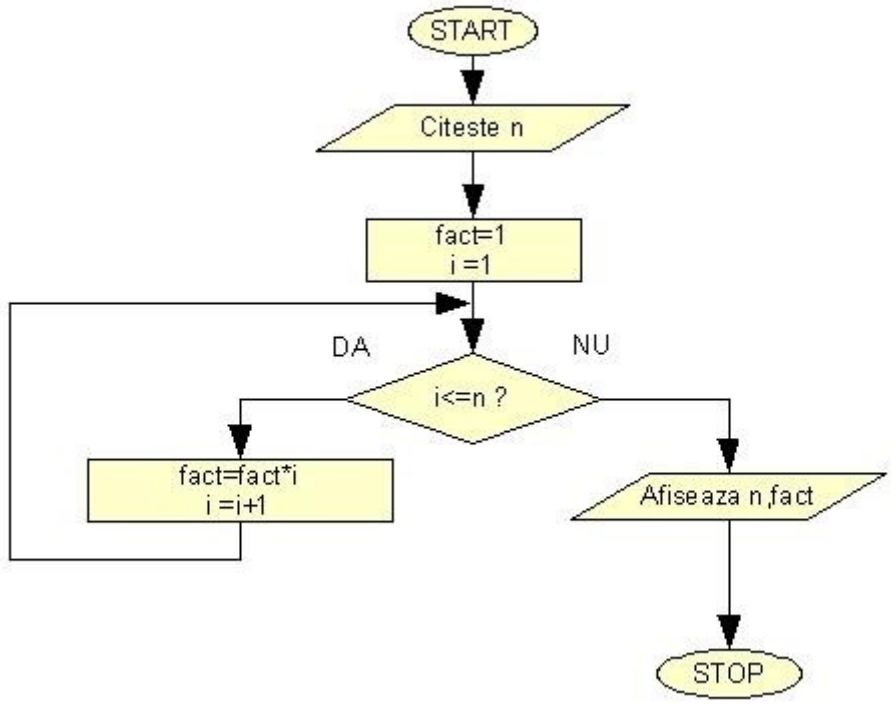
Sa realizam un program care afiseaza primele 10 numere naturale.

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// Programul afiseaza primele 10 numere naturale

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

int main(void)
{
        int i=1;
        cout <<" \n\tNumerele sunt: ";
        while (i<=10)
        {
        cout << i << " ";
        i+=1;
        }
        cin.ignore();
        cin.get();
        return 0;
}
```

Dupa rularea programului in "Command Prompt" se afiseaza:

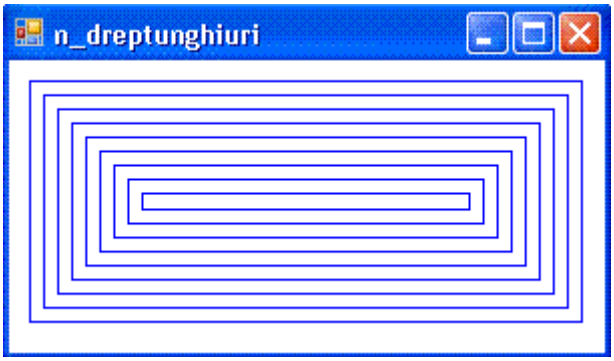


Schema logica de de mai jos, calculeaza n!. Sa realizam programul care implementeaza aceasta schema logica.




```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// Programul calculeaza n factorial.

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

int main(void)
{
        int i=1,n;
        float fact=1;
        cout <<" \n\tProgramul calculeaza n Factorial ";
        cout <<" \n\n\tIntroduceti numarul n: ";
        cin >> n;
        while (i<=n)
        {
        fact*=i;
        i+=1;
        }
        cout <<" \n\n\tRezultat: "<< n << "! =" << fact;
        cin.ignore();
        cin.get();
        return 0;
}
```

Instructiunea **while** se foloseste la fel si in modul grafic Windows Forms Application. Sa folosim instructiunea while pentru a afisa o serie de dreptunghiuri concentrice de forma celor de mai jos:



Deschidem un nou proiect Windows Forms Application intitulat "n_dreptunghiuri". De aceasta data nu mai plasam nici un Buton. Vom pune procedura noastra sa se execute odata cu deschiderea aplicatiei.

Completam procedura deschisa pe evenimentul paint al form-ului cu:

```
 int i,j,h,w,d,n=10;
 // i contor dreptunghiuri concentrice
 // n nr dreptunghiuri concentrice
 System::Drawing::Graphics^ Desen;
 Desen = this->CreateGraphics();
 System::Drawing::Pen^ Creion_albastru;
 Creion_albastru =gcnew System::Drawing::Pen(System::Drawing::Color::Blue);
 Desen->Clear(System::Drawing::Color(this->BackColor));
 h=this->Height-30;    //inaltimea maxima a dreptunghiurilor
 w=this->Width;                //latimea maxima a dreptunghiului
 d=h/2/n;              // distanta intre doua dreptunghiuri concentrice

 i=0;
 while (i<=d*n){
       Desen->DrawRectangle(Creion_albastru, i+10, i+10, (w-30-2*i),
(h-30-2*i));
       i+=d;
       }
 delete Creion_albastru;
 delete Desen;
```

Se observa ca inainte de trasarea dreptunghiurilor s-a sters ecranul cu instructiunea:

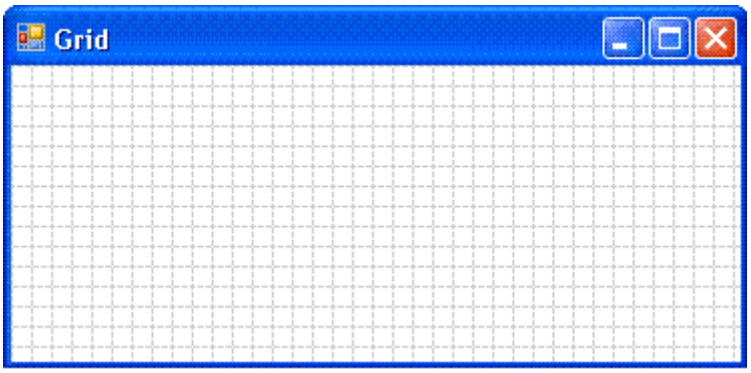
Desen->Clear(System::Drawing::Color(this->BackColor));

Instructiunea: **Desen->DrawRectangle(Creion_albastru, i+10, i+10, (w-30-2*i), (h-30-2*i));** traseaza un dreptunghi incepand cu pozitia x1=i la care s-a adaugat o margine de 10 pixeli si y1=i la care s-a adaugat o margine de 10 pixeli. Latimea dreptunghiului este w latimea maxima calculata - 2 margini de 15 pixeli adica 30 -2*i adica. I se incrementeaza cu d, d fiind distanta calculata intre doua dreptunghiuri Inaltimea este h -30-2*i similar cu latimea dreptunghiului.

Ingineria electrica se ocupa in general de studiul si utilizarea semnalelor electice de diferite forme, valori si intensitati. Un mare rol important il are analiza semnalelor electrice, analiza care presupune in primul rand vizualizarea acestora. Exista o serie de dispozitive menite sa

afiseze formele de unda a diferitelor semnale electrice (osciloscoape, vobuloscoape, analizoare spectrale etc..). De multe ori aceste semnale pot fi modelate matematic si simulate pe calculator. Vom incerca in continuare sa folosim programarea in mod grafic pentru a afisa diverse forme de unda ale diferitelor semnale electrice. Vom incepe prin a genera un grid peste care urmeaza in viitor sa trasam formele de unda.

Sa incercam sa realizam aplicatia care traseaza urmatorul grid pe ecran:



Deschidem un nou proiect Windows Forms Application intitulat "grid". Nu inseram nici un obiect pe form.

Completam procedura deschisa pe evenimentul paint al form-ului cu:

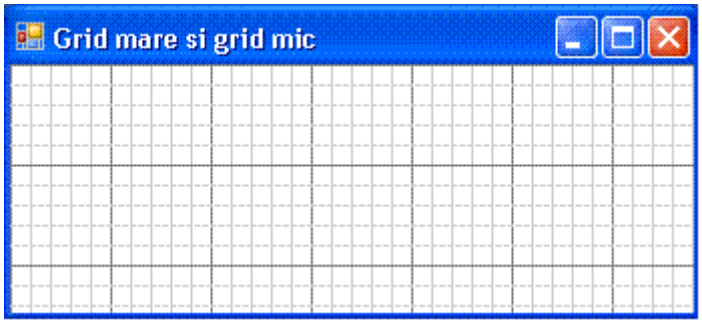
```
        int i=0;
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_gri;
                Creion_gri=gcnew System::Drawing::Pen
( System::Drawing::Color::LightGray);
        Desen->Clear(System::Drawing::Color(this->BackColor));

        // linii verticale

        while (i<=this->Width){
                Desen->DrawLine(Creion_gri, i, 0, i,this->Height);
                i+=10;
                }
        // linii orizontale

        i=0;
        while (i<=this->Height){
                Desen->DrawLine(Creion_gri, 0, i,this->Width,i);
                i+=10;
                }
        delete Creion_gri;
        delete Desen;
```

Se observa ca inainte de trasarea liniilor orizontale si verticale s-a sters ecranul cu instructiunea:

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

Daca dorim sa accentuam unele linii adica sa realizam un grid mai mare peste gridul mic sa zicem tot la 5 griduri mici sa trasam cate un grid mai

mare procedam astfel: daca ne aflam pe pozitia 10*5=50 schimbam culoarea cu un gri mai intens si optinem imaginea:

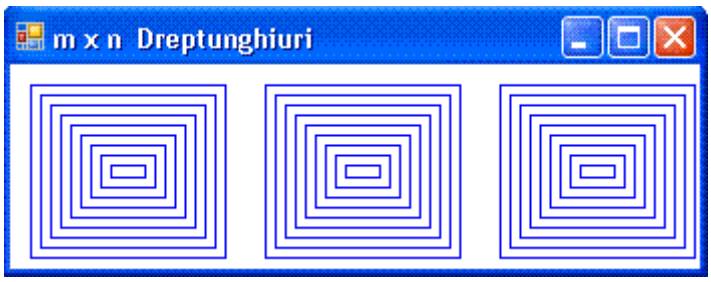



```
	int i=0;
	System::Drawing::Graphics^ Desen;
	Desen = this->CreateGraphics();
	System::Drawing::Pen^ Creion_gri_d;
	Creion_gri_d=gcnew System::Drawing::Pen
( System::Drawing::Color::LightGray);
	System::Drawing::Pen^ Creion_gri;
	Creion_gri=gcnew System::Drawing::Pen(System::Drawing::Color::Gray);
	Desen->Clear(System::Drawing::Color(this->BackColor));

	// linii verticale

	while (i<=this->Width){
		if (i%50==0)
			Desen->DrawLine(Creion_gri, i, 0, i,this->Height);
		else
			Desen->DrawLine(Creion_gri_d, i, 0, i,this->Height);
	i+=10;
	}

	// linii orizontale

	i=0;
	while (i<=this->Height){
		if (i%50==0)
			Desen->DrawLine(Creion_gri, 0, i,this->Width,i);
		else
			Desen->DrawLine(Creion_gri_d, 0, i,this->Width,i);
	i+=10;
	}
	delete Creion_gri_d;
	delete Creion_gri;
	delete Desen;
```

Se observa ca s-a folosit functia **%** numita si functia **mod** functie care ne returneaza restul impartirii. In cazul de fata am testat daca restul impartirii lui i cu 50 este 0 si atunci am schimbat culoare de la gri_d la gri'

Instructiuni whilw imbricate

Instructiunile **while** se pot insera unele in altele pe oricate nivele. Se spune ca s-a imbricat o instructiune while in alta instructiune while. Sa reluam aplicatia de desenare a n dreptunghiuri concentrice si sa desenam m grupuri de cite n dreptunghiuri concentrice. Vezi imaginea de jos!



Singura diferenta se gaseste in procedura atasata evenimentului pain al form-ului care devine:

```
        int i,j,h,w,d,m=3,n=10;
        // j contor dreptunghiuri
        // i contor dreptunghiuri concentrice
        // m nr dreptunghiuri
        // n nr dreptunghiuri concentrice
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_albastru;
        Creion_albastru =gcnew System::Drawing::Pen
( System::Drawing::Color::Blue);
        Desen->Clear(System::Drawing::Color(this->BackColor));
        h=this->Height-30;      //inaltimea maxima a dreptunghiurilor
        w=this->Width/m;        //latimea maxima a dreptunghiului
        d=w/2/n;                // distanta intre doua dreptunghiuri concentrice
        j=0;
        while (j<=m*w){
                i=0;
                while (i<=d*n){
                                Desen->DrawRectangle(Creion_albastru, j+i+10,
i+10, (w-20-2*i), (h-20-2*i));
                                i+=d;
                        }
                j+=w;
                }
        delete Creion_albastru;
        delete Desen;
```

Instructiunea do while

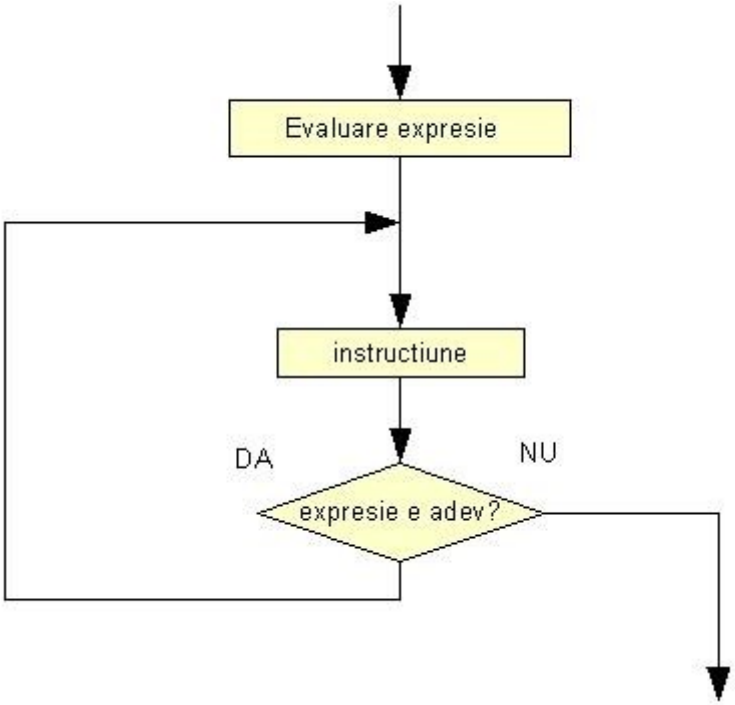
Instructiunea **do while** Se foloseste pentru a executa repetitiv o instructiune sau o secventa de instructiuni atata timp cat o expresie este adevarata, insa testul are loc dupa executia instructiunii deci ea se va executa cel putin o data.

- **Formatul instructiunii:**

Instructiunea **do while** are urmatorul format:

```
do instructiune;
while (expresie);
```

Instructiunea se executa repetat pîna cînd expresia devine falsa. Testul are loc dupa fiecare executie a instructiunii.



Sa presupunem ca vrem sa calculam aria sectiunii unui conductor electric. Programul cere diametrul conductorului si afiseaza aria sectiunii conductorului. In cazul ca avem de calculat mai multe sectiuni pentru diferite cabluiri, trebuie de fiecare data sa lansam programul, sa dam diametrul si dupa afisarea rezultatului, programul se termina si trebuie lansat din nou. Vom completa programul cu interbarea :Continuati(D/N) ? si daca raspunsul este D sa reluam executia pana se raspunde cu "N". Practic este ideal de folosit instructiunea do while care executa instructiunea si pe urma testeaza daca e adevarata expresia.

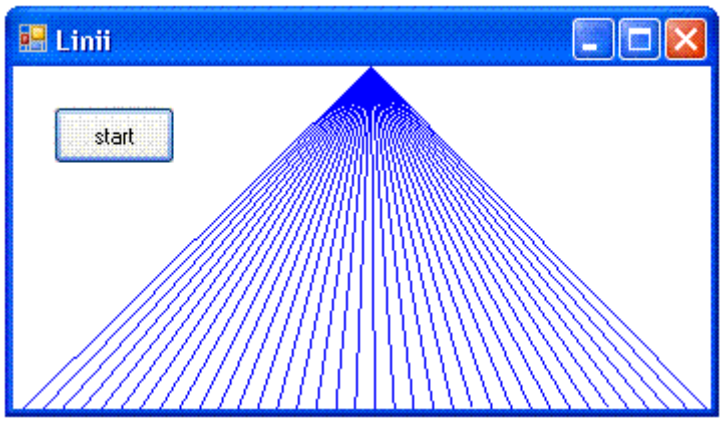
```
// Program pentru calculul sectiunii unui conductor electric
// Se citeste diametrul conductorului electric se afiseaza aria sectiunii lui
// Se reia programul pana se raspunde cu N la intrebarea : "Continuati ?"
#include "stdafx.h"
#include < iostream >
using namespace std;
int main(void)
{
        double d,s;
        char c='D';
        cout <<" \nProgram pentru calculul sectiunii unui conductor electric";
        do {
                cout <<" \n\n\tIntroduceti diametrul conductorului:";
                cin >> d;
                s=System::Math::PI*(d/2)*(d/2);
                cout <<" \n\n\tSectiunea conductorului de diametru : "<< d << "
este : " << s;
                cin.ignore();
                cout <<" \n\n\tContinuati (D/N) ?:";
                cin >> c;
        } while (c=='D' || c=='d' );
        return 0;
}
```

Pentru optimizare am putea sa eliminam intrebarea :Continuati? si sa continuam programul pana se introduce 0 pentru diametru.

In acest caz programul devine:

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// Program pentru calculul sectiunii unui conductor electric
// Se citeste diametrul conductorului electric se afiseaza aria sectiunii lui
// Programul se reia pana se introduce valoarea 0 la diametru

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

int main(void)
{
        double d,s;
        cout <<" \nProgram pentru calculul sectiunii unui conductor electric";
        do {
        cout <<" \n\n\tIntroduceti diametrul conductorului (0=terminare
program):";
        cin >> d;
        s=System::Math::PI*(d/2)*(d/2);
        cout <<" \n\n\tSectiunea conductorului de diametru : "<< d << " este : "
<< s;
        cin.ignore();
        } while (d!=0 );
        return 0;
}
```

Folosin instructiunea **do while** , sa realizam programul care traseaza mai multe linii pe formul deschis de forma imaginii de jos:



Deschidem un nou proiect Windows Forms Application intitulat "linii_m" si plasam un Buton cu numele button1

Completam procedura deschisa pe evenimentul click al butonului1 cu:

```
        int i=0;
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_albastru;
        Creion_albastru =gcnew System::Drawing::Pen
( System::Drawing::Color::Blue);
        do {
            Desen->DrawLine(Creion_albastru, this->Width/2, 0, i, this-
>Height-30);
            i+=10;
        }
        while (i<=this->Width);

        delete Desen;
        delete Creion_albastru;
```

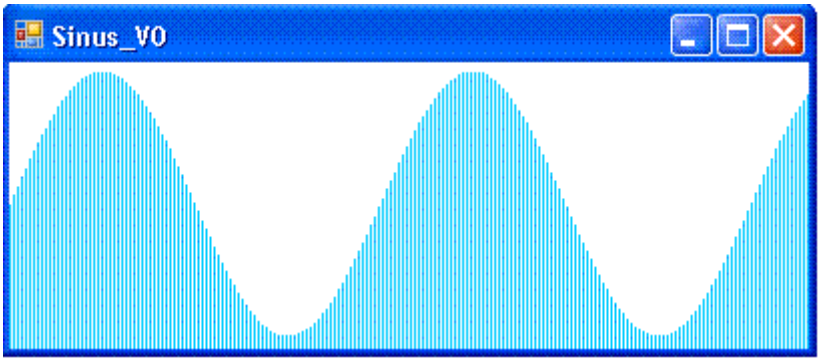
Sa desenam acum graficul functiei sinus

Deschidem un nou proiect Windows Forms Application intitulat "sinus_v0". Procedura de desenare se va executa odata cu deschiderea aplicatiei pe evenimentul paint.

Completam procedura deschisa pe evenimentul paint al form-ului cu:

```
        int i=0;
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_blue;
        Creion_blue=gcnew System::Drawing::Pen
(System::Drawing::Color::DeepSkyBlue);
        Desen->Clear(System::Drawing::Color(this->BackColor));
        do {
                int factor_s=this->Height/2-22;
                int y=5+factor_s*(1-System::Math::Sin(0.0314*i));
                Desen->DrawLine(Creion_blue, i, this->Height, i, y);
                i+=2;
        }
        while (i<=this->Width);
        delete Creion_blue;
        delete Desen;
```

Dupa rularea programului ecranul aplicatiei afieaza:



Dupa cum se vede in procedura ce se activeaza pe evenimentul "paint" al form-ului s-a folosit variabila "factor_s" pentru a defini factorul de scala. Avand in vedere ca vrem sa afisam graficul functiei sin(i) si sa dimensionam graficul in functie de marimea ferestrei deschise, definim acest factor de scala ca fiind:
this->Height/2-22 adica inaltimea form-ului curent/2-22(22 de pixeli pentru a scadea latimea benzii de sus a formului in jur de 17 pixeli +5 pixeli loc liber) pentru a marca mijlocul form-ului.

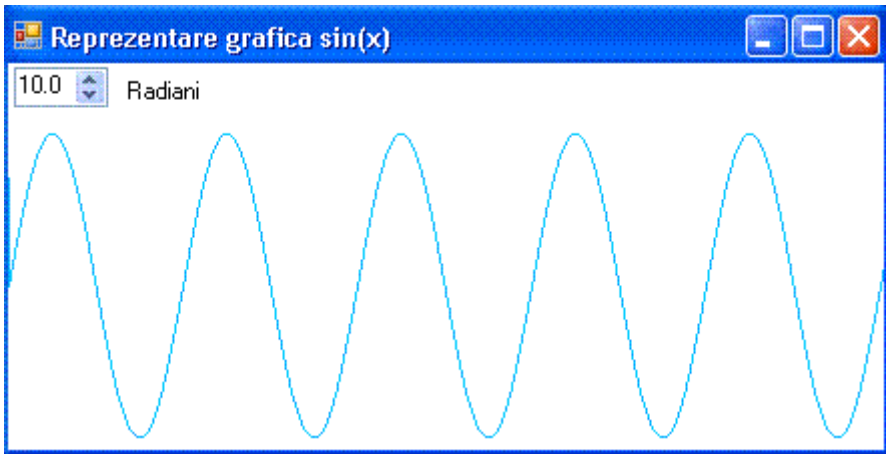
Stiind ca sin(i) poate lua valori inte +1 si -1, inmultim cu factorul de scala si graficul va fi desfasurat pe aproape toata zona libera din form.

Valorile functiei sin(i) se calculeaza cu expresia:**int y=5+factor_s*(1-System::Math::Sin(0.0314*i));**, unde am adaugat 5 pixeli pentru margine si valoarea functiei sin inmultita cu factorul de scala. I ia valori intre 0 si latimea form-ului cu increment 2. Am folosit increment 2 deoarece pentru increment 1 nu s-ar mai vedea liniile, ele fiind una langa alta, s-ar vedea o zona compacta. De ce am folosit sin(0.0314*i) si nu sin(i)?

I ia valori intre 0 si this->Width. Am vrea sa afisam o semi-perioada pe 100 de puncte. Perioada functiei sin fiid 2 pi radiani ar rezulta 3,14/100 adica 0.0314. Putem imbunatati aplicatia de sus si sa cerem numarul de perioade care sa fie afisate, iar afisarea sa nu se mai faca folosind linii verticale ci linii care sa uneasca doua puncte consecutive. Vom introduce un obiect NumericUpDown cu numele "numericUpDown1" iar pe evenimentul "ValueChanged" vom introduce procedura:

```
        int i=0,y=0,y_v=0,lat,factor_s;
        double x,rad;
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_blu;
        Creion_blu=gcnew System::Drawing::Pen
( System::Drawing::Color::DeepSkyBlue);
        Desen->Clear(System::Drawing::Color(this->BackColor));
        factor_s=this->Height/2-37;
        rad=System::Convert::ToDouble(this->numericUpDown1->Value);
        lat=this->Width;
        do {
                x=i*rad*System::Math::PI/(lat-10);
                y=35+factor_s*(1-System::Math::Sin(x));
                Desen->DrawLine(Creion_blu, i-1,y_v, i, y);
                y_v=y;
                i+=1;
        }
        while (i<=lat);
        delete Creion_blu;
        delete Desen;
```

Rulam aplicatia, setam butonul numericUpDown1 la valoarea 10 si obtinem:



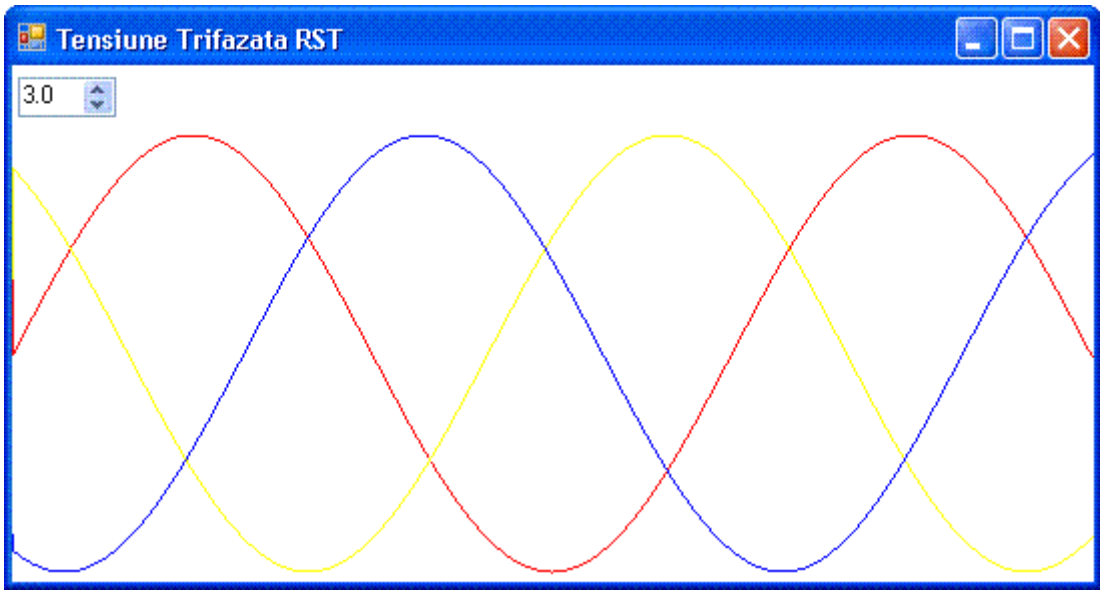
Se observa ca:
-la factorul de scala : **factor_s=this->Height/2-37;** am scazut 37 pentru a avea loc si pentru butonul numericUpDown1
-numarul de radiani pentru care se afiseaza functia s-a calculat cu:
rad=System::Convert::ToDouble(this->numericUpDown1->Value);
-valorile lui x se calculeaza pentru fiecare i astfel:
x=i*rad*System::Math::PI/(lat-10);10 fiind latimea border-ului form-ului
-latimea ferestrei in care se afiseaza este: **lat=this->Width;**
-pentru i parcurgand intervalui i...lat, x ia valorile 0...rad

Sa incercam sa afisam forma de unda a tensiunii trifazate, compusa din trei sinusoide defazate intre ele cu cate 2*pi/3

Aplicatia este asemanatoare cu cea anterioara cu deosebirea ca initializam trei creoane de culorile rosu, galben, albastru ,culori consacrate pentru cele 3 faze: R S T si afisarea a trei sinusoide defazate intre ele cu cate 2*pi/3

```
int i=0,y=0,y_v=0,lat,factor_s;
double x,rad;
System::Drawing::Graphics^ Desen;
Desen = this->CreateGraphics();
System::Drawing::Pen^ Creion_r;
Creion_r=gcnew System::Drawing::Pen(System::Drawing::Color::Red);
System::Drawing::Pen^ Creion_g;
Creion_g=gcnew System::Drawing::Pen(System::Drawing::Color::Yellow);
System::Drawing::Pen^ Creion_a;
Creion_a=gcnew System::Drawing::Pen(System::Drawing::Color::Blue);
Desen->Clear(System::Drawing::Color(this->BackColor));
factor_s=this->Height/2-37;
rad=System::Convert::ToDouble(this->numericUpDown1->Value);
lat=this->Width;
i=0;
do {
        x=i*rad*System::Math::PI/(lat-10);
        y=35+factor_s*(1-System::Math::Sin(x));
        Desen->DrawLine(Creion_r, i-1,y_v, i, y);
        y_v=y;
        i+=1;
}
while (i<=lat);
i=0;
do {
        x=(i+2*lat/rad/3)*rad*System::Math::PI/(lat-10);
        y=35+factor_s*(1-System::Math::Sin(x));
        Desen->DrawLine(Creion_g, i-1,y_v, i, y);
        y_v=y;
        i+=1;
}
while (i<=lat);
i=0;
do {
        x=(i+4*lat/rad/3)*rad*System::Math::PI/(lat-10);
        y=35+factor_s*(1-System::Math::Sin(x));
        Desen->DrawLine(Creion_a, i-1,y_v, i, y);
        y_v=y;
        i+=1;
}
while (i<=lat);
delete Creion_r;
delete Creion_g;
delete Creion_a;
delete Desen;
```

Rulam aplicatia, setam butonul numericUpDown1 la valoarea 3 si obtinem:



Graficele desenate pana acum au fost realizate in coordonate carteziene. Modul de reprezentare in sistemul cartezian de coordonate, nu este singurul mod in care se pot reprezenta grafic functiile matematice.

Sistemul polar de coordonate este un sistem de coordonate bidimensional în care fiecarui punct din plan i se asociaza un unghi si o distanta. Fiecare punct este determinat de doua coordonate polare: coordonata radiala si coordonata unghiulara.

- Coordonata radiala (notata de obicei cu r) reprezinta distanta unui punct fata de un punct central, numit pol
- Coordonata unghiulara (cunoscuta si sub numele de unghi polar, sau azimut, si notata cu t) reprezinta unghiul

Cele doua coordonate polare r si t pot fi convertite în coordonate carteziene x si y prin utilizarea functiilor trigonometrice sinus si cosinus astfel:

x=r*cos(t)
y=r*sin(t)

Sa incercam sa reprezentam grafic in coordonate polare functia: r(t)=a*cos(3*t+fi)

Functia se mai numeste si roza polara cu 3 petale.

Roza polara este o curba matematica celebra care arata ca o floare cu petale si care poate fi exprimata ca o ecuatie polara simpla, de forma r(t)=a*cos(3*t).Daca n este întreg, aceasta ecuatie produce o roza cu n petale, daca n este impar, sau cu 2n petale daca este par. Daca n este rational dar nu întreg, o forma asemanatoare cu roza ar putea aparea, dar va avea petale suprapuse.

Tinand cont de ecuatiile de transformare in coordonate carteziene, vom obtine expresiile lui x si y de forma:

x=Cos(3*t)*Cos(t);
y=Cos(3*t)*Sin(t);

T ia valori intre o si 2pi adica 0 si 6.3 radiani

Tinand cont ca x si y vor lua valori intere -1 si 1 trebuie sa le inmultim cu un factor de scala=inaltimea form-ului/2.

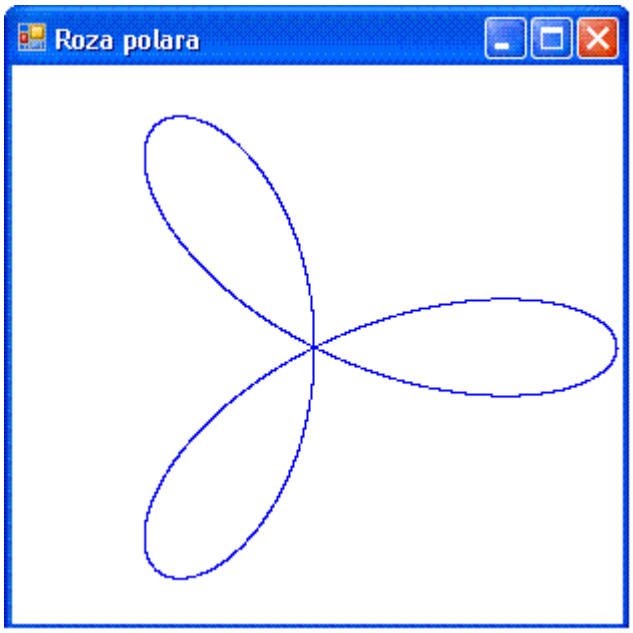
Pentru a incepe din mijlocul formulul x si y trebuie adunati cu x0 respectiv y0

x0=latimea formului/2
y0=inaltimea form-ului/2

Deschidem un nou proiect in Windows Forms Application numit **r_polara**
Pe evenimentul paint punem procedura:

```
double t=0;
int x_v,y_v,x,y,x0,y0;
System::Drawing::Graphics^ Desen;
Desen = this->CreateGraphics();
System::Drawing::Pen^ Creion_albastru =
gcnew System::Drawing::Pen(System::Drawing::Color::Blue);
Desen->Clear(System::Drawing::Color(this->BackColor));
x0=(this->Width-10)/2;
y0=(this->Height-50)/2;
x_v=x0+x0*System::Math::Cos(3*t)*System::Math::Cos(t);
y_v=y0+10+y0*System::Math::Cos(3*t)*System::Math::Sin(t);
do {
x=x0+x0*System::Math::Cos(3*t)*System::Math::Cos(t);
y=y0+10+y0*System::Math::Cos(3*t)*System::Math::Sin(t);
Desen->DrawLine(Creion_albastru, x_v, y_v, x, y);
x_v=x;
y_v=y;
t+=0.001;
}
while (t<=6.3);
delete Desen;
delete Creion_albastru;
```

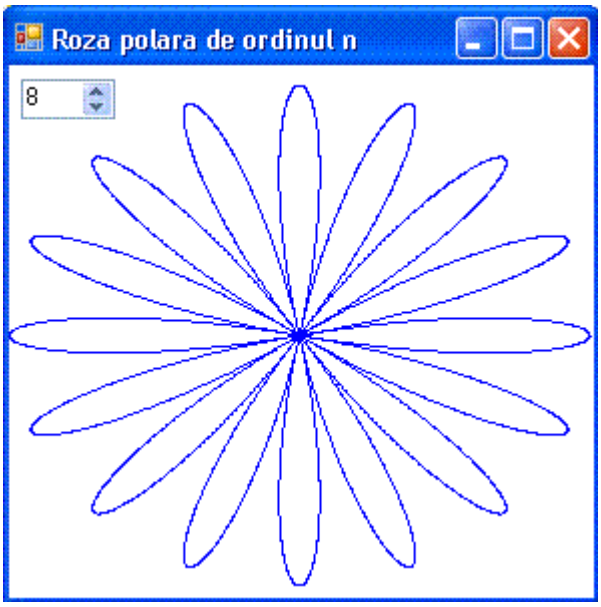
Dupa rularea programului, obtinem:



- Pe baza aplicatiei de mai sus sa realizam un nou proiect in Windows Forms Application numit **r_polara_n**
- Plasam un obiect de tip NumericUpDown cu numele numericUpDown1 pentru a permite introducerea ordinului n
- Pe evenimentul ValueChanged plasam procedura:

```
double t=0,n;
int x_v,y_v,x,y,x0,y0;
System::Drawing::Graphics^ Desen;
Desen = this->CreateGraphics();
System::Drawing::Pen^ Creion_albastru =
gcnew System::Drawing::Pen(System::Drawing::Color::Blue);
Desen->Clear(System::Drawing::Color(this->BackColor));
x0=(this->Width-10)/2;
y0=(this->Height-50)/2;
n=System::Convert::ToDouble(this->numericUpDown1->Value);
x_v=x0+x0*System::Math::Cos(n*t)*System::Math::Cos(t);
y_v=y0+10+y0*System::Math::Cos(n*t)*System::Math::Sin(t);
do {
x=x0+x0*System::Math::Cos(n*t)*System::Math::Cos(t);
y=y0+10+y0*System::Math::Cos(n*t)*System::Math::Sin(t);
Desen->DrawLine(Creion_albastru, x_v, y_v, x, y);
x_v=x;
y_v=y;
t+=0.001;
}
while (t<=6.3);
delete Desen;
delete Creion_albastru;
```

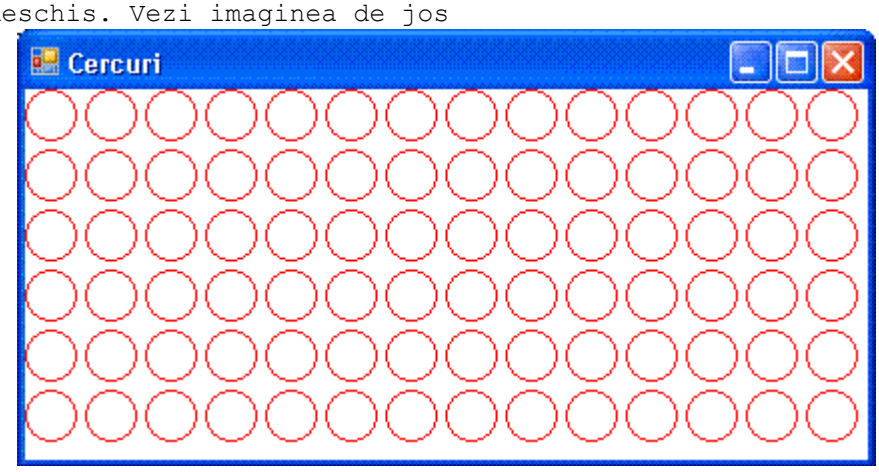
Dupa rularea programului, modificam NumericUpDown la 8 si obtinem:



Instructiuni do while implicate

Instructiunile **do while** pot fi imbricate putindu-se astfel implementa algoritmi cu un grad inalt de complexitate.

Sa incercam sa afisam pe ecran cercuri de diametru 25, cercuri asezate pe linii si coloane astfel incat sa umplem tot ecranul indiferent de marimea form-ului deschis. Vezi imaginea de jos



Deschidem un nou proiect Windows Forms Application intitulat "cercuri". Completam procedura deschisa pe evenimentul paint al form-ului cu:

```
int i=0,j=0;
System::Drawing::Graphics^ Desen;
Desen = this->CreateGraphics();
System::Drawing::Pen^ Creion_rosu;
Creion_rosu=gcnew System::Drawing::Pen(System::Drawing::Color::Red);
Desen->Clear(System::Drawing::Color(this->BackColor));
do{
        i=0;
        do{
                Desen->DrawEllipse( Creion_rosu, i, j, 25,25);
                i+=30;
        }while (i<=this->Width-30);
        j+=30;
}while (j<=this->Height-50);
delete Creion_rosu;
delete Desen;
```

Instructiunea for

Instructiunea **for** Se foloseste pentru a executa repetitiv o instructiune sau o secventa de instructiuni. De obicei implementeaza structura ciclica cu numar cunoscut de pasi.

- **Formatul instructiunii:**

Instructiunea **for** are urmatorul format:

for ([expresie1]; [expresie2]; [expresie3]) instructiune

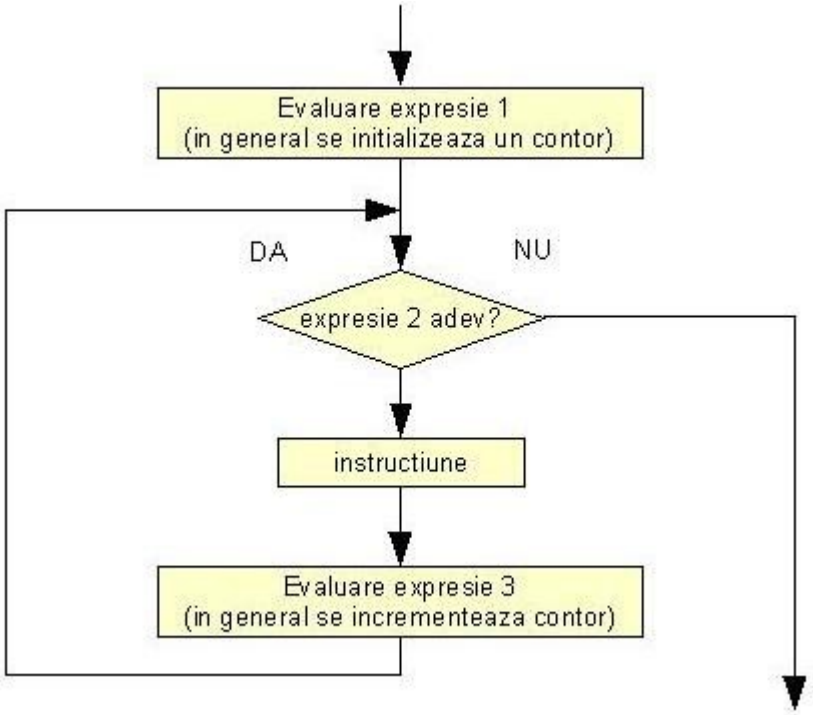
Aceasta instructiune este echivalenta cu:

```
expresie1;
while (expresie2) {
    instructiune;
    expresie3;
    }
```

Unde :

expresie1 constituie initializarea ciclului si se executa o singura data înaintea ciclului.

expresie2 specifica testul care controleaza ciclul. El se executa înaintea fiecarei iteratii. Daca conditia din test este adevarata atunci se executa corpul ciclului, dupa care se executa expresie3, care consta de cele mai multe ori în modificarea valorii variabilei de control al ciclului. Se revine apoi la reevaluarea conditiei. Ciclul se termina cînd expresie2 devine falsa.



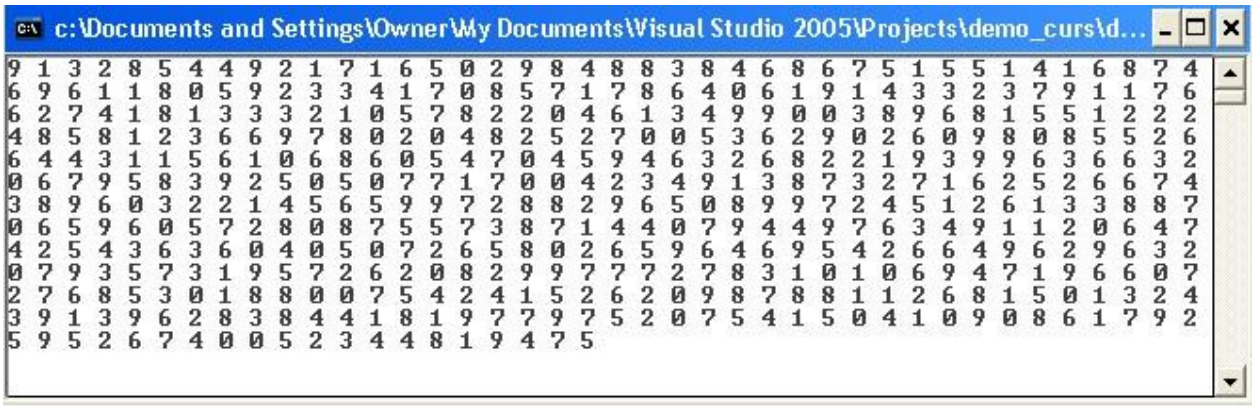
Oricare dintre expresiile instructiunii for sau chiar toate pot lipsi. Daca lipseste expresie2, atunci clauza while este echivalenta cu while

(1), ceea ce înseamna o conditie totdeauna adevarata. Alte omisiuni de expresii sînt pur si simplu eliminate din expandarea de mai sus.
    Instructiunile while si for executa testul de control la începutul ciclului si înaintea intrarii în corpul instructiunii.
    Instructiunea do executa cel putin o data instructiunea sau secventa de instructiuni.

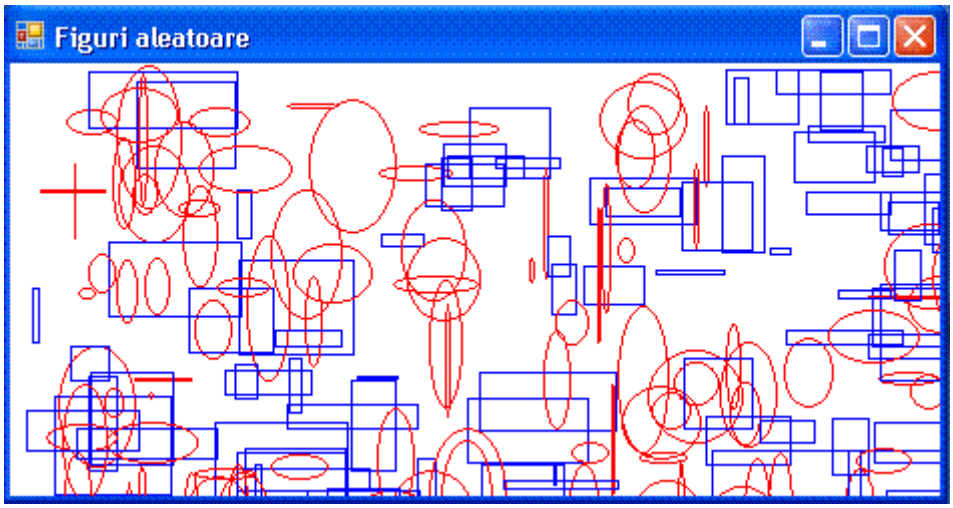
    sa realizam un program care afiseaza 500 de cifre aleatoare intre 0-9 despartite prin spatiu

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// Se utilizeaza spatiul de nume System::
// programul afiseaza 500 de cifre aleatoare intre 0-9 despartite prin spatiu

#include "stdafx.h"

using namespace System;

int main(void)
{
        System::Random^ n = gcnew System::Random(8);
        for ( int i=1; i<=500; i++)
                Console::Write(n->Next(10)+" ");
    Console::ReadLine();
        return 0;
}
```

    Dupa rularea programului in "Command Prompt" se afiseaza:



    Folosind generatorul de numere aleatoare descris mai sus, sa incercam sa realizam o aplicatie care deseneaza pe ecran figuri geometrice aleatoare de forma:



Deschidem un nou proiect Windows Forms Application intitulat "random_dr" .

Completam procedura deschisa pe evenimentul paint al form-ului cu:

```
        int i=0;
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_rosu;
        Creion_rosu=gcnew System::Drawing::Pen(System::Drawing::Color::Red);
        System::Drawing::Pen^ Creion_albastru;
        Creion_albastru=gcnew System::Drawing::Pen
(System::Drawing::Color::Blue);
        System::Random^ n = gcnew System::Random(8);
        Desen->Clear(System::Drawing::Color(this->BackColor));
        i=0;
        for ( int i=1; i<=100; i++){
         Desen->DrawRectangle(Creion_albastru,n->Next(this->Width),n->Next(this-
>Height),n->Next(75),n->Next(50));
         Desen->DrawEllipse(Creion_rosu,n->Next(this->Width),n->Next(this-
>Height),n->Next(50),n->Next(75));
        }
        delete Creion_rosu;
        delete Creion_albastru;
        delete Desen;
```

Pentru a afisa texte in mod grafic desigur se pot folosi etichete carora sa le atribuim textele dorite. De multe ori trebuie sa scriem multe texte si la pozitii diferite. Solutia este sa desenam textele la fel cum desenam formele geometrice. Va trebui sa definim si sa initializam un obiect grafic pe care sa "desenam" aceste texte. Ne propunem deci sa scriem texte pe form-ul deschis la diferite coordonate, fara a folosi obiecte de tip label.

Deschidem un nou proiect Windows Forms Application intitulat "text_grafic" si plasam un Buton "Start" cu numele button1.

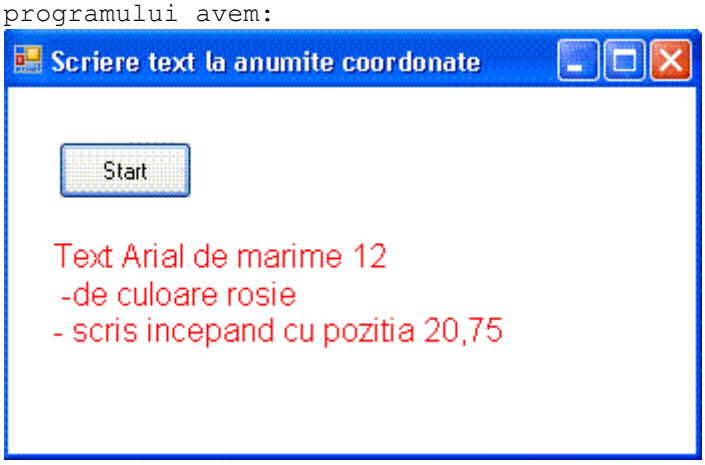
Completam procedura deschisa pe evenimentul click al butonului1 cu:

```
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::SolidBrush^ Pensula;
        Pensula=gcnew System::Drawing::SolidBrush(System::Drawing::Color::Red);
        System::Drawing::Font^ font_arial;
        font_arial=gcnew System::Drawing::Font("Arial",12);
        System::String^ text_t="Text Arial de marime 12 \n -de culoare rosie \n-
scris incepand cu pozitia 20,75";
        Desen->DrawString(text_t,font_arial,Pensula,20,75);
```

Avem deci nevoie de un obiect grafic de tip **System::Drawing::Graphics^** pe care sa "desenam" numit:"Desen" si o pensula de tip **System::Drawing::SolidBrush^** numita chiar "Pensula". Mai trebuie sa definim fontul cu care lucram adica **:System::Drawing::Font^ font_arial;** si sa initializam fontul respectiv **:font_arial=gcnew System::Drawing::Font("Arial",12);** setand numele fontului si inaltimea.

Pentru a scrie textul dorit, vom insera linia de cod:**Desen->DrawString(text_t,font_arial,Pensula,20,75);**, unde textul de tiparit text_t este definit ca: **System::String^ text_t="Text Arial de marime 12 \n -de culoare rosie \n- scris incepand cu pozitia 20,75";** iar 20, 75 reprezinta coordonatele (x1,y1 in pixeli) unde incepe textul de afisat.

Dupa rularea programului avem:



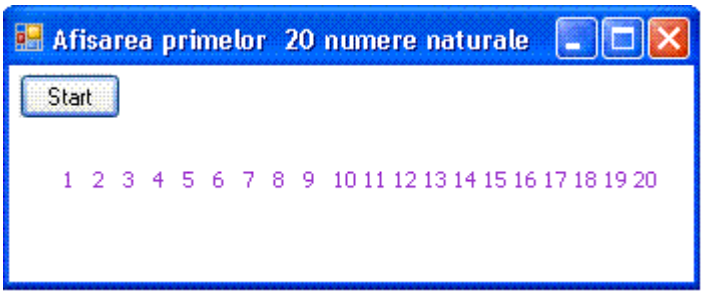
Folosindu-ne de modul de "desenare" ale caracterelor sa incercam sa scriem cifrele de la 1 la 20 pe un form la apasarea unui buton "Start"

Deschidem un nou proiect Windows Forms Application intitulat "linie_numere" si plasam un Buton "Start" cu numele button1.

Completam procedura deschisa pe evenimentul click al butonului1 cu:

```
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::SolidBrush^ Pensula;
        Pensula=gcnew System::Drawing::SolidBrush
(System::Drawing::Color::DarkOrchid );
        System::Drawing::Font^ font_nina;
        font_nina=gcnew System::Drawing::Font("Nina",8);
        for ( int i=1; i<=20; i++)
                Desen-
>DrawString(System::Convert::ToString(i),font_nina,Pensula,10+i*15,50);
```

Dupa rularea programului ecranul aplicatiei arata astfel:



Folosim din nou sistemul de coordonate polare pentru a afisa o spirala. Spirala are o ecuatie extrem de simpla in sistemul de coordonate polare, fiind echivalentul dreptei in sistemul de coordonate carteziene.

Ecuatia spiralei este:

$r(t)=a*t$, a fiind o constanta ce controleaza distanta dintre brate.

Pe baza ecuatiilor de conversie in sistemul cartezian:

$x=r*\cos(t)$

$y=r*\sin(t)$

Obtinem:

$x=a*t*\cos(t)$

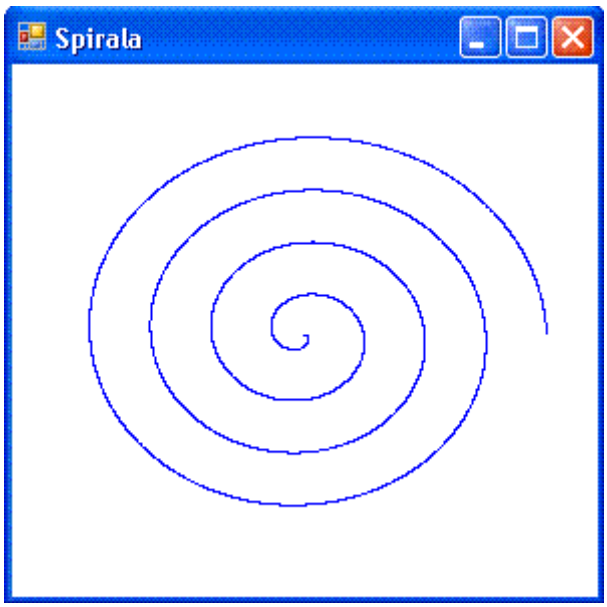
$y=a*t*\sin(t)$

Pentru a desena 4 spirale, t trebuie sa ia valori intre 0 si 8 radiani .

Deschidem un nou proiect Windows Forms Application intitulat "spirala".
Completam procedura deschisa pe evenimentul paint al form-ului cu:

```
double t=0;
int x_v,y_v,x,y,x0,y0;
System::Drawing::Graphics^ Desen;
Desen = this->CreateGraphics();
System::Drawing::Pen^ Creion_albastru =
gcnew System::Drawing::Pen(System::Drawing::Color::Blue);
Desen->Clear(System::Drawing::Color(this->BackColor));
x0=(this->Width-10)/2;
y0=(this->Height-50)/2;
x_v=x0+x0*t/30*System::Math::Cos(t);
y_v=y0+10+y0*t/30*System::Math::Sin(t);
for (t=0; t<=8*System::Math::PI; t+=0.001){
x=x0+x0*t/30*System::Math::Cos(t);
y=y0+10+y0*t/30*System::Math::Sin(t);
Desen->DrawLine(Creion_albastru, x_v, y_v, x, y);
x_v=x;
y_v=y;
}
delete Desen;
delete Creion_albastru;
```

- a=1/30 constanta care da departarea intre brate
- xo si yo realizeaza scalarea imaginii pe centrul ecranului, imaginea fiind amplasata aproximativ pe centrul form-ului, dupa cum se poate vedea si in imaginea de jos:



Pentru a controla numarul de spire plasam un obiect NumericUpDown si plasam urmatoarea procedura pe evenimentul Value Changed cu:

```
        double t=0,n;
        int x_v,y_v,x,y,x0,y0;
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_rosu =
        gcnew System::Drawing::Pen(System::Drawing::Color::Red);
        Desen->Clear(System::Drawing::Color(this->BackColor));
        x0=(this->Width-10)/2;
        y0=(this->Height-50)/2;
        n=System::Convert::ToDouble(this->numericUpDown1->Value);
        x_v=x0+x0*t/30*System::Math::Cos(t);
        y_v=y0+10+y0*t/30*System::Math::Sin(t);
        for (t=0; t<=n*System::Math::PI; t+=0.001){
        x=x0+x0*t/30*System::Math::Cos(t);
        y=y0+10+y0*t/30*System::Math::Sin(t);
        Desen->DrawLine(Creion_rosu, x_v, y_v, x, y);
        x_v=x;
        y_v=y;
        }
        delete Desen;
        delete Creion_rosu;
```

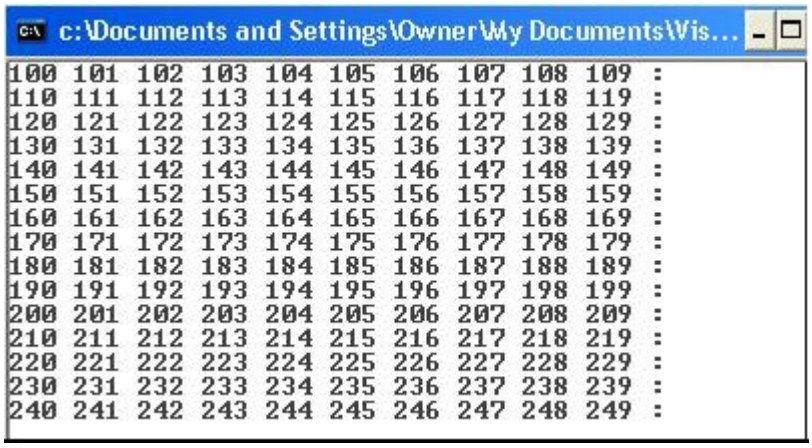
Imbricarea instructiunilor for

Instructiunile **for imbricate** ne dau posibilitatea sa rezolvam aplicatii in care sunt necesare repetitii in cadrul unor repetitii. In cazul masivelor de date, al matricilor este nevoie de astfel de instructiuni imbricate. O simpla afisare de numere pe ecran sub forma de linii si coloane necesita instructiuni for imbricate. Instructiunile for imbricate se folosesc de obicei atunci cand cunoastem numarul de iteratii pentru fiecare ciclu.

Sa afisam de exemplu numerele de la 100 la 150 pe 15 linii a cate 10 caractere.

```
// Program scris in C++ Visual Studio 2005 de tipul:CLR console application
// Se utilizeaza spatiul de nume System::
// programul afiseaza 500 de cifre aleatoare intre 0-9 despartite prin spatiu

#include "stdafx.h"

using namespace System;

int main(void)
{
        int k=100;
        for (int j=1; j<=15 ; j++){
                for ( int i=1; i<=10; i++)
                        Console::Write(k++ + " ");
        Console::WriteLine(":");
        }
        Console::ReadLine();
        return 0;
}
```

Dupa rularea programului in "Command Prompt" se afiseaza:



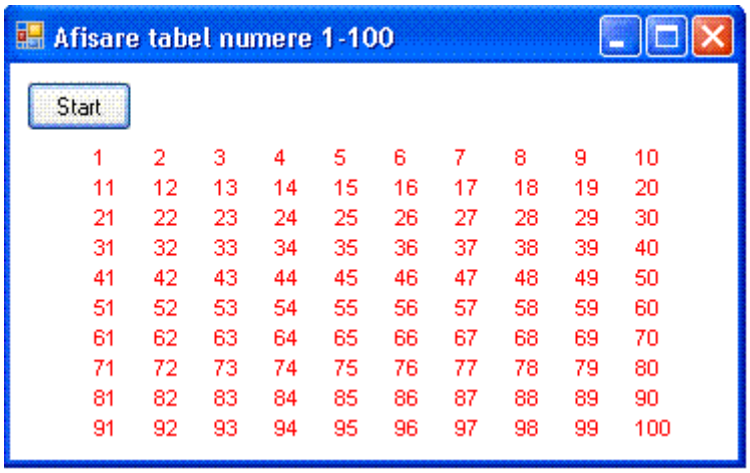
Sa afisam acum primele 100 de numere naturale, cate 10 pe un rand, folosind modul grafic.

Deschidem un nou proiect Windows Forms Application intitulat "tabel_numere" si plasam un Buton "Start" cu numele button1.

Completam procedura deschisa pe evenimentul click al butonului1 cu:

```
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::SolidBrush^ Pensula;
        Pensula=gcnew System::Drawing::SolidBrush(System::Drawing::Color::Red);
        System::Drawing::Font^ font_arial;
        font_arial=gcnew System::Drawing::Font("Arial",8);
        for (int j=1; j<=10 ; j++ ){
                for ( int i=1; i<=10; i++)
                        Desen-
>DrawString(System::Convert::ToString(i+10*(j-1)),font_arial,Pensula,10+i*30,25
+j*15);
                }
```

Dupa rularea programului, obtinem:



Structuri repetitive realizate cu timere

Utilizarea timerelor

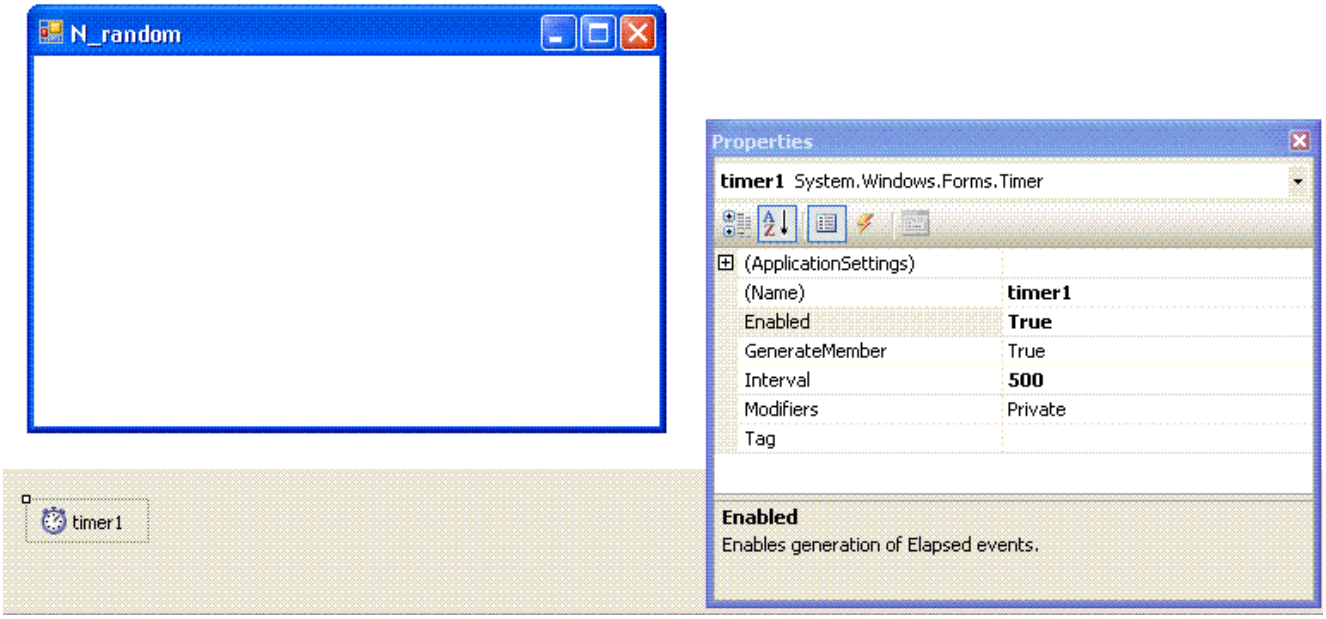
Reluand aplicatia care deseneaza figuri aleatoare si dorind sa executam operatia intr-o bucla infinita adica sa afisam un anumit numar de figuri dupa care sa stergerm ecranul si sa redesenam alte figuri, s-ar parea ca solutia ar fi sa imbricam procedura scrisa pe evenimentul paint intr- bucla infinita de forma:

```
        int i=0;
        bool repeta=true;
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_rosu;
        Creion_rosu=gcnew System::Drawing::Pen(System::Drawing::Color::Red);
        System::Drawing::Pen^ Creion_albastru;
        Creion_albastru=gcnew System::Drawing::Pen
( System::Drawing::Color::Blue);
        System::Random^ n = gcnew System::Random();
        while (repeta=true){
         Desen->Clear(System::Drawing::Color(this->BackColor));
         i=0;
          for ( int i=1; i<=100; i++){
                Desen->DrawRectangle(Creion_albastru,n->Next(this->Width),n-
>Next(this->Height),n->Next(75),n->Next(50));
                Desen->DrawEllipse(Creion_rosu,n->Next(this->Width),n-
>Next(this->Height),n->Next(50),n->Next(75));
          }
        }
        delete Creion_rosu;
        delete Creion_albastru;
        delete Desen;
        delete n;
```

Daca generam un nou proiect cu numele bucla_random si completam procedura generata pe evenimentul paint cu grupul de instructiuni de sus aplicatia ruleaza dar dupa un timp este blocata de sistemul de operare care sesizeaza buclarea infinita.

Solutia nu este acceptabila, asa ca pentru a realiza astfel de programe ce ruleaza in bucla infinita vom alege o alta abordare. Vom folosi timere pentru a rula intr-o bucla infinita aplicatia de sus vom folosi timere. Timerele sunt obiecte care se activeaza la intervale prestabilite de timp. Pe evenimentul Tick (lansat la scurgerea unui anumit interval de timp) putem atasa secventa de program pentru a fi rulata.

;Deschidem un nou proiect Windows Forms Application intitulat "n_random". Vom plasa un obiect de tipul "Timer" numit timer1.



Setam proprietatea "Enabled" la "True" iar proprietatea "Interval" la 500. Completam procedura deschisa pe evenimentul "Tick" al butonului timer1 cu:

```
        int i=0;
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_rosu;
        Creion_rosu=gcnew System::Drawing::Pen(System::Drawing::Color::Red);
        System::Drawing::Pen^ Creion_albastru;
        Creion_albastru=gcnew
System::Drawing::Pen( System::Drawing::Color::Blue);
        System::Random^ n = gcnew System::Random();
        Desen->Clear(System::Drawing::Color(this->BackColor));
        i=0;
        for ( int i=1; i<=100; i++){
         Desen->DrawRectangle( Creion_albastru,n->Next(this->Width),n-
>Next(this->Height),n->Next(75),n->Next(50));
         Desen->DrawEllipse( Creion_rosu,n->Next(this->Width),n->Next(this-
>Height),n->Next(50),n->Next(75));
        }
        delete Creion_rosu;
        delete Creion_albastru;
        delete Desen;
        delete n;
        }
```

Combinarea instructiunilor repetitive

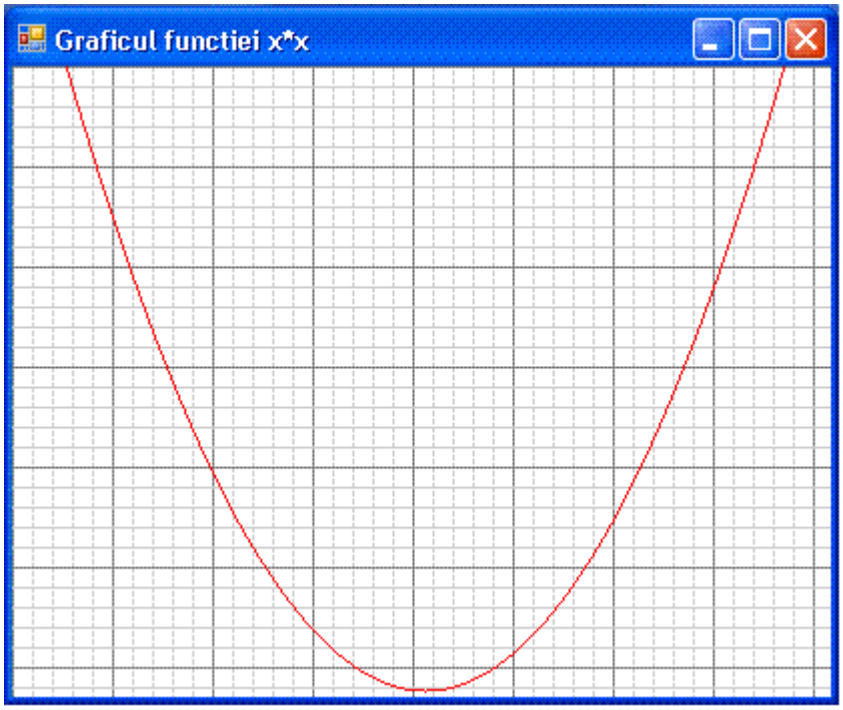
Folosindu-ne de instructiunea for si de programul care deseneaza un grid mic suprapus cu un grid mare, sa trasam graficul functiei x patrat. Tinand cont ca fereastra are latimea sa zicem lat si inaltimea h aproximativ egala cu latimea lat, am putea sa dam valori lui x incepand de la valoarea -lat/2 la valoarea lat/2. Valoarea maxima a lui y=x*x va fi practic (lat/2)*(lat/2)=lat*lat/4. Pentru a putea vizualiza toate valorile lui y intr-o fereastra aproximativ patrata (lat=h), pe domeniul -lat/2... lat/2 va trebui sa scalam valoarea lui y adica sa o impartim cu lat/4 pentru a se afisa toate valorile lui y iar imaginea graficului sa fie ajustata la dimensiunea form-ului.

Deschidem un nou proiect Windows Forms Application intitulat "grafic_x_p"

Completam procedura deschisa pe evenimentul paint al form-ului cu:

```
        int i=0,y=0,y_v=0,lat,h;
        double x;
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_gri_d;
        Creion_gri_d=gcnew
System::Drawing::Pen( System::Drawing::Color::LightGray);
        System::Drawing::Pen^ Creion_gri;
        Creion_gri=gcnew System::Drawing::Pen(System::Drawing::Color::Gray);
        Desen->Clear(System::Drawing::Color(this->BackColor));
        System::Drawing::Pen^ Creion_rosu;
        Creion_rosu=gcnew System::Drawing::Pen(System::Drawing::Color::Red);
        //Grid linii verticale
        for (i=0; i<=this->Width; i+=10){
                if (i%50==0)
                        Desen->DrawLine(Creion_gri, i, 0, i,this->Height);
                else
                        Desen->DrawLine(Creion_gri_d, i, 0, i,this->Height);
        }
        //Grid  linii orizontale

        for (i=0; i<=this->Height; i+=10){
                if (i%50==0)
                        Desen->DrawLine(Creion_gri, 0, i,this->Width,i);
                else
                        Desen->DrawLine(Creion_gri_d, 0, i,this->Width,i);
        }
        // Trasare grafic x*x
        h=this->Height-37;
        i=0;
        lat=this->Width-5;
        for(x=-lat/2; x<=lat/2; x++) {
                y=h-x*x/(lat/4);
                Desen->DrawLine(Creion_rosu, i-1,y_v, i, y);
                y_v=y;
                i+=1;
        }
        delete Desen;
```

Dupa rularea programului obtinem:



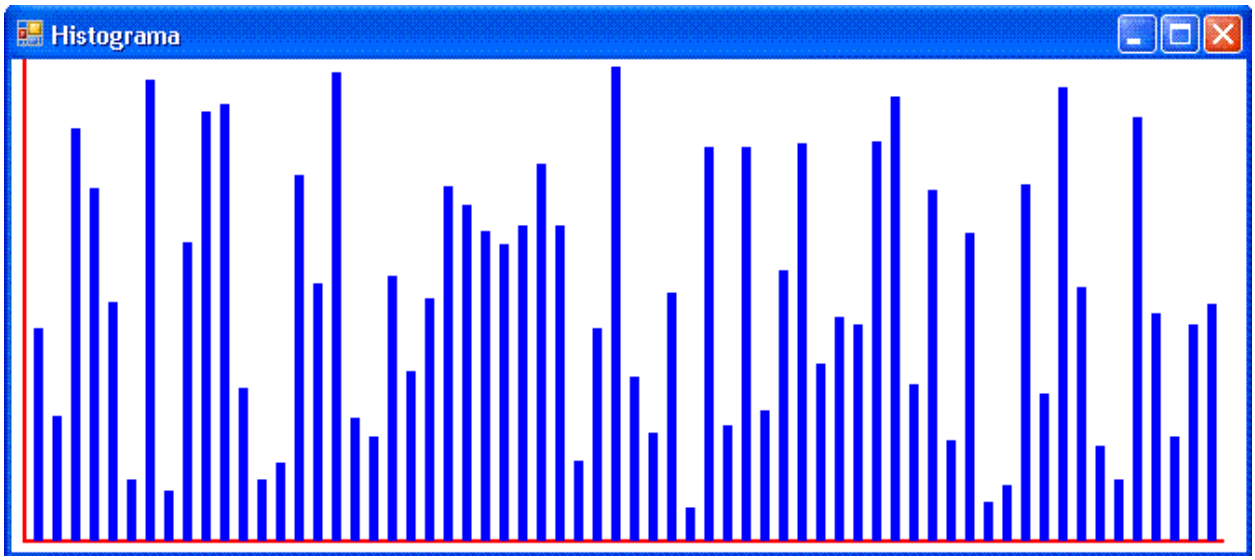
De multe ori e nevoie sa afisam pe un grafic valorile mai multor parametrii sau componente. La analiza spectrala de exemplu e necesara afisarea continutului de diferite substante dintr-un anumit compus. Un semnal electric poate contine o serie de armonici adica semnale sinusoidale de diferite frecvente si amplitudini. Un spectrometru poate afisa sub forma de histograma aceste armonici. Ne propunem sa folosim generatorul de numere aleatoare pentru a simula o histograma. Sa presupunem ca analiza spectrala se face la intervale de 1 secunda. Pentru a afisa o astfel de histograma vom folosi un timer setat la intervalul de 1 secunda.

Deschidem un nou proiect Windows Forms Application intitulat "histograma"

Vom plasa un obiect de tipul "Timer" numit timer1 setal cu valoarea intervalului la 1000. Completam procedura deschisa pe evenimentul "Tick" al butonului timer1 cu:

```
   int i=0;
   float w_r=2,w_a=5;
   System::Drawing::Graphics^ Desen;
   Desen = this->CreateGraphics();
   System::Drawing::Pen^ Creion_rosu;
   Creion_rosu=gcnew System::Drawing::Pen(System::Drawing::Color::Red,w_r);
   System::Drawing::Pen^ Creion_albastru;
              Creion_albastru=gcnew
System::Drawing::Pen( System::Drawing::Color::Blue,w_a);
   System::Random^ n = gcnew System::Random();
   Desen->Clear(System::Drawing::Color(this->BackColor));
   Desen->DrawLine( Creion_rosu,7,0,7,this->Height-40);
   Desen->DrawLine( Creion_rosu,6,this->Height-40,this->Width-20,this-
>Height-40);
   for ( int i=14; i<=this->Width-20; i+=10){
        Desen->DrawLine( Creion_albastru,i,this->Height-40,i,n->Next(this-
>Height-50));
   }
   delete Creion_rosu;
   delete Creion_albastru;
   delete Desen;
   delete n;
```

Dupa rularea programului obtinem:



Functii si proceduri

Functiile si procedurile sunt elementele de baza ale unui program scris în C++. Orice program, de orice dimensiune, contine una sau mai multe functii sau proceduri. Functiile si procedurile sunt de fapt un grup de instructiuni care impreuna executa o anumita operatie.

Exista cazuri cand trebuie sa executam de mai multe ori o anumita operatie, operatia fiind compusa dintr-un grup de instructiuni. O solutie ar fi inserarea grupului de instructiuni in program ori de cate ori e nevoie. In acest caz programele devin lungi si greoaie. Solutia eleganta e reprezentata de utilizarea functiilor si procedurilor. In majoritatea cazurilor spre grupul de instructiuni se transmit anumiti parametri, parametri necesari in executarea operatiei definita de grupul de instructiuni. In urma rularii grupului de instructiuni, de cele mai multe ori rezulta valori reprezentand rezultatul rularii instructiunilor din grupul de instructiuni. In cazul cand in urma rularii grupul de instructiuni returneaza valori, numim acest grup de instructiuni, **functie** . In cazul cand in urma rularii grupul de instructiuni nu se returneaza valori, numim acest grup de instructiuni, **procedura** . Procedurile se intalnesc de obicei in cazurile cand avem operatii de afisare. Grupul de instructiuni folosit in general pentru diverse afisari de date primeste date de intrare insa in urma prelucrarii lor se executa afisarea datelor prelucrate fara ca acestea sa intoarca nici un fel de valori. Majoritatea aplicatiilor scrise in "Windows Forms Aplication" au necesitat completarea unor proceduri atasate unui eveniment. Prcatic o procedura este o functie care nu intoarce valori. Vom numi atunci generic toate grupurile de instructiuni care impreuna executa anumite operatii, **functii** . Pana acum am scris programe cu o singura functie si anume functia **main**. In continuare vom utiliza functii pentru a realiza programe mai compacte si mai usor de controlat. Functiile ofera un mod convenabil de încapsulare a anumitor calcule într-o cutie neagra care poate fi utilizata apoi fara a avea grija continutului ei. Functiile sunt într-adevar singurul mod de a face fata complexitatii programelor mari, permitand desfacerea programelor mari în module mai mici, dand utilizatorului posibilitatea de a dezvolta programe, reutilizand codul scris de alte persoane.

Limbajul C a fost conceput sa permita definirea de functii eficiente si usor de manuit. În general e bine sa concepem programe constituite din mai multe functii mici decat din putine functii de dimensiuni mari.

Cand utilizam functii tinem cont ca:

- Un program poate fi împartit în mai multe fisiere sursa în mod convenabil, iar fisierele sursa pot fi compilate separat.
- Un program C consta dintr-o secventa de definitii externe de functii si de date.
- În fiecare program trebuie sa existe o functie cu numele impus **main**.
- Orice program îsi începe executia cu functia main. Celelalte functii sunt apelate din interiorul functiei main.
- Unele dintre functiile apelate sunt definite în acelasi program, altele sunt continute într-o biblioteca de functii.

Definirea unei functii

O functie este formata din antet si corp astfel:

```
antet_functie
{
corpul_functiei

}
```

Daca revenim la prima aplicatie scrisa observam ca de fapt avem o singura functie si anume funcia **main**

```
// Primul program scris in C++ Visual Studio 2005 de tipul:CLR (Common Language
Runtime) console application

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

int main(void)
{
        cout <<" Primul program CLR console application\n\n Felicitari!!";
        cin.get();
        return 0;
}
```

Prima linie **int main(void)** este antetul functiei (nu este urmat de ;) constand dintr-un tip de date returnat, un nume al functiei si o lista cu argumente.

Corpul functiei se pune intre acolade si consta din declaratii de variabile locale, una sau mai multe instructiuni printre care si instructiuni de revenire . In cazul de sus instructiunea de revenire fiind return 0.

O functie poate fi deci descrisa mai general astfel:

```
tip_val_return nume_func (lista_declaratiilor_param_ formali)
{
    declaratii_variabile_locale
    instructiuni
    return valoare
}
```

Prima linie reprezinta antetul functiei, în care se indica: tipul

functiei, numele acesteia si lista declaratiilor parametrilor formali. La fel ca un operand sau o expresie, o functie are un tip, care este dat de tipul valorii returnate de functie în functia apelanta. Daca functia nu întoarce nici o valoare, în locul tip_vali_return se specifica void. In acest caz avem de-a face cu o procedura

Daca tip_val_return lipseste, se considera, implicit, ca acesta este int. Nume_functie este un identificator.

Lista_declaratiilor_param_formali (încadrata între paranteze rotunde) consta într-o lista (enumerare) care contine tipul si identificatorul fiecarui parametru de intrare, despartite prin virgula. Daca lista parametrilor formali este vida, în antet, dupa numele functiei, apar doar parantezele ( ), sau (void).

Corpul functiei este un bloc, care implementeaza algoritmul de calcul folosit de catre functie. În corpul functiei apar (în orice ordine) declaratii pentru variabilele locale si instructiuni. Daca functia întoarce o valoare, se foloseste instructiunea return valoare. La executie, la întâlnirea acestei instructiuni, se revine în functia apelanta.

În limbajul C/C++ se utilizeaza declaratii si definitii de functii.

Declaratia contine antetul functiei si informeaza compilatorul asupra tipului, numelui functiei si a listei parametrilor formali (în care se poate indica doar tipul parametrilor formali, nu si numele acestora).

Declaratiile de functii se numesc prototipuri, si sunt constituite din antetul functiei, din care pot lipsi numele parametrilor formali.

Definitia contine antetul functiei si corpul acesteia. Nu este admisa definirea unei functii în corpul altei functii.

Prima aplicatie scrisa ar putea fi despartita in doua functii astfel:

```
// Program care foloseste functii
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application

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

// Incepe definirea functiei afiseaza

void afis(void)
{
	cout <<" \n\n\tPrimul program ce utilizeaza functii.\n\n";
}
int main(void)
{
	afis(); // se apeleaza functia afis
	cin.get();
	return 0;
}
```

Functia **afis** este definita prima astfel : void afis(void) . Primul void inseamna ca nu returneaza o valoare, iar afis(void) inseamna ca functia afis nu are nevoie de parametri cand se apeleaza. Corpul functiei afis are o singura instructiune si anume cout. Nu are instructiune de revenire return deoarece nu intoarce nici o valoare.

Tinand cont ca functia afis() nu returneaza nici un parametru o putem numi procedura.

Prototipul unei functii

De obicei un program incepe cu functia main(). In aplicatia de sus este definita prima data procedura afis(), dupa care apare functia main(). Daca schimbam locul lor programul devine:

```
// Primul program care foloseste functii
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application

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

int main(void)
{
	afis(); // se apeleaza functia afis
	cin.get();
	return 0;
}
// Incepe definirea functiei afiseaza

void afis(void)
{
	cout <<" \n\n\tPrimul program ce utilizeaza functii.\n\n";
}
```

Incercand sa compilam aplicatia de sus vom obtine o eroare deoarece in functia main() se apeleaza functia afis(), functie care inca nu a fost definita. Solutia este sa declaram functia inaintea functiei main(), dupa care putem pune functia main() urmata de procedura afis(). Declararea functiilor si procedurilor se realizeaza prin utilizarea prototipurilor de functii.

Aplicatia anterioara va rula cu conditia sa scriem inaintea functiei main() prototipul.

```
// Programul foloseste functii si prototipuri functii
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application

#include "stdafx.h"
#include < iostream >
using namespace std;
void afis(void);// prototipul functiei
int main(void)
{
	afis(); // se apeleaza functia afis
	cin.get();
	return 0;
}
// Incepe definirea functiei afiseaza
```

```
void afis(void)
{
	cout <<" \n\n\tProgramul utilizeaza functii si prototipuri.\n\n";
}
```

Prin aceasta metoda reusim sa scriem programe mai "clare" in sensul ca la inceput se trec in revista toate functiile prin declararea prototipurilor, urmeaza functia main(), dupa care urmeaza definitiile functiilor declarate prin prototip

Proceduri fara parametri

In Windows Forms Application functiile se definesc la fel. Vom incerca sa desenam mai multe cercuri concentrice distantate la 10 pixeli. Numarul cercurilor depinde de latimea form-ului curent, astfel la fiecare redimensionare a ferestrei sa se traseze in alt numar de cercuri concentrice. Daca punem o singra procedura de desanare pe evenimentu "resize" desenarea se va face numai daca redimensionam fereastra curenta, astfel la pornirea programului nu se afiseaza nimic. Procedura trebuie lansata si pe evenimentul paint.

Vom scrie o procedura numita desenare pe care o vom lansa din procedurile create pe cele doua evenimente.

Procedura deseneaza arata astfel:

```
void deseneaza(void){
		int i=0;
		System::Drawing::Graphics^ Desen;
		Desen = this->CreateGraphics();
		System::Drawing::Pen^ Creion_rosu;
		Creion_rosu=gcnew
System::Drawing::Pen( System::Drawing::Color::Red);
		Desen->Clear(System::Drawing::Color(this->BackColor));
		i=0;
		for ( int i=1; i<=this->Width/4; i+=10){
			Desen->DrawEllipse( Creion_rosu, this->Width/2-i, this-
>Height/2-30-i,2*i,2*i);
		}
		delete Creion_rosu;
		delete Desen;
		}
```

Deschidem un nou proiect numit "f_cercuri". Procedura de sus va fi adaugata procedurilor generate la deschiderea proiectului.

Procedura va fi amplasata in zona #pragma region unde sunt amplasate si procedurile de initializare ale componentelor

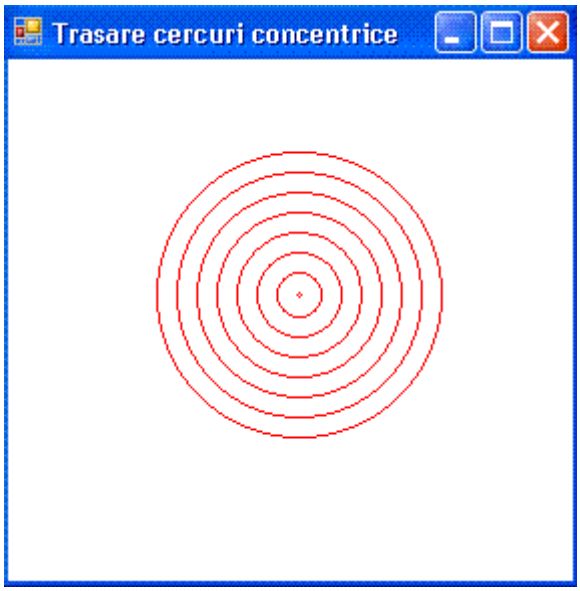
Creem proceduri pe evenimentele paint si resize.

Procedurile create pe evenimentele paint si resize vor contine apelul procedurii de desenare adica deseneaza();

Ultima parte a codului afisat cu "view code" va arata astfel:

```
        void deseneaza(void){
                int i=0;
                System::Drawing::Graphics^ Desen;
                Desen = this->CreateGraphics();
                System::Drawing::Pen^ Creion_rosu;
                Creion_rosu=gcnew
System::Drawing::Pen( System::Drawing::Color::Red);
                Desen->Clear(System::Drawing::Color(this->BackColor));
                i=0;
                for ( int i=1; i<=this->Width/4; i+=10){
                        Desen->DrawEllipse( Creion_rosu, this->Width/2-i, this-
>Height/2-30-i,2*i,2*i);
                }
                delete Creion_rosu;
                delete Desen;
                }

#pragma endregion

        private: System::Void Form1_Paint(System::Object^  sender,
System::Windows::Forms::PaintEventArgs^  e) {
                                deseneaza();
                        }
        private: System::Void Form1_Resize(System::Object^  sender,
System::EventArgs^  e) {
                                deseneaza();
                        }
        };
}
```

Dupa rularea aplicatiei se afiseaza imediat imaginea de jos iar la redimensionarea ferestrei se redeseneasza cercurile.



In multe cazuri e necesar sa efectuam aceleasi operatii pe mai multe

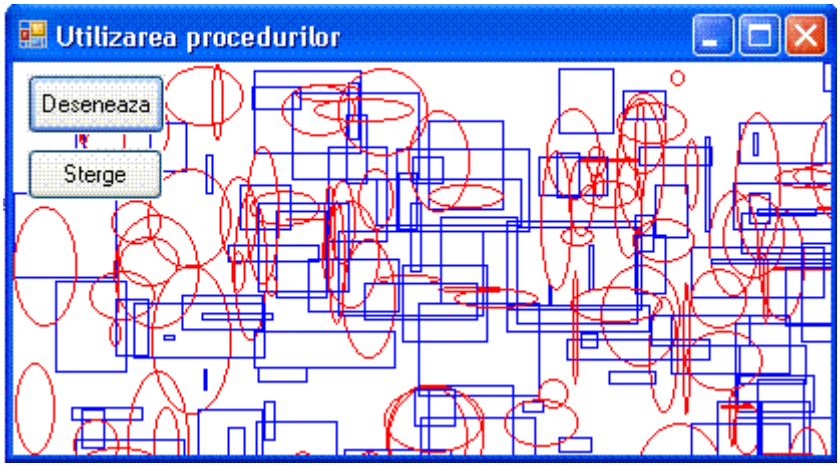
evenimente. Spre exemplu in cazul in care afisam ceva intr-o fereastra dupa apasarea unui buton, am dori sa reafisam continutul ferestrei si dupa operatia de redimensionare a ferestrei. Vom defini deci o procedura numita de exemplu deseneaza() pe care o apelam din porocedurile lansate pe evenimentele dorite. Sa realizam o aplicatie in care "Form-ul" principal contine doua butoane: "Deseneaza" si "Sterge". Procedura deseneaza() trebuie sa se declanseze atat pe evenimentul Click al butonului "Deseneaza" cat si pe evenimentul resize al form-ului. Procedura "Sterge" trebuiesa se declanseze atat pe evenimentul Click al butonului "Sterge" cat si pe evenimentul resize al form-ului.

Vom crea un nou proiect numit: **functii_vo** in Windows Forms Application . Plasam doua butoane si anume button1 pentru "Deseneaza" si button2 pentru "Sterge". Definim procedurile deseneaza() si sterge() si le apelam pe evenimentele mai sus amintite.

Ultima parte a codului afisat cu "view code" va arata astfel:

```
void deseneaza(void){
        int i=0;
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        System::Drawing::Pen^ Creion_rosu;
        Creion_rosu=gcnew System::Drawing::Pen(System::Drawing::Color::Red);
        System::Drawing::Pen^ Creion_albastru;
        Creion_albastru=  System::Drawing::Pen(System::Drawing::Color::Blue);
        System::Random^ n = gcnew System::Random();
        for ( int i=1; i<=100; i++){
         Desen->DrawRectangle( Creion_albastru,n->Next(this->Width),n-
>Next(this->Height),n->Next(75),n->Next(50));
         Desen->DrawEllipse( Creion_rosu, n->Next(this->Width), n->Next(this-
>Height), n->Next(50),n->Next(75));
        }
        delete Creion_rosu;
        delete Creion_albastru;
        delete Desen;
}
void sterge(void){
        System::Drawing::Graphics^ Desen;
        Desen = this->CreateGraphics();
        Desen->Clear(System::Drawing::Color(this->BackColor));
}
#pragma endregion
        private: System::Void button1_Click(System::Object^  sender,
System::EventArgs^  e) {
                 sterge();
                 deseneaza();
        }
        private: System::Void button2_Click(System::Object^  sender,
System::EventArgs^  e) {
                 sterge();
        }
        private: System::Void Form1_ResizeEnd(System::Object^  sender,
System::EventArgs^  e) {
                 sterge();
                 deseneaza();
        }
};
}
```

Dupa lansarea aplicatiei si apasarea butonului "Deseneaza" se afiseaza imaginea:



Sa incercam sa afisam forma de unda a semnalelor modulate in amplitudine. La transmiterea undelor radio se foloseste modulatia in amplitudine, in frecventa sau in faza pentru transmiterea la mare distanta a undelor radio. In principiu daca vrem sa transmitem la distanta un semnal electric de joasa frecventa trebuie sa ne folosim de un semnal de inalta frecventa (numit purtatoare) peste care sa suprapunem semnalul de joasa frecventa numit (modulatoare) . In cazul cand vrem sa transmitem un semnal sinusoidal, acesta va modula un semnal purtator tot de forma sinusoidala dar de frecventa mult mai mare. Sa afisam atunci un semnal sinusoidal modulat tot cu o sinusoida. Trebuie sa afisam deci functia sin(mx)*sin(sp) unde xm este frecventa undei modulatoare iar xp este frecventa undei purtatoare. De obicei xp>>xm.

Generam un nou proiect in Windows Forms Application numit **modulatie** . Plasam doua obiecte de tip NumericUpDown care vor permite introducerea numarului de radiani pentru modulatoare respectiv factor de multiplicare pentru purtatoare in raport cu frecventa modulatoarei, cu numele numericUpDown1 respectiv numericUpDown2.

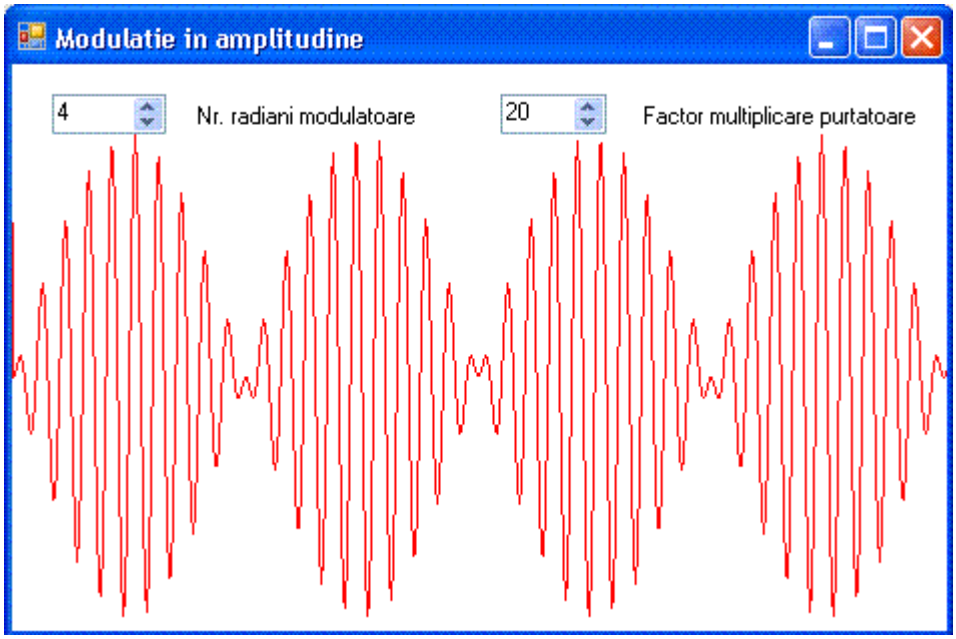
Cu click dreapta pe form -- aleg optiiunea View Code -- inserez urmatoarea procedura:

```
void sin_mod(void){
      int i=0,y=0,y_v=0,lat,factor_s;
      double x,rad,xm,f;
   System::Drawing::Graphics^ Desen;
      Desen = this->CreateGraphics();
      System::Drawing::Pen^ Creion_blu;
      Creion_blu=gcnew System::Drawing::Pen( System::Drawing::Color::Red);
      Desen->Clear(System::Drawing::Color(this->BackColor));
      factor_s=this->Height/2-37;
      rad=System::Convert::ToDouble(this->numericUpDown1->Value);
      f=System::Convert::ToDouble(this->numericUpDown2->Value);
      lat=this->Width;
   do {
        x=i*rad*System::Math::PI/(lat-10);
        xm=f*x;
        y=35+factor_s*(1-System::Math::Sin(xm)*System::Math::Sin(x));
        Desen->DrawLine(Creion_blu,i-1,y_v, i, y);
        y_v=y;
        i+=1;
```

```
        }
    while (i<=lat);
    delete Creion_blu;
    delete Desen;
}
```

Procedura **sin_mod** trebuie sa se apeleze atat la evenimentul paiant cat si pe evenimentele ValueChanged ale celor doua butoane. Vom pune deci pe aceste evenimente apelul procedurii sin_mod();

Dupa rularea aplicatiei obtinem:



Din cele doua butoane putem schimba frecventele celor doua unde si acestea se redeseneaza automat.

Proceduri cu argument (parametric)

Desi s-ar parea ca aplicatia anterioara primeste parametri in sensul ca apeleaza aceeasi procedura sin_mod() dar afiseaza imagini diferite in functie de valorile butoanelor NumericUpDown nu avem de-a face cu o procedura cu argument (parametri). Argumentele (parametrii) pot fi transmisi spre proceduri in diverse moduri. Cel mai simplu mod de transmitere a parametrilor spre proceduri este transmiterea argumentelor (parametrilor) prin valoare.

- **Transmiterea argumentelor (parametrilor) spre proceduri, prin valoare.**

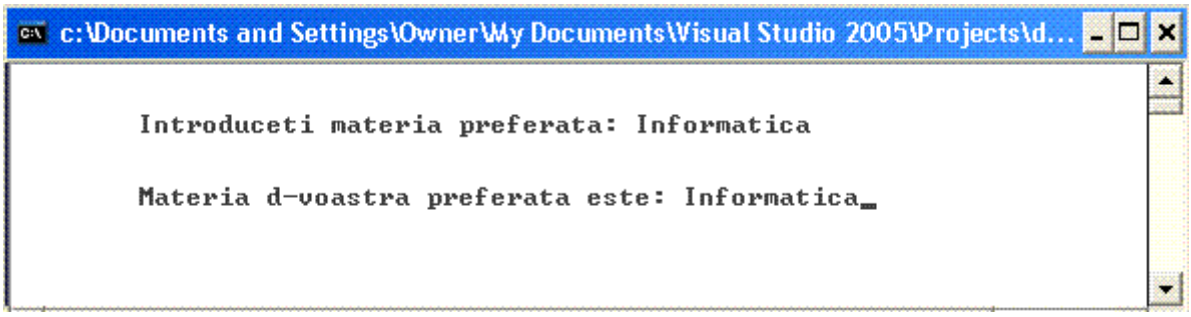
Vom folosi o functie numita **afis_mes** pentru a afisa un mesaj. Textul pentru afisat este transmis ca argument procedurii.

```
// Programul foloseste procedura afis_mes careia i se transmit parametrii prin
valoare
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application

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

using namespace std;
void afis_mes(string);// prototipul
int main(void)
{
        string str;
        cout << " \n\n\tIntroduceti materia preferata: ";
        cin >> str;
        afis_mes(str); // se apeleaza procedura afis
        cin.ignore();
        cin.get();
        return 0;
}
// Incepe definirea procedurii afis_mes
void afis_mes(string s)
{
        cout <<" \n\n\tMateria d-voastra preferata este: " << s;
}
```

Atat antetul cat si prototipul functiei au un singur argument. Argumentul prototipului contine numai tipul argumentului.

Cand am definit procedura **afis_mes** am utilizat atat tipul argumentului (string) cat si parametrul s . Aceast parametru se numeste parametru formal sau simulat si este utilizat pentru definirea interna procedurii. Acesta nu are semnificatie pentru programul principal. Acest parametru formal va fi inlocuit de argumentul transmis la apelarea procedurii.

Dupa rularea programului si introducerea textulii "Informatica" Fereastra "Command" arata astfel:



Am putea modifica functia **afis_mes** astfel incat sa o putem folosi pentru a afisa orce text care a fost definit in prealabil.

```
// Programul foloseste procedura afis_mes careia i se transmit parametrii prin
valoare
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application

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

using namespace std;
void afis_mes(string);// prototipul
int main(void)
{
        string str;
        string mesaj=" \n\n\tMateria d-voastra preferata este: ";
        cout << " \n\n\tIntroduceti materia preferata: ";
        cin >> str;
        afis_mes(mesaj);// se apeleaza procedura afis_mes
        afis_mes(str); // se apeleaza procedura afis_mes
        cin.ignore();
        cin.get();
        return 0;
}
// Incepe definirea procedurii afis_mes
void afis_mes(string s)
{
        cout << s;
}
```

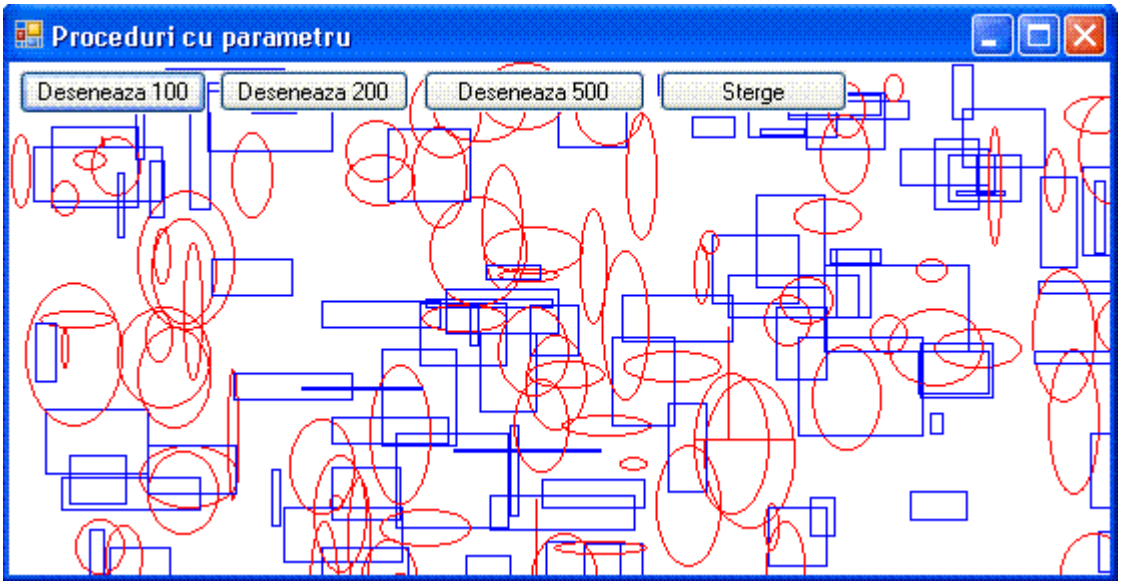
Vom realiza in continuare o aplicatie in care se apeleaza proceduri carora li se transmit parametri prin valoare. Reluam aplicatia cu desenarea figurilor aleatoare dar plasam niste butoane cu care alegem numarul de figuri desenate pe ecran.

Vom crea un nou proiect numit: **functii_v1** in Windows Forms Application . Plasam patru butoane si anume: button1 pentru "Deseneaza 100", button2 pentru "Deseneaza 200", button3 pentru "Deseneaza 500" si button4 pentru "Sterge". Definim procedurile deseneaza() si sterge() si le apelam pe evenimentele butoanelor definite.

Ultima parte a codului afisat cu "view code" va arata astfel:

```
void deseneaza(int nr){
    int i=0;
    System::Drawing::Graphics^ Desen;
    Desen = this->CreateGraphics();
    System::Drawing::Pen^ Creion_rosu;
    Creion_rosu=gcnew System::Drawing::Pen(System::Drawing::Color::Red);
    System::Drawing::Pen^ Creion_albastru;
    Creion_albastru=gcnew System::Drawing::Pen(System::Drawing::Color::Blue);
    System::Random^ n = gcnew System::Random();
    for ( int i=1; i<=nr; i++){
       Desen->DrawRectangle( Creion_albastru,n->Next(this->Width),n->Next(this-
>Height),n->Next(75),n->Next(50));
       Desen->DrawEllipse( Creion_rosu,n->Next(this->Width),n->Next(this-
>Height),n->Next(50),n->Next(75));
     }
delete Creion_rosu;
delete Creion_albastru;
delete Desen;
}
void sterge(void){
       System::Drawing::Graphics^ Desen;
       Desen = this->CreateGraphics();
       Desen->Clear(System::Drawing::Color(this->BackColor));

}
#pragma endregion
private: System::Void button1_Click(System::Object^  sender, System::EventArgs^
e) {
sterge();
       deseneaza(100);
}
private: System::Void button2_Click(System::Object^  sender, System::EventArgs^
e) {
       sterge();
       deseneaza(200);
}
private: System::Void button3_Click(System::Object^  sender, System::EventArgs^
e) {
       sterge();
       deseneaza(500);
}
private: System::Void button4_Click(System::Object^  sender, System::EventArgs^
e) {
       sterge();
}
private: System::Void Form1_ResizeEnd(System::Object^  sender,
System::EventArgs^  e) {
       sterge();
       deseneaza(10);
}
};
}
```

Dupa lansarea aplicatiei si apasarea butonului "Deseneaza 100 " se afiseaza imaginea:



- **Utilizarea argumentelor multiple.**

```
// Programul foloseste procedura afis_mes careia i se transmit parametrii
multiplii prin valoare
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application

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

using namespace std;
void afis_mes(string,string,string);// prototipul
int main(void)
{
        string univ,facult,sec;
        string mesaj=" \n\n\tUnde sunteti student ? ";
        string raspuns=" \n\n\tSunteti student la : ";
        afis_mes(mesaj,"","");// se apeleaza proc. afis_mes pentru afisare mesaj
        cout << " \n\n\tUniversitatea: ";
        cin >> univ;
        cin.ignore();
        cout << " \n\tFacultatea: ";
        cin >> facult;
        cin.ignore();
        cout << " \n\tSectia: ";
        cin >> sec;
        cin.ignore();
        afis_mes(raspuns,"","");// se apeleaza proc.afis_mes pentru afisare
raspuns
        afis_mes(univ,facult,sec); // se apeleaza proc. afis_mes  pentru afisare
univ,facult,sec
        cin.get();
        return 0;
```

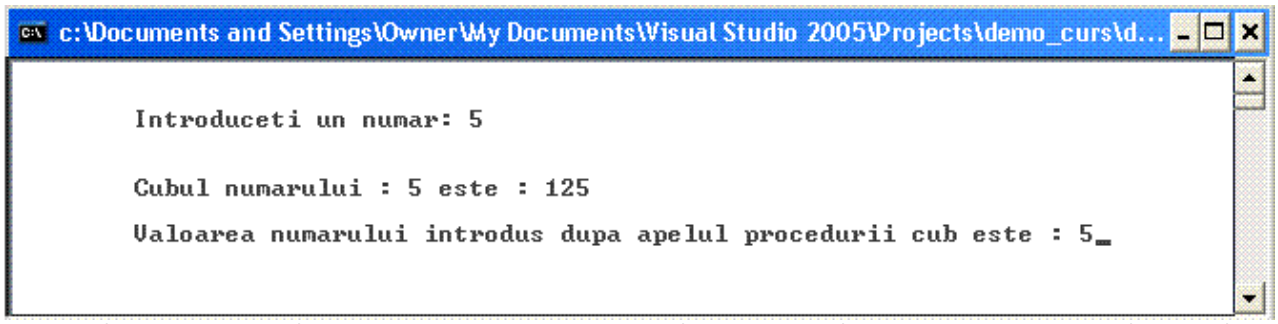
```
}
// Incepe definirea procedurii afis_mes
void afis_mes(string u, string f, string s)
{
	cout << u ; cout << "  "; cout << f; cout << "  "; cout << s;
}
```

- **Transmiterea argumentelor prin referinta.**

In cazul cand se doreste modificarea argumentelor in procedura apelanta transmiterea argumentelor (parametrilor) se face prin referinta.

Sa presupunem ca realizam o aplicatie care cere un numar de la tastatura si apeleaza o functie care inlocuieste numarul introdus cu cubul acestuia, dupa care afiseaza rezultatul. La revenirea din procedura valoarea numarului introdus ramane neschimbata.

```
// Programul foloseste procedura cub careia i se transmite un parametru.
// Dupa apelul procedurii se testeaza valoarea parametrului transmis
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application

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

using namespace std;
void cub(int);// prototipul
int main(void)
{
	int x;
	cout << " \n\n\tIntroduceti un numar: ";
	cin >> x;
	cub(x);
	cout << "\n\n\tValoarea numarului introdus dupa apelul functiei 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;
}
```



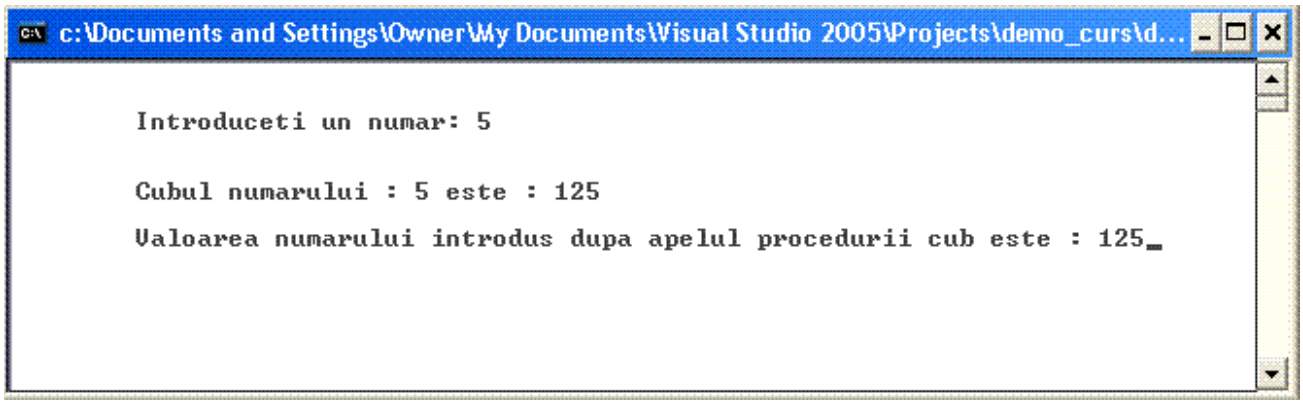
Chiar daca schimbam numele parametrului formal din procedura **cub** din nr in x , rezultatul este acelasi.

```
// Programul foloseste procedura cub careia i se transmite un parametru.
// Dupa apelul procedurii se testeaza valoarea parametrului transmis.
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application

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

using namespace std;
void cub(int);// prototipul
int main(void)
{
        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 x)
{
        cout << "\n\n\tCubul numarului : ";
        cout << x;
        cout << " este : ";
        x=x*x*x;
        cout << x;
}
```

Pentru a beneficia de o variabila definita in programul principal dar modificata de o procedura apelata se foloseste metoda transmiterii parametrilor prin referinta.

```
// Programul foloseste procedura cub careia i se transmite un parametriu prin
referinta
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application

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

using namespace std;
void cub(int&);// prototipul
int main(void)
{
        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;
}
```

De data aceasta dupa apelul procedurii valoare numarului introdus este modificata de catre procedura.



Singura diferenta este in modul de declarare al prototipului si antetului.

```
void cub(int&);// prototipul
void cub(int& nr) // definirea procedurii cub
```

Prin referinta se transmit numai acele valori care trebuiec modificate de catre procdura. Constantele nu se trimit prin referinta. Spre o procedura se

pot trimite atat valori prin referinta cat si prin valoare. Sa realizam acum o aplicatie care cere raza unui cerc si afiseaza aria lui. Se va sctie o procedura numita: aria_c spre care se vor trimite constanta pi,raza si aria. Aria se initializeaza cu 0 si va fi trimisa prin referinta pentru a putea fi afisata la intoarcerea din procedura.

```
// program pentru calculul ariei cercului
// se cere raza cercului si se afiseaza aria acestuia
// se foloseste procedura aria_c pentru a calcula aria cercului
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application

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

using namespace std;
void aria_c(float,float,float&);// prototipul
int main(void)
{
	float pi=3.1415,r,a=0;
	cout << " \n\n\tIntroduceti raza cercului: ";
	cin >> r;
	aria_c(pi,r,a);
	cout << "\n\n\tAria cercului de raza : ";
	cout << r;
	cout << " este : ";
	cout << a;
	cin.ignore();
	cin.get();
	return 0;
}
// Incepe definirea procedurii aria_c
void aria_c(float p,float raza, float& aria)
{
	aria=p*raza*raza;
}
```

Constanta pi poate fi trimisa si direct ca argument:

```
// program pentru calculul ariei cercului
// se cere raza cercului si se afiseaza aria acestuia
// se foloseste procedura aria_c pentru a calcula aria cercului
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application

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

using namespace std;
void aria_c(float,float,float&);// prototipul
int main(void)
{
	float r,a=0;
	cout << " \n\n\tIntroduceti raza cercului: ";
```

```
        cin >> r;
        aria_c(3.1415,r,a);
        cout << "\n\n\tAria cercului de raza : ";
        cout << r;
        cout << " este : ";
        cout << a;
        cin.ignore();
        cin.get();
        return 0;
}
// Incepe definirea procedurii aria_c
void aria_c(float p,float raza, float& aria)
{
        aria=p*raza*raza;
}
```

Functii

O procedura care returneaza valori, se numeste functie. Argumentele functiei transmit valori unei functii apelate. Valoarile returnate se poat folosi pentru a transmite valori functiei apelante.

- **Returnarea valorilor dintr-o functie.**

In programul precedent in care am folosit o procedura pentru a calcula aria unui cerc am folosit variabila a (aria cercului, pe care am initializat-o cu zero) transmisa prin referinta pentru a putea fi afisata la intoarcerea din procedura. Aplicatie este oarecum fortata pentru ca ar trebui folosita o functie spre care sa se transmita argumentele pi si raza si ea sa returneze o valoare adica aria cercului. Vom rescrie aplicatia, folosin de data aceasta o functie.

```
// program pentru calculul ariei cercului
// se cere raza cercului si se afiseaza aria acestuia
// se foloseste functia aria_c pentru a returna aria cercului
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application

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

using namespace std;
double aria_c(double,double);// prototipul functiei
int main(void)
{
        double r,a=0;
        cout << " \n\n\tIntroduceti raza cercului: ";
        cin >> r;
        a=aria_c(3.1415,r);
        cout << "\n\n\tAria cercului de raza : ";
        cout << r;
        cout << " este : ";
```

```
        cout << a;
        cin.ignore();
        cin.get();
        return 0;
}
// Incepe definirea functiei aria_c
double aria_c(double p,double raza)
{
        return p*raza*raza;
}
```

Sa calculam acum, sirul lui Fibonacci.
Sirul are urmatoarea structura: f(0)=0, f(1)=1, f(n)=f(n-1)+f(n-2), daca n>1

```
// Se defineste si se utilizeaza functia fib
// Programul calculeaza elenentul n din sirul lui Fibonacci
// adica:  f(0)=0, f(1)=1, f(n)=f(n-1)+f(n-2), daca n>1
#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
long int fib(long int n); // prototipul
int main(void)
{
        int nr;
        long int f;
        cout << "\n\n\tProgramul calculeaza elementul n din sirul lui
Fibonacci";
        cout << "\n\n\tIntroduceri n: ";
        cin >> nr;
        f=fib(nr);
        cout << "\n\n\tValoarea sirului pentru n=" << nr << " este:" << f <<
"\n";
        cin.ignore();
        cin.get();
        return 0;
}
long int fib(long int n)
{
        if (n==0) return 0;
        if (n==1) return 1;
        int i; long int a, b, c; a=0; b=1;
        for (i=2; i<=n+1; i++){
                a=b;
                b=c;
                c=a+b;
        }
        return c;
}
```

Sa afisam acum n elemente din sirul lui Fibonacci.
Sirul are urmatoarea structura: f(0)=0, f(1)=1, f(n)=f(n-1)+f(n-2), daca n>1

```
// Se defineste si se utilizeaza functia fib
// Programul afiseaza n elemente din sirul lui Fibonacci
// adica:  f(0)=0, f(1)=1, f(n)=f(n-1)+f(n-2), daca n>1
#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
long int fib(long int n); // prototipul
int main(void)
{
        int i,nr;
        long int f;
        cout << "\n\n\tProgramul afiseaza n elemente din sirul lui Fibonacci";
        cout << "\n\n\tIntroduceri n: ";
        cin >> nr;
        cout << "\n\n";
        for (i=0; i<=nr ;i++){
        f=fib(i);
        cout << "\tf(" << i << ")= " << f << "\n";
        }
        cin.ignore();
        cin.get();
        return 0;
}

long int fib(long int n)
{
        if (n==0) return 0;
        if (n==1) return 1;
        int i; long int a, b, c; a=0; b=1;
        for (i=2; i<=n+1; i++){
                a=b;
                b=c;
                c=a+b;
        }
        return c;
}
```

In aplicatiile realizate pana acum de tipul tipul:CLR console application unde am utilizat spatiul de nume System sau in aplicatii de tipul Windows Forms Application am utilizat de multe ori functii. Am calculat de exemplu lungimea unui cerc si am convertit raza in double cu functia **raza = System::Convert::ToDouble(raza_s);** . In acest caz am apelat defapt o functia **Convert::ToDouble(raza_s);** definita in spatiul de nume System, careia i-am transmis raza_s sub forma de string, iar functia intoarce raza sub forma double. Sa reluam aplicatia de sus si sa utilizam spatiul de nume System.

```
// program pentru calculul ariei cercului
// se cere raza cercului si se afiseaza aria acestuia
// se foloseste functia aria_c pentru a returna aria cercului
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application
// Se utilizeaza spatiul de nume System
```

```
#include "stdafx.h"
using namespace System;
double aria_c(double,double);// prototipul functiei

int main(void)
{
        double raza ,a;
        String^ raza_s;
        double pi= System::Math::PI;
        Console::Write( L"\n\n\tIntroduceti raza cercului:" );
        raza_s= Console::ReadLine();
        raza = System::Convert::ToDouble( raza_s );
        a=aria_c(pi,raza);
        Console::WriteLine( "\n\n\tAria cercului de raza : "+ raza +" este:
"+a);
        Console::ReadLine();
    return 0;
}
// Incepe definirea functiei aria_c
double aria_c(double p,double r)
{
        return p*r*r;
}
```

Am putea chiar sa definim constanta pi in interiorul functiei aria_c

```
// program pentru calculul ariei cercului
// se cere raza cercului si se afiseaza aria acestuia
// se foloseste functia aria_c pentru a returna aria cercului
// Programul este scris in C++ Visual Studio 2005 de tipul:CLR
// (Common Language Runtime) console application
// Se utilizeaza spatiul de nume System

#include "stdafx.h"
using namespace System;
double aria_c(double);// prototipul functiei

int main(void)
{
        double raza ,a;
        String^ raza_s;
        Console::Write( L"\n\n\tIntroduceti raza cercului:" );
        raza_s= Console::ReadLine();
        raza = System::Convert::ToDouble( raza_s );
        a=aria_c(raza);
        Console::WriteLine( "\n\n\tAria cercului de raza : "+ raza +" este:
"+a);
        Console::ReadLine();
    return 0;
}
// Incepe definirea functiei aria_c
double aria_c(double r)
{
        double pi= System::Math::PI;
        return pi*r*r;
}
```

- **Utilizarea functiilor in Windows Forms Application .**

In aplicatiile de tipul Windows Forms Application am utilizat de multe ori functii. La calculul valorii sin(), de exemplu am introdus linii de program de genul:

```
double rad2;
rad2=System::Convert::ToDouble(this->numericUpDown1->Value);
this->label3->Text =System::Convert::ToString(System::Math::Sin(rad2));
```

Sa realizam o aplicatie in Windows Forms Application care afiseaza valoarea coordonatelor stanga sus ale ferestrei principale a aplicatiei.

Deschidem un nou proiect Windows Forms Application intitulat **poz_wind** si plasam 4 obiecte de tip label numite label1..label4. Completam procedura deschisa pe evenimentul mov al Form1 cu :

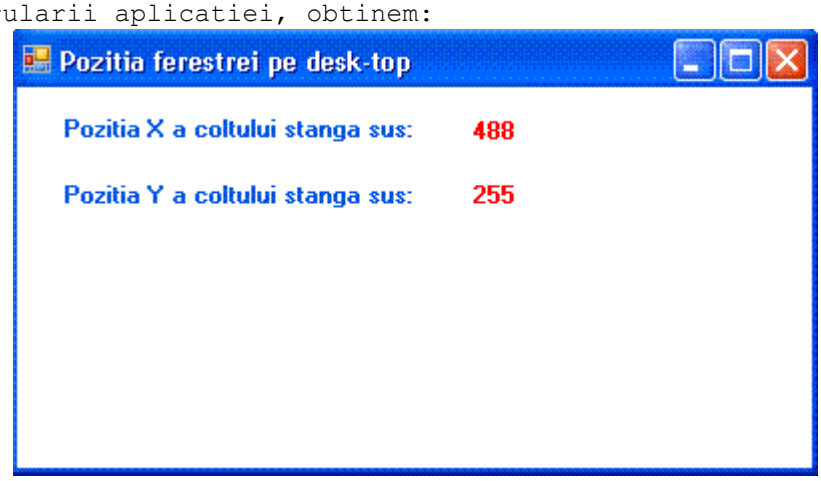
```
this->label3->Text= System::Convert::ToString(this->Location.X);
this->label4->Text= System::Convert::ToString(this->Location.Y);
```

Se apeleaza deci functia **System::Convert::ToString** cu argumentul(this->Location.X) respectiv (this->Location.Y)

Functia returneaza un sir de caractere ce reprezinta pozitia x respectiv y a coltului stanga sus a ferestei curente.

Valoare returnata de functie este atribuita atributului text al etichetelor label3 respectiv label4 pentru a fi afisate

In urma rularii aplicatiei, obtinem:



Schimband pozitia ferestrei curente se afiseaza in mod corespunzator pozitia ferestrei.

Sa realizam acum o aplicatie in Windows Forms Application care afiseaza pozitia curenta a mouse-ului pe desk-top.

Deschidem un nou proiect Windows Forms Application intitulat **mouse** si plasam 2 obiecte de tip label numite pozx si pozy.

Pentru a afisa pozitia x si y a mouse-ului vom face afisarea la intervale de 0.1 secunde va trebui sa plasam si un timer.

Completam procedura deschisa pe evenimentul Tick al timerului cu:

```
        this->pozx->Text = String::Concat(Control::MousePosition.X);
        this->pozy->Text = String::Concat(Control::MousePosition.Y);
```

Se apeleaza deci functia **System::Convert::ToString** dar cu argumentul(Control::MousePosition.X) respectiv (Control::MousePosition.Y)
In urma rularii aplicatiei, obtinem:

Pentru a afisa pozitia mouse-ului in cadrul ferestrei curente, deschidem un nou proiect Windows Forms Application intitulat **poz_mouse** asemanator cu aplicatia anterioara dar procedura deschisa pe evenimentul Tick al timer-uli devine:

```
        if((Control::MousePosition.X>=this->Location.X)
                &&(Control::MousePosition.X-this->Location.X<=this->Size.Width )
                &&(Control::MousePosition.Y>=this->Location.Y)
                &&(Control::MousePosition.Y-this->Location.Y<=this-
>Size.Height ))
        {
                this->pozx->Text=
System::Convert::ToString(Control::MousePosition.X-this->Location.X);
                this->pozy->Text=
System::Convert::ToString(Control::MousePosition.Y-this->Location.Y);
        }
        else{
                this->pozx->Text="Mouse-ul nu este in fereastra";
                this->pozy->Text="";
        }
```

Functii recursive

O functie este numita functie recursiva daca ea se autoapeleaza, fie direct (în definitia ei se face apel la ea însasi), fie indirect (prin apelul altor functii). Limbajele C/C++ dispun de mecanisme speciale care permit suspendarea executiei unei functii, salvarea datelor si reactivarea executiei la momentul potrivit. Pentru fiecare apel al functiei, parametrii si variabilele automatice se memoreaza pe stiva, având valori distincte. Orice apel al unei functii conduce la o revenire în functia respectiva, în punctul

urmator instructiunii de apel. La revenirea dintr-o functie, stiva este curatata (stiva revine la starea dinaintea apelului).

- **Exemple de functii recursive**

Unul dintre cele mai simple exemple de utilizare ale functiilor recursive este functia de calcul factorial. Programul urmator defineste si utilizeaza functia recursiva factorial.

```
// Se utilizeaza functii recursive
// Programul calculeaza factorialul unui numar
#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
int fact(int n); // prototipul
int main(void)
{
        int nr, f;
        cout << "\n\n\tProgramul calculeaza factorialul unui numar";
        cout << "\n\n\tIntroduceri nr: ";
        cin >> nr;
        f=fact(nr);
        if (f!=0){
                cout << "\n\n\t" << nr << "!=" << f << "\n";
        }
        cin.ignore();
        cin.get();
        return 0;
}

int fact(int n)
{
        if (n<<"\n\n\tArgument negativ!\n";
                return 0;
        }
        else if (n==0)
                return 1;
        else
                return n*fact(n-1);
}
```

Sa calculam acum, utilizand functii recursive, elementul n din sirul lui Fibonacci. Dupa cum se stie, sirul are urmatoarea structura:

f(0)=0, f(1)=1, f(n)=f(n-1)+f(n-2), daca n>1

```
// Se utilizeaza functii recursive
// Programul calculeaza elementul n din sirul lui Fibonacci
// f(0)=0, f(1)=1, f(n)=f(n-1)+f(n-2), daca n>1
#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
long int r_fib(long int n); // prototipul
int main(void)
{
	int nr;
	long int f;
	cout << "\n\n\tProgramul calculeaza elementul n din sirul lui
Fibonacci";
	cout << "\n\n\tIntroduceti n: ";
	cin >> nr;
	f=r_fib(nr);
	cout << "\n\n\tValoarea sirului pentru n=" << nr << " este:" << f <<
"\n";
	cin.ignore();
	cin.get();
	return 0;
}

long int r_fib(long int n)
{
	if (n
```

Sa afisam acum, utilizand functii recursive, n elemente din sirul lui Fibonacci.

```
// Se defineste si se utilizeaza functia recursiva r_fib
// Programul afiseaza n elemente din sirul lui Fibonacci
// adica:  f(0)=0, f(1)=1, f(n)=f(n-1)+f(n-2), daca n>1
#include "stdafx.h"
#include < iostream >
#include < string >

using namespace std;
long int r_fib(long int n); // prototipul
int main(void)
{
        int i,nr;
        long int f;
        cout << "\n\n\tProgramul afiseaza n elemente din sirul lui Fibonacci";
        cout << "\n\n\tIntroduceri n: ";
        cin >> nr;
        cout << "\n\n";
        for (i=0; i<=nr ;i++){
           f=r_fib(i);
           cout << "\tf(" << i << ")= " << f << "\n";
        }
        cin.ignore();
        cin.get();
        return 0;
}

long int r_fib(long int n)
{
        if (n<0) return 0;
        if (n==0) return 0;
        if (n==1) return 1;
        long int i1=r_fib(n-1);
        long int i2=r_fib(n-2);
        return i1+i2;
}
```

Vizibilitatea si durata de viata a unei variabile

Odata cu aparitia mai multor functii intr-un program se pune problema vizibilitatii si duratei de viata a variabulelor definite in diferite functii inclusiv in cadrul functiei **main** .

- **Variabile locale**

Sa reluam aplicatia pentru calcularea ariei cercului dar sa o modificam astfel incat programul sa se reia atata timp cat se introduce o valoare diferita de 0 pentru raza cercului. Vom introducem o variabila nr_rel care contorizeaza numarul de reluari pentru a putea afisa pentru a cate-a ora se reia aplicatia:

```
// program pentru calculul ariei cercului
// se cere raza cercului si se afiseaza aria acestuia
// programul se reia pana se introduce valoarea 0 pentru raza
// se folseste variabila locala nr_rel pentru a afisa numarul de reuari ale
aplicaiei

#include "stdafx.h"
using namespace System;
double aria_c(double);// prototipul functiei

int main(void)
{
    int nr_rel=1;
    double raza ,a;
    String^ raza_s;
   do {
    Console::WriteLine( "\n\n\tApelarea nr : "+ nr_rel);
    Console::Write( L"\n\tIntroduceti raza cercului ( 0 pentru iesire):" );
    raza_s= Console::ReadLine();
    raza = System::Convert::ToDouble( raza_s );
    a=aria_c(raza);
    Console::WriteLine( "\n\n\tAria cercului de raza : "+ raza +" este: "+a);
    //Console::ReadLine();
       nr_rel++;
   } while (raza !=0);
    return 0;
}
// Incepe definirea functiei aria_c
double aria_c(double r)
{
    double pi= System::Math::PI;
    return pi*r*r;
}
```

In cazul in care am incrementa valoarea variabilei locale nr_rel, adica functia aria_c devine:

```
// Incepe definirea functiei aria_c
double aria_c(double r)
{
        double pi= System::Math::PI;
        nr_rel++;
        return pi*r*r;
}
```

La compilare ar aparea o eroare pentru ca variabila nr_rel nu este definita in cadrul functiei aria_c ci este definita in functia main. Cu alte cuvinte variabila nr_rel este o variabila locale iar domeniul sau de vizibilitate se rezuma la functia main.

Pentru ca variabila nr_rel sa fie vazuta atat in functia main cat si in functia calc_c, ea va trebui sa fie definita inaintea functiei main deci ea sa fie o variabila de tip global

- **Variabile globale**

O variabila de tip global are vizibilitate pe tot intreg parcursul programului. Aplicatia de sus va fi modificata, si va fi folosita variabila globala nr_rel pentru a fi accesata atat din functia main cat si din functia aria_c.

```
// program pentru calculul ariei cercului
// se cere raza cercului si se afiseaza aria acestuia
// programul se reia pana se introduce valoarea 0 pentru raza
// se folseste variabila globala nr_rel pentru a afisa numarul de reuari ale
aplicaiei
#include "stdafx.h"
using namespace System;
double aria_c(double);// prototipul functiei
int nr_rel=1;
int main(void)
{
   double raza ,a;
   String^ raza_s;
   do {
    Console::WriteLine( "\n\n\tApelarea nr : "+ nr_rel);
    Console::Write( L"\n\tIntroduceti raza cercului ( 0 pentru iesire):" );
    raza_s= Console::ReadLine();
    raza = System::Convert::ToDouble( raza_s );
    a=aria_c(raza);
    Console::WriteLine( "\n\n\tAria cercului de raza : "+ raza +" este: "+a);
    //Console::ReadLine();
    } while (raza !=0);
   return 0;
}
// Incepe definirea functiei aria_c
double aria_c(double r)
{
        double pi= System::Math::PI;
        nr_rel++;
        return pi*r*r;
}
```

Variabilele globale au avantajul ca sunt vizibile in tot programul dar au dezavantajul ca trebuie tinuta evidenta stricta a lor pentru a se evita modificarea lor in diverse functii. Pentru programele mari, operatia devine destul de dificila.

- **Variabile locale statice**

Durata de viata a unei variabile, este data de domeniul sau de vizibilitate, durata de viata a unei variabile locale se inchei odata cu functia. O variabila glogala are vizibilitate in tot programul deci se pastreaza pe toata durata programului. Variablele locale statice au vizibilitatea variabilelor locale dar durata de viata ete durata de viata a variabilelor globale. Variablele locale statice se declara in interiorul unei functii dar sub forma **static** nume_variabila. Aplicatia anterioara se rescrie astfel:

```
// program pentru calculul ariei cercului
// se cere raza cercului si se afiseaza aria acestuia
// programul se reia pana se introduce valoarea 0 pentru raza
// se folseste variabila locala statica nr_rel pentru a afisa numarul de reuari
ale aplicaiei

#include "stdafx.h"
using namespace System;
double aria_c(double);// prototipul functiei


int main(void)
{
    double raza ,a;
    String^ raza_s;
    do {
       Console::Write( L"\n\tIntroduceti raza cercului ( 0 pentru iesire):" );
       raza_s= Console::ReadLine();
       raza = System::Convert::ToDouble( raza_s );
       a=aria_c(raza);
       Console::WriteLine( "\tAria cercului de raza : "+ raza +" este: "+a);
       //Console::ReadLine();

    } while (raza !=0);
   return 0;
}
// Incepe definirea functiei aria_c
double aria_c(double r)
{
       double pi= System::Math::PI;
       static int nr_rel=1;
       Console::WriteLine( "\n\n\tLa apelarea nr : "+ nr_rel);
       nr_rel++;
       return pi*r*r;
}
```

Bibliografie

1.Catrina,O.,Cojocaru, I., Turbo C++, Ed.Teora, Bucuresti,1993.

2.Cozac,I., Programare în limbajul C, uz intern, Univ."Petru Maior" ,Tg.Mures,2004

3.Kernighan,B.W., Ritchie,D.M., The C programming language, Prentice Hall,1988

4.Petrovici,V., Programarea în limbajul C, Ed.Tehnica, Bucuresti,1993.

5.Runceanu,A., Programarea si utilizarea calculatoarelor.Limbajul C++. Ed.Academica Brâncusi, Tg.Jiu,2003.

6.Stefanescu,D., Segal,C., Initiere in limbajele C/C++,Ed.Fundatiei Universitare"Dunarea de Jos" Galati,2000

7.Mircea Popovici,Tehnologia orientata pe obiecte.Aplicatii,Ed.Teora 1996.

8 Namir C. Shammas, Curs rapid de Borland C++, Ed.Teora 1996.

9.Jeff Kent, C++ fara mistere,Ed.Rosetti Educational 2004 .

Link-uri

1. http://davos.science.upm.ro/~traian - Site Traian Turc - 2008.
2. http://www.ibs.ro/~bela/Teachings/Programming/programming.html Site Genge Bela - Tematica laboratoare C++ - 2008
3. http://www.scribd.com/doc/517796/Manual-C - Manual C# - 2008
4. - Expert C++-CLI .NET for Visual C++ - 2008
5. http://www.developerfusion.co.uk/show/1887/2/ - Programare C -2008
6. http://www.pcguide.com - Ghid PC - 2008
7 http://davos.science.upm.ro/~cozac - Site Ioan Cozac - 2008.
8. http://users.utcluj.ro/~baruch/carte_ac.html – Arhit. calculatoarelor - 2008
9. http://www.cs.uow.edu.av/people/nabg/ABC/ABC.html - Stroustrup, B., A Beginners C++ - 2008.
10. http://www.icce.rug.nl/docs/cplusplus/cplusplus.html - Stroustrup, B., C++ Annotations - 2008.
11. http://www.developerfusion.co.uk/show/1887/ - Tutorial C# - 2008
12. http://www.microsoft.com/express/vc/ - Link Visual C++ Express - 2008.
13. http://www.microsoft.com/express/download/" - Link-ul pentru download Visual C++ 2008 Express - 2008
14. http://msdn.microsoft.com/en-us/library/ms235291(VS.80).aspx - How to: Deploy using XCopy - 2008.
15. http://msdn.microsoft.com/en-us/library/ms235317(VS.80).aspx - How to: Deploy using a Setup and Deployment Project - 2008

Cuprins