

Îndrumător de laborator

Disciplina:

Programare orientată pe obiecte

Autor: Lect. univ. *Horea Oros*

Cuprins

INTRODUCERE	4
LABORATOR 1 - <i>LIMBAJUL C</i>	5
LABORATOR 2 - <i>INTRARE-IEȘIRE LA CONSOLĂ</i>	9
LABORATOR 3 - <i>INTRARE – IEȘIRE PE FIȘIERE ȘI PE ȘIRURI</i>	13
LABORATOR 4 - <i>INTRODUCERE ÎN CREAREA CLASELOR</i>	17
LABORATOR 5 - <i>NUMERE COMPLEXE</i>	22
LABORATOR 6 - <i>NUMERE RAȚIONALE</i>	26
LABORATOR 7 - <i>STIVĂ DE ÎNTREGI</i>	29
LABORATOR 8 - <i>ȘIRURI DE CARACTERE</i>	32
LABORATOR 9 - <i>ORA</i>	38
LABORATOR 10 - <i>DATA</i>	42
LABORATOR 11 - <i>MATRICE</i>	47
LABORATOR 12 - <i>LISTĂ DE PERSOANE</i>	53
LABORATOR 13 - <i>ARBORE OARECARE</i>	58
LABORATOR 14 - <i>NUMERE MARI</i>	62
LABORATOR 15	71
LABORATOR 16	84
LABORATOR 17	94
LABORATOR 18	97

LABORATOR 19	103
LABORATOR 20	124
LABORATOR 21	135
LABORATOR 22	143
LABORATOR 23	149
LABORATOR 24	150
LABORATOR 25	151
LABORATOR 26	152
LABORATOR 27	153
LABORATOR 28	157

Introdúcere

Laborator 1 - limbajul C

Cum limbajul C++ este bazat pe limbajul C, pentru a programa în C++ este absolut necesară cunoașterea sintaxei limbajului C. Nu este obligatorie însă învățarea limbajului C înainte de C++ (fapt confirmat chiar și de creatorul limbajului C++, Bjarne Stroustrup și de mulți alți programatori profesioniști) dar având în vedere că aceste note de laborator se adresează studenților din anul III profilul matematică-informatică, studenți ce au studiat anterior limbajul C, considerăm că aceștia pot rezolva următoarele probleme C cu ușurință.

- 1) Scrieți un program C care calculează factorialul unui număr natural introdus de la tastatură.
- 2) Scrieți un program C care generează aleator elementele unui tablou de întregi și sortează tabloul crescător folosind metoda bulelor. (funcția. random(n) din stdlib.h)
- 3) Ce va tipări următorul program:

```
#include <stdio.h>
int main(){
    float a;
    a=10/3;
    printf("%.2f\n", a);
    return 0;
}
```

- 4) Ce valoare are expresia: `!(1 && 0 || !1);`
- 5) Scrieți un program C care interschimbă valorile a două variabile de tip întreg cu ajutorul unei funcții.
- 6) Ce valoare returnează funcția următoare dacă este apelată prin `f(7)` :

```
int f(int v){
    if(v==1 || v==0)
        return 1;
    if(v%2 == 0)
        return f(v/2)+2;
    else
        return f(v-1)+3;
}
```

- 7) Care din următoarele fragmente de cod sunt corecte? Explicați.

- a) `char s[100];`
`s="hello";`
`printf("%s", s);`
- b) `char *s;`
`s="hello";`
`printf("%s", s);`
- c) `char *s;`
`s=(char *)malloc(100);`
`s="hello";`
`free(s);`
- d) `char *s;`
`s=(char *)malloc(100);`

Laborator C++

```
strcpy(s, "hello");
```

```
free(s);
```

8) Scrieți un program C care tipărește argumentele din linia de comandă cu care a fost apelat.

9) Definiți o funcție care returnează suma parametrilor de tip int cu care a fost apelată.

Declarați un pointer la o astfel de funcție și apelați funcția utilizând pointerul.

10) Explicați următoarele declarații complexe

a) `void *(*fp1)(int)[10];`

b) `float (*fp2)(int, int, int)(int);`

c) `typedef double (*fp3())[10]();` `fp3 a;`

d) `int (*fp4())[10]();`

Rezolvare:

1) Calculul factorialului:

```
#include <stdio.h>
```

```
main()
```

```
{
```

```
    int n, i, f;
```

```
    printf("n=");
```

```
    scanf("%d", &n);
```

```
    f=1;
```

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

```
        f = f * i;
```

```
    printf("n! = %d", f);
```

```
    return 0;
```

```
}
```

2) Sortare prin metoda bulelor:

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
#define SIZE 10 /* dimensiunea tabloului */
```

```
#define MAX 1000 /* valoarea maxima a unui element al tabloului */
```

```
main()
```

```
{
```

```
    int i, j, t, a[SIZE];
```

```
    randomize();
```

```
    for(i=0; i<SIZE; i++)
```

```
        a[i] = random(MAX+1);
```

```
    printf("Tabloul inainte de sortare:\n");
```

```
    for(i=0; i<SIZE; i++)
```

```
        printf("%d %d\n", i+1, a[i]);
```

```
    for(i=0; i<SIZE-1; i++)
```

```
        for(j=i+1; j<SIZE; j++)
```

```
            if(a[i]>a[j])
```

```
            {
```

```

        t = a[i]; a[i] = a[j]; a[j] = t;
    }
    printf("Tabloul sortat crescator:\n");
    for(i=0; i<SIZE; i++)
        printf("%d %d\n", i+1, a[i]);
    return 0;
}

```

3) Programul va tipări 3.00. Explicația este că în expresia 10/3 ambii operanzi fiind tip întreg și rezultatul va fi tot de tip întreg, așadar 3. Faptul că variabila a este de tip float nu influențează evaluarea expresiei 10/3.

4) Operatorul de negație logică ! este cel mai prioritar dintre cei trei operatori logici, folosiți în expresie, fiind și singurul operator unar dintre cei trei. Operatorul && (și logic) este mai prioritar decât operatorul || (sau logic).

$!(1 \ \&\& \ 0 \ || \ !1) \Rightarrow !(1 \ \&\& \ 0 \ || \ 0) \Rightarrow !(0 \ || \ 0) \Rightarrow !(0) \Rightarrow 1$

5) Funcție pentru interschimbarea a două valori:

```

#include <stdio.h>
void swap(int *, int *);
main()
{
    int a = 1, b = 2;
    printf("Inainte de interschimbare:\na = %d b = %d\n", a, b);
    swap(&a, &b);
    printf("Dupa interschimbare:\na = %d b = %d\n", a, b);
    return 0;
}
void swap(int *x, int *y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
}

```

6) $f(7) = ?$

$f(7) = f(6) + 3$

$f(6) = f(3) + 2$

$f(3) = f(2) + 3$

$f(2) = f(1) + 2$

$f(1) = 1 \Rightarrow f(7) = 11.$

7) Explicația:

- greșit** pentru că s, fiind numele unui tablou, este un pointer constant la o zonă de memorie de 100 int și care nu poate fi făcut să poarte la o altă zonă de memorie
- corect**
- greșit** pentru că *free(s)* nu funcționează întrucât s nu mai poartă la zona de memorie alocată cu *malloc*. Zona de memorie alocată cu *malloc* rămâne orfană după instrucțiunea *s = "hello"*, apare fenomenul de "memory leak" (scurgere de memorie)
- corect**,

Laborator C++

8) Linia de comandă

```
#include <stdio.h>
main(int argc, char *argv[])
{
    int i;
    for(i=1; i<argc; i++) // primul argument este numele
                           //programului
        printf("%s\n", argv[i]);
    return 0;
}
```

9) Pointer la funcție

```
#include <stdio.h>
int f(int, int);
main()
{
    int (*fp)(int, int);
    fp = f;
    printf("%d", fp(1, 2));
    return 0;
}
int f(int a, int b)
{
    return a+b;
}
```

10) Declarațiile complexe se citesc din interior spre exterior. Pentru interpretarea declarațiilor complexe există programul cdecl.

- a) fp1 este pointer la funcție care are un singur argument de tip int și returnează pointer la tablou de 10 pointeri la void
- b) fp2 este pointer la funcție (cu trei argumente) care returnează pointer la funcție cu un singur argument de tip int și care returnează o valoare de tip float
- c) a este pointer la funcție fără argumente care returnează pointer la tablou de 10 pointeri la funcții fără argumente care returnează o valoare de tip double
- d) fp4 este funcție fără argumente ce returnează pointer la tablou de 10 pointeri la funcții fără argumente care returnează o valoare de tip int.

Laborator 2 - intrare-ieșire la consolă

Oricine învață un nou limbaj de programare, primul lucru pe care vrea să-l știe este modul în care poate afișa date pe ecran și cum poate introduce date de la tastatură, pentru a realiza un program de consolă (program de consolă = program la care interfața cu utilizatorul se realizează în mod text). Cu toate că programele de consolă fac parte din epoca de piatră a programării, (la ora actuală majoritatea programelor dispun de interfețe grafice standardizate care le fac mai ușor și mai intuitiv de utilizat) noi vom folosi în programele noastre intrarea-ieșirea la consolă dar mai ales intrarea-ieșirea pe fișiere. Limbajul C++ nu dispune de facilități de intrare-ieșire, aceste facilități fiind oferite prin intermediul bibliotecii **iostream**. Avantajele utilizării acestei biblioteci, față de utilizarea bibliotecii standard de intrare-ieșire din C, **stdio** sunt mai multe:

- 1) este orientată pe obiecte;
- 2) se utilizează aceeași interfață indiferent dacă intrarea-ieșirea se face la consolă, pe fișiere sau pe șiruri de caractere;
- 3) tipurile de date pe care le creăm pot fi adaptate pentru a fi compatibile cu **iostream** ceea ce duce la extensibilitate facilităților de I/O. Funcțiile **printf** și **scanf** știu să lucreze doar cu un număr limitat de tipuri (tipurile fundamentale din C – char, int, float, double), aceste funcții nefiind extensibile.

Primul program pe care-l scrie orice novice în C++ este celebrul program “Hello World!”.

```
#include <iostream.h>
main()
{
    cout << "Hello World!" << endl;
    return 0;
}
```

Prima linie a programului include, în programul sursă, fișierul header **iostream.h** ce conține declarațiile pentru facilitățile de intrare-ieșire la consolă. **cout** (console output) este un obiect asociat cu stream-ul standard de ieșire (în mod implicit este ecranul dar poate fi redirectat). **endl** este un manipulator al cărui efect va fi trecerea pe linia următoare. După cum știm operatorul << este operatorul de deplasare la stânga pe biți în C. Acest operator are aceeași semnificație și în C++ dar când este folosit împreună cu **cout** își schimbă semnificația, numindu-se operator de inserție care permite trimiterea datelor ce se află în partea dreaptă a operatorului la obiectul din partea stângă, în acest caz **cout**. Sensul săgeții indică direcția de deplasare a datelor. Așadar următorul program este corect: și se va afișa 32:

```
#include <iostream.h>
main()
{
    cout << (2<<4) << endl;
    return 0;
}
```

Alături de manipulatorul **endl** există și alți manipulatori simpli și parametrizați. Utilizarea manipulatorilor parametrizați necesită includerea în program a fișierului header **iomanip.h**.

```
#include <iostream.h>
#include <iomanip.h>
main()
```

Laborator C++

```
{  
    cout << dec << setw(5) << 255 << endl;  
    cout << oct << setw(5) << 255 << endl;  
    cout << hex << setw(5) << 255 << endl;  
    cout << setprecision(2) << 1.234 << endl;  
    return 0;  
}
```

Manipulatorul **dec** convertește valoarea ce se afișează în baza 10, **oct** în baza 8 și **hex** în baza 16, **setw** stabilește dimensiunea minimă a spațiului pe care vor fi afișate datele alinierea implicită fiind la dreapta, **setprecision** permite stabilirea numărului de zecimale cu care va fi afișat un număr în virgulă mobilă.

Problemă: Scrieți un program C++ ce afișează la consolă conținutul unui tablou de caractere în format hexa. Conținutul tabloului va fi generat aleator, folosind funcția din biblioteca standard C **random()**. Datele vor fi afișate la consolă sub forma:

76 72 53 8B 69 AB BB 9B 98 8D 6D D 6E B6 28 30	vrS.i.....m.n.(0
5A 97 4A 5C 4A 1E B0 A2 DB 4C 73 BF C5 28 63 24	Z.J\J....Ls..(c\$
95 FA 98 2F 6D 60 F1 66 B5 E D5 D9 82 E4 F 70	.../m`.f.....p
A4 23 21 4F 11 54 E5 19 AC CE D2 41 1E 5F B8 1B	.#!O.T.....A._...
B5 F7 55 F1 0 F3 98 CD 2E D7 E8 34 CD F 44 7C	..U.....4..D
B 9B 97 17 F3 2D 20 5B EB A0 37 2C E3 55 52 20- [...7,.UR
5E 54 9B 6B	^T.k

Sursa de octeți nu e foarte importantă. Problema o vom relua când vom trata lucrul cu fișiere.

Pentru fișiere vom putea crea un viewer sau chiar un editor hexa.

În partea stângă apar valorile hexa ale caracterelor din partea dreaptă. În partea dreaptă am înlocuit cu punct caracterele ce au codul ASCII în afara intervalului 32..127.

Programul ce rezolvă problema de mai sus:

```
#include <iostream.h>  
#include <stdlib.h>  
#include <iomanip.h>  
#include <ctype.h>  
main()  
{  
    const int MAX=100;  
    unsigned char s[MAX+1];  
    int i;  
    // initializarea generatorului de numere aleatoare  
    randomize();  
    for(i=0; i<MAX; i++)  
        s[i]=(unsigned char)random(255);  
    s[MAX]='\0';  
    i=0;  
    int j;  
    do{  
        for(j=0; j<16 && i<MAX; j++, i++)  
            cout << setiosflags(ios::uppercase) << setw(3)  
                << hex << (unsigned)s[i];
```

```

        if(i==MAX)
            for(int t=0; t<16-j; t++)
                cout << setw(3) << " ";

        i-=j;
        cout << '\t';
        for(j=0; j<16 && i<MAX; j++, i++)
            if (isprint(s[i]))
                cout << s[i];
            else
                cout << '.';

        cout << endl;
    }while(i<MAX);
    return 0;
}

```

Intrarea de la consolă se realizează prin obiectul **cin** (console input). **cin** este un obiect asociat cu stream-ul standard de intrare (în mod implicit este tastatura dar poate fi redirectat).

```

#include <iostream.h>
main()
{
    char c, tab[10];
    int n;
    float f;
    cin >> c >> tab >> n >> f;
    cout << "c=" << c << endl;
    cout << "tab=" << tab << endl;
    cout << "n=" << n << endl;
    cout << "f=" << f << endl;
    return 0;
}

```

Dacă introducem de la tastatură șirul **"x abc 123 1.234"** programul va afișa:

```

c=x
tab=abc
n=123
f=1.234

```

Datele trebuie să fie separate prin spații albe (spațiu, tab sau linie nouă). Pentru a citi un șir de caractere ce conține spații albe sau alți delimitatori, trebuie folosite metodele **get** sau **getline**. Pentru a citi un singur caracter folosim metoda **getc** iar pentru a afișa un singur caracter **putc**. Pentru a inspecta următorul caracter din intrare se poate folosi metoda **peek** (caracterul nu este scos din intrare), iar pentru a repune în intrare un caracter ce a fost scos se folosește **putback**.

Exercițiu: Studiați modul de utilizare al metodelor descrise mai sus.

Intrarea standard se redirectează utilizând sintaxa: "nume program executabil" > fisier.

Ieșirea standard se redirectează utilizând sintaxa: "nume programului executabil" < fisier.

Pentru a fi siguri că rezultatele vor ajunge pe ecran și nu într-un fișier, chiar dacă utilizăm redirectarea putem trimite datele la obiectul **cerr** (console error).

Laborator C++

Problemă:

1. Scrieți un program C++ care elimină comentariile dintr-un fișier sursă C++. Se va folosi redirectarea intrării pentru a da programului numele fișierului sursă ce trebuie procesat și redirectarea ieșirii pentru a da numele fișierului rezultat (cel ce nu conține comentarii).

Laborator 3 - intrare – ieșire pe fișiere și pe șiruri

Pentru a putea folosi intrare-ieșire pe fișiere trebuie să includem în programele noastre fișierul header `<fstream.h>` și să declarăm variabile de tip **ifstream** și **ofstream**. Următorul program copiază conținutul fișierului text cu numele “xxx” în fișierul cu numele “yyy”.

```
#include <iostream.h>
#include <fstream.h>
main()
{
    char c;
    ifstream i_file("xxx");
    ofstream o_file("yyy");
    if(i_file && o_file) // adevarat doar daca ambele fisiere
                        //au putut fi deschise
        while(i_file >> c)
            o_file << c;
}
```

Programul încearcă să deschidă cele două fișiere și dacă reușește copiază caracter cu caracter datele din fișierul “xxx” în fișierul “yyy”.

Probleme:

- 1) Scrieți un program C++ care copiază conținutul unui fișier text într-un alt fișier text. Numele celor două fișiere sunt date în linia de comandă. Citirea se face pe linii pentru a reduce numărul de accesări ale discului și pentru a crește viteza de rulare a programului.
- 2) Rezolvați problema de mai pentru fișiere binare (citirea și scrierea datelor din/în fișiere binare se face cu metodele **read** respectiv **write**).
- 3) Scrieți un program C++ care scrie într-un fișier text, cu numele **include.dat**, numele tuturor fișierelor incluse într-un fișier sursă C++. Numele fișierului sursă se dă ca unic argument în linia de comandă
- 4) Scrieți un program care găsește toate palindroamele dintr-un fișier text, al cărui nume se dă în linia de comandă. (Palindrom = cuvânt pentru care literele egal depărtate de mijlocul lui sunt identice).
- 5) Scrieți un program C++ care afișează pe ecran conținutul un fișier binar în format hexa.

Rezolvare:

- 1) Copiere de fișiere text

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
    ifstream i;
    ofstream o;
    if(argc!=3){
        cerr << "Utilizare copy <sursa> <destinatia>" << endl;
        exit(1);
    }
}
```

Laborator C++

```
i.open(argv[1]);
if(i.fail()){
    cerr << "Nu pot deschide fisierul " << argv[1] << endl ;
    exit(1);
}
o.open(argv[2]);
if(o.fail()){
    cerr << "Nu pot deschide fisierul " << argv[2] << endl ;
    exit(1);
}
const int MAXLINE=1000;
char buf[MAXLINE];
while (!i.eof()){
    i.getline(buf, MAXLINE, '\n');
    o << buf << endl;
}

i.close();
o.close();
return 0;
}

2) Copiere de fişiere binare
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <stdlib.h>
main(int argc, char *argv[])
{
    ifstream i;
    ofstream o;
    if(argc!=3){
        cerr << "Utilizare " << argv[0]
            << " <sursa> <destinatie>" << endl;
        exit(1);
    }
    i.open(argv[1], ios::binary);
    if(i.fail()){
        cerr << "Nu pot deschide fisierul "
            << argv[1] << endl ;
        exit(1);
    }
    o.open(argv[2], ios::binary);
    if(o.fail()){
        cerr << "Nu pot deschide fisierul "
            << argv[2] << endl ;
        exit(1);
    }
}
```

Laborator C++

```
}
unsigned char c[1];
while (!i.eof()){
    i.read(c, 1); // citesc cate un caracter
    if(!i.eof()) // testul este necesar pentru ca
        // eof va fi setat doar dupa ce se citește un
        // caracter dincolo de sfarsitul fiserului
        // fara acest test dimensiunea fiserului destinatie
        // va fi cu un octet mai mare decat a fiserului sursa
    o.write(c, 1);
}

i.close();
o.close();
return 0;
}
```

Intrarea-ieșirea se poate face și pe șiruri din memorie, într-un mod echivalent cu cel al funcțiilor din biblioteca Standard C **sscanf** și **sprintf**. Pentru a putea folosi aceste facilități trebuie să includem în program fișierul header **strstream.h** și să declarăm obiecte de tip **ostrstream** pentru ieșire, respectiv **istrstream** pentru intrare.

```
#include <iostream.h>
#include <strstream.h>
main()
{
    char s[100];
    ostrstream o(s, 100);
    o << "pi=" << 3.14 << ends;
    cout << s;
    return 0;
}
```

Programul de mai sus afișează **pi=3.14**. Manipulatorul **ends** adaugă terminatorul de șir (caracterul NUL).

Dacă există un șir de caractere și vrem să citim din el putem folosi un stream de intrare pe șiruri ca mai jos.

```
#include <iostream.h>
#include <strstream.h>
#include <string.h>
main()
{
    char *s = "3.14 12 abc";
    istrstream i(s, strlen(s));
    float f;
    int n;
    char t[100];
```

Laborator C++

```
i >> f >> n >> t;  
cout << f << " " << n << " " << t;  
return 0;  
}
```

Programul de mai sus va afișa **3.14 12 abc**.

Utilizând intrarea ieșirea pe șiruri putem realiza, conversii asemănătoare cu cele realizate de funcțiile `atoi`, `itoa`, `ltoa` etc.

Problemă:

1. Scrieți un program C++ care să realizeze conversia unui șir de caractere, reprezentând un număr, în numărul corespunzător (întreg sau real) și conversia unui număr întreg sau real în șirul de caractere echivalent. Utilizați intrarea-ieșirea pe șiruri.

Laborator 4 - *introducere în crearea claselor.*

Clasa este un concept fundamental în programarea orientată pe obiecte. În C++ tipurile fundamentale sunt char, int, float, double și void. Cu acestea putem crea tipuri compuse sub formă de tablouri și structuri. Construcțiile typedef nu introduc noi tipuri de date ele introduc în spațiul unui program doar noi nume pentru tipuri existente. Clasele sunt tipuri de date definite de utilizator. Prin intermediul claselor putem extinde la infinit mulțimea tipurilor oferite de limbajul de programare. Așadar când declarăm o clasă stabilim datele membre și operațiile care le putem executa asupra acelor date. Clasele reprezintă doar o abstractizare a unui concept; pentru a putea lucra efectiv cu acel concept trebuie să creăm o instanță a clasei instanță care în jargonul programării orientate pe obiecte se numește chiar **obiect**.

O operație importantă care se execută asupra datelor în orice program este inițializarea, datele neinițializate reprezentând în orice program o sursă de erori. Datele unei clase se inițializează cu ajutorul funcțiilor constructor. Funcția constructor alocă resurse unui obiect. Funcția destructor eliberează resursele alocate obiectului.

Reluăm programul "Hello World!" și îl vom scrie din punctul de vedere al POO.

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

```
class World{
```

```
    public:
```

```
        World(int id): _id(id)
```

```
        {
```

```
            cout << "Hello ! de la " << _id << endl;
```

```
        }
```

```
        ~World()
```

```
        {
```

```
            cout << "Bye bye ! de la " << _id << endl;
```

```
        }
```

```
    private:
```

```
        int _id;
```

```
};
```

```
World w(1); // obiect global
```

```
int main()
```

```
{
```

```
    cout << "Nu asta e prima linie care se va afisa" << endl;
```

```
    World w2(2); // obiect local
```

```
    for(int i=3; i<5; i++)
```

```
        World w3(i);
```

```
    World w3(5);
```

```
    return 0;
```

```
}
```

Probleme:

- 1) Descoperiți modul de rulare pas cu pas al unui program în mediului pentru dezvoltarea de aplicații pe care îl folosiți și rulați programul de mai sus pas cu pas pentru a vedea când sunt apelate funcțiile constructor și destructor pentru fiecare din obiectele definite.
- 2) Scrieți un program C++ care să afișeze următorul tipar:

Laborator C++

```
*  
* *  
* * *  
* * * *  
* * * * *
```

- 3) Descoperiți modul de creare a unui proiect în mediul pentru dezvoltare de aplicații pe care îl folosiți și creați un proiect pentru problema de mai sus. Se vor crea trei fișiere: un fișier header ce conține declarația clasei cu numele stars.h , un fișier ce conține implementarea clasei (definiția clasei) cu numele stars.cpp și un fișier cu numele driver.cpp ce conține funcția main.

Rezolvare:

```
// stars.h  
#ifndef STARS_H // aceste directive pentru preprocesor  
#define STARS_H // ne garanteaza faptul ca declaratia  
// clasei nu va fi inclusa de doua ori  
// intr-un fisier sursa C++, ceea ce constituie  
// eroare  
  
class Star  
{  
    public:  
        Star(int);  
    private:  
        int _i;  
};  
#endif  
  
// stars.cpp  
#include "stars.h"  
#include <iostream.h>  
Star::Star(int i): _i(i)  
{  
    for(int j=0; j<i; j++)  
        cout << '*';  
    cout << endl;  
}  
  
// driver.cpp  
#include <stars.h>  
main()  
{  
    for(int i=1; i<=5; i++)  
        Star s(i);  
    return 0;  
}
```

Laborator C++

Probleme:

- 1) Definiți o clasă HorzBar al cărei constructor tipărește la consolă +-----+. Numărul de semne minus se dă argument constructorului.
- 2) Definiți o clasă VertBar al cărei constructor tipărește:

```
|  
|  
|
```

Numărul de semne “|” se dă ca argument constructorului.

- 3) Definiți o clasă Frame care conține două obiecte de tip HorzBar și între ele unul de tip VertBar

```
+-----+  
|  
|  
|  
+-----+
```

- 4) Definiți o clasă Ladder ce conține două obiecte de tip Frame și între ele un obiect de tip VertBar.

```
+-----+  
|  
|  
|  
+-----+  
|  
|  
|  
+-----+  
|  
|  
|  
+-----+
```

Rezolvare:

```
#include <iostream.h>  
class HorzBar  
{  
public:  
    HorzBar(int h): _h(h)  
    {  
        cout << '+';  
        for(int i=0; i<h; i++)  
            cout << '-';  
        cout << '+' << endl;  
    }  
private:  
    int _h;  
};
```

Laborator C++

```
class VertBar
{
public:
    VertBar(int v): _v(v)
    {
        for(int i=0; i<v; i++)
            cout << '|' << endl;
    }
private:
    int _v;
};

class Frame
{
public:
    Frame(int h, int v): h1(h), vert(v), h2(h)
    {
        //Acesta este un exemplu care ilustreaza foarte bine
        //rolul listei de initializatori a constructorului.
        //Obiectele declarate intr-o clasa pot fi initializate doar
        //prin intermediul acestei liste.
        //Ordinea in care apar variabilele in lista nu este importanta
        //ele vor fi initializate in ordinea in care sunt declarate
        //in clasa. Lista de initializatori a constructorului este de
        //asemenea importanta in cazul mostenirii, prin intermediul ei
        //se initializeaza clasele de baza.
    }
private:
    HorzBar h1;
    VertBar vert;
    HorzBar h2;
};

class Ladder
{
public:
    Ladder(int h, int v): f1(h, v), vert(v), f2(h, v)
    {
    }
private:
    Frame f1;
    VertBar vert;
    Frame f2;
};

main()
{
    Ladder l(3, 4);
}
```

Laborator C++

```
return 0;  
}
```

Problemă: Creați un proiect pentru programul de mai sus. Pentru fiecare clasă creați un fișier header ce va conține declarația clasei și un fișier cu implementarea clasei (definiția metodelor). Funcția main va fi pusă într-un fișier separat cu numele driver.cpp.

Laborator 5 - numere complexe

Cu toate că majoritatea mediilor pentru dezvoltarea de aplicații C++ au o bibliotecă de clase ce conține și o clasă pentru lucrul cu numere complexe, considerăm ca fiind un exercițiu util definirea acestui tip de date.

Problemă: Creați un tip de date definit de utilizator pentru numere complexe.

Pentru inițializarea unui număr complex vom folosi trei constructori: unul fără parametri (implicit) care va inițializa numărul complex la zero, unul cu un singur parametru de tip double cu care se va inițializa partea reală (în acest caz partea imaginară se inițializează cu zero) și un constructor cu doi parametri de tip double cu care se vor inițializa partea reală respectiv partea imaginară. Vom implementa operațiile cu numere complexe: adunare, scădere, înmulțire, comparare.

```
// complex.h
#ifndef COMPLEX_H
#define COMPLEX_H
enum bool {false, true};
class Complex
{
    public:
        // Constructori
        Complex();
        Complex(double);
        Complex(double, double);
        // Funcții accesori
        double getRe();
        double getIm();

        void tiparire();
        // Operatori aritmetici
        Complex adun(Complex &);
        Complex scad(Complex &);
        Complex inmult(Complex &);
        double modul();
        // Operatori relationali
        bool esteEgalCu(Complex &);
        bool esteDiferitDe(Complex &);
    private:
        double _re, _im;
};
#endif
```

```
//complex.cpp
#include "complex.h"
#include <iostream.h>
#include <math.h> // pentru sqrt
// C-tor implicit
```

Laborator C++

```
Complex::Complex(): _re(0), _im(0)
{
}
// C-tor cu un singur argument
Complex::Complex(double re): _re(re), _im(0)
{
}
// Constructor cu doua argumente
Complex::Complex(double re, double im): _re(re), _im(im)
{
}
// Declaram doua functii membre accesori.
// Accesul la datele private se va face prin aceste metode,
// chiar daca este mai putin eficient decat accesarea directa a lor
// din functii membre.
// Orice apel de functie adauga o suprasarcina (overhead) programului.
// Cand lucram cu clase mai sofisticate aceste metode
// permit schimbarea cu usurinta a implemetarii.
double Complex::getRe()
{
    return _re;
}

double Complex::getIm()
{
    return _im;
}

// Functie membra care are rolul de a afisa numarul complex
void Complex::tiparire()
{
    cout << getRe();
    if(getIm()>=0)
        cout << "+i*";
    cout << getIm();
}
// Operatiile de adunare, scadere, inmultire nu modifica
// operanzii. Se creaza un obiect temporar care se returneaza
// prin valoare.
Complex Complex::adun(Complex &op2)
{
    Complex temp(getRe() + op2.getRe(), getIm() + op2.getIm());
    return temp;
    // sau return Complex(getRe + op2.getRe(), getIm() + op2.getIm());
    // varianta mai eficienta decat prima
```

}

Complex Complex::scad(Complex &op2)

```
{  
    Complex temp(getRe() - op2.getRe(), getIm() - op2.getIm());  
    return temp;  
    // sau return Complex(getRe() - op2.getRe(), getIm() - op2.getIm());  
}
```

Complex Complex::inmult(Complex &op2)

```
{  
    Complex temp(getRe()*op2.getRe() - getIm()*op2.getIm(),  
                getRe()*op2.getIm() + getIm()*op2.getRe());  
    return temp;  
}
```

// Functia returneaza modulul numarului complex

double Complex::modul()

```
{  
    return sqrt(getRe()*getRe() + getIm()*getIm());  
}
```

// Implementarea operatorilor relationali

bool Complex::esteEgalCu(Complex &op2)

```
{  
    // Operatiile de comparare de mai jos  
    // ar trebui sa tina cont si de reprezentarea  
    // numerelor in virgula mobila  
    if(_re == op2._re && _im == op2._im)  
        return true;  
    else  
        return false;  
}
```

bool Complex::esteDiferitDe(Complex &op2)

```
{  
    if(_re != op2._re || _im != op2._im )  
        return true;  
    else  
        return false;  
}
```

// test.cpp

#include "complex.h"

#include <iostream.h>

main()

```
{  
    Complex c1, c2(1.23), c3(2.34, 3.45), c4(1.21, 3.25);
```



```
cout << "c1="; c1.tiparire(); cout << endl;
cout << "c2="; c2.tiparire(); cout << endl;
cout << "c3="; c3.tiparire(); cout << endl;
cout << "c4="; c4.tiparire(); cout << endl;
c1 = c2.adun(c3);
cout << "|c1| = " << c1.modul() << endl;
cout << "c2+c3="; c1.tiparire(); cout << endl;
c1 = c2.scad(c4);
cout << "c2-c4="; c1.tiparire(); cout << endl;
c1 = c3.inmult(c4);
cout << "c3*c4="; c1.tiparire(); cout << endl;
if (c1.esteEgalCu(c3.inmult(c4)))
    cout << "Egale" << endl;
else
    cout << "Nu sunt egale" << endl;
return 0;
}
```

După cum se vede din programul test.cpp de mai sus utilizarea clasei “Complex” este foarte simplă, dacă cineva (creatorul de clase) a depus efortul de a o implementa. Tot ceea ce trebuie să facă programatorul client (cel care utilizează clasa) este să includă în fișierul său sursă fișierul header al clasei și să acceseze funcțiile de interfață. Implementarea clasei de obicei va fi într-o bibliotecă de clase de a cărei existență trebuie informat compilatorul.

Utilizând programare orientată pe obiecte putem deveni programatori mult mai eficienți (prin eficiență în programare înțelegem că un program care cu alte paradigme ale programării este finalizat de o echipă de zece programatori în două luni de zile, programând pe obiecte, doi programatori îl termină în trei săptămâni). Eficiența se realizează prin reutilizarea codului. Programarea orientată pe obiecte ne permite să “pictăm cu o pensulă mai mare”, putem exprima într-un mod mai compact și mai clar mult mai multe operații. Limbajul în care se exprimă soluția problemei este mult mai apropiat de limbajul în care este formulată problema decât de limbajul mașinii pe care se rezolvă problema.

Probleme:

1. Adăugați la clasa Complex, definită mai sus, operațiile de împărțire a două numere complexe și ridicare la putere a unui număr complex. Adăugați de asemenea și metode pentru transformarea unui număr complex din forma algebrică în forma trigonometrică.
2. Studiați modul de creare a unei biblioteci în mediul pentru dezvoltare de aplicații C++ pe care îl utilizați. Cu clasa definită mai sus creați o bibliotecă pe care să o utilizați într-un program.

Laborator 6 - numere raționale

Creați un tip de date definit de utilizator pentru numere raționale. Numele tipului va fi **Rational**. Definiți pentru acest tip de date operațiile de adunare, scădere, înmulțire, împărțire, inversare, egalitate. Puneți acest tip de date într-o bibliotecă.

```
// rational.h
#ifndef RATIONAL_H
#define RATIONAL_H
enum bool {false, true};
class Rational
{
public:
    Rational(); // Constructor implicit
    Rational(int, int);
    // Operatori aritmetici
    Rational Adun(Rational &);
    Rational Scad(Rational &);
    Rational Inmult(Rational &);
    void invers(); // Inverseaza un numar rational;
    // Operatori relationali
    bool egalCu(Rational &);

    void tipar();
    // Metode accesori
    int getNuma(); // Numaratorul
    int getNumi(); // Numitorul
private:
    int numarator, numitor;
};
#endif
// rational.cpp
#include "rational.h"
#include <iostream.h>
Rational::Rational()
{
    numarator = numitor = 0;
}
Rational::Rational(int numarator, int numitor)
{
    // Pentru a face distincție între argumentele
    // funcției constructor și numele datelor membre
    // folosim operatorul de rezoluție a domeniului
    // de valabilitate ::
    Rational::numarator = numarator;
    Rational::numitor = numitor;
}
```

Laborator C++

```
// Metode operator
Rational Rational::Adun(Rational &r2)
{
    Rational temp;
    temp.numarator = getNuma()*r2.getNumi() + getNumi()*r2.getNuma();
    temp.numitor = getNumi() * r2.getNumi();
    return temp;
}

Rational Rational::Scad(Rational &r2)
{
    Rational temp;
    temp.numarator = getNuma() * r2.getNumi() - getNumi()*r2.getNuma();
    temp.numitor = getNumi() * r2.getNumi();
    return temp;
}

Rational Rational::Inmult(Rational &r2)
{
    Rational temp;
    temp.numarator = getNuma() * r2.getNuma();
    temp.numitor = getNumi() * r2.getNumi();
    return temp;
}

void Rational::tipar()
{
    cout << getNuma() << '/' << getNumi();
}

// Metode accesor
int Rational::getNuma()
{
    return numarator;
}

int Rational::getNumi()
{
    return numitor;
}

void Rational::invers()
{
    int temp;
    temp = numarator;
    numarator = numitor;
    numitor = temp;
}
```

Laborator C++

}

```
bool Rational::egalCu(Rational &r2)
{
    if(numarator*r2.getNume()==numitor*r2.getNume())
        return true;
    else
        return false;
}
// driver.cpp
#include <iostream.h>
#include "rational.h"
main()
{
    Rational r1(1, 2), r2(6, 8), r3;
    r3 = r1.Adun(r2);
    r3.tipar(); cout << endl;

    r1.tipar(); cout << endl;
    r1.invers();
    r1.tipar(); cout << endl;

    if(r1.egalCu(r2))
        cout << "Egale" << endl;
    else
        cout << "Nu sunt egale" << endl;
    return 0;
}
```

Probleme:

1. Adăugați programului de mai sus operația de aducere la forma ireductibilă a unui număr rațional.
2. Creați metode pentru compararea a două numere raționale. (țineți cont și de numerele negative).
3. Modificați metodele de adunare și scădere a două numere raționale, determinând cel mai mic multiplu comun al numitorilor și amplificând fiecare fracție cu valoarea corespunzătoare (Procedând astfel putem evita în unele cazuri depășirile).
4. Adăugați cod pentru tratarea erorilor: numitori nuli, depășiri.

Laborator 7 - stivă de întregi

Creați un tip de date definit de utilizator care să simuleze o stivă de numere întregi. Într-o stivă datele se introduc și se extrag la un singur capăt după principiul ultimul intrat-primul ieșit (Last In First Out - LIFO). Implementarea se va face utilizând un tablou.

```
// stack.h
#if !defined(STACK_H)
#define STACK_H
#define SIZE 100 // Stiva este limitata la 100 de elemente
enum bool {false, true};
// Numele clasei este decorat cu int pentru
// a scoate in evidenta faptul ca aceasta clasa
// reprezinta o stiva de intregi. Pentru a crea
// o stiva de alt tip trebuie sa copiem tot
// codul si sa facem modificarile de rigoare
// decorand noul nume de tip cu altceva.
// Stiva este un concept general, care nu depinde
// de tipul datelor ce le contine.
// C++ ofera facilitati pentru crearea de tipuri
// parametrizate prin asa numitele "sabioane"
// (template) ce vor fi studiate mai tarziu.
// Sabioanele permit programarea generica.
class Stack_int
{
public:
    // Constructor implicit
    Stack_int();
    // Metode ce constituie interfata stivei
    // Introducerea unui element in stiva
    void push(int);
    // Extragerea unui element din stiva
    void pop();
    // Obținerea elementului din varfului
    // fara a fi extras
    int top();
    // Pentru obținerea numarului de elemente din stiva
    unsigned size();
    bool empty();
private:
    int tab[SIZE];
    unsigned _top; // indica prima pozitie libera din stiva
};
#endif
```

Laborator C++

```
// stack.cpp
#include "stack.h"
#include <iostream.h>
Stack_int::Stack_int()
{
    _top = 0;
}

void Stack_int::push(int x)
{
    if(_top < SIZE - 1)
        tab[_top++] = x;
    else
        cerr << "Stiva plina";
}

void Stack_int::pop()
{
    if(!empty())
        _top--;
}

int Stack_int::top()
{
    if(!empty())
        return tab[_top-1];
    else
        cerr << "Stiva goala" << endl;
    // C++ ofera facilitati pentru tratarea erorilor.
    // Acestea vor fi studiate
    // ulterior astfel incat situatii cum sunt
    // cele din aceasta functie si din functia
    // push, de mai sus, se vor trata mai elegant.
}

unsigned Stack_int::size()
{
    return _top;
}

bool Stack_int::empty()
{
    if(_top == 0)
        return true;
    else
        return false;
}
```

}

```
// stack_drv.cpp
#include "stack.h"
#include <iostream.h>
main()
{
    Stack_int s;
    s.push(1);
    s.push(2);
    s.push(3);
    cout << "Elementul din varful stivei: "
         << s.top() << endl;
    s.pop();
    cout << "Numarul de elemente din stiva: "
         << s.size() << endl;
    s.pop();
    if (s.empty())
        cout << "Stiva goala" << endl;
    else
        cout << "Stiva nu este goala. Elementul "
             "din varful stivei: " << s.top() << endl;
    return 0;
}
```

Probleme:

1. Scrieți un program C++ în care să folosiți o stivă de numere raționale. (Utilizați clasa **Rational**)
2. Modificați implementarea clasei **Stack_int** astfel încât datele să fie păstrate într-o structură dinamică ce se poate redimensiona după necesități și nu într-un tablou alocat în mod static. Mesajul de eroare “Stiva plina” va dispărea și va fi înlocuit cu mesajul “Memorie insuficienta”. Când stiva se umple se face o nouă alocare de memorie pentru mai multe elemente, se copiază datele în noua zonă de memorie și se eliberează vechea zonă de memorie.

Laborator 8 - șiruri de caractere

Lucrul cu șiruri de caractere în C este supus erorilor. Erorile ce pot apărea sunt de mai multe tipuri: utilizarea unor șiruri de caractere ne terminate cu caracterul NUL ('\\0'), accesarea unui caracter dincolo de sfârșitul șirului, returnarea dintr-o funcție a unui șir de caractere local funcției etc. Din acest motiv este utilă crearea unei clase pentru lucrul cu șiruri de caractere. Biblioteca standard C++ conține clasa **string** pentru lucrul cu șiruri de caractere. Utilizarea acestei clase ușurează mult munca programatorilor, peste tot unde s-a folosit tipul **char*** se folosește **string**. Pentru programatorii începători, crearea unei clase pentru șiruri de caractere este un exercițiu util chiar dacă în programe se va folosi clasa **string** din biblioteca standard C++ sau din alte biblioteci pe care le oferă mediul pentru dezvoltarea de aplicații pe care îl folosiți.

Problemă: Creați o clasă pentru lucrul cu șiruri de caractere. Limita pentru dimensiunea unui șir de caractere va fi memoria disponibilă. Un șir de caractere se va putea inițializa în mai multe moduri. Se va crea și constructorul de copiere care se utilizează când: inițializăm o instanță a clasei cu un obiect de același tip, transmitem un obiect ca argument unei funcții prin valoare, returnăm un obiect dintr-o funcție prin valoare. Se vor supraîncărca operatorii: + (pentru concatenarea a două șiruri), << (operatorul de inserție), operatorii relaționali (==, !=, <, >, <=, >=), operatorul [] (pentru obținerea unui caracter din șir), operatorul = (de atribuire).

```
// str.h
#ifndef STR_H
#define STR_H
#include <iostream.h>
enum bool {false, true};
class Str
{
public:
    // C-tor implicit
    Str();
    // C-tor de copiere
    Str(const Str&);
    // C-tor de conversie
    Str(char *);
    // C-tori ce creeaza subsiruri ale unor siruri existente
    Str(const Str&, int, int);
    Str(const Str&, int);
    Str(char *, int, int);
    Str(char *, int);
    Str(char, int); // Repeta caracterul de un numar de ori
    // D-tor
    ~Str();
    // Operatorul de atribuire
    void operator=(const Str&);
    // Operator pentru concatenarea a doua siruri
    friend Str operator+(const Str&, const Str&);
    // Operatorii relationali
```



```
friend bool operator<(const Str&, const Str&);
friend bool operator<=(const Str&, const Str&);
friend bool operator<(const Str&, const Str&);
friend bool operator<=(const Str&, const Str&);
friend bool operator==(const Str&, const Str&);
friend bool operator!=(const Str&, const Str&);
// Operatorul de insertie
friend ostream& operator<<(ostream&, const Str&);
// Operatorul subscript
char &operator[](long);
long size() const;
private:
    char *p;
    long _size;
};
#endif

// str_impl.cpp
#include "str.h"
#include <string.h>
#include <iostream.h>
#include <stdlib.h>
// C-tor implicit = c-tor care poate fi apelat fara argumente.
// Asadar un constructor implicit poate avea argumente dar toate
// trebuie sa aiba valori implicite.
Str::Str()
{
    p = new char[1];
    p[0] = '\0';
    _size = 0;
}
// C-tor de copiere
Str::Str(const Str &s2)
{
    p = new char[s2.size() + 1];
    strcpy(p, s2.p);
    _size = s2._size;
}
Str::Str(char *s)
{
    p = new char[strlen(s) + 1];
    strcpy(p, s);
    _size = strlen(s);
}
Str::Str(const Str &s2, int poz, int lng)
{

```

Laborator C++

```
// Exercițiu:  
// Copiază lng caractere începând de la poziția poz.  
// Verifică dacă se pot copia caracterele.  
}  
Str::Str(const Str &s2, int poz)  
{  
    if(poz+1<=s2.size())  
    {  
        p = new char[s2.size() - poz + 1];  
        strncpy(p, s2.p+poz, s2.size() - poz + 1);  
        _size = s2.size() - poz;  
    }  
    else  
        cerr << "Nu pot crea sirul cu parametrul dat";  
}  
Str::Str(char *s2, int poz, int lng)  
{  
    // Exercițiu:  
    // Copiază lng caractere începând de la poziția poz.  
    // Verifică dacă se pot copia caracterele.  
  
}  
Str::Str(char *s2, int poz)  
{  
    if(poz+1<=strlen(s2))  
    {  
        p = new char[strlen(s2) - poz + 1];  
        strncpy(p, s2+poz, strlen(s2) - poz + 1);  
    }  
    else  
        // Dacă executia programului ajunge in acest loc  
        // obiectul va fi nedefinit. Trebuie gasita o alta  
        // modalitate de tratare a acestui tip de exceptie,  
        // nu doar afisarea unui mesaj de avertizare.  
        // De ex. crearea unui obiect gol, la fel ca si  
        // constructorul implicit.  
        cerr << "Nu pot crea sirul cu parametrul dat";  
}  
Str::Str(char ch, int n)  
{  
    p = new char[n+1];  
    for(int i=0; i<n; i++)  
        p[i] = ch;  
    p[n] = '\0';  
}  
Str::~Str()
```

Laborator C++

```
{
    delete []p;
    p = 0;
}
void Str::operator=(const Str &s2)
{
    // Pentru a preveni auto-asignarea folosim
    // testul de mai jos.
    // this este pointer constant la obiectul pentru
    // care a fost apelata functia
    if(this!=&s2)
    {
        delete []p;
        p = new char[s2.size() + 1];
        strcpy(p, s2.p);
        _size = s2.size();
    }
}
Str operator+(const Str& s1, const Str &s2)
{
    char *s = new char[s1.size() + s2.size() + 1];
    strcpy(s, s1.p);
    strcat(s, s2.p);
    return Str(s);
}
bool operator<(const Str& s1, const Str &s2)
{
    if(strcmp(s1.p, s2.p)<0)
        return true;
    else
        return false;
}
bool operator<=(const Str &s1, const Str &s2)
{
    // Exercițiu
}
bool operator>(const Str &s1, const Str &s2)
{
    // Exercițiu
}
bool operator>=(const Str &s1, const Str &s2)
{
    // Exercițiu
}
bool operator==(const Str &s1, const Str &s2)
{

```

Laborator C++

```
// Exercitiu
}
bool operator!=(const Str &s1, const Str &s2)
{
    // Exercitiu
}

char &Str::operator[](long i)
{
    if(i<_size)
        return p[i];
    else
    {
        cerr << "Incercare de accesare a unui caracter dincolo"
            " de sfarsitul sirului";
        exit(1);
    }
};
long Str::size() const
{
    return _size;
}

ostream& operator<<(ostream& out, const Str &s)
{
    out << s.p;
    return out;
}

//driver.cpp
#include "str.h"
#include <iostream.h>
#include <conio.h>
main()
{
    clrscr();
    Str s1("abcdefg"); // Se utilizeaza c-torul Str(char*);
    Str s2(s1); // Se utilizeaza constructorul de copiere
    Str s3(s1, 2); // Se utilizeaza c-torul Str(Str&,int);
    Str s4 = s3; // Se utilizeaza constructorul de copiere.
                                // Obiectul s4 este creat ceea ce determina
                                // apelul unui constructor. Analog cu Str s4(s3);
    Str s5; // Se utilizeaza constructorul implicit
    s5 = s1; // Se utilizeaza operatorul de atribuire supraincarcat
    cout << "s1 = " << s1 << endl;
    cout << "s2 = " << s2 << endl;
```

```
cout << "s3 = " << s3 << endl;
cout << "s4 = " << s4 << " Lungimea = "
    << s4.size() << endl;
cout << "s5 = " << s5 << endl;
// Operatorul [] l-am declarat ca returnand o referinta
// la char asadar poate fi folosit si ca r-valoare
// si ca l-valoare dupa cum se vede ma jos
cout << "s5[0] = " << s5[0] << endl;
s5[0]='b';
cout << "s5[0] = " << s5[0] << endl;
cout << "s5[s5.size()-1] = " << s5[s5.size()-1] << endl;
// Sintaxa folosita de cel care utilizeaza clasa,
// pentru compararea, concatenarea etc, a doua siruri
// este mult mai simpla si mai intuitiva.
if(s1 < s3)
    cout << "Sirul s1 este mai mic decat s3" << endl;
else
    cout << "Sirul s1 nu este mai mic decat s3" << endl;
// Concatenarea a doua siruri
cout << "s1+s3=" << (s1+s3) << endl;

return 0;
}
```

Exerciții:

1. Completați toate metodele cărora le lipsește implementarea și testați aceste metode.
2. Adăugați o metodă, cu numele **swap** care să permită interschimbarea conținutului a două obiecte de tip **Str**, folosind sintaxa: **s1.swap(s2)**.
3. Adăugați o metodă cu numele **empty** care să returneze **true** dacă un șir nu conține nici un caracter și **false** în caz contrar.
4. Adăugați una sau mai multe metode **erase** supraîncărcate, care să permită ștergerea unei porțiuni dintr-un șir.
5. Adăugați una sau mai multe metode **insert** supraîncărcate care să permită inserarea într-un obiect de tip **Str** a unui șir sau subșir de caractere.
6. Adăugați una sau mai multe metode **replace** care să permită înlocuirea unei porțiuni dintr-un șir cu un alt șir de caractere sau cu un subșir al unui șir de caractere.
7. Adăugați una sau mai multe metode **find** care să permită căutarea într-un șir a unui subșir.
8. Adăugați metodele **upper** și **lower** care transformă toate literele dintr-un șir în litere mari respectiv mici.
9. Adăugați o metodă **reverse** care inversează caracterele dintr-un șir (caracterul de pe prima poziție se interschimbă cu cel de pe ultima, al doilea cu cel de pe penultima ș.a.m.d.)

Laborator 9 - ora

Creați un tip de date definit de utilizator (**Time**) pentru reprezentarea timpului sub forma (ora:minutul:secunda.sutimea). Pentru acest tip de date implemetați operația de adunare a doi timpi. Se vor utiliza două modalități de inițializare a unui obiect de tip **Time**:

```
Time t1(2, 34, 05, 89);
Time t2("2:34:05.89");
```

Rezolvare:

```
//main.cpp
#include <mtime.h>
int main()
{
    MTime t1(2, 30, 59,10);
    MTime t2("2.30:59.11");
    MTime t3;
    t3 = t1 + t2;
    cout << endl << t1 << '+' << endl << t2 << '='
         << endl << t3 << endl;
    if(t1==t2)
        cout << "t1 == t2" << endl;
    else
        cout << "t1!=t2" << endl;
    return 0;
}

//mtime.h
#ifndef MTIME_H
#define MTIME_H
#include <iostream.h>
enum bool {false, true};
class MTime
{
public:
    MTime();
    MTime(unsigned, unsigned, unsigned, unsigned);
    MTime(char *);
    bool operator==(const MTime &);
    bool operator!=(const MTime &);
    bool operator<(const MTime &);
    bool operator>(const MTime &);
    bool operator<=(const MTime &);
    bool operator>=(const MTime &);
    MTime operator+(const MTime &);
    friend ostream & operator<<(ostream &, const MTime &);
private:
```

Laborator C++

```
        unsigned ora;
        unsigned minut;
        unsigned secunda;
        unsigned sutime;
};
#endif
//mtime.cpp
#include "mtime.h"
#include <iostream.h>
#include <strstream.h>
#include <iomanip.h>
MTime::MTime(): ora( 0 ), minut( 0 ), secunda( 0 ), sutime( 0 )
{
}

MTime::MTime(unsigned ora, unsigned min, unsigned sec, unsigned sut )
{
    MTime::ora = ora;
    minut = min;
    secunda = sec;
    sutime = sut;
}

MTime::MTime(char *str )
{
    istrstream t(str);
    t >> ora;
    t.ignore();
    if (t.peek()=='0')
        t.ignore();
    t >> minut;
    t.ignore();
    if (t.peek()=='0')
        t.ignore();
    t >> secunda;
    t.ignore();
    if (t.peek()=='0')
        t.ignore();
    t >> sutime;
}

MTime MTime::operator+(const MTime & op2 )
{
    MTime temp; int c;
    temp.sutime = (sutime + op2.sutime) % 100;
    c = (sutime + op2.sutime) / 100;
```

Laborator C++

```
temp.secunda = (secunda + op2.secunda + c) % 60;
c = (secunda + op2.secunda + c) / 60;
temp.minut = (minut + op2.minut + c) % 60;
c = (minut + op2.minut + c) / 60;
temp.ora = ora + op2.ora + c;
return temp;
}
ostream & operator<<(ostream & out, const MTime & t )
{
    out << t.ora << ':'
        << setw(2) << setfill('0') << t.minut << ':'
        << setw(2) << setfill('0') << t.secunda << '.'
        << setw(2) << setfill('0') << t.sutime;
    out << setfill(' ');
    return out;
}

bool MTime::operator==(const MTime &t )
{
    if(ora == t.ora && minut == t.minut &&
        secunda == t.secunda && sutime == t.sutime)
        return true;
    else
        return false;
}

bool MTime::operator!=(const MTime &t )
{
    if(ora != t.ora || minut != t.minut ||
        secunda != t.secunda || sutime != t.sutime)
        return true;
    else
        return false;
}

bool MTime::operator<(const MTime &t )
{
    if(ora < t.ora)
        return true;
    else
        if(ora > t.ora)
            return false;
        else
            if(minut < t.minut)
                return true;
            else
                if(minut > t.minut)
                    return false;
```



```
        else
            if(secunda < t.secunda)
                return true;
            else
                if(secunda > t.secunda)
                    return false;
                else
                    if(sutime < t.sutime)
                        return true;
                    else
                        return false;
    }
    bool MTime::operator>(const MTime &t )
    {
        if( (*this!=t) && !(*this < t) )
            return true;
        else
            return false;
    }
    bool MTime::operator<=(const MTime &t )
    {
        if( !(*this > t) )
            return true;
        else
            return false;
    }
    bool MTime::operator>=(const MTime &t )
    {
        if( !(*this < t) )
            return true;
        else
            return false;
    }
}
```

Probleme:

1. Supraîncărcați operatorul – pentru clasa **MTime** pentru a calcula diferența dintre doi timpi. Operatorul va returna prin valoare un obiect de tip **MTime**.
2. Implementați o metodă care să determine timpul scurs de la o anumită ora până la ora curentă. “Ora curentă” este cea dată de ceasul intern al calculatorului.

Laborator 10 - data

Creați un tip de date definit de utilizator (**Date**) pentru reprezentarea datei sub forma (zi.luna.an). Pentru acest tip de date implementați o operație pentru determinarea numărului de zile dintre două date. Se vor utiliza două modalități de inițializare a unui obiect de tip

Date:

```
Date d1(3, 4, 1940);  
Date d2("17.11.1997");
```

```
//main.cpp  
#include "date.h"  
int main()  
{  
    Date d1(12, 12, 2030);  
    Date d2("13.12.2040");  
    cout << endl << d1 << endl << d2 << endl;  
    cout << "d1-d2 = " << d1-d2 << endl;  
    cout << "d1-d2 = " << d2-d1 << endl;  
    if( (d1-d2) == (d2-d1) )  
        cout << "OK!";  
    return 0;  
}  
  
//date.h  
#if !defined(DATE_H)  
#define DATE_H  
#include <iostream.h>  
enum bool {false, true};  
class Date  
{  
public:  
    Date();  
    Date(unsigned, unsigned, unsigned);  
    Date(char *);  
  
    long operator-(const Date&);  
  
    bool operator==(const Date&);  
    bool operator!=(const Date&);  
    bool operator<(const Date&);  
    bool operator>(const Date&);  
    bool operator<=(const Date&);  
    bool operator>=(const Date&);  
    friend ostream& operator<<(ostream&, const Date&);  
private:  
    unsigned zi, luna, an;
```

Laborator C++

```
};  
#endif  
  
//date.cpp  
#include "date.h"  
#include <strstream.h>  
#include <iomanip.h>  
Date::Date(): zi(0), luna(0), an(0)  
{  
}  
  
Date::Date(unsigned z, unsigned l, unsigned a):  
    zi(z), luna(l), an(a)  
{  
}  
  
Date::Date(char *s)  
{  
    istrstream is(s);  
    is >> zi;  
    is.ignore();  
    if(is.peek() == '0')  
        is.ignore();  
    is >> luna;  
    is.ignore();  
    is >> an;  
}  
  
long Date::operator-(const Date& d)  
{  
    Date d1, d2;  
    if( *this < d )  
    {  
        d1 = d;  
        d2 = *this;  
    }  
    else  
    {  
        d1 = *this;  
        d2 = d;  
    }  
    long nr = 0;  
    while(d1 > d2)  
    {  
        nr++;  
        if( d1.zi>1)
```

```
        d1.zi--;
    else
        switch(d1.luna){
            case 1:
                d1.zi = 31;
                d1.luna = 12;
                d1.an--;
                break;
            case 2: case 4: case 6: case 8: case 9: case 11:
                d1.zi = 31;
                d1.luna--;
                break;
            case 5: case 7: case 10: case 12:
                d1.zi = 30;
                d1.luna--;
                break;
            case 3:
                d1.luna--;
                if(d1.an%400 == 0 || (d1.an%100!=0 && d1.an%4==0))
                    d1.zi = 29;
                else
                    d1.zi = 28;
        }
    }
    return nr;
}

bool Date::operator==(const Date& d)
{
    if(zi == d.zi && luna == d.luna && an == d.an)
        return true;
    else
        return false;
}

bool Date::operator!=(const Date& d)
{
    if( !(*this == d) )
        return true;
    else
        return false;
}

bool Date::operator<(const Date& d)
{
    if(an < d.an)
```

Laborator C++

```
        return true;
    else
        if(an > d.an)
            return false;
        else
            if(luna < d.luna)
                return true;
            else
                if(luna > d.luna)
                    return false;
                else
                    if(zi < d.zi)
                        return true;
                    else
                        return false;
    }

    bool Date::operator>(const Date& d)
    {
        if( (*this!=d) && !(*this<d) )
            return true;
        else
            return false;
    }

    bool Date::operator<=(const Date& d)
    {
        if( (*this==d) || (*this<d) )
            return true;
        else
            return false;
    }

    bool Date::operator>=(const Date& d)
    {
        if( (*this==d) || (*this>d) )
            return true;
        else
            return false;
    }

    ostream& operator<<(ostream& out, const Date& d)
    {
        out << d.zi << '.'
            << setw(2) << setfill('0') << d.luna << '.'
            << setw(4) << d.an;
```

Laborator C++

```
out << setfill(' ');  
                                return out;  
  
}
```

Probleme:

1. Pentru clasa **Date** de mai sus implementați o metodă care să determine vârsta unei persoane în an, luni și zile. Data curentă va fi cea dată de ceasul intern al calculatorului.
2. Pentru clasa **Date** de mai sus implementați o metodă care să determine în ce zi a săptămânii cade o anumită dată.

Laborator 11 - matrice

Creați un tip de date definit de utilizator pentru reprezentarea matricilor de numere reale. Vor fi implementate operațiile de adunare, scădere și înmulțire a două matrici. Se va testa dacă operațiile se pot executa. Operația de accesare a unui element al matricii va fi realizată supraîncărcând operatorul apel de funcție cu doi parametri (folosind sintaxa `m(1, 2)` vom accesa elementul de pe linia 2 coloana 3). Acest operator trebuie să returneze referință la `double` pentru a putea fi folosit și ca l-valoare și ca r-valoare. Dacă s-ar returna doar `double` atunci expresia `m(1, 2)` poate fi folosită doar ca r-valoare (în partea dreaptă a semnului egal).

```
// main.cpp
#include "matrice.h"
#include <iostream.h>
#include <stdlib.h>
#define MAX 4
int main()
{
    randomize();
    Matrice m1(2, 2), m2(2, 2);
    unsigned i, j;
    // Elementele celor doua matricile generam aleator
    for(i=0; i<2; i++)
        for(j=0; j<2; j++)
            m1(i, j) = random(MAX);
    cout << "m1" << endl << m1;

    for(i=0; i<2; i++)
        for(j=0; j<2; j++)
            m2(i, j) = random(MAX);
    cout << "m2" << endl << m2;
    // In expresia de mai jos cu toate ca apare operatorul de atribuire nu va fi apelata
    // functia operator= supraincercata pentru clasa Matrice. In acest caz este apelat
    // constructorul de copiere.
    Matrice m = m1 + m2; // echivalent cu Matrice m( m1 + m2 );
    cout << "m1+m2" << endl << m;

    // In expresiile de mai jos obiectul m exista deja (a fost creat si initializat mai sus) prin
    // urmare va fi apelata functia operator= supraincercata pentru clasa Matrice
    m = m1 - m2;
    cout << "m1-m2" << endl << m;

    m = m1 * m2;
    cout << "m1*m2" << endl << m;
    return 0;
}
```

Laborator C++

```
// matrice.h
#ifndef MATRICE_H
#define MATRICE_H
#include <iostream.h>
class Matrice
{
public:
    Matrice();
    Matrice(unsigned);
    Matrice(unsigned, unsigned);
    Matrice(const Matrice &);
    ~Matrice();

    // Funcțiile operator +, -, *, () pot fi const pentru ca ele nu modifica operandul
    // pentru care sunt apelate.
    // Cand facem o functie const presupunem ca acea functie nu trebuie sa modifice
    // obiectul pentru care este apelata. In aceste functii tipul lui this va fi:
    // const Matrice * const adica this este pointer constant la obiect constant.
    // In funcțiile ce nu au atributul const tipul lui this este Matrice *const adica this
    // este pointer constant la un obiect ce poate fi modificat.
    Matrice operator+(const Matrice &) const;
    Matrice operator-(const Matrice &) const;
    Matrice operator*(const Matrice &) const;
    Matrice& operator=(const Matrice &);
    double& operator()(unsigned, unsigned) const;
    friend ostream& operator<<(ostream&, const Matrice&);

private:
    double *p;
    unsigned l, c;
    // O matrice bidimensionala poate fi reprezentata ca un sir unidimensional, pe linii sau pe
    // coloane. In acest program am optat pentru reprezentarea pe linii.
    //Matricea |1  2|
    //          |3  4|
    // se reprezinta ca sir unidimensional pe linii sub forma 1 2 3 4
    // Pentru a accesa elementul de pe pozitia (i, j) se foloseste formula i*c + j unde c este
    // numarul de coloane al matricii.
};
#endif

// matrice.cpp
#include "matrice.h"
#include <iostream.h>
// Constructor implicit
Matrice::Matrice()
{
    l = c = 0;
}
```


Laborator C++

```
p = 0;
}

Matrice::Matrice(unsigned n)
{
    l = c = n;
    p = new double[ n * n ];
}

Matrice::Matrice(unsigned l, unsigned c)
{
    Matrice::l = l;
    Matrice::c = c;
    p = new double[ l * c ];
}

// Constructor de copiere
Matrice::Matrice(const Matrice &m)
{
// Aloc memorie pentru noul obiect ...
    p = new double[ m.l * m.c ];
// ... initializez numarul de linii si coloane
    l = m.l; c = m.c;
// ... si copiez valorile matricii in noua matrice.
    unsigned i, j;
    for(i=0; i<l; i++)
        for(j=0; j<c; j++)
            p[ i*c + j ] = m.p[ i*c + j ];
}

// Destructor
Matrice::~~Matrice()
{
    delete []p;
}

// Operatorii +, - si * nu trebuie sa modifice operanzii asupra carora opereaza, asa ca
// vor returna un obiect de tip Matrice prin valoare, ceea ce va cauza apelul
// constructorului de copiere.
// Apelul constructorului de copiere este o operatie costisitoare, dar trebuie sa acceptam
// compromisul daca vrem sa folosim in programul principal o sintaxa simpla si intuitiva.
Matrice Matrice::operator+(const Matrice &m) const
{
    Matrice temp;
    // Adun doua matrici doar daca se poate
    if(l == m.l && c == m.c)
```

Laborator C++

```
{
    temp.p = new double[l*c];
    temp.l = l;
    temp.c = c;
    unsigned i, j;
    for(i=0; i<l; i++)
        for(j=0; j<c; j++)
            temp.p[ i*c + j ] = p[ i*c + j ] + m.p[ i*c + j ];
}
return temp;
}
```

Matrice Matrice::operator-(const Matrice &m) const

```
{
    Matrice temp;
    // Scad doua matrici doar daca se poate
    if(l == m.l && c==m.c)
    {
        temp.p = new double[l*c];
        temp.l = l;
        temp.c = c;

        unsigned i, j;
        for(i=0; i<l; i++)
            for(j=0; j<c; j++)
                temp.p[ i*c + j ] = p[ i*c + j ] - m.p[ i*c + j ];
    }
    return temp;
}
```

Matrice Matrice::operator(const Matrice &m) const*

```
{
    Matrice temp;
    if( c != m.l)
        return temp;
    temp.p = new double[ l*m.c ];
    temp.l = l;
    temp.c = m.c;
    unsigned i, j, k;
    double aux = 0;
    for(i=0; i<l; i++)
        for(j=0; j<m.c; j++)
        {
            aux = 0;
            for(k=0; k<c; k++)
                aux += p[ i*c + k ] * m.p[ k*m.c + j ];
        }
}
```

Laborator C++

```
        temp.p[ i*m.c + j] = aux;
    }
    return temp;
}

// Operatorul de atribuire modifica operandul din partea stanga a semnului egal asa ca
// aici putem returna o referinta la obiectul care a fost modificat.
//Aceasta functie poate sa nu returneze nimic (void) dar in acest caz expresia ce contine
// operatorul de atribuire nu va avea nici o valoare. Nu se va putea folosi nici o
// expresie care foloseste rezultatul evaluarii atribuirii.
Matrice& Matrice::operator=(const Matrice &m)
{
    if( this == &m )
        return *this;
    delete []p;
    p = new double[ m.l * m.c ];
    l = m.l;
    c = m.c;
    unsigned i, j;
    for(i=0; i<l; i++)
        for(j=0; j<c; j++)
            p[ i*c + j ] = m.p[ i*c + j ];
    return *this;
}

double& Matrice::operator()(unsigned lin, unsigned col) const
{
    return p[ lin*c + col ];
}

ostream& operator<<(ostream& out, const Matrice& m)
{
    unsigned i, j;
    for(i=0; i<m.l; i++)
    {
        for(j=0; j<m.c; j++)
            out << m.p[i*m.c+j] << " ";
        out << endl;
    }
    out << endl;
    return out;
}
```

Probleme:

- 1) Adăugați clasei **Matrice** operația de transpunere a unei matrici.

Laborator C++

- 2) Adăugați clasei **Matrice** operația de calculare a determinantului unei matrici, pentru matricile pătratice.
- 3) Adăugați clasei **Matrice** operația de inversare a unei matrici.

Laborator 12 - listă de persoane

Creați o listă de persoane. Vor fi implementate două clase: una se va numi **Persoana** și va avea ca date membre numele, prenumele și vârsta unei persoane, cealaltă se va numi **ListaDePersoane** și va avea ca date membru un pointer la obiect de tip **Persoana**, numărul actual de Persoane din lista și numărul maxim de obiecte ce pot fi păstrate în listă la un moment dat. Introducerea unei noi persoane în listă se va face cu metoda **Add**. Când lista se umple se va face o re-alocare de memorie pentru a permite introducerea de noi persoane în listă. Țineți cont de faptul că re-alocarea memoriei, pentru a păstra o nouă listă este o operație costisitoare, întrucât trebuie copiate toate persoanele din lista veche (cea plină) într-o altă zonă de memorie.

```
// pers.h
#ifndef PERS
#define PERS
#include <iostream.h>
class Persoana
{
public:
    Persoana(char *, char *, unsigned short);
    Persoana();
    Persoana(const Persoana&);
    Persoana& operator=(const Persoana&);
    friend ostream& operator<<(ostream&, const Persoana&);
    ~Persoana();
private:
    char * nume;
    char * prenume;
    unsigned short varsta;
};
#endif

// pers.cpp
#include "pers.h"
#include <string.h>
#include <iomanip.h>
Persoana::Persoana(char *nume, char *prenume, unsigned short varsta)
{
    //daca numele unei date membre a clasei coincide cu numele //unui argument al unei
    //functii membre atunci pentru a elimina //ambiguitatea se precede numele datei membre de
    //numele clasei //si operatorul de rezolutie a domeniului de valabilitate //(scope resolution
    //operator)
    Persoana::nume = new char[strlen(nume)+1];
    strcpy(Persoana::nume, nume);

    Persoana::prenume = new char[strlen(prenume)+1];
```

Laborator C++

```
    strcpy(Persoana::prenume, prenume);
    Persoana::varsta = varsta;
}

//Constructor implicit
Persoana::Persoana()
{
    nume = prenume = 0;
    varsta = 0;
}

//C-tor de copiere. Este folosit cand transmitem un obiect //prin valoare unei functii, cand
returnam un obiect prin //valoare dintr-o functie sau cand initializam un obiect cu un
//altul de acelasi tip
Persoana::Persoana(const Persoana& p)
{
    nume = new char[strlen(p.nume) + 1];
    strcpy(nume, p.nume);
    prenume = new char[strlen(p.prenume) + 1];
    strcpy(prenume, p.prenume);
    varsta = p.varsta;
}

//Supraincarcam operatorul de inserare
ostream& operator<<(ostream& out, const Persoana& p)
{
    out.setf(ios::left);
    out << setw(20) << p.nume
    << setw(20) << p.prenume
    << p.varsta << endl;
    return out;
}

Persoana::~Persoana()
{
    delete []nume;
    delete []prenume;
}

//Obs. 1 Operatorul de atribuire trebuie scris in asa fel //incat sa previna de auto-asignarea.
//Obs. 2 Daca ar returna void nu am putea folosi o expresie de //genul a=b=c; expresia b=c;
nu ar avea nici o valoare.
Persoana& Persoana::operator=(const Persoana& pers)
{
    if(this == &pers)
        return *this;
    else
```

Laborator C++

```
{
    delete []nume;
    delete []prenume;
    nume = new char[strlen(pers.nume) + 1];
    strcpy(nume, pers.nume);
    prenume = new char[strlen(pers.prenume) + 1];
    strcpy(prenume, pers.prenume);
    varsta = pers.varsta;
}
}
```

```
//lst.h
#ifndef LIST_PERS
#define LIST_PERS
#include "pers.h"
//Decoram numele clasei cu "DePersoana" pentru a scoate in //evidenta foarte clar ce fel
de lista am creat. Dar dupa cum //se vede din modul in care este implementate (mai jos
//lst.cpp), aceasta clasa poate fi modificata foarte usor //pentru a crea liste de orice alt tip.
Programarea generica //(cuv. cheie template) ne va permite sa creem o singura clasa
//parametrizata.
```

```
class ListaDePersoane
{
public:
    ListaDePersoane();
    ~ListaDePersoane();
    void Add(const Persoana&);
    friend ostream& operator<<(ostream&, const ListaDePersoane&);
    unsigned Numar() const;
    unsigned MaxNumar() const;
private:
    unsigned numar;
    unsigned maxnumar;
    Persoana *p;
};
#endif
```

```
// lst.cpp
#include "lst.h"
#include <math.h>
#include <iomanip.h>
#define INCREMENT 3 // numarul cu care creste dimensiunea //listei la o re-alocare
ListaDePersoane::ListaDePersoane()
{
    maxnumar = INCREMENT;
    numar = 0;
    p = new Persoana[maxnumar]; // initial lista poate contine //INCREMENT elemente
```

Laborator C++

```
}
```

```
ListaDePersoane::~~ListaDePersoane()
```

```
{  
    delete []p; // operatorul delete [] va apela destructorul //pentru fiecare obiect de tip  
    Persoana din lista  
    numar = maxnumar = 0;  
}
```

```
void ListaDePersoane::Add(const Persoana& pers)
```

```
{  
    Persoana* q;  
    if(numar < maxnumar-1) // Daca mai am loc pentru inca o  
                           //persoana  
    {  
        p[numar++] = pers; // o adaug  
    }  
    else // altfel fac o re-alocare  
    {  
        q = new Persoana[maxnumar += INCREMENT];  
        for(int i=0; i<numar; i++)  
            q[i] = p[i];  
        q[i] = pers;  
        numar++;  
        delete []p;  
        p = q;  
    }  
}
```

```
ostream& operator<<(ostream& out, const ListaDePersoane& lst)
```

```
{  
    for(int i=0; i<lst.numar; i++)  
    {  
        out.setf(ios::right);  
        //functia log10 imi permite sa determin numarul de cifre al //unui numar intreg  
        out << setw(int(log10(lst.numar))+1) << i+1 << " " << lst.p[i];  
    }  
    return out;  
}
```

```
unsigned ListaDePersoane::Numar() const
```

```
{  
    return numar;  
}
```

```
unsigned ListaDePersoane::MaxNumar() const
```


Laborator C++

```
{
    return maxnumar;
}

//main.cpp

#include "lst.h"
#include "pers.h"
#include <iostream.h>
int main()
{
    ListaDePersoane lst;
    Persoana p("Georgescu", "Ion", 25);
    lst.Add(p);
    lst.Add(Persoana("Petrescu", "Gheorghe", 29));
    lst.Add(Persoana("Ionescu", "Marin", 26));
    lst.Add(Persoana("Popescu", "Petre", 28));
    cout << lst;
    cout << "Numarul de persoane din lista: " << lst.Numar();
    return 0;
}
```

Probleme:

1. Adăugați clasei **ListaDePersoane** o metodă pentru ștergerea întregii liste.
2. Adăugați clasei **ListaDePersoane** facilitatea de căutare în listă a unei persoane căreia îi știm numele.
3. Adăugați clasei **ListaDePersoane** o metodă de sortare a listei în ordine alfabetică. Această sortare va fi ineficientă pentru că vor trebui inversate obiecte de tip **Persoana**.
4. Pentru ca sortarea să devină eficientă trebuie modificată implementarea clasei **ListaDePersoane** într-o listă înlănțuită. Faceți această modificare.
5. Dacă am modificat implementarea, listei într-o listă înlănțuită atunci putem elimina obiecte din listă foarte eficient (fără să facem o re-alocare). Adăugați o metodă pentru eliminarea unui obiect din listă. Metoda va primi ca argument numele persoanei ce va fi eliminată.
6. Adăugați clasei **ListaDePersoane** un constructor de copiere.

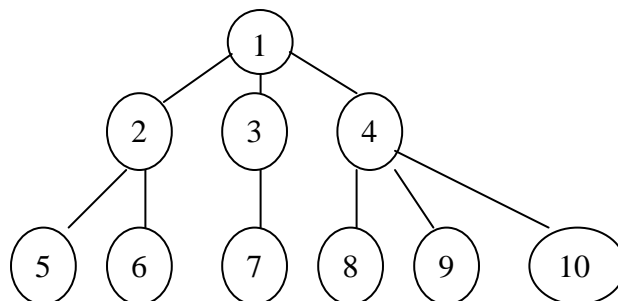
Laborator 13 - arbore oarecare

Problemă:

Scrieți un program C++ pentru crearea unui arbore oarecare și parcurgeți acest arbore în adâncime și în lățime.

Un arbore oarecare este un arbore ale cărui noduri pot avea oricâți descendenți.

De ex:



Pentru a parcurge în adâncime un arbore se parcurge prima dată rădăcina după care se parcurg toți fii, acesta fiind un proces recursiv.

Parcurgerea în adâncime a arborelui de mai sus: 1, 2, 5, 6, 3, 7, 4, 8, 9, 10.

Parcurgerea în lățime a unui arbore înseamnă de fapt parcurgere pe nivele. Pentru aceasta vom folosi o coadă. O coadă este o structură liniară de date în care intrările se fac la un capăt iar ieșirile la celălalt capăt. Algoritmul pentru parcurgerea în lățime a unui arbore poate fi dedus din listingul de mai jos.

Parcurgerea în lățime a arborelui de mai sus: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

Programul care rezolvă problema de mai sus:

```
// ar.cpp
#include <iostream.h>
#include <stdlib.h>
class nod
{
public:
    char inf; // Informatia poate fi orice alta structura de
              // date
    int dim;
    nod *tab[100]; // e indicat a se folosi containerul
                  // vector din biblioteca standard C++
};
nod * crearb(); // Functia pentru crearea arborelui
void padinc(nod *); // Functia pentru parcurgerea in adincime
void plat(nod *); // Functia pentru parcurgerea in latime

class queue{ // e indicat sa se foloseasca containerul
```

```

                                //queue din biblioteca standard C++
public:
    queue(): size(0), end(0)
    {
    }
    void pop(); //functia elimina un element din coada
    void push(nod *); //aduaga un element in coada
    nod * front(); // returneaza un element din coada
                        //fara sa-l elimine
    int empty();
private:
    int size, end;
    nod *t[100]; // structura de date folosita pentru
                // implementarea cozii poate fi o lista dinamica
};
void queue::pop()
{
    if (end>0)
        end--;
    else
    {
        cerr << "queue empty" << endl;
        exit(1);
    }
}
void queue::push(nod *p)
{
    for(int i=end; i>0; i--)
        t[i]=t[i-1];
    t[0]=p;
    end++;
}

nod * queue::front()
{
    if (end>0)
        return t[end-1];
    else
    {
        cerr << "queue empty" << endl;
        exit(1);
    }
}
int queue::empty()
{
    if (end == 0)

```

Laborator C++

```
        return 1;
    else
        return 0;
}

int main()
{
    //Creez arborele
    nod * root;
    root = crearb();
    cout << "Parcurgere in adincime:" << endl;
    padinc(root);
    cout << endl;
    cout << "Parcurgere in latime:" << endl;
    plat(root);
    return 0;
}

nod * crearb()
{
    char ch;
    int n,i;
    nod *p;
    cout << "Inf="; cin >> ch;
    p = new nod;
    p->inf = ch;
    cout << "Cati fii are?"; cin >> n;
    p->dim = n;
    for(i=0; i < n; i++)
        p->tab[i] = crearb();
    return p;
}

// Parcurgerea in adincime este de fapt parcurgerea in
// preordine
//se parcurge radacina urmata de fii de la stanga la dreapta
void padinc(nod *p)
{
    int i;
    // if(p!=NULL) { testul este inutil
        cout << p->inf;
        for(i=0; i<p->dim; i++)
            padinc(p->tab[i]);
    // }
}
```

Laborator C++

```
// Parcurgerea in latime utilizeaza structura de coada  
// Algoritmul este nerecursiv si poate fi inteles destul de  
// simplu din codul de mai jos;  
void plat(nod *p)  
{  
    queue Q; nod *t; int i;  
    Q.push(p);  
    while (!Q.empty())  
    {  
        t = Q.front();  
        Q.pop();  
        cout << t->inf;  
        for(i=0; i<t->dim; i++)  
            Q.push(t->tab[i]);  
    }  
}
```

Probleme:

1. Dacă, în mediul pentru dezvoltarea de aplicații pe care îl folosiți, aveți la dispoziție biblioteca standard C++, modificați programul de mai sus astfel încât să folosiți clasa **queue** (pentru coadă) și clasa **vector** (pentru fii unui nod) din această bibliotecă.

Laborator 14 - numere mari

Problemă:

Scrieți un program C++ care calculează primele 100 de elemente din șirul lui Fibonacci.

Rezolvare:

Numerele cu care trebuie să operăm în această problemă sunt numere întregi cu multe cifre, numere ce nu se pot reprezenta folosind tipurile fundamentale de numere întregi furnizate de C++ (cu long int putem reprezenta numere întregi cuprinse între -2^{31} și $2^{31}-1$ adică $2^{31} = 2147483648$). Din acest motiv vom crea un tip de date definit de utilizator cu numele NumarMare. Un număr mare (în baza 10) este o listă de cifre cuprinse între 0 și 9. Relația dintre NumarMare și listă este de tip “este o” ceea ce corespunde relației de moștenire public în C++. Prin urmare vom crea o listă dublu înălțuită pentru care vom implementa o serie de operații cu lista, ce sunt esențiale pentru problema pe care o rezolvăm. Clasa **List** va fi clasă de bază pentru clasa **NumarMare**. Pentru clasa **NumarMare** vom implementa operația de adunare a două numere mari.

```
// main.cpp
#include "bignum.h"
#include <fstream.h>
int main()
{
    // Programul va scrie toate datele in fisierul
    // fib.dat
    ofstream file("fib.dat");

    // Secvente ce testeaza functionalitatea clasei
    // Crearea unui obiect si initializarea lui
    NumarMare n1("123456"), n2;

    // Copierea numarului la un stream de iesire
    file << n1 << endl;

    // Operatorul de atribuire a fost supraincarcat
    // doar pentru clasa List, cea care contine datele
    n2 = n1;
    file << n2 << endl;

    NumarMare n3("8"), n4("13");
    file << n3 << '+' << n4 << '=';

    // Pentru clasa NumarMare a fost supraincarcat
    // doar operatorul += ceea ce e suficient pentru
    // problema pe care o rezolvam
    n3+=n4;
    file << n3 << endl;
```

Laborator C++

```
// Secventa pentru calculul primelor  
// 100 de numere din sirul lui Fibonnaci  
NumarMare f1("0"), f2("1"), f("0");  
long i;  
for(i=1; i<100; i++)  
{  
    f1+= f2;  
    f = f1;  
    f1 = f2;  
    f2 = f;  
    file << i+1 << " " << f << endl;  
}  
return 0;  
}
```

```
//node.h  
#if !defined(NODE_H)  
#define NODE_H  
#include <stdlib.h>  
struct Node  
{  
    Node(int c): ch(c)  
    {  
        next = previous = NULL;  
    }  
    int getCh()  
    {  
        return ch;  
    }  
    int ch;  
    Node *next, *previous;  
};  
#endif
```

```
//list.h  
#if !defined(LIST_H)  
#define LIST_H  
#include "node.h"  
class List  
{  
public:  
    List();  
    List(char *);  
    ~List();  
    void pushBack(Node *);  
};
```

Laborator C++

```
void pushFront(Node *);
void moveFront();
void moveBack();
Node* getBack();
Node* getFront();
Node* getPrevious();
Node* getNext();
unsigned size() const;
void operator=(List&);
void clear();
protected:
    Node *front, *back, *point;
    unsigned _size;
};
#endif

// list.cpp
#include "list.h"
#include "node.h"
#include <string.h>
List::List()
{
    _size = 0;
    front = back = point = NULL;
}

List::List(char *s)
{
    _size = strlen(s);
    front = back = point = NULL;
    if(_size>0)
    {
        for(int i=0; i<_size; i++)
        {
            Node *n = new Node(s[i]-'0');
            pushBack(n);
        }
    }
}

void List::operator=(List &lst)
{
    Node *p;
    clear();
    lst.moveFront();
```


Laborator C++

```
point = back = front = new Node((lst.getFront()->ch);  
while(lst.getNext()!=NULL)  
{  
    p = new Node((lst.point)->ch);  
    back->next = p;  
    p->previous = back;  
    back = p;  
}  
}
```

```
List::~~List()  
{  
    Node *p;  
    while(front!=NULL)  
    {  
        p = front->next;  
        delete front;  
        front = p;  
    }  
}
```

```
void List::pushBack(Node *n)  
{  
    if(front == NULL && back == NULL)  
        front = back = n;  
    else  
    {  
        back->next = n;  
        n->previous = back;  
        back = n;  
    }  
}
```

```
void List::pushFront(Node *n)  
{  
    if(front == NULL && back == NULL)  
        front = back = n;  
    else  
    {  
        n->next = front;  
        front->previous = n;  
        front = n;  
    }  
}
```

Laborator C++

```
void List::moveFront()  
{  
    point = front;  
}
```

```
void List::moveBack()  
{  
    point = back;  
}
```

```
Node* List::getBack()  
{  
    return back;  
}
```

```
Node* List::getFront()  
{  
    return front;  
}
```

```
unsigned List::size() const  
{  
    return _size;  
}
```

```
Node* List::getPrevious()  
{  
    if(point==NULL)  
        return NULL;  
    if(point->previous != NULL)  
        point = point->previous;  
    else  
        point = NULL;  
    return point;  
}
```

```
Node* List::getNext()  
{  
    if(point==NULL)  
        return NULL;  
    if(point->next != NULL)  
        point = point->next;  
    else  
        point = NULL;  
    return point;  
}
```

Laborator C++

```
}
```

```
void List::clear()
{
    Node *p;
    while(front!=NULL)
    {
        p = front->next;
        delete front;
        front = p;
    }
    front = back = point = NULL;
}
```

```
//bignum.h
#ifndef NUMAR_MARE
#define NUMAR_MARE
#include "list.h"
#include <iostream.h>
class NumarMare: public List
{
public:
    NumarMare();
    NumarMare(char *);
    ~NumarMare();
    void operator+=(NumarMare &);
    friend ostream& operator<<(ostream &, NumarMare &);
};
#endif
```

```
//bignum.cpp
#include "bignum.h"
NumarMare::NumarMare(): List()
{
}

NumarMare::NumarMare(char *s): List(s)
{
}
NumarMare::~~NumarMare()
{
}
ostream& operator<<(ostream &out, NumarMare &n)
{
    n.moveFront();
    if(n.point!=NULL)
```

Laborator C++

```
        out << (n.point)->ch;
    else
        return out;
    while(n.getNext()!=NULL)
        out << (n.point)->ch;
    return out;
}

void NumarMare::operator+=(NumarMare &n2)
{
    moveBack();
    n2.moveBack();
    int carry = 0, temp, op1, op2;
    Node *p;
    while(point!=NULL || n2.point!=NULL)
    {
        if (point!=NULL)
            op1 = point->ch;
        else
            op1 = 0;
        if (n2.point!=NULL)
            op2 = (n2.point)->ch;
        else
            op2 = 0;
        temp = (op1 + op2 + carry) % 10;
        carry = (op1 + op2 + carry) / 10;
        if (point!=NULL)
            point->ch = temp;
        else
        {
            p = new Node(temp);
            pushFront(p);
        }
        getPrevious();
        n2.getPrevious();
    }

    if(carry!=0)
    {
        p = new Node(carry);
        pushFront(p);
    }
}
```

Probleme:

1. Extindeți programul de mai sus pentru a putea înmulți un număr mare cu o cifră. Veți introduce în clasa **NumarMare** o metodă cu prototipul `void operator*=(char) ;` unde caracterul va trebui să reprezinte o cifră.
 2. Calculați 1000! Pentru aceasta va trebui introdusă o metodă pentru înmulțirea unui număr mare cu un întreg. Prototipul acestei metode va fi `void operator*=(long) ;`
- Pentru a generaliza problema de mai sus, creați o metodă pentru înmulțirea a două numere mari.

Semestrul II

Programare Win32 folosind biblioteca MFC

Laborator 15

Obiective:

Familiarizare cu AppWizard – explicarea tuturor celor 6 opțiuni puse la dispoziție de acest Wizard
Prezentarea claselor create de AppWizard pt. o aplicație de tip SDI.

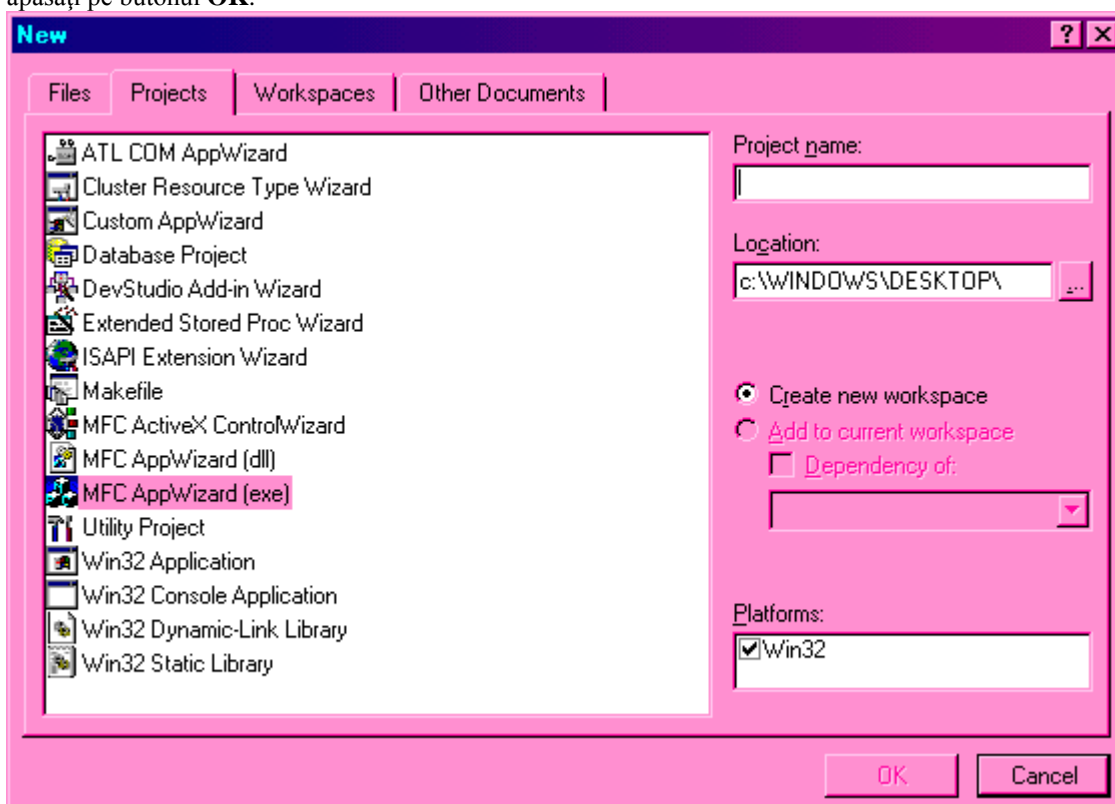
Funcția OnDraw().

Desenare: linii

Clasa CRect

Funcția GetClientRect()

Pentru a crea un nou proiect trebuie să selectați comanda **New** a meniului **File** -astfel se va afișa caseta de dialog **New** (vezi imaginea de mai jos)- după care selectați opțiunea **MFC AppWizard (exe)** (care reprezintă tipul proiectului pe care doriți să-l creați), introduceți un nume pentru acest nou proiect sub eticheta **Project name:**, selectați (doar dacă doriți) o altă adresă pentru directorul în care să fie creat acest proiect, după care apăsați pe butonul **OK**.

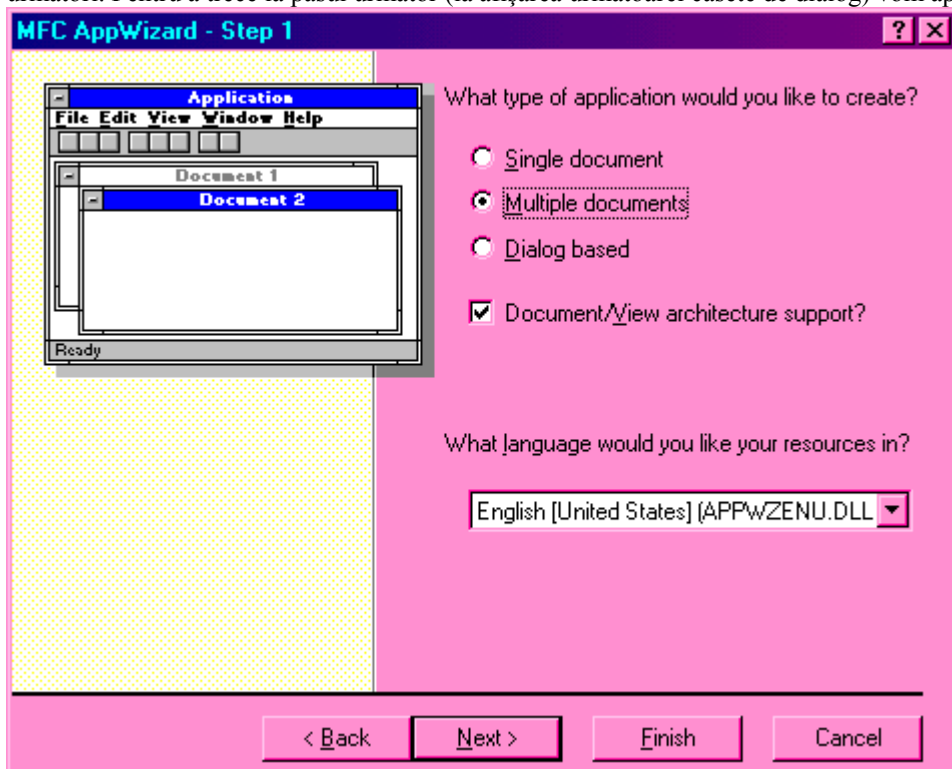


MFC provine de la **Microsoft Foundation Classes** și reprezintă o bibliotecă ce conține o mulțime de clase C++ predefinite.

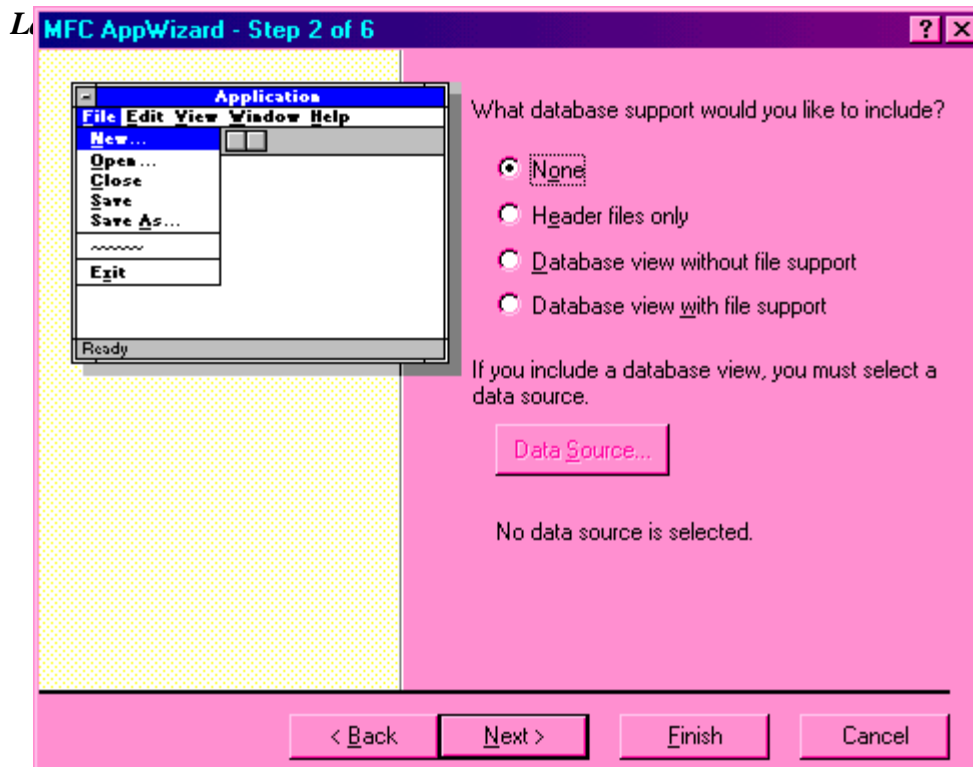
După stabilirea opțiunilor din cadrul casetei de dialog **New** și apăsarea butonului **OK** veți fi ajutați de **AppWizard** (astfel că se va deschide caseta de dialog **MFC AppWizard – Step 1**). Scopul său este de a crea un “schelet de program” care va putea fi apoi extins cu ușurință. Acest “schelet de program”, format din fișiere-sursă și fișiere-resursă, (după compilare) va fi funcțional chiar fără a adăuga nimic în codul său sursă. **AppWizard** este un instrument puternic care vă permite precizarea mai multor caracteristici personalizabile pe care le implementează în codul sursă pe care-l creează. Aceste caracteristici (personalizabile) se vor putea preciza prin intermediul a șase casete (aceasta se întâmplă în cazul arhitecturilor **Single document** și **Multiple documents**; în cazul **Dialog based** existând doar patru casete) de dialog care se vor afișa succesiv (existând posibilitatea de a sări peste aceste casete de dialog -deci acceptând datele predefinite- apăsând pe butonul **Finish**). La fiecare pas(i.e. o casetă de dialog din cele șase) există posibilitatea de a sări peste un pas sau de a

reveni la un pas anterior prin apăsarea unuia dintre butoanele **Next >** sau **Back <**. Vom explica pe rând rolul acestor șase casete de dialog.

Caseta de dialog **MFC AppWizard – Step 1** (vezi imaginea de mai jos) vă oferă posibilitatea alegerii tipului interfeței aplicației și a limbii în care să fie scris cod sursă. Noi vom alege opțiunea **Single document** în majoritatea programelor care urmează. Mai jos va trebui să decideți dacă doriți sau nu suport pentru arhitectura Document/Reprezentare al claselor **MFC**. Dacă nu doriți acest lucru nu veți putea folosi clasele **MFC** pentru a deschide un fișier de pe disc. De asemenea nu veți putea nici să selectați anumite opțiuni care vor urma în pașii următori. Pentru a trece la pasul următor (la afișarea următoarei casete de dialog) vom apăsa pe butonul **Next >**.



Caseta de dialog **MFC AppWizard – Step 2** (vezi imaginea de mai jos) oferă anumite opțiuni referitoare la diferite tipuri de suport pentru bazele de date. Acestea sunt: **None**, **Header file only**, **Database view without file support** și **Database view with file support**. În cazul opțiunii suportului bazei de date apăsați pe butonul **Data Source...** și alegeți o bază de date externă de tipul **ODBC**, **DAO**, **OLE DB** împreună cu sursa de date și opțiunilor specifice existente fiecărui caz în parte. Pentru a trece la pasul următor apăsați pe butonul **Next >**.

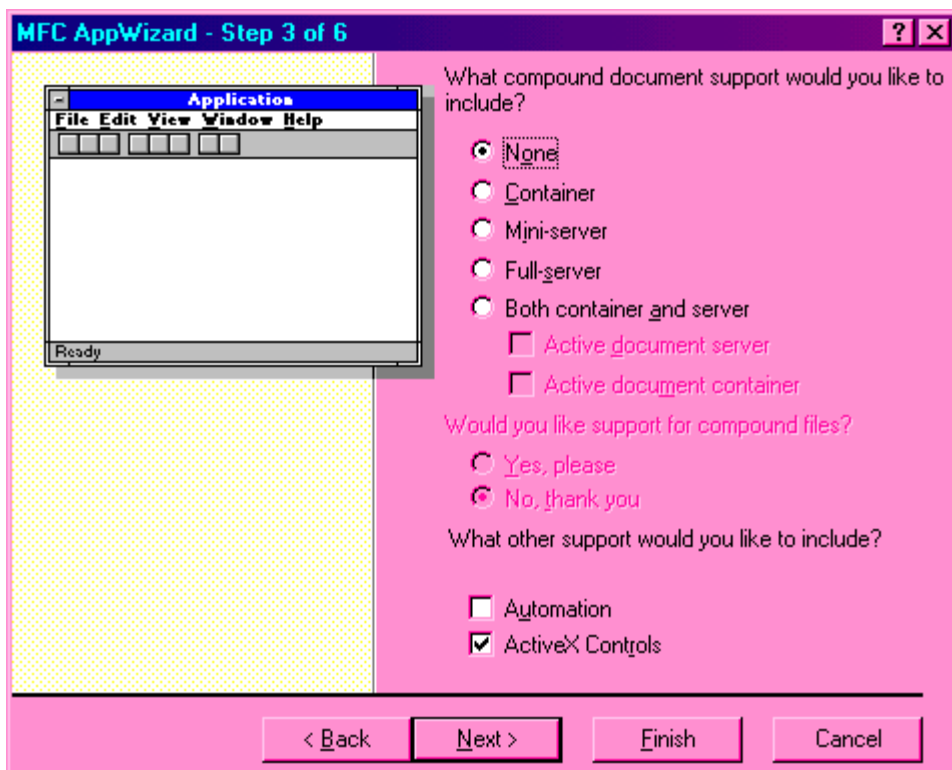


Caseta de dialog **MFC AppWizard – Step 3** (vezi imaginea de mai jos) oferă opțiuni ce privesc adăugarea unor facilități privind automatizarea **OLE** (Object Linking and Embedding). În funcție de destinația aplicației pe care o veți crea aveți de ales între **None**, **Container**, **Mini-server**, **Full-server** și **Both container and server**.

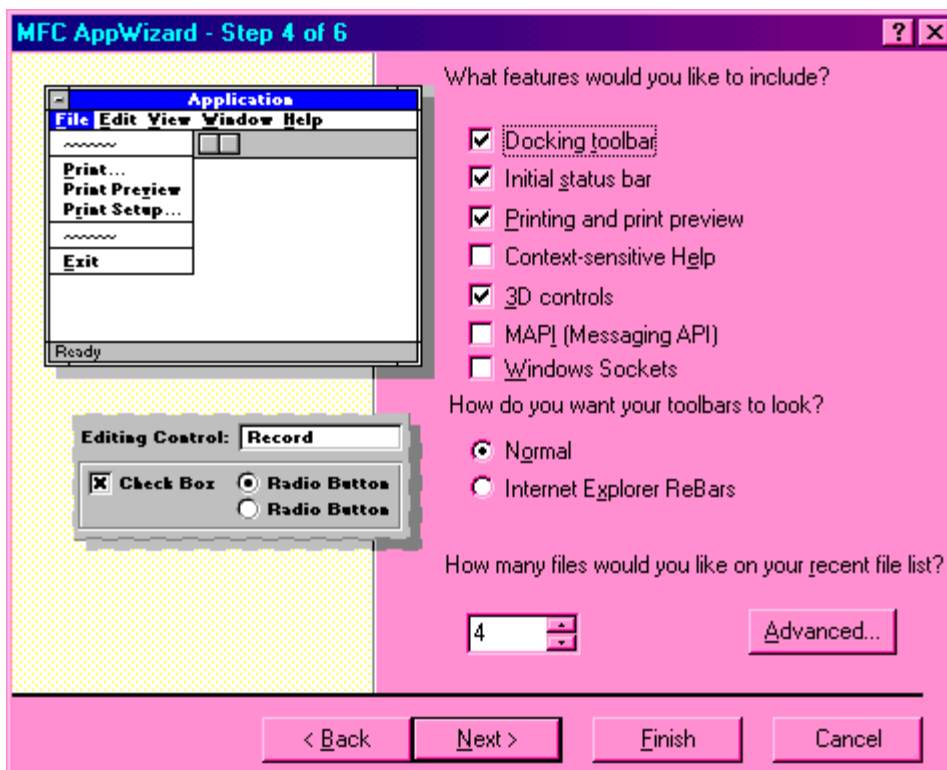
Pentru a înțelege puțin fiecare opțiune vom porni de la un exemplu. Foile de calcul din **Excel** pot fi inserate și editate în **Word**. În acest exemplu **Word** va reprezenta aplicația *container* **OLE** iar **Excel** aplicația *server* **OLE**. Desigur că unele aplicații pot fi în același și *containere* **OLE** și *server* **OLE** (chiar în exemplul nostru, rulând **Excel** puteți insera documente **Word** și invers).

Un *server complet* poate să ruleze atât ca aplicație de sine stătătoare cât și ca obiect inserat pe când *mini-serverele* pot fi lansate doar din cadrul unui program container.

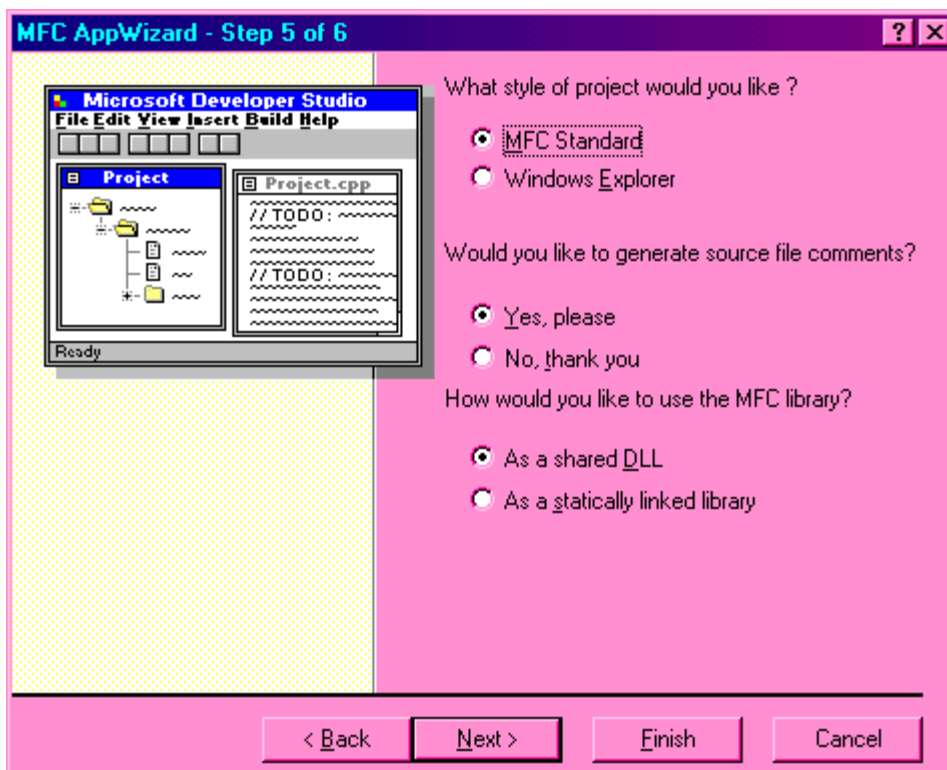
Pentru a trece la pasul apăsați pe butonul **Next >**.



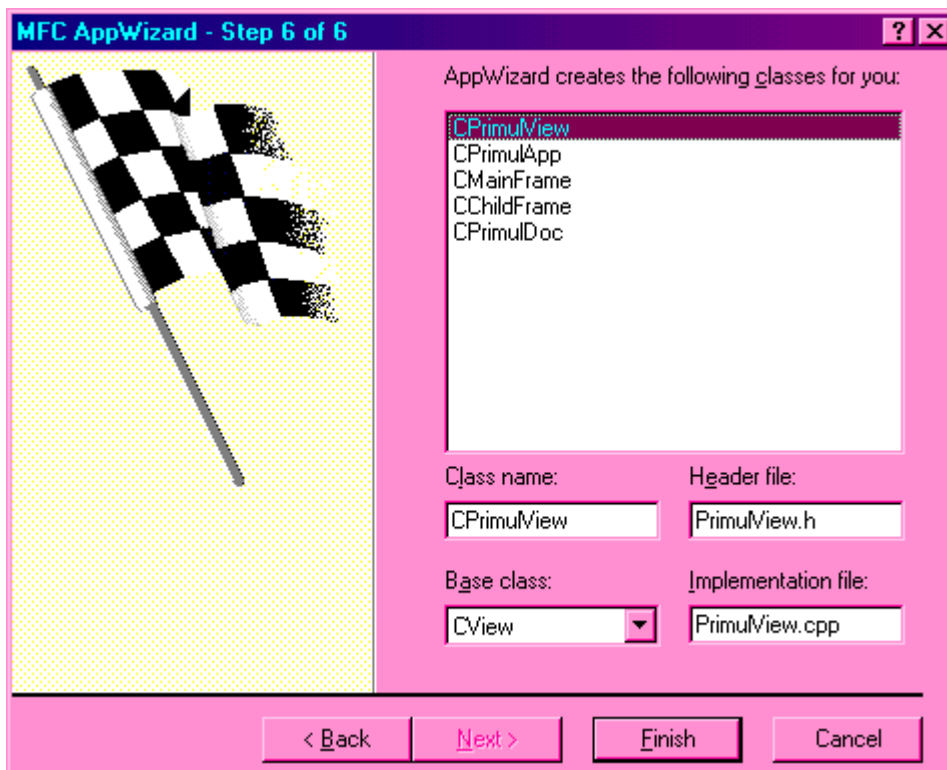
Caseta de dialog **MFC AppWizard – Step 4** (vezi imaginea de mai jos) oferă posibilitatea alegerii câtorva opțiuni de bază referitoare la interfața cu utilizatorul și la bara cu instrumente. Dacă doriți să schimbați numele fișierelor și/sau a extensiilor pe care să le folosească aplicația apăsați pe butonul **A**dvanced... și selectați opțiunile dorite. Pentru a trece la pasul următor apăsați pe butonul **N**ext >.



Caseta de dialog **MFC AppWizard – Step 5** (vezi imaginea de mai jos) oferă posibilitatea alegerii stilului proiectului (**Explorer**, care afișează o reprezentare de tip arbore în partea stângă și o reprezentare de tip listă în partea dreaptă, sau **MFC Standard** care ne oferă o singură regiune pentru reprezentarea unui fișier), a generării comentariilor în fișierele sursă (care vă pot ghida atunci când scrieți codul sursă a programului) și modul în care doriți să utilizați biblioteca **MFC**. Pentru a trece la pasul apăsați pe butonul **Next >**.



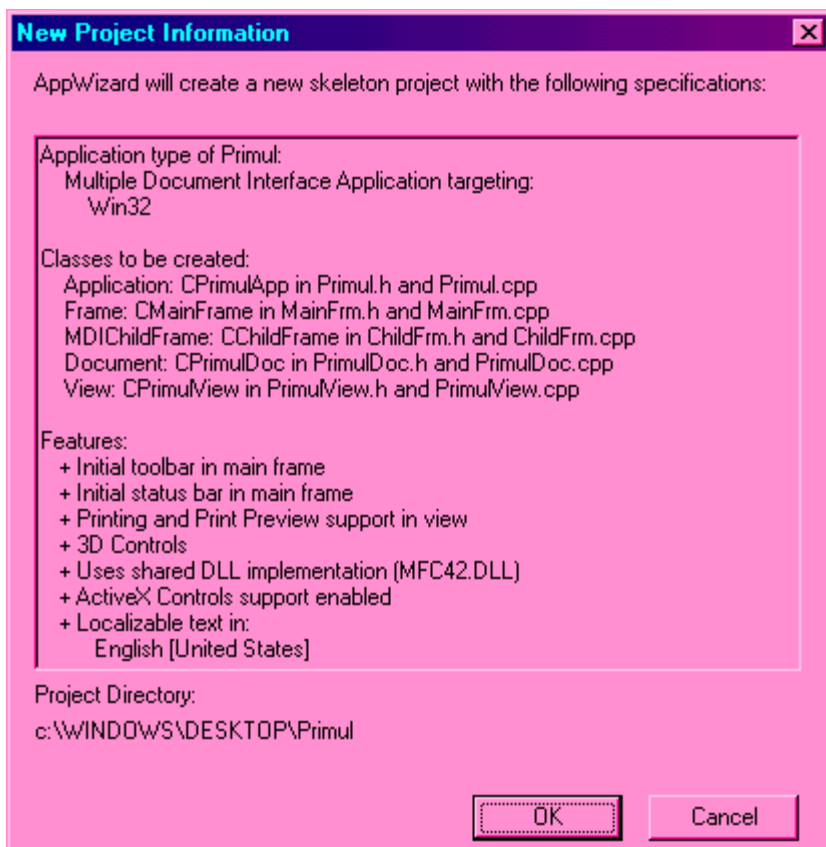
Caseta de dialog **MFC AppWizard – Step 6** (vezi imaginea de mai jos) permite modificarea *numelui clasei*, a *clasei de bază*, a *numelui fișierului antet* și a *fișierului sursă* care sunt implicate.




Mai jos afișăm lista numelor claselor de bază ce pot fi alese:

Clasa	Descriere
CView	Clasa de bază pentru toate reprezentările.
CCtrlView	Clasa de bază pentru CTreeView , CListView , CEditView , și CRichEditView . Aceste clase permit folosirea arhitecturii document/reprezentare împreună cu cele mai folosite controale Windows .
CEditView	O reprezentare simplă bazată pe controlul de editare Windows . Permite introducerea și editarea textului și poate fi folosit ca punct de pornire pentru un editor de texte simplu.
CRichEditView	O reprezentare ce conține un obiect de tipul CRichEditView . Această clasă este analoagă clasei CEditView , dar spre deosebire de ultima CRichEditView poate utiliza formatarea textului.
CListView	O reprezentare ce conține un obiect de tipul CListView .
CTreeView	O reprezentare ce conține un obiect de tipul CTreeView , util pentru reprezentările asemănătoare ferestrei Workspace din Visual C++ .
CScrollView	Clasă de bază pentru CFormView , CRecordView , și CDaoRecordView . Oferă facilități pentru derularea conținutului reprezentării (bara de derulare verticală, etc).
CFormView	Folosește o machetă de dialog ca reprezentare, permițând aplicației să se prezinte ca un formular de bază de date.
CHtmlView	Este o reprezentare de tip browser de Web cu ajutorul căreia utilizatorul aplicației poate accesa pagini de pe World Wide Web și de asemenea și directoare din sistemul de fișiere local sau din rețea.
CRecordView	Această clasă este derivată din clasa CFormView și a fost dezvoltată pentru a face legătura între reprezentare și o mulțime de înregistrări dintr-o bază de date. Dacă optați pentru suport ODBC în proiectul creat atunci clasa de bază a clasei reprezentare va fi CRecordView .
CDaoRecordView	Această clasă este derivată din clasa CFormView și se folosește la baze de date DAO . Dacă optați pentru suport DAO în proiectul creat atunci clasa de bază a clasei reprezentare va fi CDaoRecordView .
COleDBRecordView	Această clasă este derivată din clasa CFormView , este nou apărută în versiunea 6.0 a Microsoft Visual Studio , și este folosită la înregistrări de tipul OLE DB. Dacă optați pentru suport OLE DB în proiectul creat atunci clasa de bază a clasei reprezentare va fi COleDBRecordView . Notă: COleDBRecordView nu este încă recunoscută de ClassWizard , nici de către New Class și nici de către comanda New Form din meniul Insert .

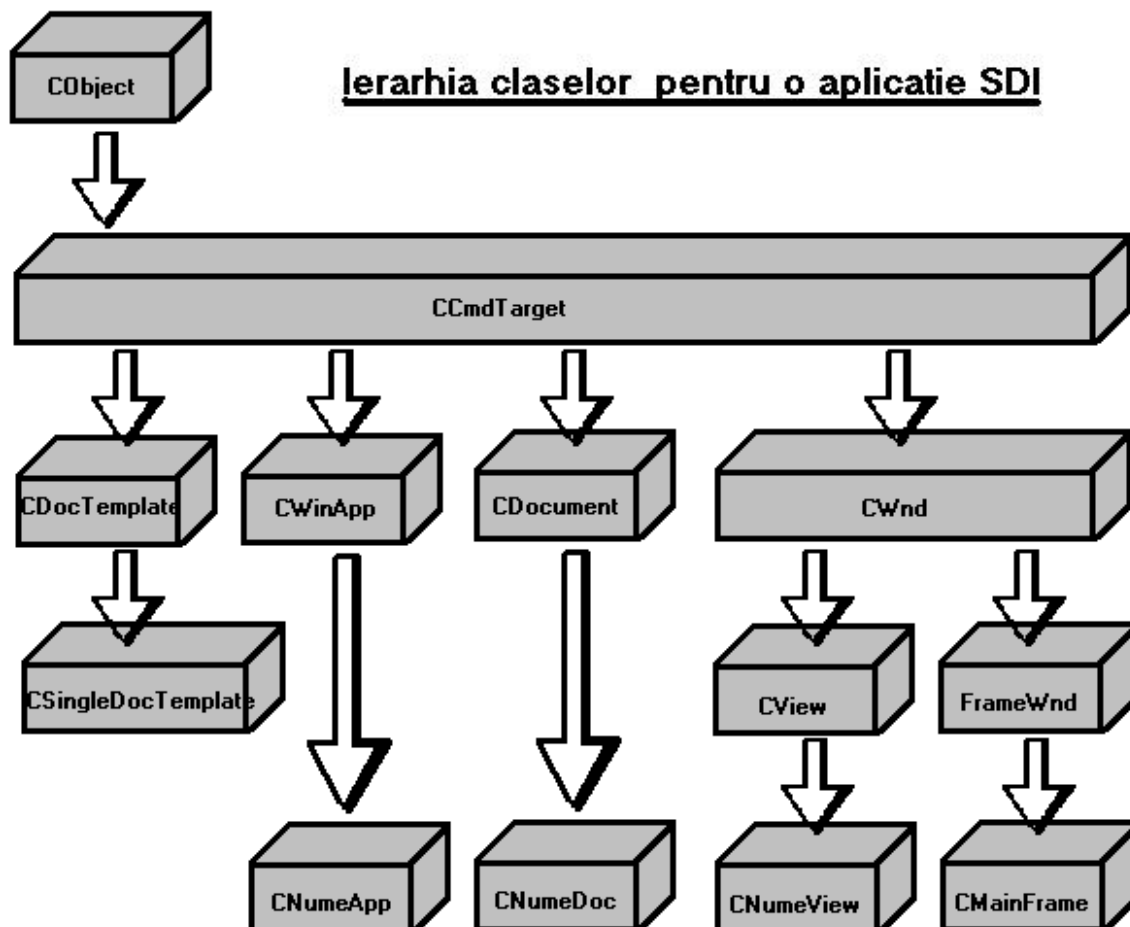
Apăsând butonul **Finish** va apărea o casetă de dialog (vezi imaginea următoare) cu un sumar al opțiunilor selectate. Apăsați pe butonul **OK** pentru ca **AppWizard** să creeze noua aplicație conform opțiunilor selectate la pașii anterior.



Pentru a compila și a rula programul apăsați pe butoanele **Ctrl+F5** (sau pe bunonul  sau selectați comanda **Execute ...** din meniul **Build**) și apăsați pe butonul **OK**.

Prin noțiunea de **resursă** vom înțelege o componentă adițională a unui proiect (pe lângă codul C++) care este stocată împreună cu aplicația și poate fi utilizată prin cod pentru afișarea de pictograme, meniuri, bara cu instrumente(toolbar), bitmap-uri, șiruri(string-uri) sau casete de dialog. Deci, meniul, bara cu instrumente ș.a. sunt resurse pe care **AppWizard** le-a adăugat proiectului atunci când l-a creat. Dacă selectați pagina **ResourceView**, și vă uitați printre componentele ce se găsesc acolo, veți observa că există patru resurse(o tabelă de acceleratori, o pictogramă, un meniu și o bară cu instrumente) care au același identificator(**IDR_MAINFRAME**). Efectuând un dublu-clic pe oricare din acestea, veți avea posibilitatea editării acestora. Mai există și o tabelă de șiruri. Aceasta reprezintă un șir de caractere cu un format special; șirul este împărțit în șapte șiruri distincte, delimitate de "\n" . Fiecare subșir descrie un anumit atribut al documentului aplicației(de exemplu primul subșir reprezintă titlul afișat în fereastra aplicației).

Dacă vă uitați la pagina **ClassView** a unei aplicații de tip **SDI**, veți observa că **AppWizard** a creat patru clase care gestionează structura aplicației(plus încă o clasă corespunzătoare aplicației de tip dialog **About**). Aceste sunt: **CNumApp**, **CNumDoc**, **CNumView**, **CMainFrame** (unde **Nume** reprezintă numele aplicației **SDI** create). Lanțul moștenirilor acestor clase este ilustrat în imaginea următoare.



- a) Clasa **CNumApp**, derivată din **CWinApp**, se ocupă de inițializarea aplicației, este responsabilă pentru întreținerea relației dintre clasele document (**CNumDoc**), reprezentare (**CNumView**) și cadru (**CMainFrame**). De asemenea, primește mesajele Windows și le expediază către fereastra țintă corespunzătoare.
- b) Într-o aplicație cu interfață **SDI**, la un moment dat există o singură instanță a clasei **CNumDoc**, derivată din **CDocument**. Această clasă are rolul de container pentru datele aplicației. Aceste date pot fi de orice formă (de exemplu pentru o aplicație de prelucrare a textului datele ar putea conține textul în sine precum și unele detalii legate de formatarea textului, în timp ce pentru o aplicație de proiectare grafică datele ar putea conține coordonate și alte detalii ale fiecărei componente ale unei imagini).
- c) Clasa **CNumDoc**, se ocupă cu reprezentarea datelor documentului și permite acestuia să interacționeze cu respectivele date. Clasa reprezentare accesează datele necesare prin intermediul funcțiilor membru oferite de către clasa document.
- d) Clasa **CMainFrame** are rolul de a furniza aplicației o fereastră. Această clasă se ocupă de asemenea și de crearea, inițializarea și distrugerea barelor cu instrumente și a barei de stare.

Infrastructura **MFC** apelează funcția membru **OnDraw** a clasei reprezentare (adică funcția **CNumView::OnDraw**) de fiecare dată când reprezentarea trebuie generată pe ecran (de exemplu în cazul redimensionării ferestrei cadru). Această funcție este apelată și pentru a gestiona operațiile necesare pentru tipărire și previzualizare. Deci pentru a implementa codul de desenare vom folosi această funcție membru **OnDraw()**.

În aplicația pe care ați început-o mai sus (ora trecută), adăugați în cadrul funcției **OnDraw()** următoarele linii de cod (pentru aceasta efectuați un dublu-clic pe numele funcției și completați cu liniile de mai jos!). Astfel vom vedea cum putem să desenăm linii, dreptunghiuri, elipse etc.

Laborator C++

```
1 void CNumView::OnDraw(CDC* pDC)
2 {
3     CLinii2Doc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5     // TODO: add draw code for native data here
6
7     //mutam pozitia curenta de desenare in punctul (200,200)
8     pDC->MoveTo(200,200);
9
10    //de aici desenam o linie pana in punctul (200,10)
11    pDC->LineTo(200,10);
12
13    //vom desena un dreptunghi incadrat intre coordonatele:
14    //coltul stanga-sus (10,10),iar cel din dreapta-jos(180,90)
15    pDC->Rectangle(10,10,180,90);
16
17    //desenam o elipsa inscrisa in acest dreptunghi
18    pDC->Ellipse(10,10,180,90);
19 }
```

Pe măsură ce scrieți aceste linii de cod vă sugerăm să compilați aplicația și să o rulați astfel încât să sesizați ce face fiecare funcție. Astfel puteți să vă opriți după ce ați completat până la linia 12, apoi respectiv până la linia 16 și 19 inclusiv.

După ce ați înțeles fiecare parte de cod de mai înainte, puteți să mai adăugați și următoarele linii:

```
19 //pentru a afisa un mesaj vom folosi functia TextOut() care primeste drept parametri pozitia si
20 //respectiv textul de afisat
21 pDC->TextOut(200,200,"Laboratorul 15 : Visual C++!!!");
22
23 //pentru a obtine o valoare de tipul COLORREF (necesara functiilor ce necesita specificarea unei culori)
24 //vom folosi macro-ul RGB care necesita trei parametri (pentru detalii vezi mai jos,
25 //dupa aceste linii de cod). Pentru a seta o culoare de fundal folosim functia SetBkColor()
26 pDC->SetBkColor(RGB(124,124,124));
27 pDC->TextOut(200,250,"Primul laborator de Visual C++!!!");
28
29 pDC->SetBkMode(TRANSPARENT);
30 pDC->TextOut(200,245,"Primul laborator de Visual C++!!!");
31 pDC->SetBkMode(OPAQUE);
32 pDC->TextOut(200,205,"Primul laborator de Visual C++!!!");
33
34 pDC->SetTextColor(RGB(125,0,0));
35 pDC->TextOut(200,300,"Primul laborator de Visual C++!!!");
36
37 CRect rect;
38 GetClientRect(&rect);
```

Macro-ul **RGB** primește trei argumente și anume trei întregi din intervalul [0, 255], corespunzători respectiv nuanțelor de roșu (**R**ed), verde (**G**reen) și albastru (**B**lue).

Liniile 29-32 arată cum se poate folosi funcția **SetBkMode()**. Dacă funcția primește parametru **OPAQUE** atunci textul este afișat numai după ce culoarea de fundal este desenată. Dacă însă va primi parametrul **TRANSPARENT**, textul afișat se va afișa fără a se mai folosi nici o culoare de umplere pe fundal. Folosind funcția **SetTextColor** putem seta culoarea cu care dorim să fie afișat un text. Această funcție primește drept parametru o valoare de tip **COLORREF**.

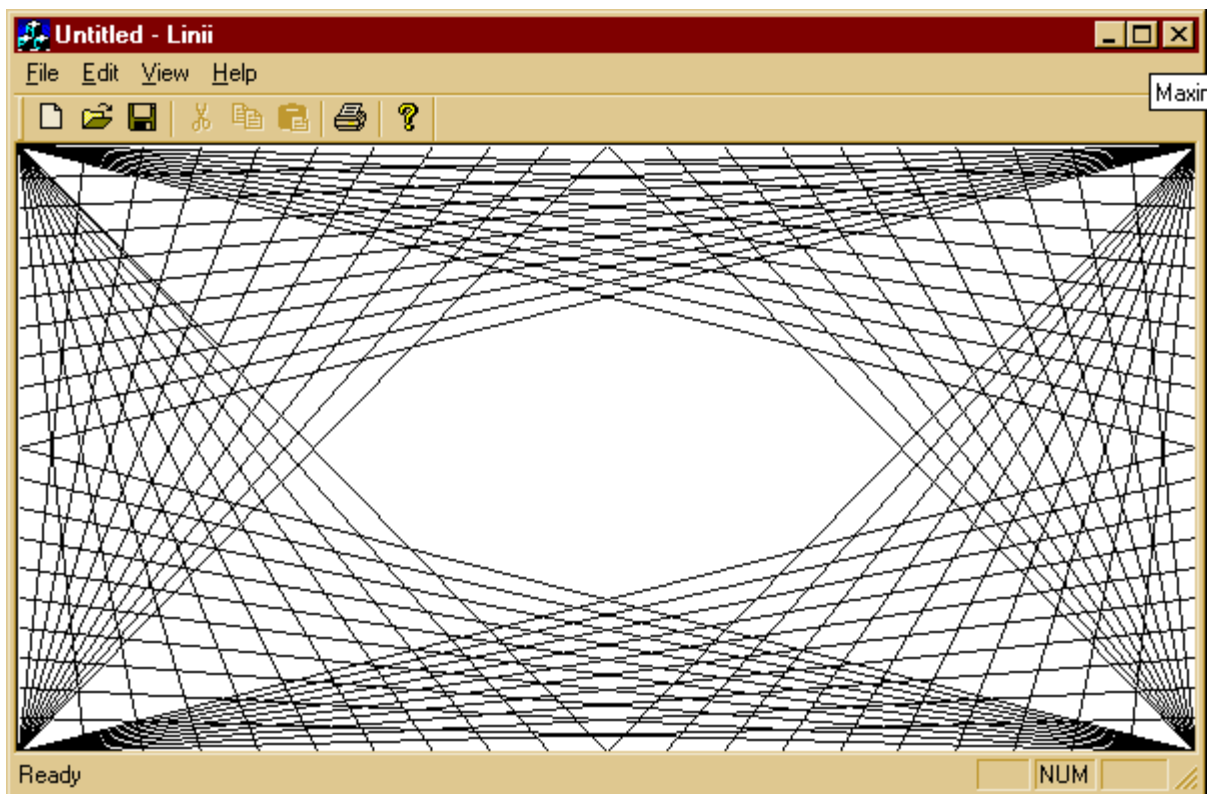
De multe ori este necesar să știm dimensiunile ferestrei cu care lucrăm. Pentru aceasta avem la dispoziție clasa **CRect** și funcția **GetClientRect()**. Un obiect de tip **CRect** este determinat de patru numere întregi (la care ne putem gândi ca la cele patru coordonate corespunzătoare colțului stânga-sus și respectiv a colțului din dreapta-

Laborator C++

jos corespunzătoare unui dreptunghi). Pentru a obține dimensiunile ferestrei transmitem o referință la un obiect de tip **CRect** funcției **GetClienRect()**. Liniile de cod 37-38 încarcă în rect dimensiunile ferestrei de lucru. În **VisualC++** pentru lucrul cu șiruri putem lucra cu clasa **CString**. Această clasă oferă facilități extinse lucrului cu șiruri. De asemenea avem o clasă definită și pentru lucrul cu puncte în coordonate bi-dimensionale: **CPoint**.

Problemă:

Scrieți un program MFC de tip SDI care realizează următorul desen:



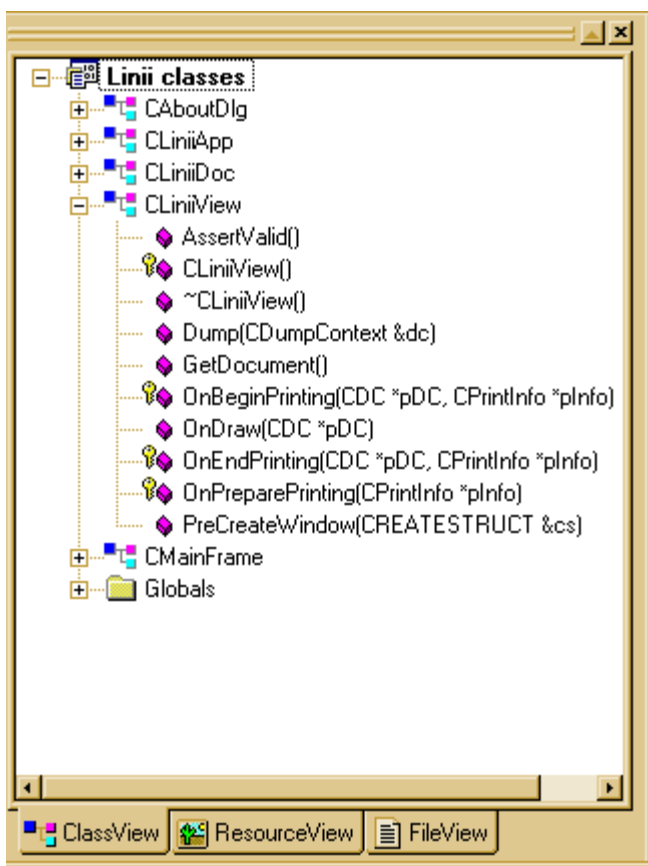
Numărul de linii care se trasează din fiecare colț este egal cu 10.

Rezolvarea problemei propuse

Pasul 1) Creați o nouă aplicație de tip SDI (pentru detalii referitoare la aceasta vezi începutul acestui laborator) și dați-i numele **Linii** (sau dacă vrei orice alt nume)

Pasul 2) Efectuați un clic pe eticheta de pagină **ClassView** de la baza ferestrei de lucru a proiectului (Vezi imaginea de mai jos)

Pasul 3) Efectuați un clic pe semnul "+" pentru afișarea tuturor claselor proiectului, apoi încă unul pentru a vizualizarea funcțiilor și variabilelor membru ale clasei **CLiniiView**.



Pasul 4) Apăsați un dublu-clic pe prototipul funcției **OnDraw(CDC *pDC)** (sau apăsați clic-dreapta pe **OnDraw(CDC *pDC)** și alegeți opțiunea **Go to Definition**)

Pasul 5) Adăugați listingul următor:

```

1      void CLiniiView::OnDraw(CDC* pDC)
2      {
3          CLiniiDoc* pDoc = GetDocument();
4          ASSERT_VALID(pDoc);
5          // TODO: add draw code for native data here
6          CRect r;
7          GetClientRect(&r);
8          int n = 10, i;
9          int dx = r.Width() / (2 * n);
10         int dy = r.Height() / (2 * n);
11         for (i = 0; i < n; i++){
12             pDC->MoveTo(0, 0);
13             pDC->LineTo(r.Width(), r.Height() / 2 - i * dy);
14         }
15         for (i = 0; i < n; i++){
16             pDC->MoveTo(0, 0);
17             pDC->LineTo(r.Width() / 2 - i * dx, r.Height());
18         }
19         for (i = 0; i < n; i++){
20             pDC->MoveTo(0, r.Height());
21             pDC->LineTo(r.Width(), r.Height() / 2 + i * dy);

```

Laborator C++

```
22     }
23     for (i = 0; i < n; i++){
24         pDC->MoveTo(0, r.Height());
25         pDC->LineTo(r.Width() / 2 - i * dx, 0);
26     }
27     for (i=0; i < n; i++){
28         pDC->MoveTo(r.Width(), 0);
29         pDC->LineTo(0, r.Height() / 2 - i * dy);
30     }
31     for (i=0; i < n; i++){
32         pDC->MoveTo(r.Width(), 0);
33         pDC->LineTo(r.Width() / 2 + i * dx, r.Height());
34     }
35     for (i=0; i < n; i++){
36         pDC->MoveTo(r.Width(), r.Height());
37         pDC->LineTo(0, r.Height() / 2 + i * dy);
38     }
39     for (i=0; i < n; i++){
40         pDC->MoveTo(r.Width(), r.Height());
41         pDC->LineTo(r.Width() / 2 + i * dx, 0);
42     }
43 }
```

Observații:

- 1) În liniile 6 și 7 declarăm un obiect de tip **CRect** și îi transmitem dimensiunile ferestrei client cu ajutorul funcției **GetClientRect** (care are următorul prototip: `void GetClientRect(LPRECT lpRect)`);
- 2) În liniile 9 respectiv 10 am calculat distanțele dintre capete a două linii consecutive, capete care se află pe axa orizontală respectiv verticală;
- 3) Cu ajutorul funcțiilor **MoveTo** și **LineTo** vom desena apoi (din fiecare colț) liniile cerute de enunțul problemei;

Laborator 16

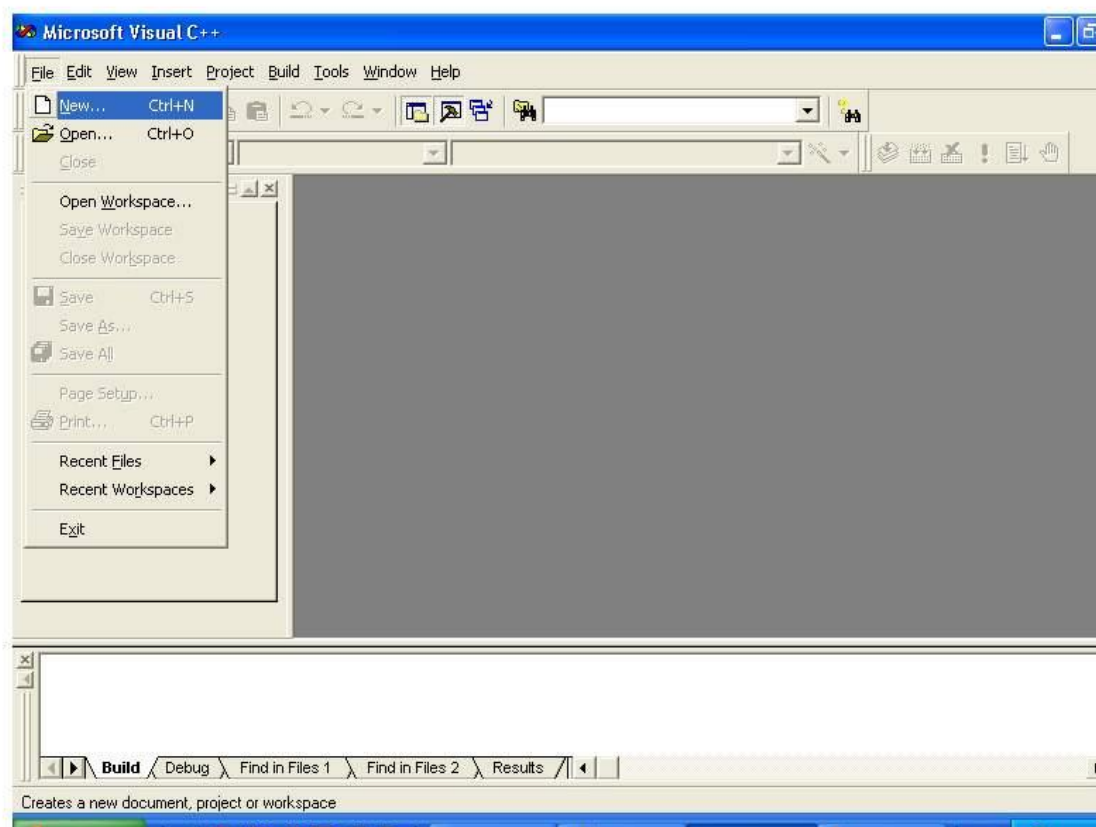
Problema 1.

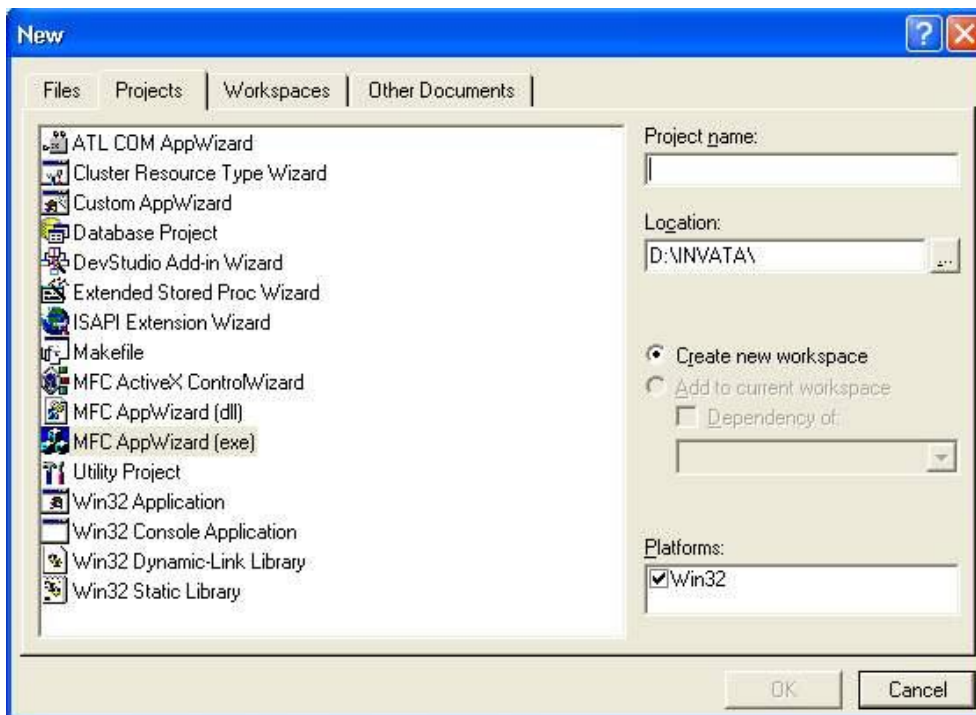
Scrieți un program MFC care aprinde fiecare pixel din zona client a ferestrei care se află într-un punct de coordonate (x, y) cu proprietatea $cmmdc(x, y) = 1$. Pentru a aprinde un pixel se folosește funcția **SetPixel()** a clasei context de dispozitiv.

Rezolvare:

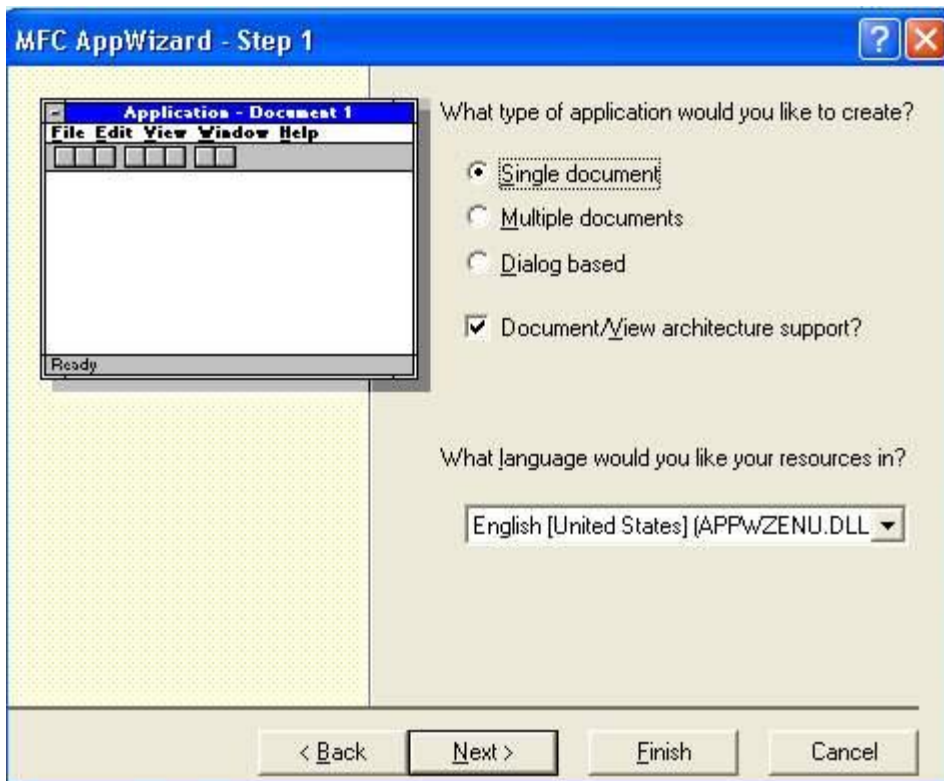
Se deschide programul Microsoft Visual C++ și se alege opțiunea **New** din meniul **File** ca în exemplul de mai jos:

după care va fi afișat un alt meniu:



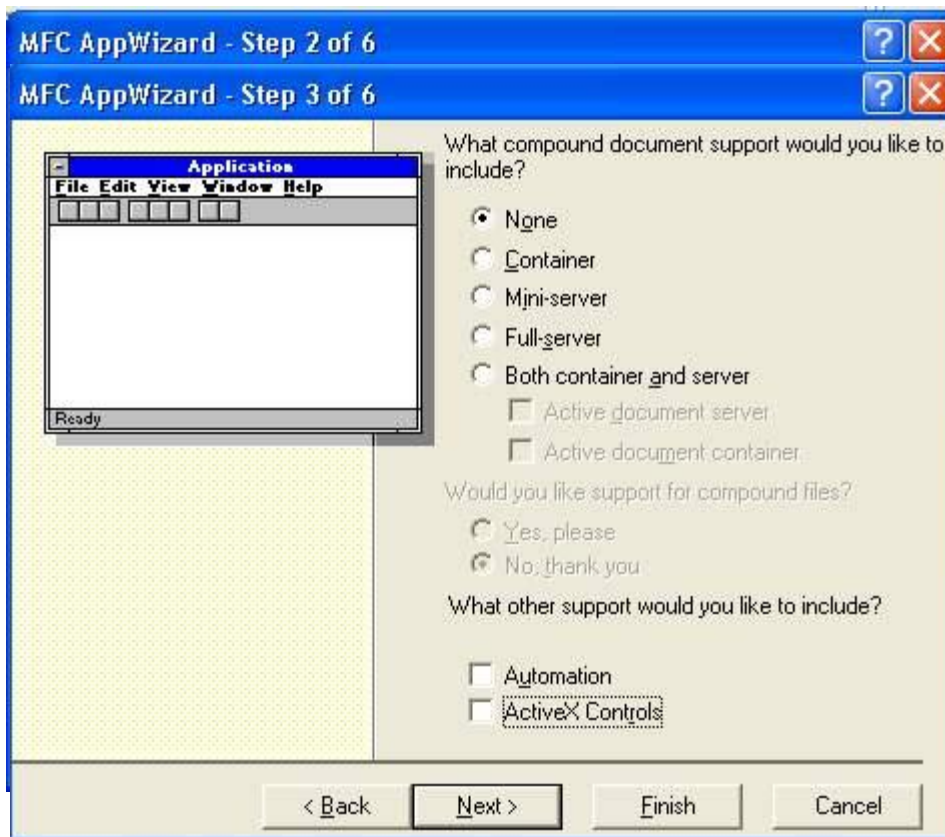


În care după câte se poate observa se alege opțiunea **Projects** iar din lista aparută se alege **MFC AppWizard(exe)**. În partea dreaptă a ferestrei se va scrie numele proiectului în caseta de editare corespunzătoare mesajului **Project name** iar dedesubt se va scrie locația de pe disc unde va fi creat proiectul, după care se va apăsa butonul **OK**. Va apărea următoarea fereastră:

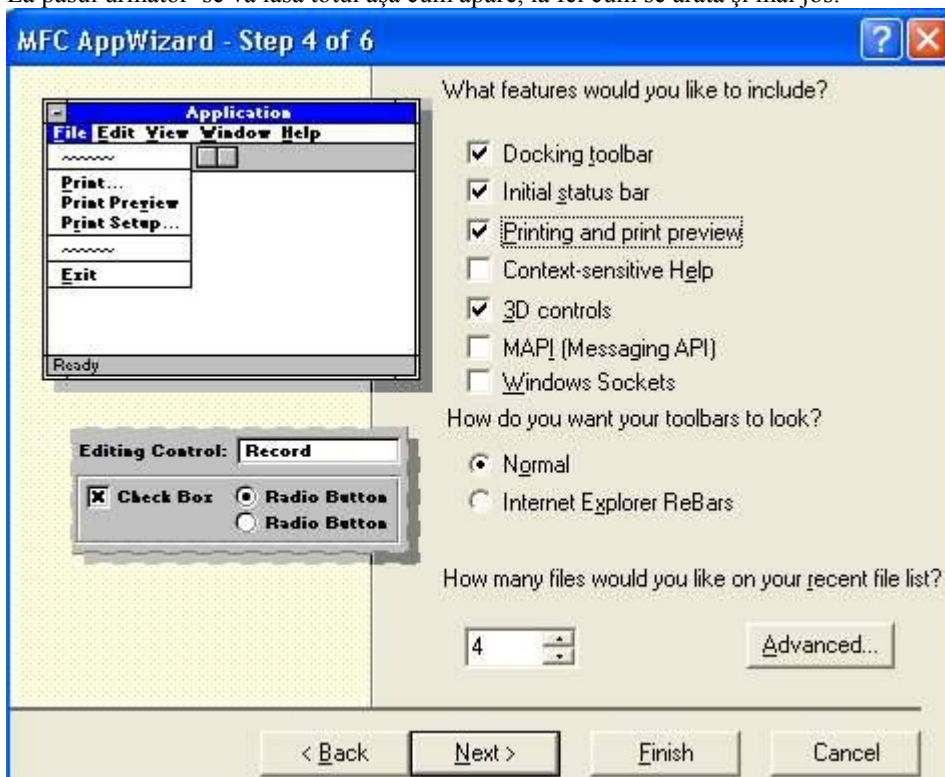


Laborator C++

În care se va selecta opțiunea **Single Document**, opțiunea **Document/View** architecture support? este activată. Apăsând Butonul **Next >** va apărea fereastra următoare, în această fereastră se va selecta opțiunea **None** și se apasă butonul **Next >**, iar în fereastra următoare opțiunea **ActiveX Controls** și **Automation** vor fi dezactivate ca mai jos:

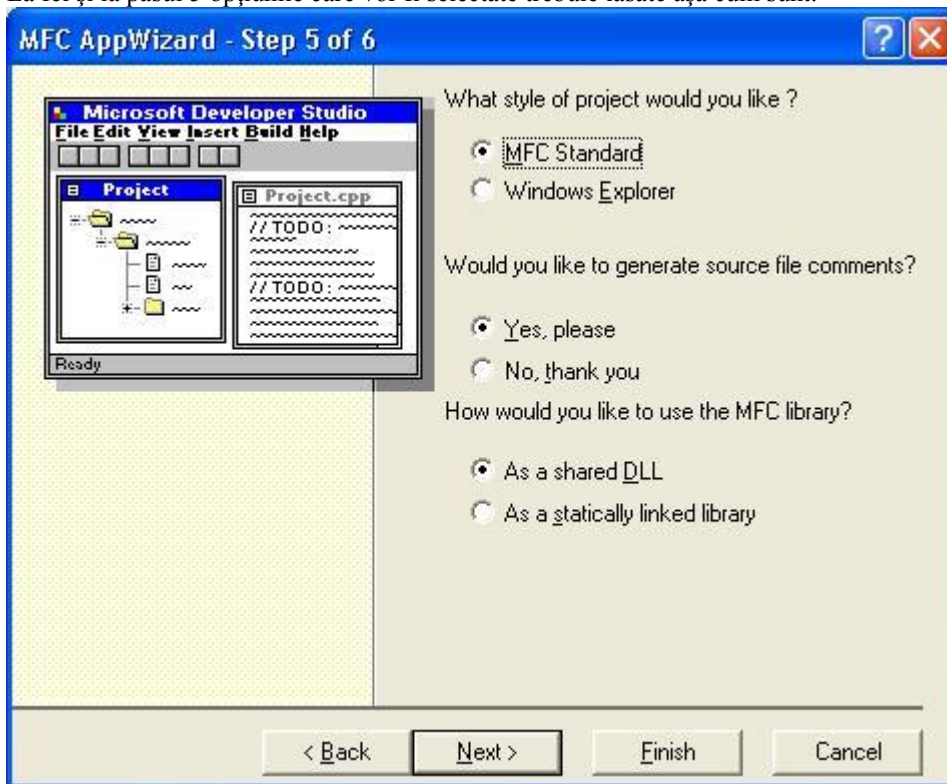


La pasul următor se va lăsa totul așa cum apare, la fel cum se arată și mai jos:

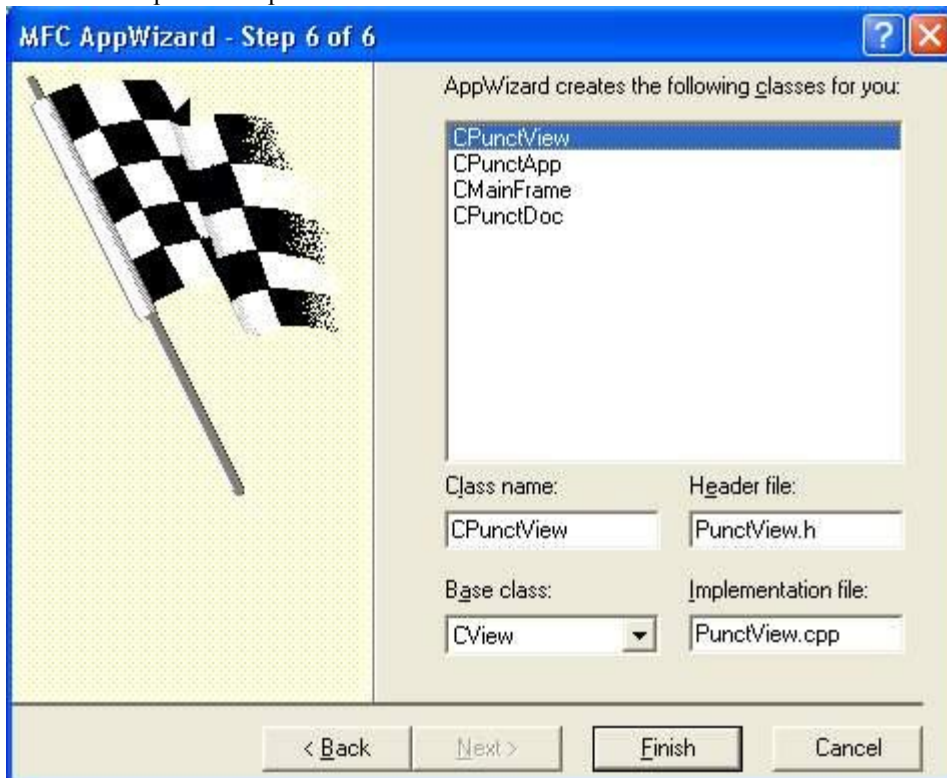


Laborator C++

La fel și la pasul 5 opțiunile care vor fi selectate trebuie lăsate așa cum sunt.

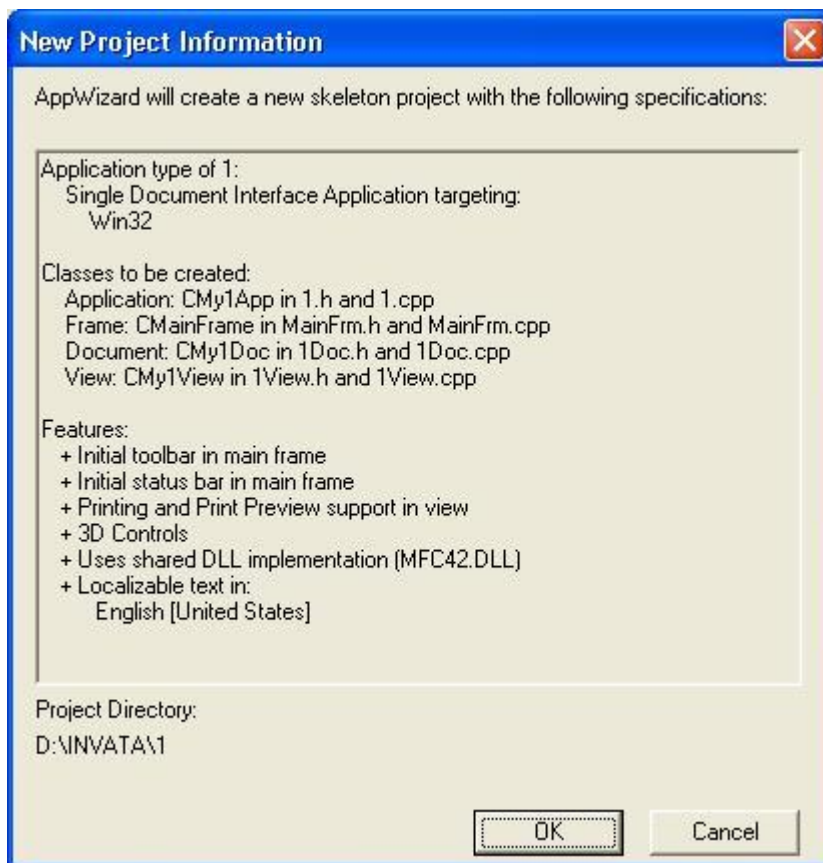


Iar în ultimul pas se va apăsa butonul **Finish** fără a modifica nimic.



Laborator C++

Înainte de a se crea proiectul va fi afișat un mesaj ca mai jos, în care se va apăsa butonul **OK**.



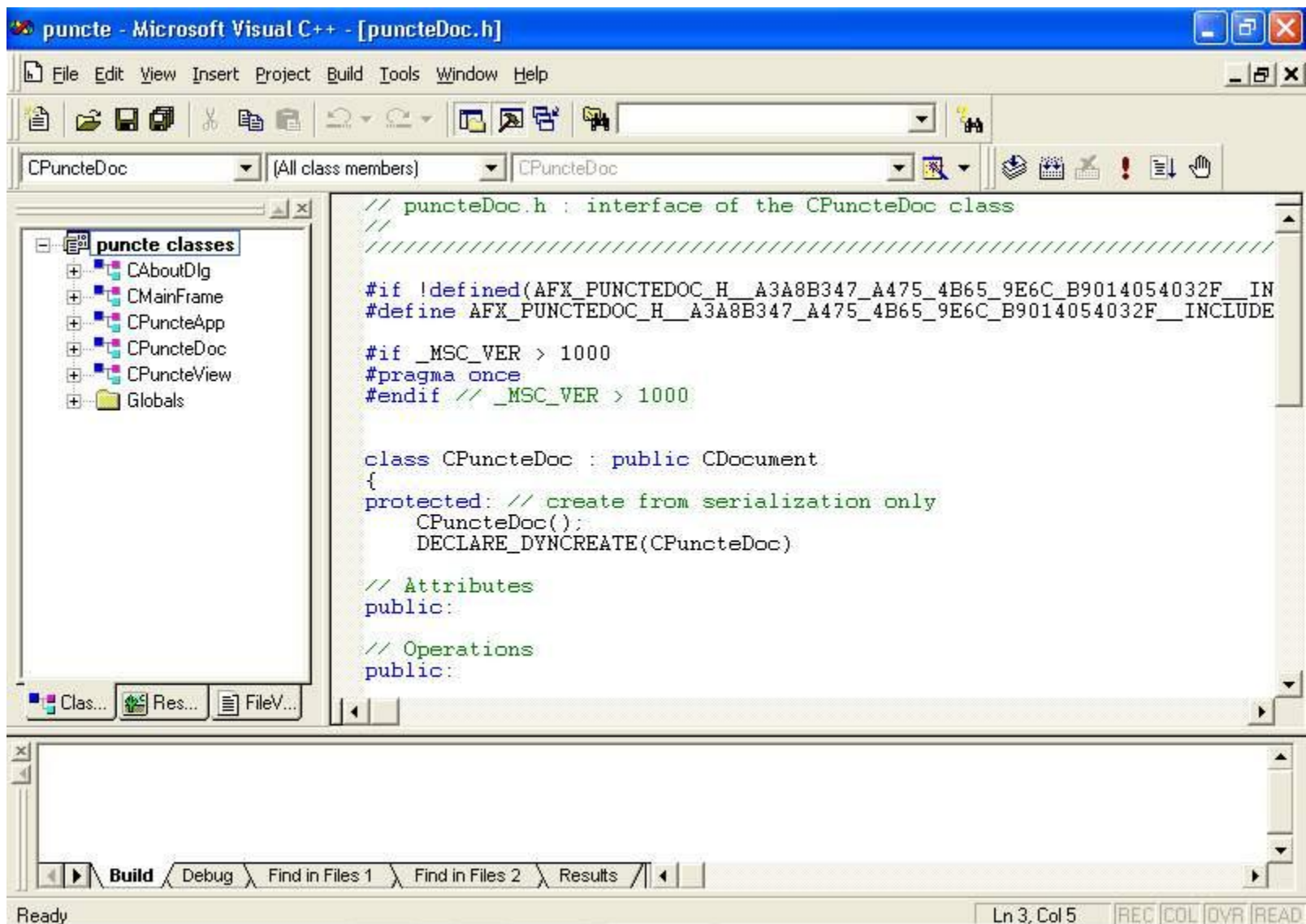
În meniul din dreapta al ferestrei de mai jos se va alege opțiunea **Class View**.

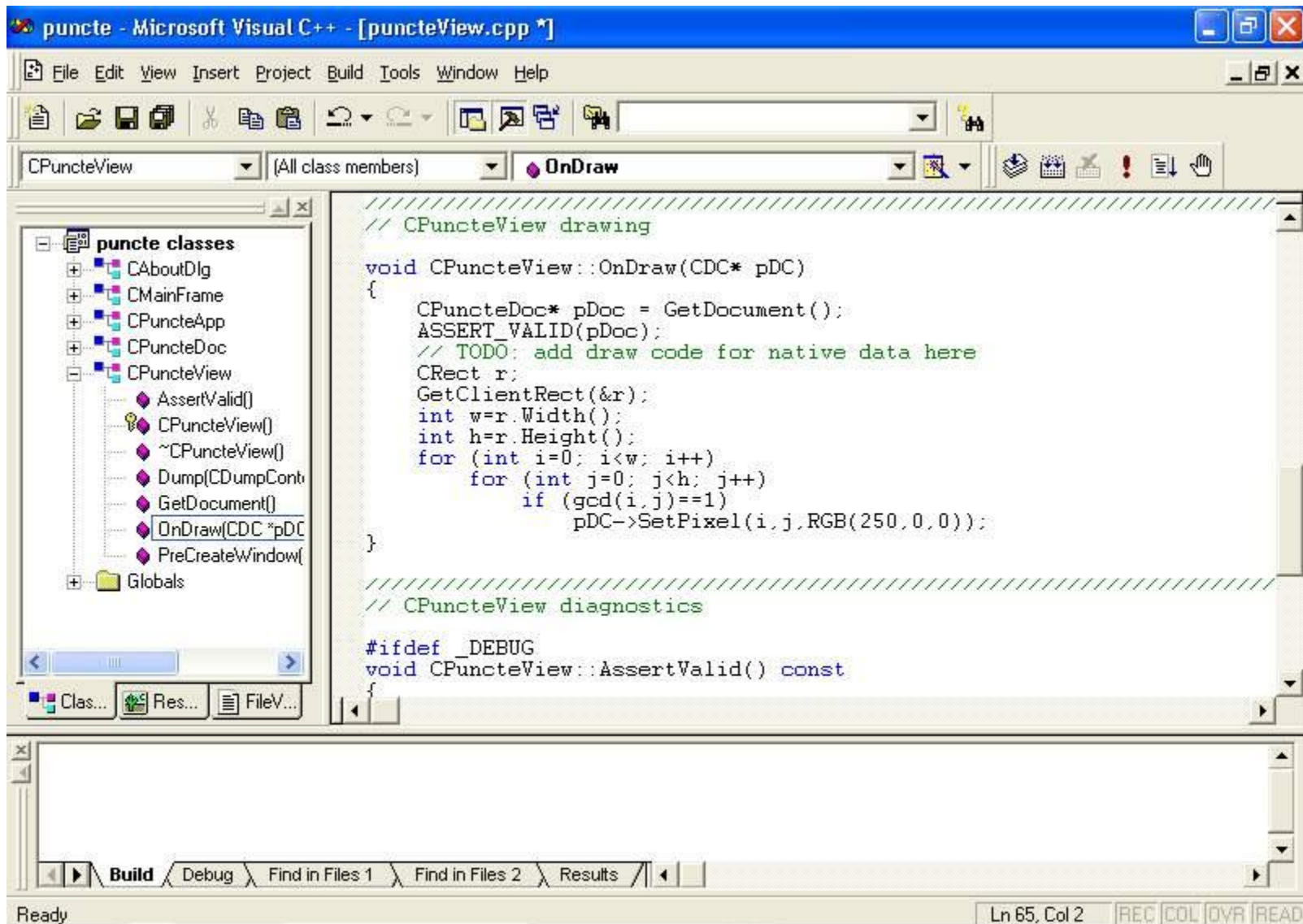
Iar în funcția **OnDraw** din clasa **CPuncteView** în linia imediat următoare liniei `:// TODO: add draw code for native data here`, se va adăuga următorul cod :

```
CRect r;  
GetClientRect(&r);  
int w=r.Width();  
int h=r.Height();  
for (int i=0; i<w; i++)  
    for (int j=0; j<h; j++)  
        if (gcd(i,j)==1)  
            pDC->SetPixel(i,j,RGB(250,0,0));
```


Laborator C++

Se crează un obiect de tip **CRect**, în cazul nostru se numește **r**. Pentru mai multe amănunte vezi biblioteca MSDN. Variabila **w** va lua valoare egală cu numărul de puncte al lungimii zonei client a ferestrei iar variabila **h** valoarea egală cu înălțimea zonei client. Mai departe se testează cu ajutorul funcției **gcd** (pe care o vom implementa imediat) dacă punctele din zona client sunt prime între ele. În cazul în care sunt prime se folosește contextul de dispozitiv **pDC** și funcția membru **SetPixel()** pentru a afișa pixelul respectiv într-o anumită culoare. În cazul de față am ales pentru funcția **RGB** care setează paleta de culori, valorile 255, 0 și 0 ceea ce înseamnă culoarea roșu.





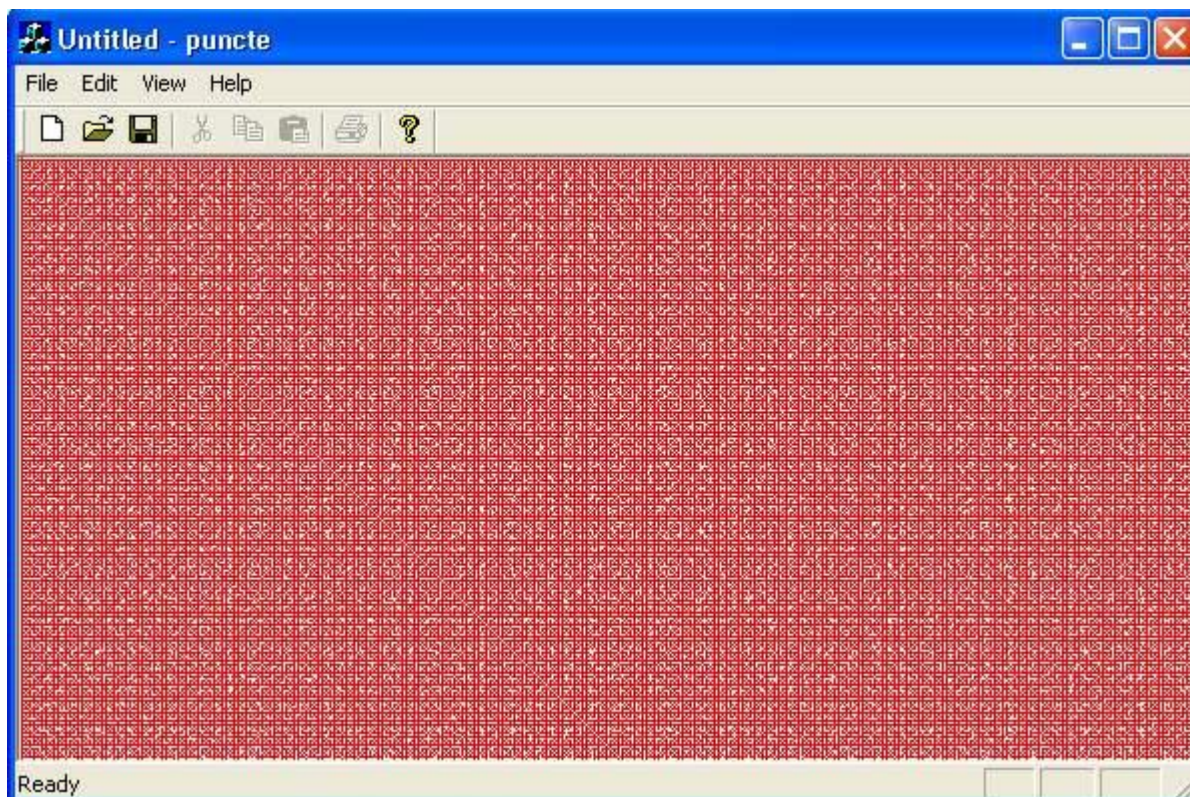
Dar să revenim la implementarea funcției **gcd**. Se adaugă în fișierul **puncteview.cpp** înaintea declarației funcției **OnDraw** următorul cod:

```
int gcd(int a, int b)
{
    if (b==0) return a;
    else return gcd(b,a%b);
}
```


Laborator C++

Fișierul **puncteview.cpp** este corespunzător clasei **CPuncteView**. Deci pentru a-l deschide vom deschide clasa **CPuncteView**.

În continuare vom compila programul și-l vom executa. Va trebui să obținem următorul rezultat :



Problema 2.

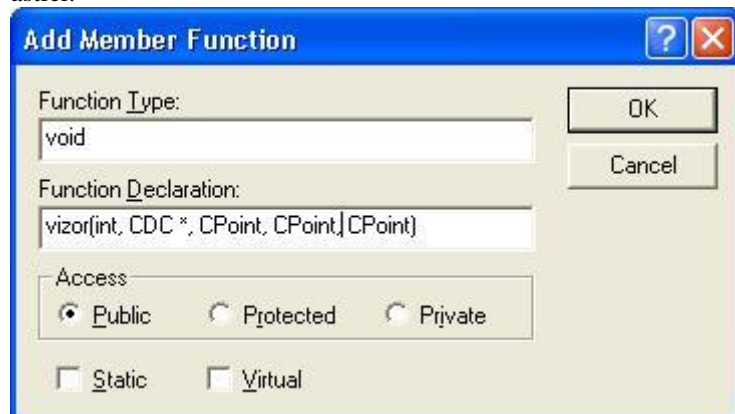
Desenați triunghiul lui Sierpinski de ordin n . (Se desenează un triunghi. Se iau mijloacele celor trei laturi și elimină triunghiul format de aceste trei puncte. Pentru cele trei triunghiuri rămase se procedează la fel.)

Rezolvare:

Se crează un proiect de tip **SDI(Single Document Interface)** cu numele Sierpinski.

Implementăm o funcție membru clasei **View** cu numele **Vizor** de tip **void** și cu 5 parametri un **int**, un pointer la **CDC**, și trei **CPoint** astfel:

apasăm click dreapta în dreptul clasei **CSierpinskiView**, selectăm **Add Member Function** și decalaram funcția astfel:



Laborator C++

Funcția va trebui să arate astfel:

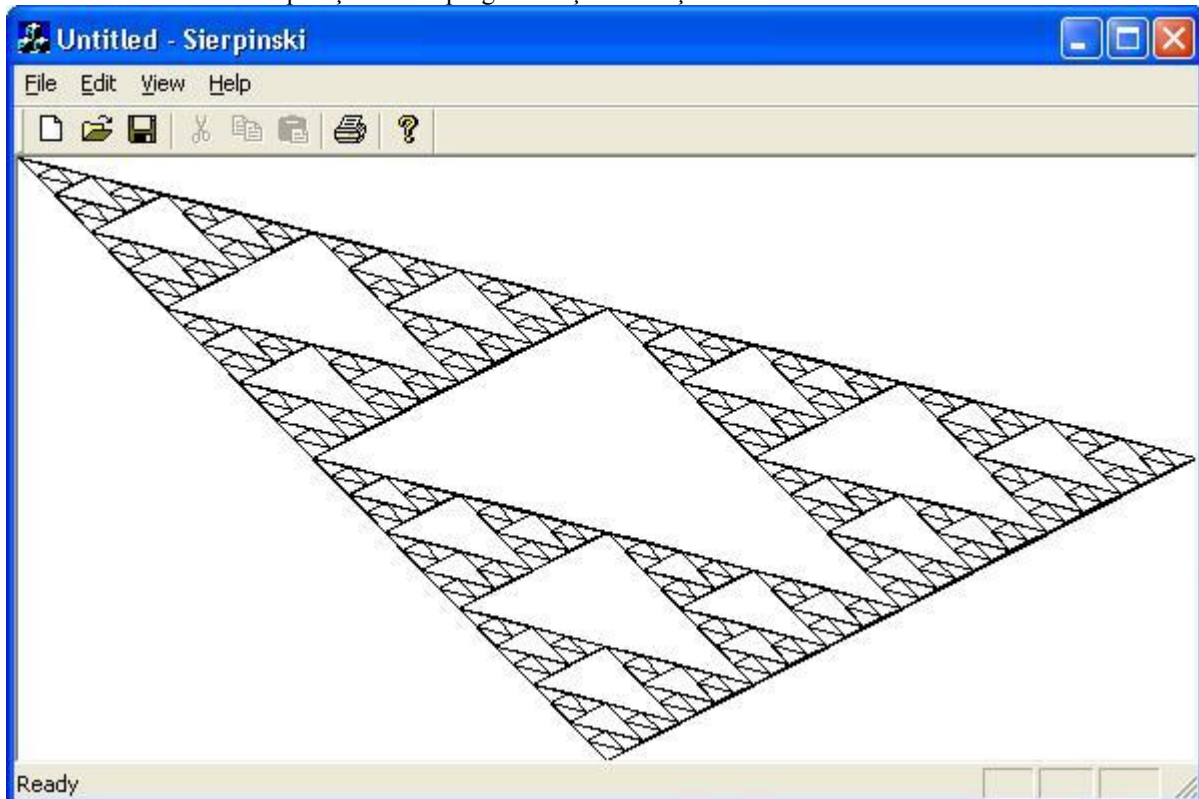
```
void CSierpinskiView::vizor(int n, CDC *pDC, CPoint p1, CPoint p2, CPoint p3)
{
    pDC->MoveTo(p1);
    pDC->LineTo(p2);
    pDC->LineTo(p3);
    pDC->LineTo(p1);
    if (n>1)
    {
        vizor(n - 1, pDC, p1,
            CPoint((p1.x + p2.x) / 2, (p1.y + p2.y)/2),
            CPoint((p1.x + p3.x) / 2, (p1.y + p3.y)/2));
        vizor(n - 1, pDC, p2,
            CPoint((p2.x + p1.x) / 2, (p2.y + p1.y)/2),
            CPoint((p2.x + p3.x) / 2, (p2.y + p3.y)/2));
        vizor(n - 1, pDC, p3,
            CPoint((p3.x + p2.x) / 2, (p3.y + p2.y)/2),
            CPoint((p3.x + p1.x) / 2, (p3.y + p1.y)/2));
    }
}
```

Se observă că această funcție se autoapelează atâta timp cât **n** este mai mare decât zero.

În funcția membru **OnDraw** a clasei **View** se adauga următorul cod:

```
int n=6;
CRect r;
GetClientRect(&r);
vizor(n, pDC, CPoint(0, 0),
    CPoint(r.Width(), r.Height() / 2),
    CPoint(r.Width() / 2, r.Height()));
```

În cele din urmă vom compila și executa programul și vom obține următorul rezultat:



Laborator C++

Observatie:

În funcția **OnDraw** variabila **n** se poate modifica, astfel vom obține un alt desen.

Laborator 17

Problema 1

Desenați un poligon regulat cu **n** laturi în zona client a ferestrei. Numărul de laturi **n** se modifică (incrementează și decrementează) prin opțiuni de meniu și prin butoane pe bara cu instrumente.

Rezolvare:

Se crează un proiect de tip **SDI** cu numele **Poligon**. La primul pas opțiunea **Document/View architecture support?** va fi inactivată.

În funcția membru **OnPaint** a clasei **CChildView** se adaugă următorul cod:

```
1 double pi = 3.1415926535;  
2 CRect r;  
3 GetClientRect(&r);  
4 dc.SetViewportOrg(r.CenterPoint());  
5 CString s;  
6 s.Format("n=%d", n);  
7 dc.TextOut(10, 10, s);  
8 CPoint pcti, pctf;  
9 for (int k=0; k<n; k++)  
10 {  
11     pcti=CPoint(100*cos((2*k*pi)/n), 100*sin((2*k*pi)/n));  
12     pctf=CPoint(100*cos((2*(k+1)*pi)/n), 100*sin((2*(k+1)*pi)/n));  
13     dc.MoveTo(pcti);  
14     dc.LineTo(pctf);  
15 }
```

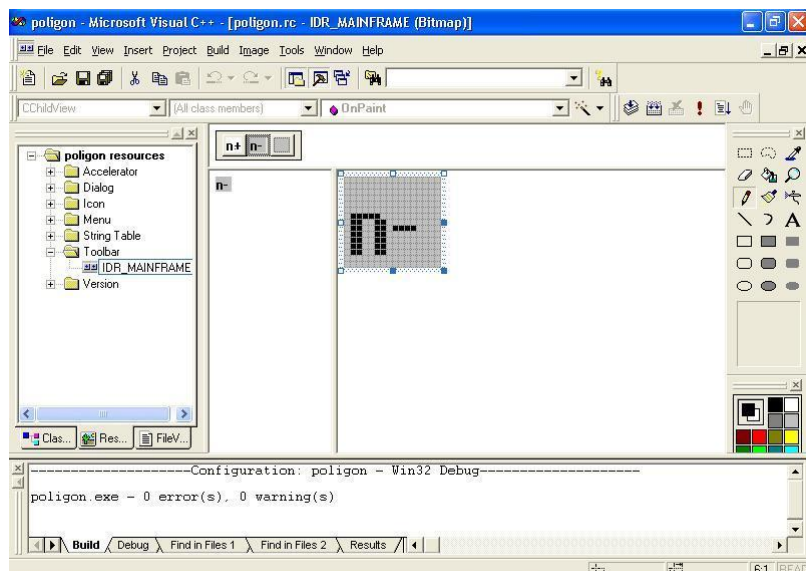
În prima linie se declară variabila **pi** de tip **double** și se inițializează.

Variabila **n** de tip **int** va fi declarată în clasa **CChildView** în secțiunea **private** și va fi inițializată cu valoarea 3 în constructorul clasei **CChildView**.

În fișierul **ChildView** vom include biblioteca **math.h** pentru a se putea folosi funcțiile **sin** și **cos** predefinite în această clasă.

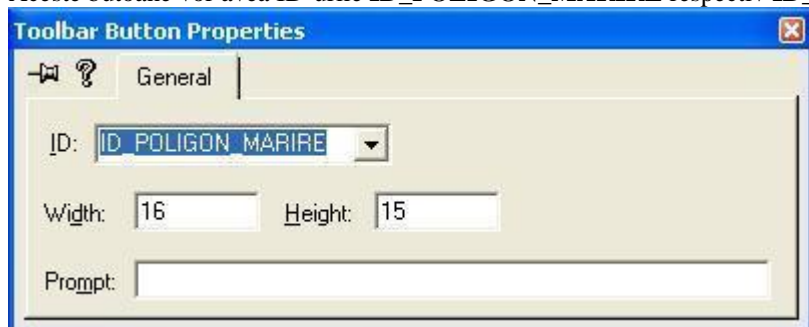
```
#include <math.h>
```

Vom crea în continuare două butoane pentru a mări și micșora numărul de laturi. Aceste butoane se vom crea astfel: apăsăm un clic stânga pe **ResourceView**, deschidem **Poligon resources**, **Toolbar** și

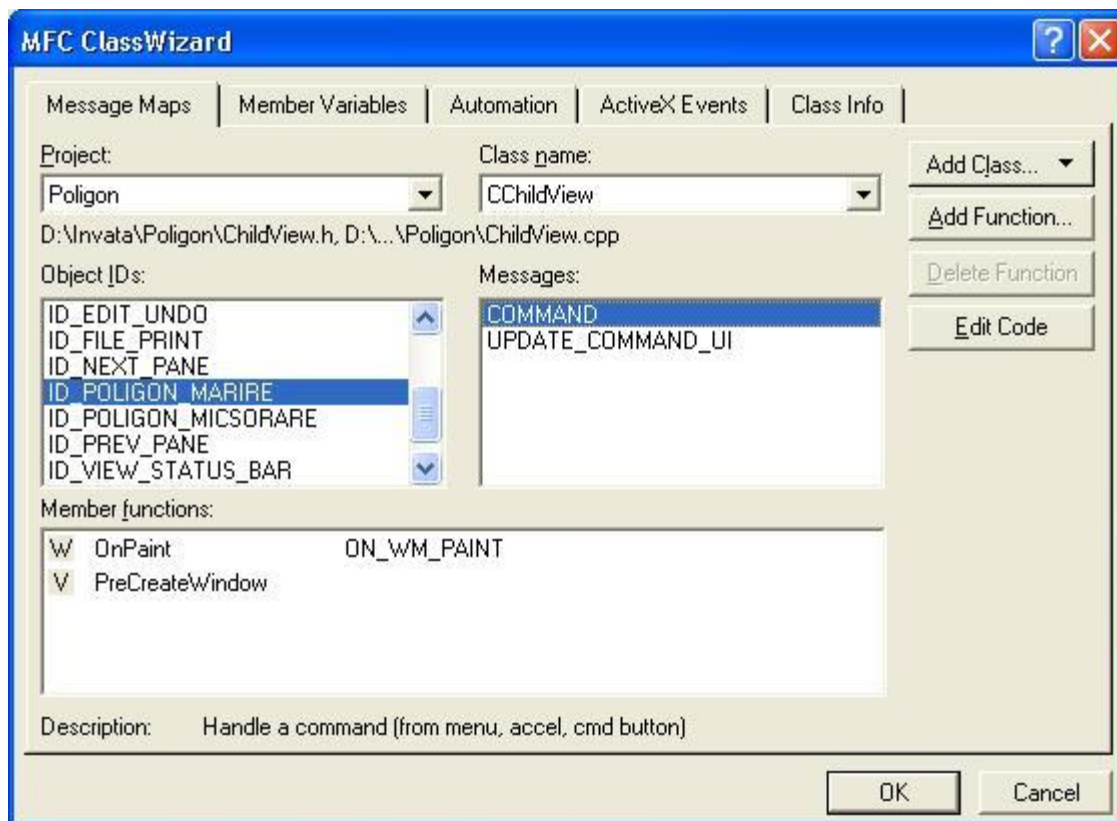


IDR_MAINFRAME și se adaugă două butoane în care va fi scris pe unul **n+** și pe celălalt **n-**.

Aceste butoane vor avea ID-urile **ID_POLIGON_MARIRE** respectiv **ID_POLIGON_MICSORARE**.



Ultimul pas al rezolvării problemei va fi crearea unor funcții care tratează apăsarea butoanelor n+ și n- . Acestea vor fi funcții membre a clasei **CChildView** și se vor adăuga folosind **ClassWizard**.



Atenție aceste funcții trebuie să fie funcții membre a clasei **CChildView** pentru aceasta se va selecta această clasă la meniul corespunzător mesajului **ClassName**(la fel ca în imaginea de mai sus) .

Se va apăsa butonul **Add Function** după ce în prealabil se va selecta **ID_POLIGON_MARIRE** și **COMMAND** și va apărea următoarea fereastră:



Laborator C++

În care se va scrie numele funcției, în cazul nostru implicit va fi scris numele **OnPoligonMarire**.

La fel se procedează și pentru adăugarea funcției de micșorare a numărului de laturi.

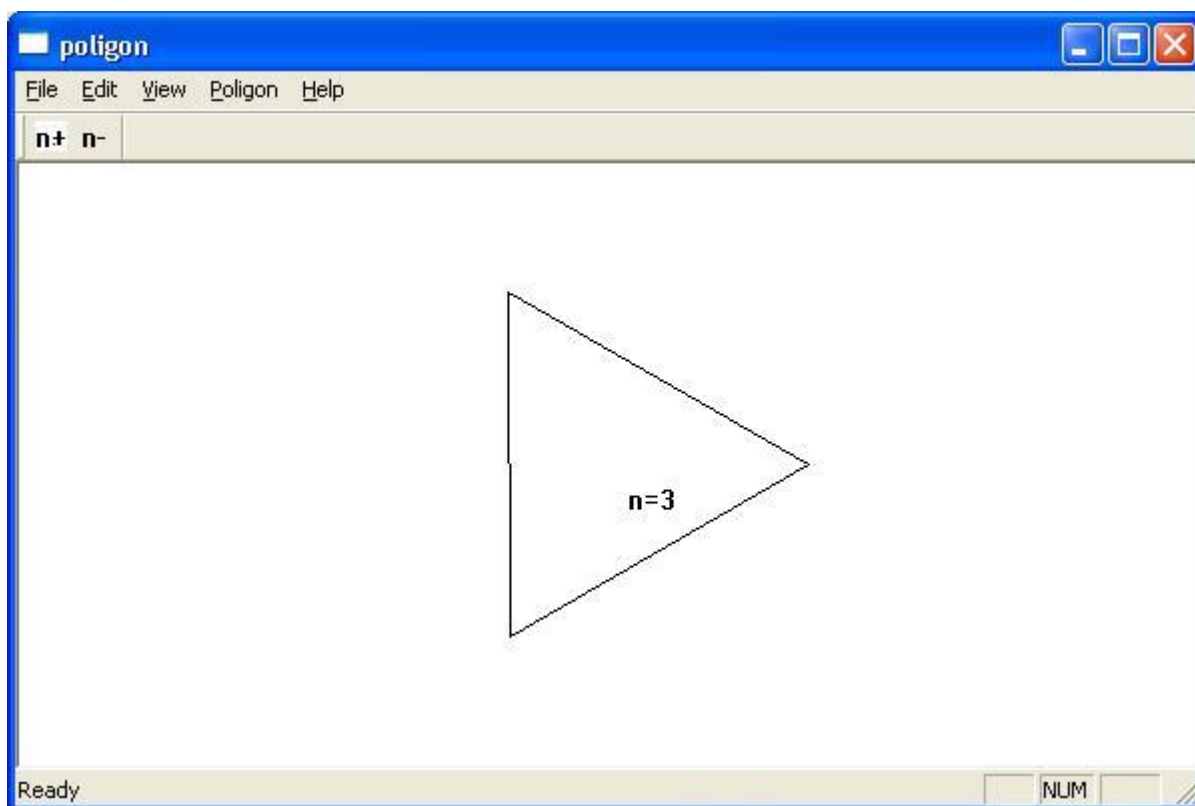
După realizarea acestor două funcții în **ClassWizard** se apasă butonul **Edit Code** și se va modifica codul astfel încât să arate astfel:

```
void CChildView::OnPoligonMarire()
{
    // TODO: Add your command handler code here
    n++;
    Invalidate();
}

void CChildView::OnPoligonMicsorare()
{
    // TODO: Add your command handler code here
    n--;
    Invalidate();
}
```

Se observă că se incrementează **n** respectiv se decrementează, iar funcția **Invalidate()** se folosește pentru a rescrie zona client a ferestrei.

După compilare și execuție vom obține următorul rezultat:



Problema 2:

Scrieți un program MFC care desenează și apoi rotește un cub.

Se crează o clasă Cpoint3D pentru reprezentarea unui punct în spațiu și rotirea punctului.

Se crează o clasă pentru cub. (Implementarea funcțiilor de rotație în jurul celor 3 axe ca răspuns la apăsarea unor taste. Eventual se va ține cont și de perspectivă).

(Funcție pentru scalarea cubului.)

Laborator 18

Obiective:

Familiarizarea cu:

Casete de dialog. Clasa document.

Funcția `AfxMessageBox()`. `CColorDialog`, `CFontDialog`.

Crearea casetelor de dialog modale. Adăugarea resurselor de tip dialog. Funcția `DoModal()`.

Problemă:

Scrieți un program MFC care cere utilizatorului introducerea unor date despre o persoană printr-o caseta de dialog (numele, prenumele, vârsta și salvează datele într-o colecție din clasa document). Toate datele introduse se afișează în zona client a ferestrei. Programul va permite și salvarea datelor aplicației într-un fișier pe disc și încărcarea datelor dintr-un fișier de pe disc (serializare).

Rezolvare:

Se crează un proiect de tip SDI urmând pași descriși în laboratorul 16 cu excepția faptului că la pasul 6 clasa de bază va fi **CScrollView** pentru a putea afișa în zona client a ferestrei pe mai multe linii și pentru a putea fi vizualizate aceste linii cu ajutorul unei bare de derulare. Pentru exemplu nostru am considerat numele proiectului **Persoane**.

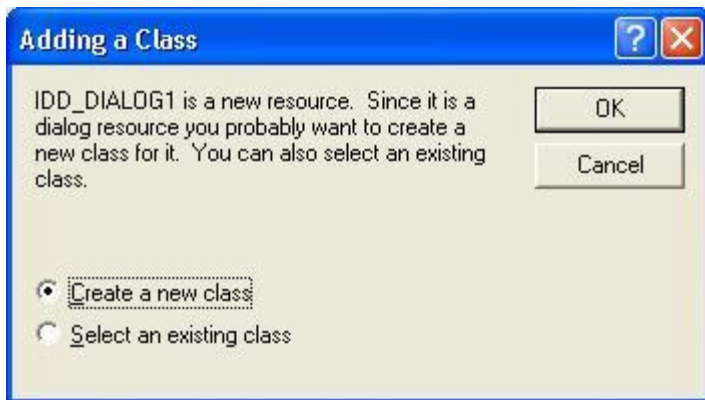
Următorul pas în rezolvarea problemei va fi inserarea unei casete de dialog astfel: se deschide **ResourceView**, se apasă un clic dreapta în dreptul directorului **Dialog** și din lista apărută se alege **Insert Dialog** iar în noul **Dialog** creat se vor adăuga trei controale de tip **Static Text** și în dreptul fiecăruia câte un control de tip **Edit Box**.

Caseta de dialog va avea următoarea formă:

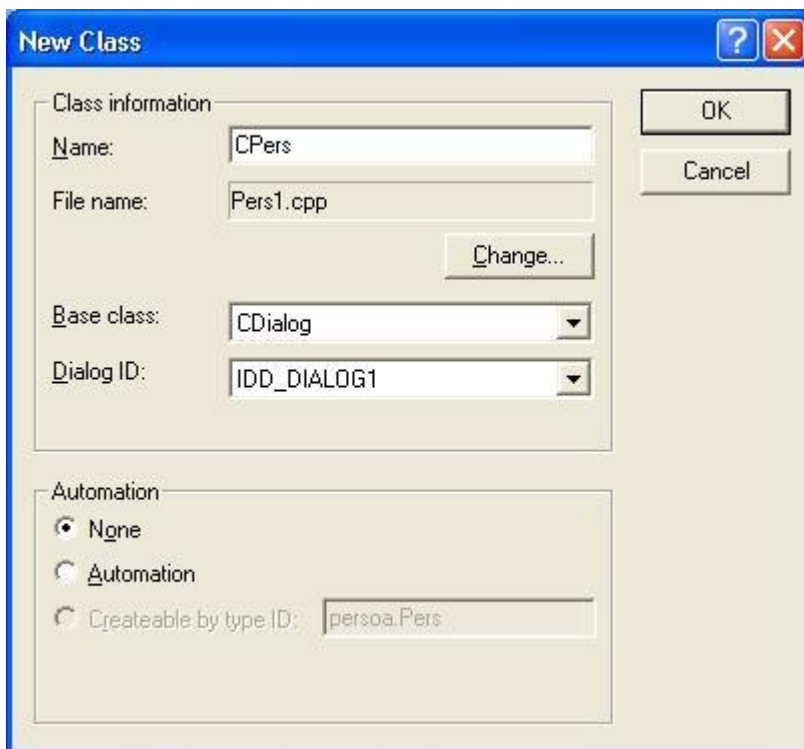


Se deschide din nou caseta de dialog creată și din meniul **View** se deschide **ClassWizard**, și ne va fi afișată o fereastră prin care putem crea o clasă corespunzătoare noului **Dialog**.

Laborator C++

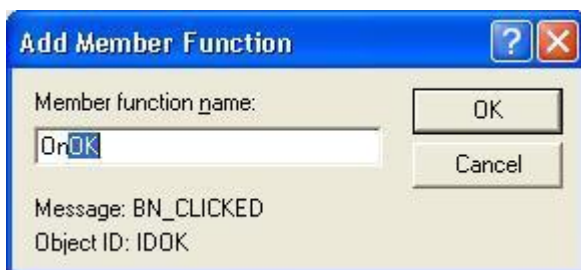


Se va selecta **Create a new class** și se va apăsa butonul **OK**, și ne va fi afișată o fereastră în care se va scrie numele Clasei, în exemplu prezentat am ales numele **CPers**, iar clasa de bază corespunzătoare va fi **CDialog**, iar prin butonul **Change** se poate modifica numele fișierele corepunzătoare acestei clase.



După ce am stabilit numele clasei, clasa de baza pentru clasa creată și ID-ul clasei se va apăsa butonul **OK**.

La următorul pas se apasă un dublu clic pe butonul **OK** pentru a se introduce o funcție care tratează apăsarea butonului respectiv. În fereastra apărută se va scrie numele funcției, implicit numele funcției va fi **OnOK** dar poate fi modificată (ceea ce nu este prea indicat).



Laborator C++

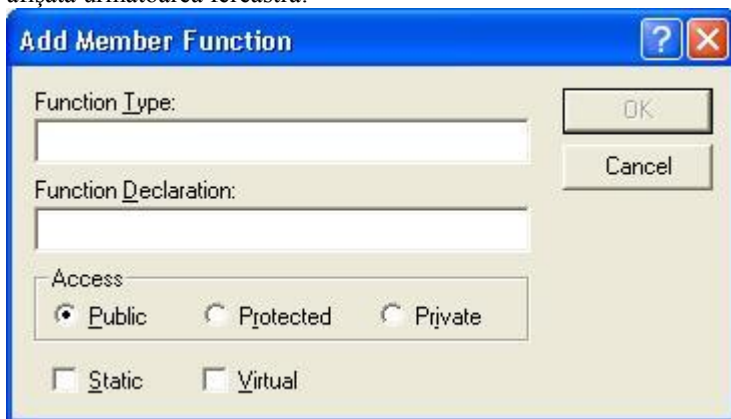
După stabilirea numelui funcției se apasă butonul **OK** din acea fereastră și va fi adăugată automat această funcție în care automat va fi introdusă următoarea linie de cod: **CDialog::OnOK()**; . La fel se adaugă și funcția care tratează apăsarea butonului **Cancel**.

În fereastra MFC **ClassWizard** se va alege pagina **Member Variables** și se va asocia casetelor de editare corespunzătoare Numelui și Prenumelui câte două variabile fiecare: una de tip **CEdit** și una de tip **CString** cu nume **m_nume** și **m_v_nume** respectiv **m_prenume** și **m_v_prenume**, iar casetei de editare corespunzător vârstei o variabilă de tip **UINT** cu numele **m_v_varsta** care va lua valori între 1 și 80 și o variabilă de tip **CEdit** cu numele **m_v** astfel:

se apasă butonul **AddVariable** după care în fereastra **Add Member Variable** se va scrie numele variabilei și va fi ales tipul ei. În cazul variabilelor de tip **CString** se va alege categoria **Value** iar în cazul variabilei de tip **CEdit** categoria **Control**.

Vom alege pe urmă în proiectul nostru **ClassView** și vom adăuga o nouă clasă apăsând un clic dreapta pe **Persoane Classes** și alegem opțiunea **New Class**. Clasa creată va fi **Generic Class** și numele **CPersoana** și care nu va fi derivată din altă clasă, iar în această clasă se va adăuga o funcție membră cu numele **Draw** care va avea ca parametri un pointer la **CDC** (Context de dispozitiv) și doi întregi. De asemenea acestei clase i se va adăuga și un constructor cu parametri: doi de tip referință la **CString** și un **unsigned int**.

Variabilele se vor introduce apăsând un click dreapta pe numele clasei și se alege **Add Member Variable** și va fi afișată următoarea fereastră:



Tipul funcției va fi **void** în cazul funcției **Draw** iar în cazul constructorului nu se va scrie nimic. La declarația funcției se va scrie numele urmat de o paranteză în care se va scrie parametri separați de virgulă.

După ce am introdus aceste funcții codul lor va fi modificat ca să arate astfel:

```
CPersoana::CPersoana(CString &nm, CString &pnm, unsigned int v)
{
    nume = nm;
    prenume = pnm;
    varsta = v;
}

void CPersoana::Draw(CDC *pDC, int x, int y)
{
    CString s;
    s.Format("%s %s %d", nume, prenume, varsta);
    pDC->TextOut(x, y, s);
}
```

Vor mai fi introduse tot în această clasă variabilele **nume** și **prenume** de tip **CString** și variabila vârsta de tip **unsigned**, prin opțiunea **Add Member Variable**.

Pentru serializare vom modifica funcția **Serialize()** a clasei **CPersoaneDoc** astfel:

Laborator C++

```
void CPersDoc::Serialize(CArchive& ar)
{
    int i, n;
    CString nume, pren;
    int v;
    if (ar.IsStoring())
    {
        // TODO: add storing code here
        ar << persoane.size();
        for (i = 0; i < persoane.size(); i++){
            ar << persoane[i].nume;
            ar << persoane[i].prenume;
            ar << persoane[i].varsta;
        }
    }
    else
    {
        // TODO: add loading code here
        ar >> n;
        for(i = 0; i < n; i++)
        {
            ar >> nume;
            ar >> pren;
            ar >> v;
            persoane.push_back(CPersoana(nume, pren, v));
        }
    }
}
```

De asemenea se va introduce o colecție cu numele **persoane** care este vector cu persoanele introduse astfel: se adaugă următorul cod în fișierul **PersoaneDoc.h** în secțiunea de declarații private

```
std::vector<CPersoana> persoane;
```

De asemenea tot în acest fișier se introduce și următorul cod

```
#include "Persoana.h"
#include <vector>
```

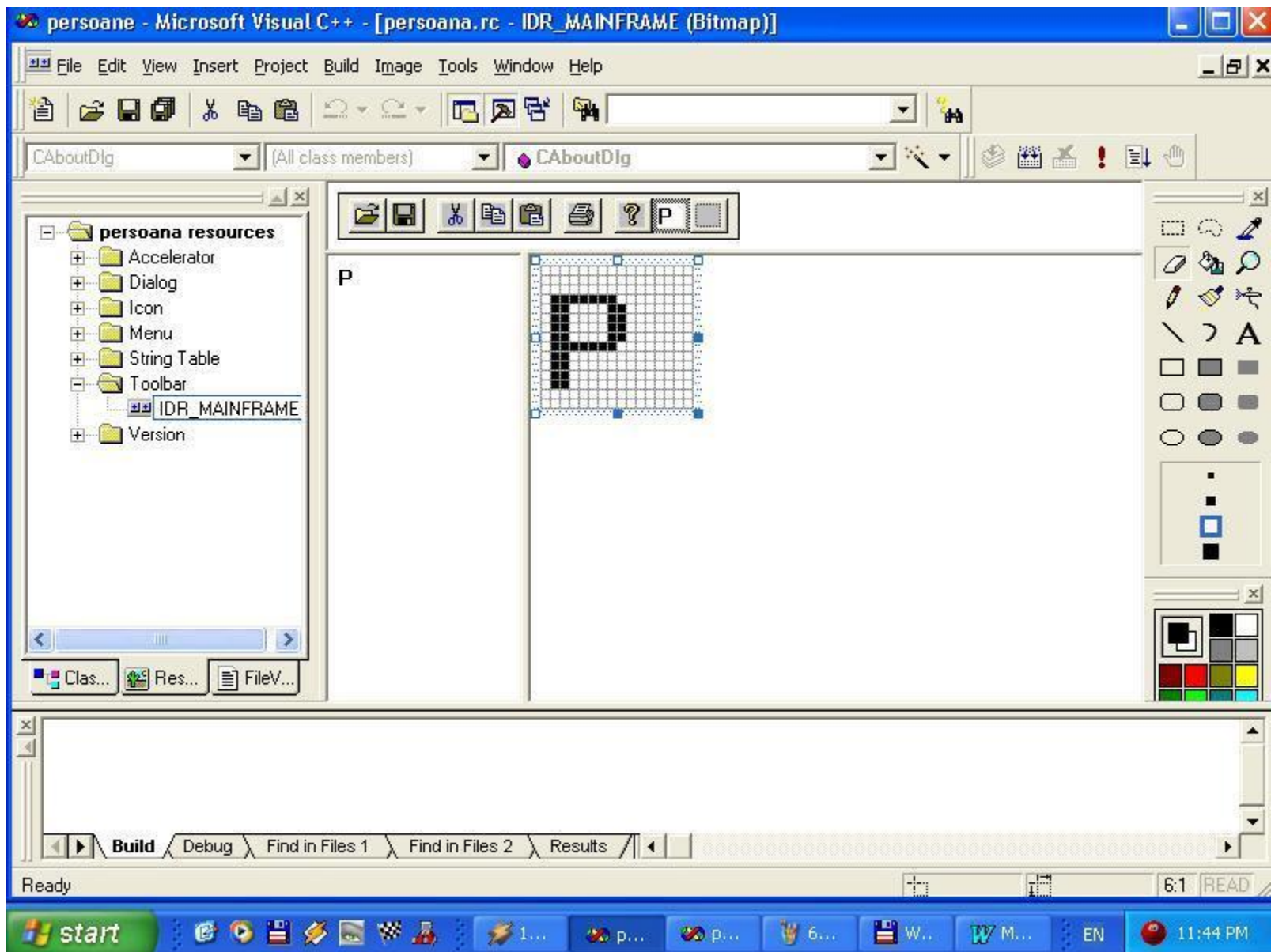
în secțiunea #include.

Funcția **OnDraw** a clasei **CPersoaneView** va fi modificată astfel:

```
void CPersoaneView::OnDraw(CDC* pDC)
{
    CPersoaneDoc* pDoc = GetDocument();
    ASSERT_VALID(pDoc);
    // TODO: add draw code for native data here
    for(int i = 0; i < pDoc->persoane.size(); i++)
        pDoc->persoane[i].Draw(pDC, 20, 20 + i * 20);
}
```

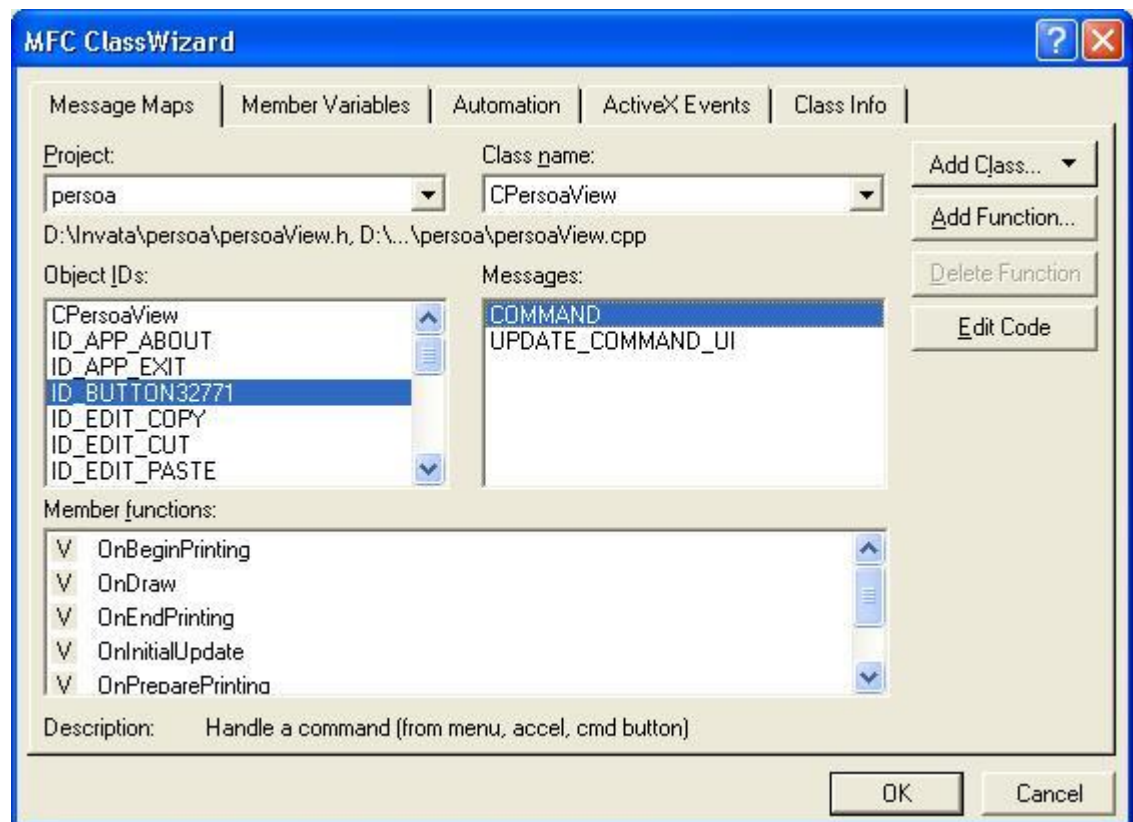
În continuare vom deschide pagina **ResourceView** din directorul **Toolbar** și în cadrul lui fișierul **IDR_MAINFRAME** și vom introduce un buton.

Laborator C++



Laborator C++

În **ClassWizard** se va asocia butonului creat o funcție.

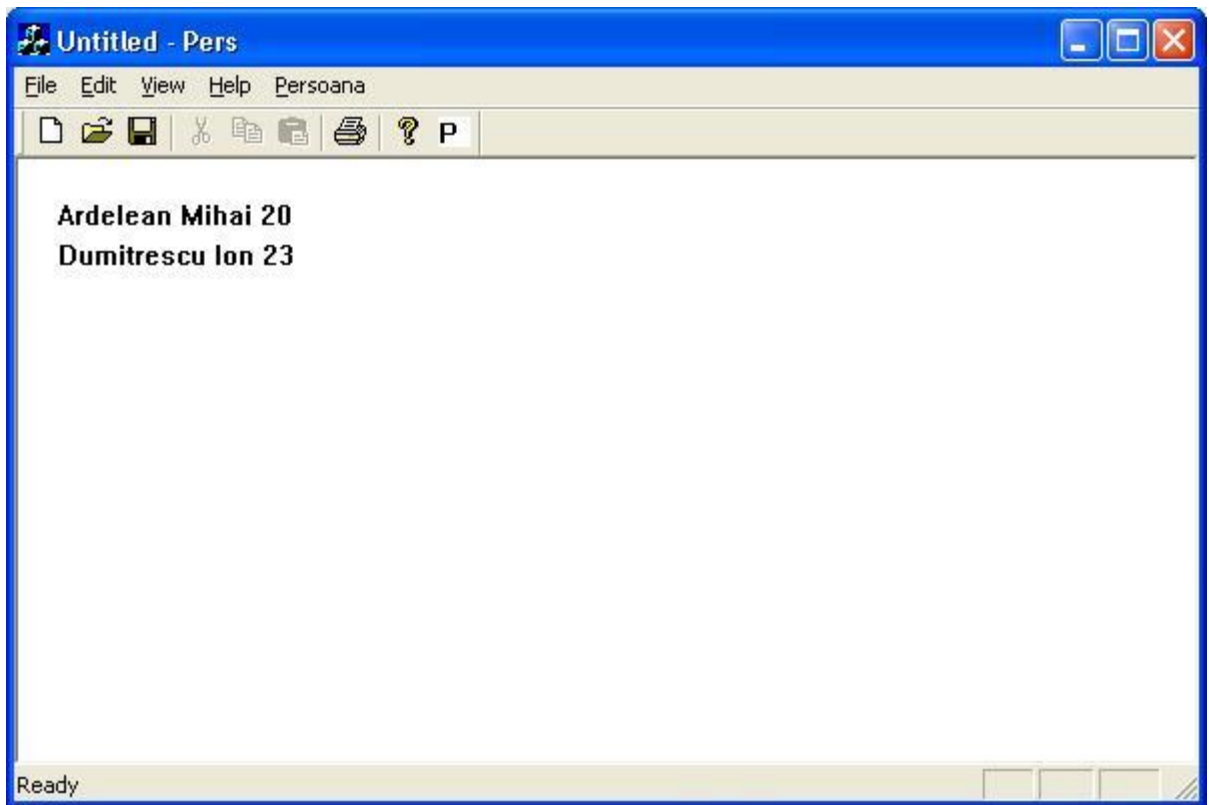


După apăsarea butonului **Add Function** se va adăuga funcția **OnButton32771** (numele acestei funcții se poate modifica, aici am lăsat numele funcției așa cum apare) care va trebui modificată prin apăsarea butonului **Edit Code** să arate astfel:

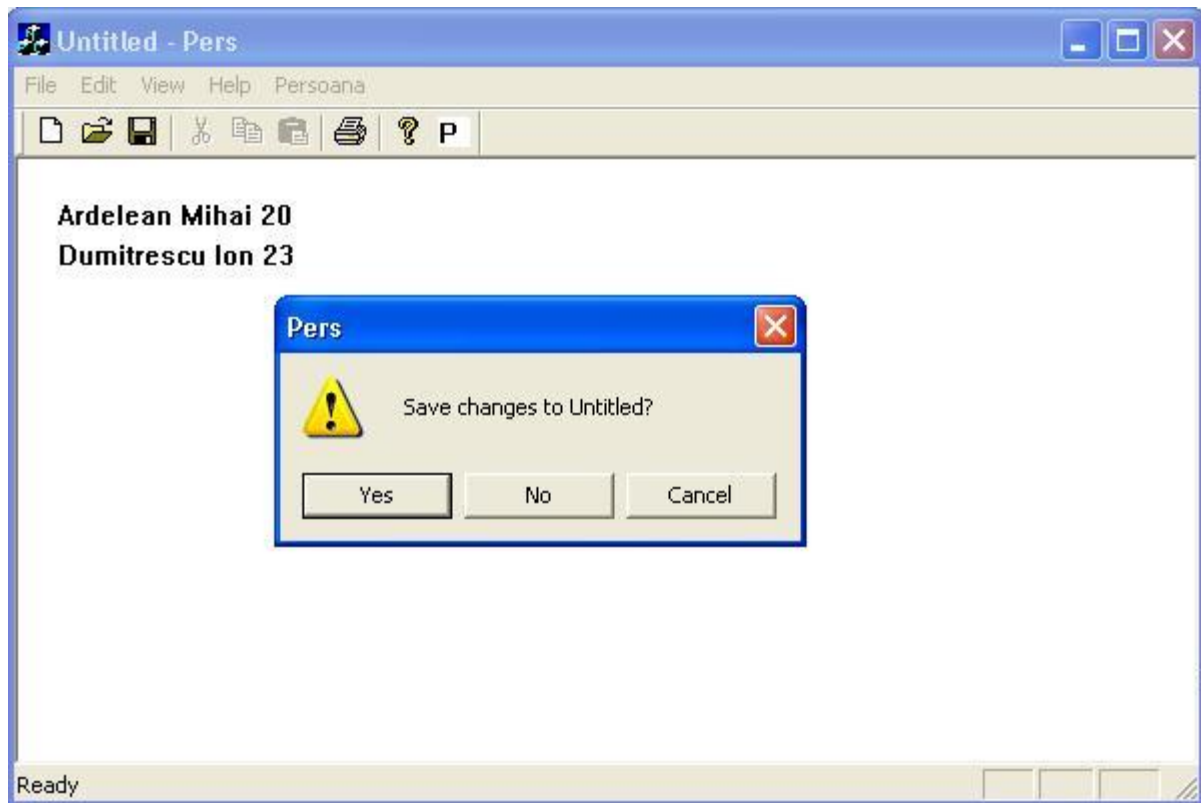
```
void CPersoaView::OnButton32771()
{
    CPers dlg;
    CPersoaDoc* pDoc = GetDocument();

    int ret = dlg.DoModal();
    if (ret == IDOK) {
        // colectez datele din controale si le pun intr-o colectie
        CPersoana p(dlg.m_v_nume, dlg.m_v_prenume, dlg.m_v_varsta);
        pDoc->p.push_back(p);
        pDoc->SetModifiedFlag();
        Invalidate();
    }
}
```

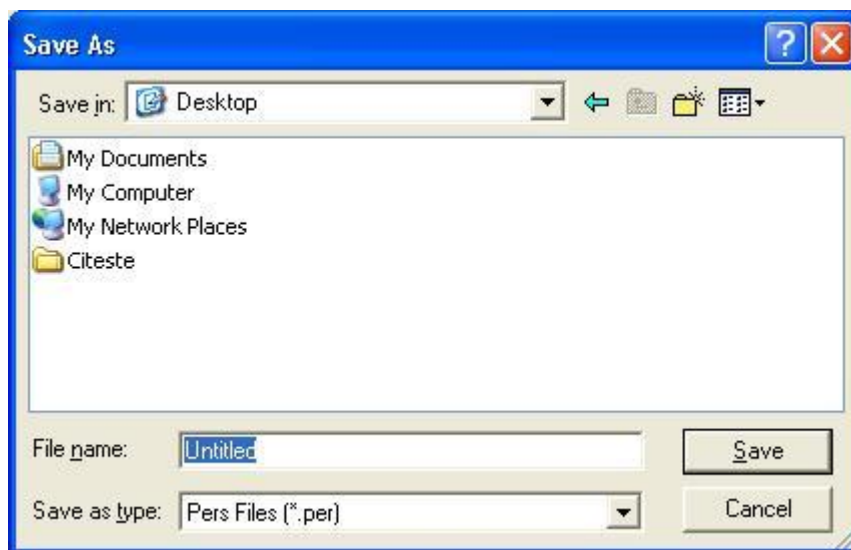
Astfel după ce vom compila și executa programul vom putea introduce date, aceste date fiind afișate ulterior în zona client a meniului astfel:



La final după introducerea datelor despre persoane dacă nu am salvat datele într-un fișier și vom dori să închidem fereastra programul ne va avertiza în legătură cu salvarea fișierului pe disc.



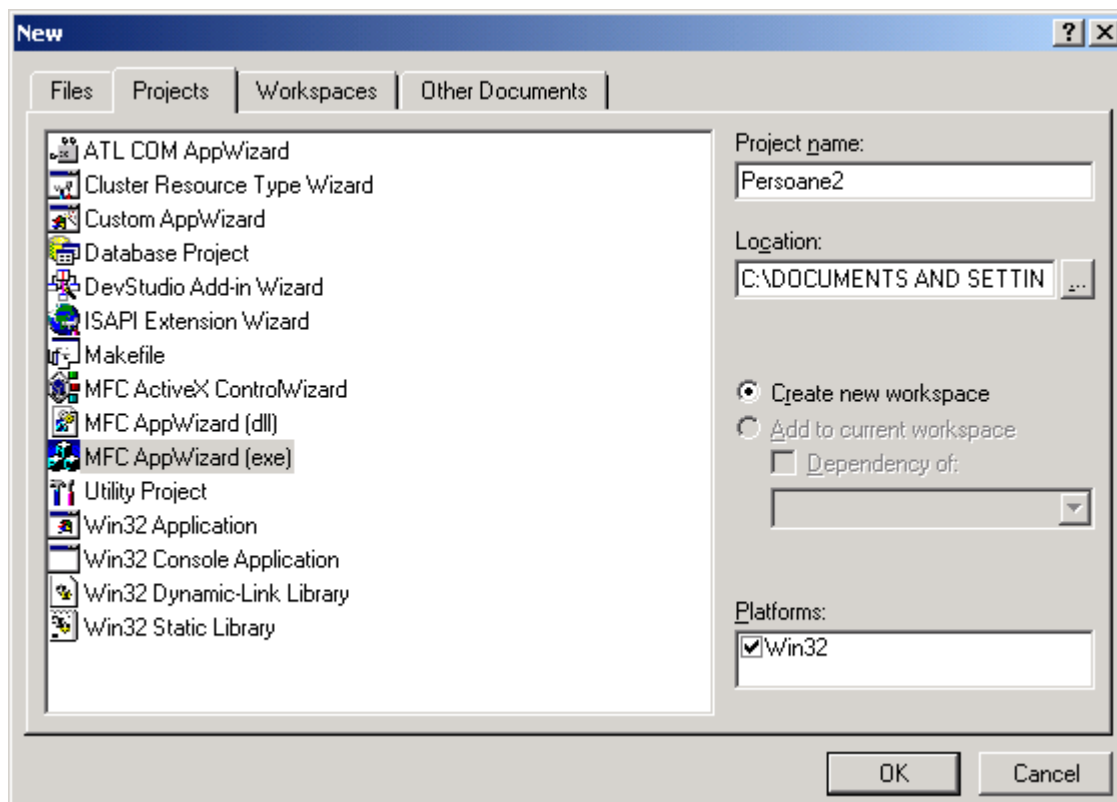
Dacă vom apăsa butonul **No** atunci vom renunța la salvarea pe disc a datelor, dacă vom apăsa **Yes** va apărea fereastra următoarea în care vom putea scrie numele și extensia fișierului în care se vor salva datele.



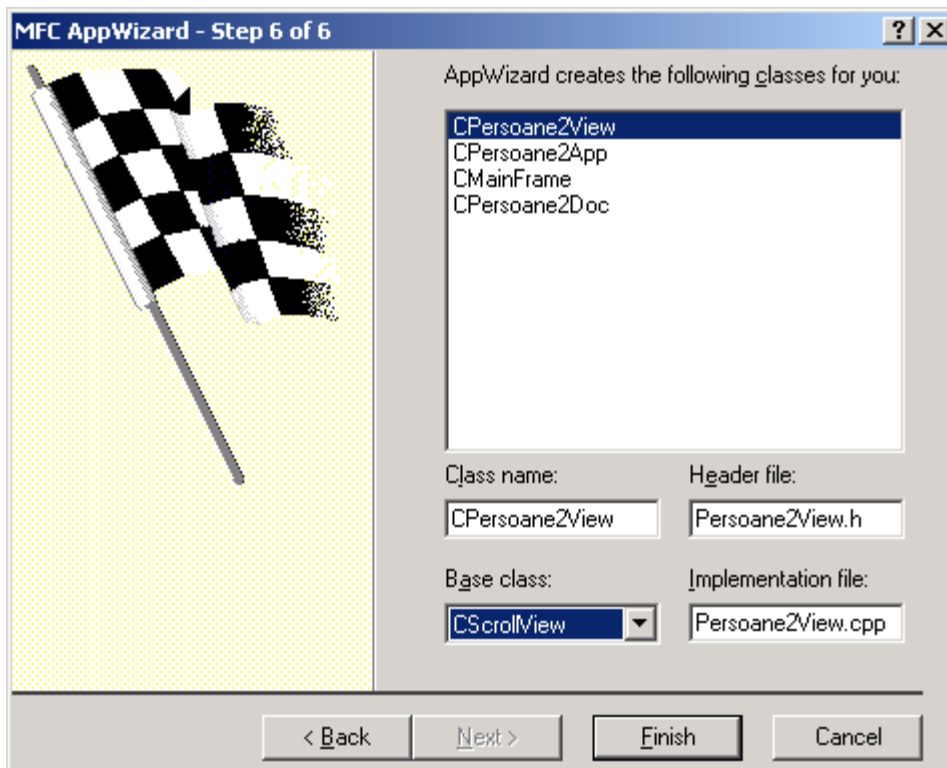
Laborator 19

Scrieți un program care cere utilizatorului introducerea datelor despre persoane (numele, prenumele, vârsta) și afișează în zona client a ferestrei datele introduse. Programul permite modificarea culorii de afișare (folosind clasa CColorDialog), modificarea fontului folosit (folosind clasa CFontDialog), modificarea culorii de fundal precum și căutarea unei persoane în colecție și ștergerea ei dacă utilizatorul cere aceasta. (clasa de baza a aplicației va fi CScrollView).

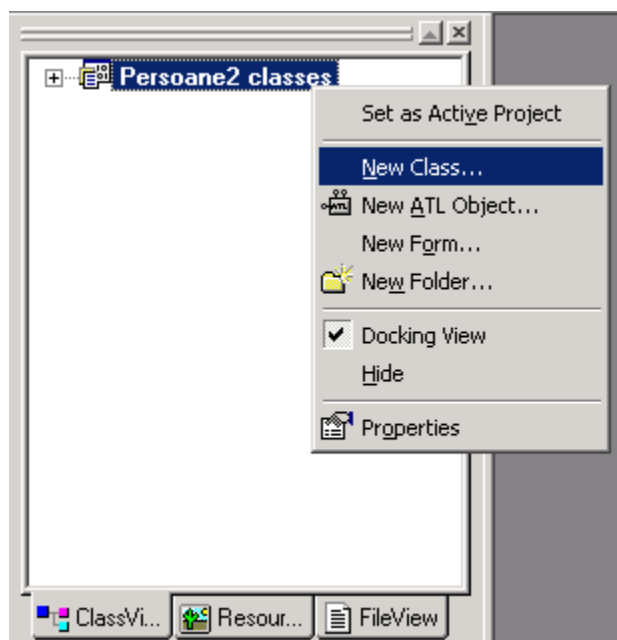
Pasul1: Creați o aplicație SDI (pe care noi am numit-o de exemplu **Persoane2**)



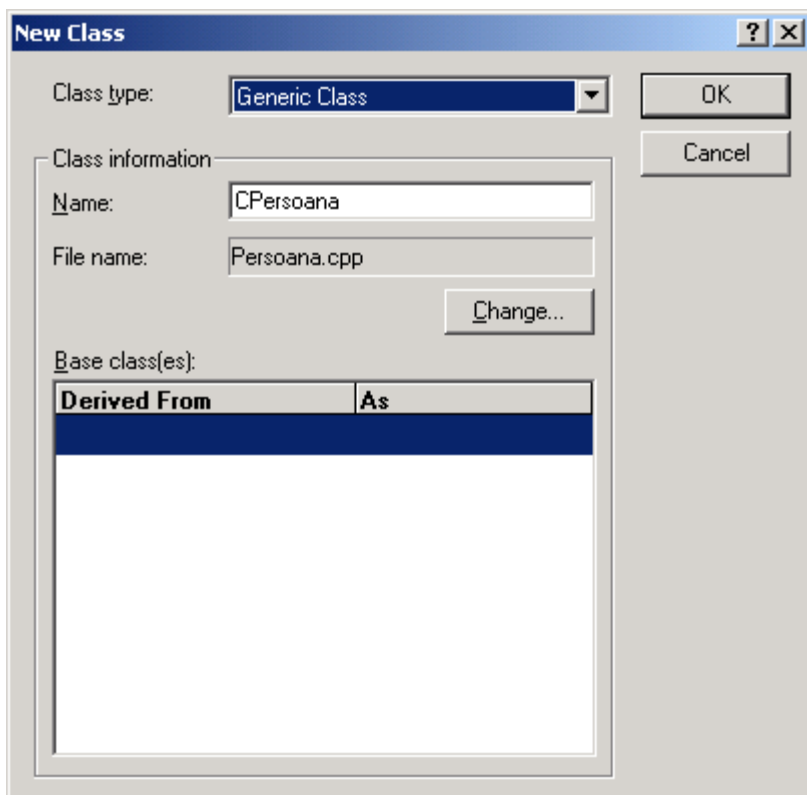
iar la **pasul 6** al lui **AppWizard** selectați astfel încât clasa de baza să fie **CScrollView** (vezi imaginea de mai jos).



Pasul 2: Vom crea o nouă clasă, **CPersoana**, care va conține următoarele date individuale: **nume**, **prenume**, respectiv **varsta**. Pentru aceasta dați un clic-dreapta pe foaia **ResourceView** și selectați opțiunea **New Class** (vezi imaginea următoare).



Acum ar trebui să vă apară caseta de dialog **New Class**, unde veți selecta ca tipul noi clase să fie una **generică** iar această nouă clasă am denumit-o **CPersoana** (vezi imaginea de mai jos). După introducerea acestor date apăsați pe butonul **OK** pentru a închide această casetă de dialog.



Efectuați, după aceea, un clic pe semnul “+” din stânga numelui **Persoane2 classes** din pagina **ClassView**. Veți vedea acum toate clasele proiectului. Efectuați un dublu-clic pe numele clasei **CPersoana** și adăugați liniile de cod de mai jos în declarația acestei clasei:

```
1) #pragma once
2) #endif // _MSC_VER > 1000
3)
4) class CPersoana
5) {
6) public:
7)     CPersoana();
8)     CPersoana( CString nume_ , CString prenume_ , int varsta_ );
9)     CString GetNume();
10)    CString GetPrenume();
11)    int GetVarsta();
12)    void SetNume( CString nume_ );
13)    void SetPrenume( CString prenume_ );
14)    void SetVarsta( int varsta_ );
15)
16)    virtual ~CPersoana();
17) private:
18)    CString Nume, Prenume;
19)    int Varsta;
20) };
```

Astfel am declarat, în liniile 18-19, variabilele membru **Nume**, **Prenume** și **Varsta** ca având tipul **CString**, **CString** respectiv **int** ; în liniile 7-8 se poate observa că vom avea doi constructori ai acestei clase (dintre care primul este implicit) .

Laborator C++

În plus vom avea încă șase funcții, câte două pentru fiecare variabilă membru a acestei clase: una care să ne returneze valoarea variabilei iar alta pentru stabilirea valorii acestei variabile.

Acum, efectuați din nou un clic pe semnul “+” din dreptul numelui clasei **CPersoana** și efectuați un dublu-clic pe numele ce se află imediat sub numele acestei clase (adică pe numele constructorului acestei clase:

CPersoana()) și completați cu liniile de cod de mai jos care reprezintă codul fiecărei funcții-membru în parte.

```
1)      CPersoana::CPersoana()
2)      {
3)          Nume="";
4)          Prenume="";
5)          Varsta=0;
6)      }
7)
8)      CPersoana::CPersoana( CString nume_ , CString prenume_ , int
varsta_ )
9)      {
10)         Nume=nume_;
11)         Prenume=prenume_;
12)         Varsta=varsta_;
13)     }
14)     CPersoana::~~CPersoana()
15)     {
16)
17)     }
18)
19)     CString CPersoana::GetNume()
20)     {
21)         return Nume;
22)     }
23)
24)     CString CPersoana::GetPrenume()
25)     {
26)         return Prenume;
27)     }
28)
29)     int CPersoana::GetVarsta()
30)     {
31)         return Varsta;
32)     }
33)
34)     void CPersoana::SetNume(CString nume_ )
35)     {
36)         Nume=nume_;
37)     }
38)
39)     void CPersoana::SetPrenume(CString prenume_ )
40)     {
41)         Prenume=prenume_;
42)     }
43)
44)     void CPersoana::SetVarsta(int varsta_ )
45)     {
46)         Varsta=varsta_;
47) }
```

Laborator C++

Pasul 3: Pentru a le ține în memorie și putea opera cu aceste date personale vom utiliza clasa predefinită **vector**. Pentru aceasta introduceți următoarele linii de cod în declarația clasei document **CPersoane2Doc** (aveți de introdus doar liniile 4, 5 și 16)

```
1) #pragma once
2) #endif // _MSC_VER > 1000
3)
4) #include <vector>
5) #include "Persoana.h"
6)
7)
8) class CPersoane2Doc : public CDocument
9) {
10)     protected: // create from serialization only
11)     CPersoane2Doc();
12)     DECLARE_DYNCREATE(CPersoane2Doc)
13)
14)     // Attributes
15)     public:
16)         std::vector<CPersoana> persoanele;
17)
18)
19)     // Operations
20)     public:
```

Absolut analog vom declara în clasa reprezentare (adică în clasa **CPersoane2View**) următoarele variabile membru: **culoare_scris**, **culoare_fundal** (pentru culoarea cu care să se scrie în zona **Client**, respectiv culoarea de fundal, ambele de tipul **COLORREF**), **If** (necesar fontului cu care să se afișeze textul, de tipul **LOGFONT**), și un întreg, **schimbare_font**, pe care îl vom folosi pentru a ști dacă pe parcursul rulării aplicației am dorit să schimbăm fontul caracterelor sau nu (în cazul din urmă vom lăsa fontul implicit). Pentru aceasta introduceți următoarele linii de cod (mai trebuie inserate doar liniile 12-14) în declarația clasei reprezentare:

```
1) class CPersoane2View : public CScrollView
2) {
3)     protected: // create from serialization only
4)     CPersoane2View();
5)     DECLARE_DYNCREATE(CPersoane2View)
6)
7)     // Attributes
8)     public:
9)         CPersoane2Doc* GetDocument();
10)
11)     public:
12)         COLORREF culoare_scris,culoare_fundal;
13)         LOGFONT lf;
14)         int schimbare_font;
15)
16)     // Operations
17)     public:
```

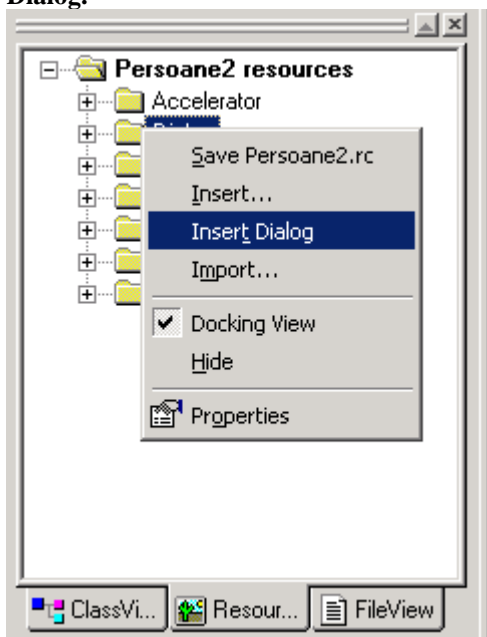
Pentru a inițializa aceste variabile-membru adăugați în constructorul clasei reprezentare următoarele linii de cod:

```
1) CPersoane2View::CPersoane2View()
2) {
```

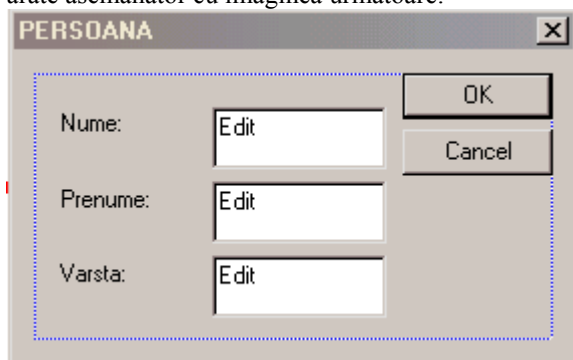
Laborator C++

```
3)      // TODO: add construction code here
4)      culoare_scri=RGB(0,0,0);
5)      culoare_fundal=RGB(255,255,255);
6)      schimbare_font=0;
7)      }
```

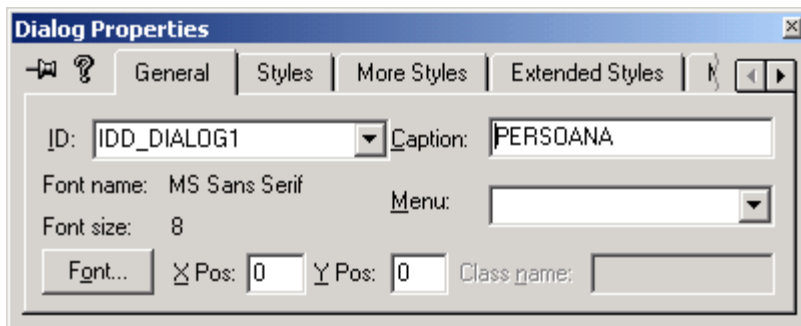
Pasul 4: Pentru ca utilizatorul să poată introduce datele unei persoane vom folosi o resursă de tip dialog. Pentru aceasta, pe pagina (lista) **ResourceView** apăsați clic-dreapta pe eticheta **Dialog** și selectați opțiunea **Insert Dialog**.



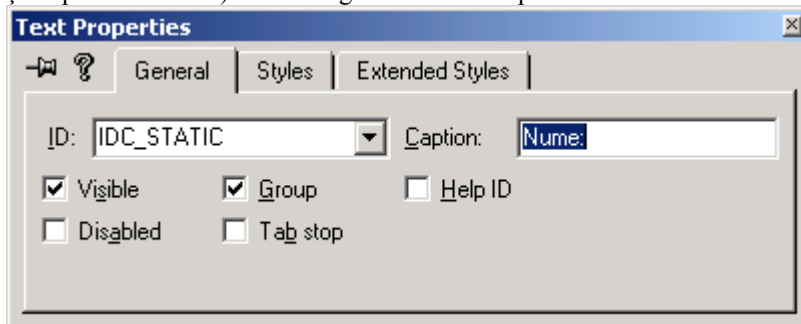
Acum adăugați pe această nouă resursă de tip dialog trei etichete statice și trei controale de editare astfel încât să arate asemănător cu imaginea următoare:



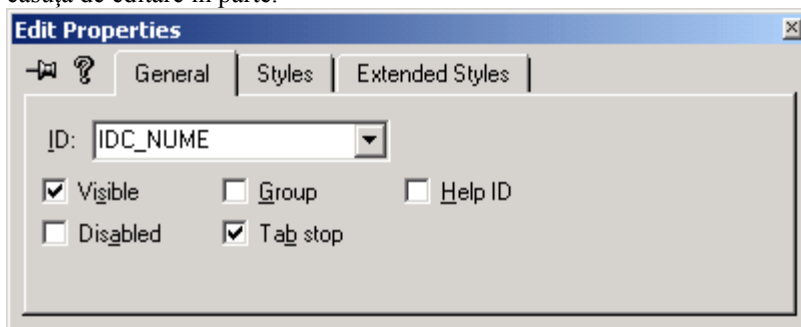
Pentru aceasta efectuați un clic-dreapta oriunde pe suprafața liberă a acestei casete de dialog și selectați opțiunea **Properties**. În căsuța de editare din dreapta etichetei **Caption** introduceți numele pe care doriți să-l aibă caseta de dialog atunci când ea va fi accesată (noi am introdus **PERSOANA**).



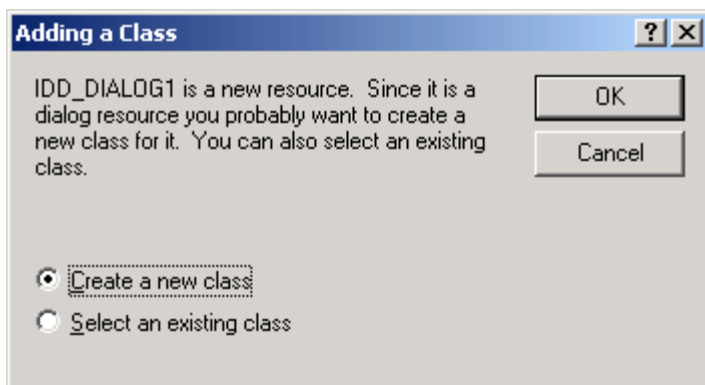
Efectuați apoi pe fiecare etichetă statică un clic-dreapta și selectați opțiunea **Properties**. În căsuța de editare din dreapta etichetei **Caption** introduceți numele pentru fiecare etichetă în parte. (noi am introdus **Nume:**, **Prenume:** și respectiv **Varsta:**). Vezi imaginea următoare pentru eventualele neclarități.



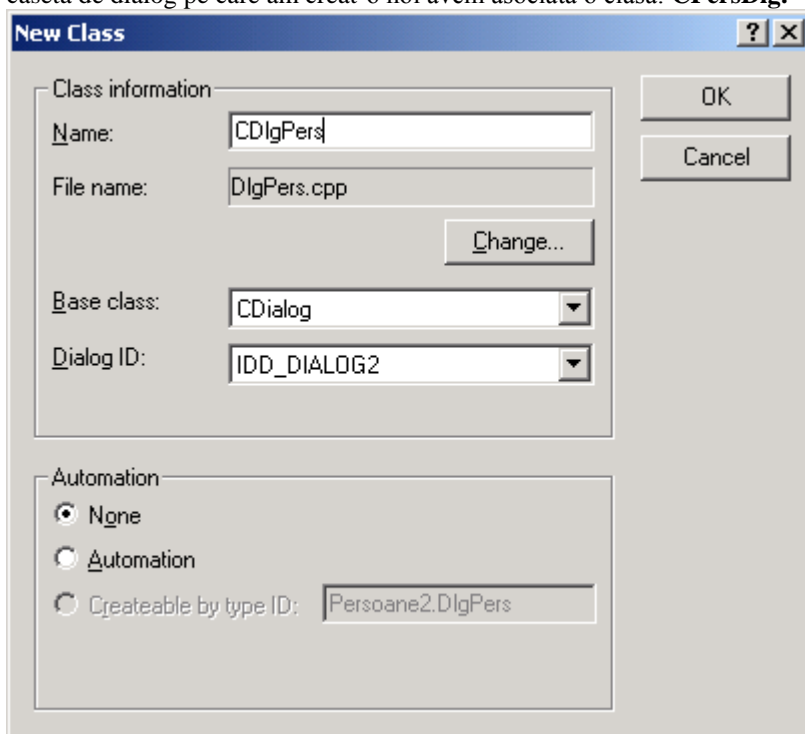
Vom asocia, în cele ce urmează, fiecare din cele trei casete de editare câte un nume și anume **IDC_NUME**, **IDC_PRENUME** și respectiv **IDC_VARSTA** astfel: efectuați pe fiecare caseta de editare câte un clic-dreapta și selectați opțiunea **Properties**. În căsuța de editare din dreapta etichetei **ID** introduceți numele pentru fiecare casuță de editare în parte.



Vom asocia (în cele ce urmează) această resursă de dialog cu o nouă clasă pe care o vom crea. Deschideți **ClassWizard** (apăsând direct **Ctrl+W** sau selectând opțiunea **ClassWizard** din meniul **View**). Se va deschide automat caseta de dialog **Adding a Class**. Apăsați pe butonul **OK** pentru a crea o nouă clasă.



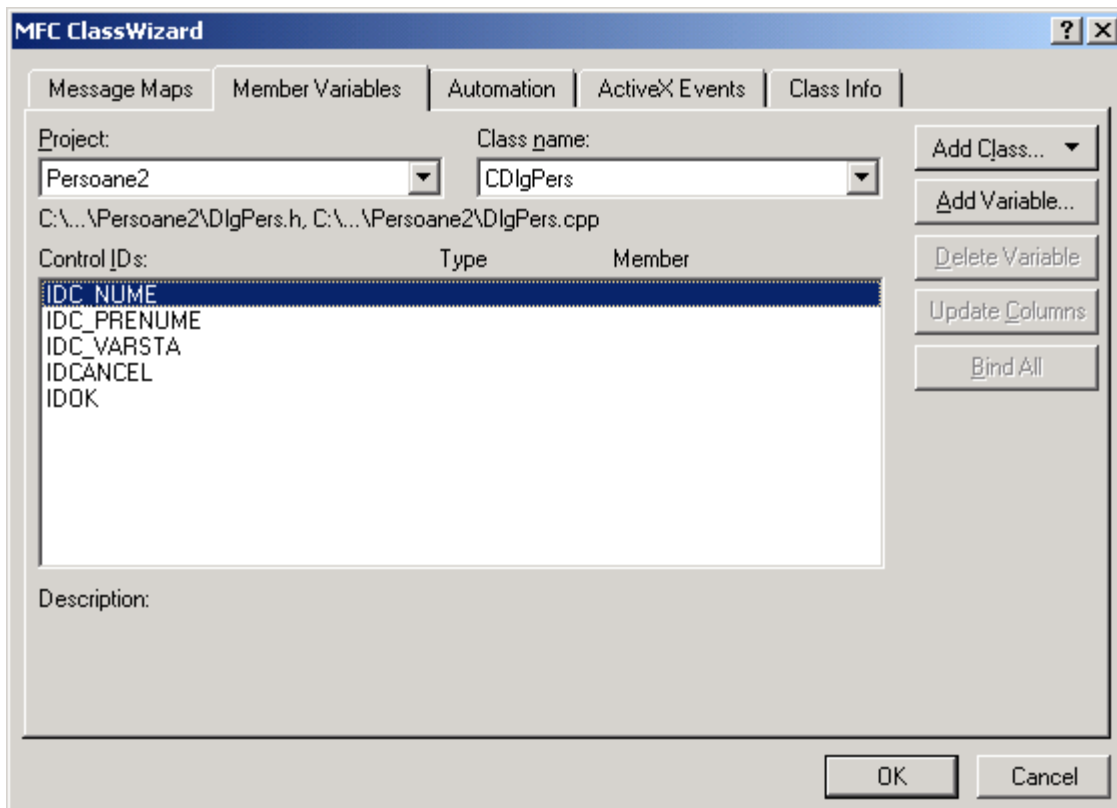
Acum se deschide caseta de dialog **New Class**. Denumiți această nouă clasă **CDlgPers**. În felul acesta, pentru caseta de dialog pe care am creat-o noi avem asociată o clasă: **CPersDlg**.



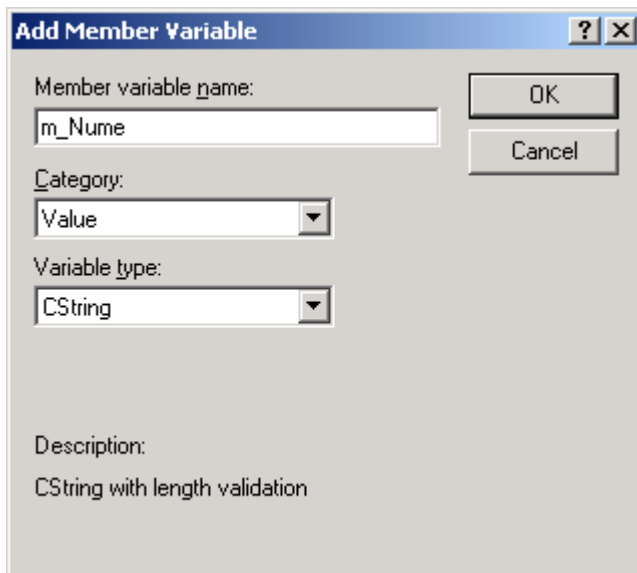
Pentru a putea lucra mai ușor cu datele introduse în caseta de dialog, vom asocia celor trei casete de editare (vom **mapa** peste ele) câte o variabilă (conform cu tabelul de mai jos):

Control ID	Variable Name	Category	Type
IDC_NUME	m_Nume	Value	CString
IDC_PRENUME	m_Prenume	Value	CString
IDC_VARSTA	m_Varsta	Value	int

Pentru aceasta, dați un clic pe oricare din aceste casete de editare după care deschideți **ClassWizard**. Ar trebui să se deschidă o casetă de dialog asemănătoare cu cea din imaginea de mai jos (daca nu este așa: selectati mai intai pagina **Members Variables** și asigurați-vă că este selectată (în câmpul **Class name**:) clasa **CPersDlg**).



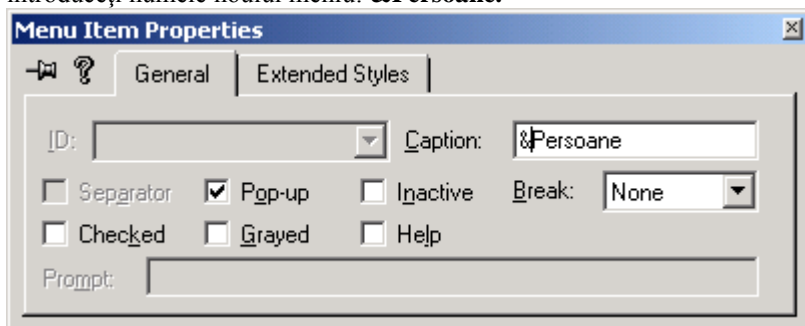
Selecțai **IDC_NUME** și efectuați un clic pe **Add Variable** (sau apăsați direct un dublu-clic pe **IDC_NUME**). Ar trebui să vă apară caseta de dialog **Add Member Variable**. Introduceți **m_Nume** pentru numele noii variabile membru pe care să o creeze **ClassWizard** și selecțai **CString** pentru tipul acestei variabile.



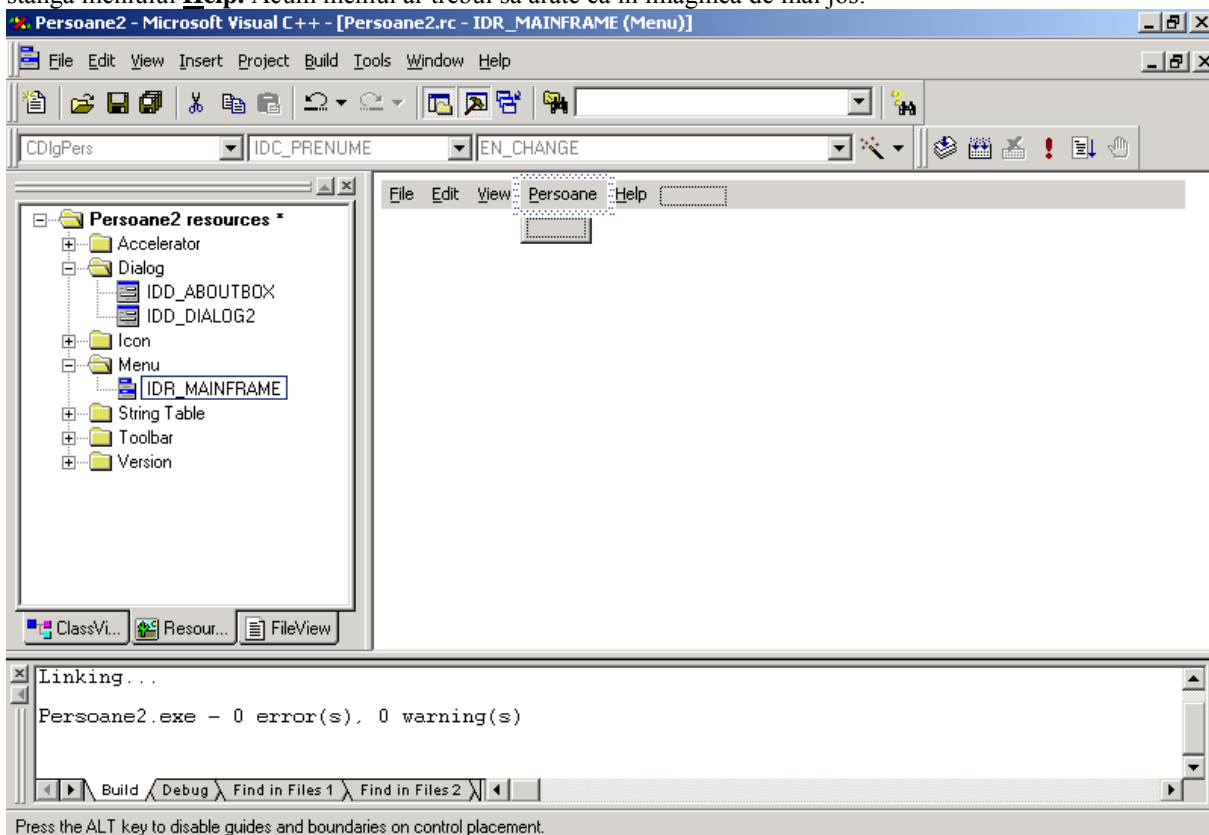
Procedați analog și pentru **IDC_PRENUME** (și dați-i numele **m_Prenume**, tipul **CString**) și respectiv **IDC_VARSTA** (și dați-i numele **m_Varsta**, tipul **int**).

Pașul 5: Acum vom crea o opțiune de meniu **Adaugare persoana** (cu ajutorul careia vom adăuga câte o persoană în lista **persoanele** de tipul **vector** de **CPersoana**) în meniul pe care tot noi îl vom crea și îl vom

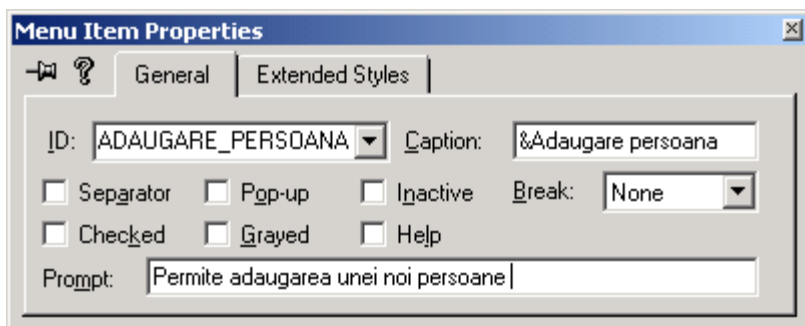
denumi **Persoane**. Mai întâi vom crea acest meniu **Persoane**. Pentru aceasta apăsați un clic pe semnul “+” ce se află în stânga etichetei **Menu** din pagina **ResourceView**. Efectuați acum un dublu-clic pe eticheta **IDR_MAINFRAME**. Pe dreptunghiul gol din dreapta etichetei **Help** de sus efectuați un clic dreapta și selectați opțiunea **Properties**. Se va deschide caseta de dialog **Menu Item Properties**. În caseta de editare **Caption** introduceți numele noului meniu: **&Persoane**.



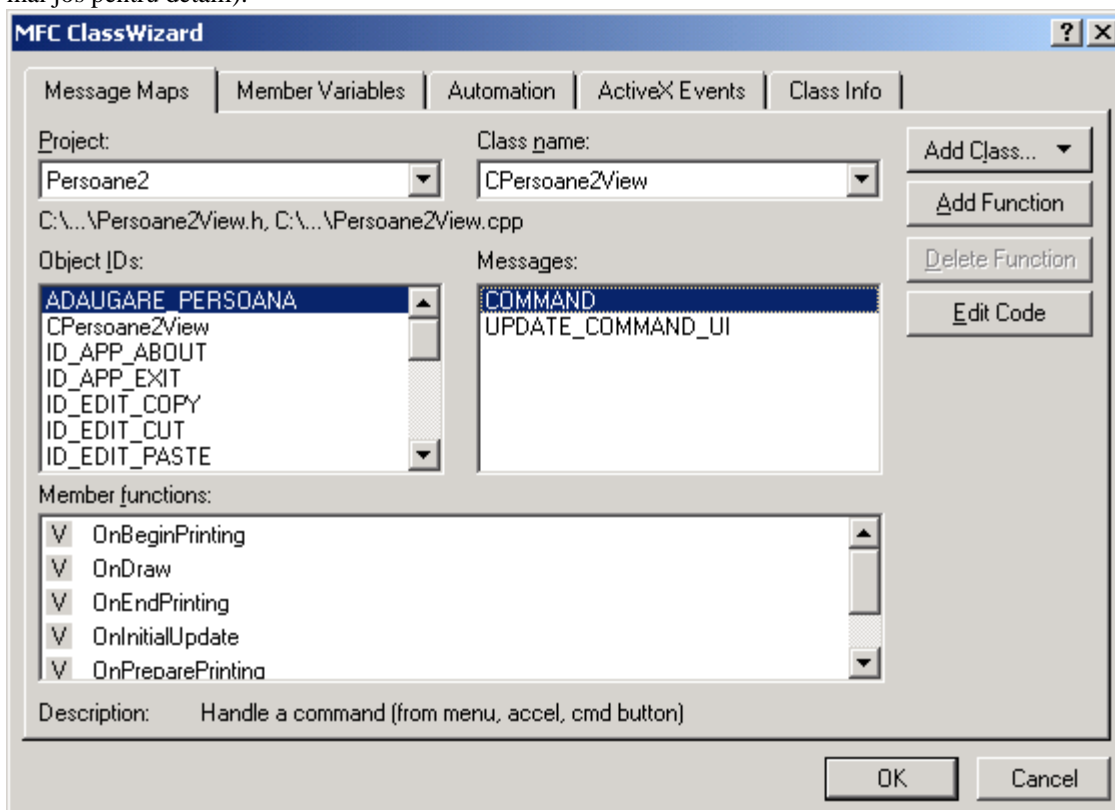
Semnul & dinaintea literei “P” va determina ca în numele acestui meniu acea literă să apară subliniată (în cazul nostru **P**). Aceasta se folosește pentru așa numitele “taste fierbinti”. Astfel, în cazul de față prin simpla apăsare simultană a tastelor **Alt+P** vom accesa direct meniul **Persoane**. Dacă dorim ca în numele unui meniu sau a unei opțiuni din meniu să apară și caracterul &, acesta trebuie dublat adică vom scrie && peste tot unde vom dori să apară &. Pentru a schimba poziția meniului **Persoane** (deoarece poziția sa firească ar fi înaintea meniului **Help**) vom apăsa un clic pe dreptunghiul unde său și, ținând apăsat butonul stâng al mouse-ului, îl “tragem” înspre stânga meniului **Help**. Acum meniul ar trebui să arate ca în imaginea de mai jos:



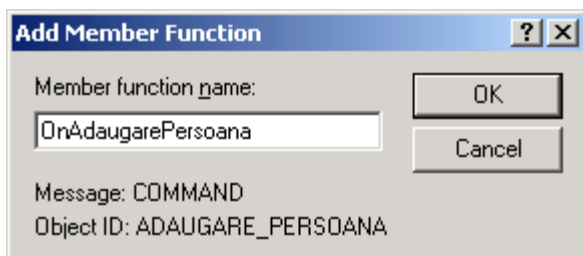
Acum vom crea opțiunea de meniu **Adaugare persoana**. Pentru aceasta vom urma aceiași pași de mai sus însă dreptunghiul gol folosit va fi cel care s-a creat automat sub numele **Persoane**. Schimbați și numele din dreapta etichetei statice **ID**: cu numele **ADAUGARE_PERSOANA** (vezi imaginea de mai jos)



Pasul 6: Pentru ca atunci când selectăm opțiunea **Adaugare persoana** din meniul **Persoane** să se efectueze ceva (de exemplu, în cazul nostru, să se deschidă o casetă de dialog de tipul **CDlgPers**) va trebui să apelăm din nou la **ClassWizard**. Pentru aceasta deschideți **ClassWizard**. Selectați pagina **Message Maps**, la numele clasei selectați **CPersoane2View** (am ales această clasă pentru a putea lucra mai ușor cu membrii acestei clase dar la fel se putea folosi oricare altă clasă), iar la **Object ID's** selectați **ADAUGARE_PERSOANA**. Selectați astfel ca tipul mesajului care să fie tratat să fie **COMMAND** apoi efectuați un clic-stânga pe **Add Function** (vezi imaginea de mai jos pentru detalii).



Schimbați numele implicit de funcție care apare în caseta de dialog **Add Member Function** (sau puteți să-l lăsați nemodificat dacă vă convine acest nume de funcție mai mult) cu numele **OnAdaugarePersoana** și apăsați pe butonul **OK**.

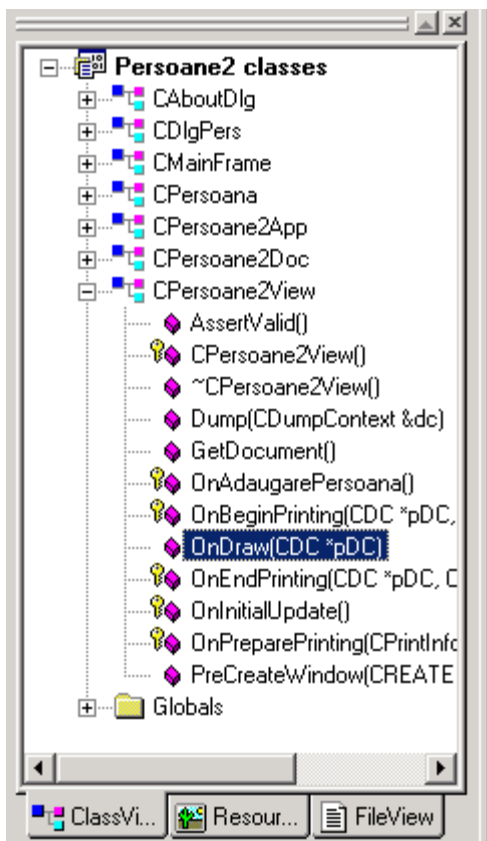


În final apăsați pe butonul **Edit Code** pentru ca **ClassWizard** să vă deschidă fișierul unde s-a creat noua funcție și să editați codul sursă al acestei funcții. Acolo adăugați liniile de cod de mai jos:

```
1)  //////////////////////////////////////
2)  // CPersoane2View message handlers
3)  #include "Persoana.h"
4)  #include "DlgPers.h"
5)  void CPersoane2View::OnAdaugarePersoana()
6)  {
7)      // TODO: Add your command handler code here
8)      CPersoane2Doc* pDoc = GetDocument();
9)      ASSERT_VALID(pDoc);
10)
11)     CDlgPers dlg;
12)     int valoare_return;
13)
14)     valoare_return=dlg.DoModal();
15)     if(valoare_return==IDOK)
16)     {
17)         pDoc->persoanele.push_back(CPersoana(dlg.m_Nume,dlg.m_Prenume,dlg.m_Varsta));
18)     }
19)     Invalidate();
20) }
```

În liniile 8-9 declarăm un pointer de tipul **CPersoane2Doc** prin care vom face o legătură între clasa reprezentare și clasa document. În linia a 11-a declarăm un obiect de tipul **CDlgPers**. În linia a 14-a folosind funcția membru **DoModal()** se va deschide caseta de dialog **dlg**. Dacă vom apăsa butonul **OK** al acestei casete de dialog atunci se va returna constanta întreagă predefinită **IDOK**, iar dacă apăsăm pe butonul **Cancel** al acestei casete de dialog se va returna **IDCANCEL**. Folosind funcția membru **push_back()**, vom introduce în vectorul de persoane **persoanele** acea nouă persoană pe care tocmai am introdus-o în această casetă de dialog. Astfel că, folosind variabilele membru ale casetei de dialog, vom transmite constructorului clasei **CPersoana** cele trei variabile: **m_Nume**, **m_Prenume** și **m_Varsta**. Pentru a redesena ecranul (zona **Client** a ferestrei) folosim funcția **Invalidate()**.

Pasul 7: Pe numele funcției **OnDraw()** pe care o găsiți pe pagina **ClassView** (vezi imaginea de mai jos)



scrieți următoarele linii de cod:

```

1) void CPersoane2View::OnDraw(CDC* pDC)
2) {
3)     CPersoane2Doc* pDoc = GetDocument();
4)     ASSERT_VALID(pDoc);
5)     // TODO: add draw code for native data here
6)
7)     CString str;
8)     UpdateData();
9)     pDC->SetTextColor(culoare_scris);
10)    pDC->SetBkColor(culoare_fundal);
11)
12)    if(schimbare_font)
13)    {
14)        CFont *fontul=new CFont();;
15)        fontul->CreateFontIndirect(&lf);
16)        pDC->SelectObject(fontul);
17)    }
18)
19)    //pentru a aparea bara de derulare verticala daca va fi cazul!!
20)
21)    TEXTMETRIC tm;
22)    pDC->GetTextMetrics(&tm);
23)    int inaltime_text=tm.tmHeight;
24)
25)    int n=pDoc->persoanele.size();

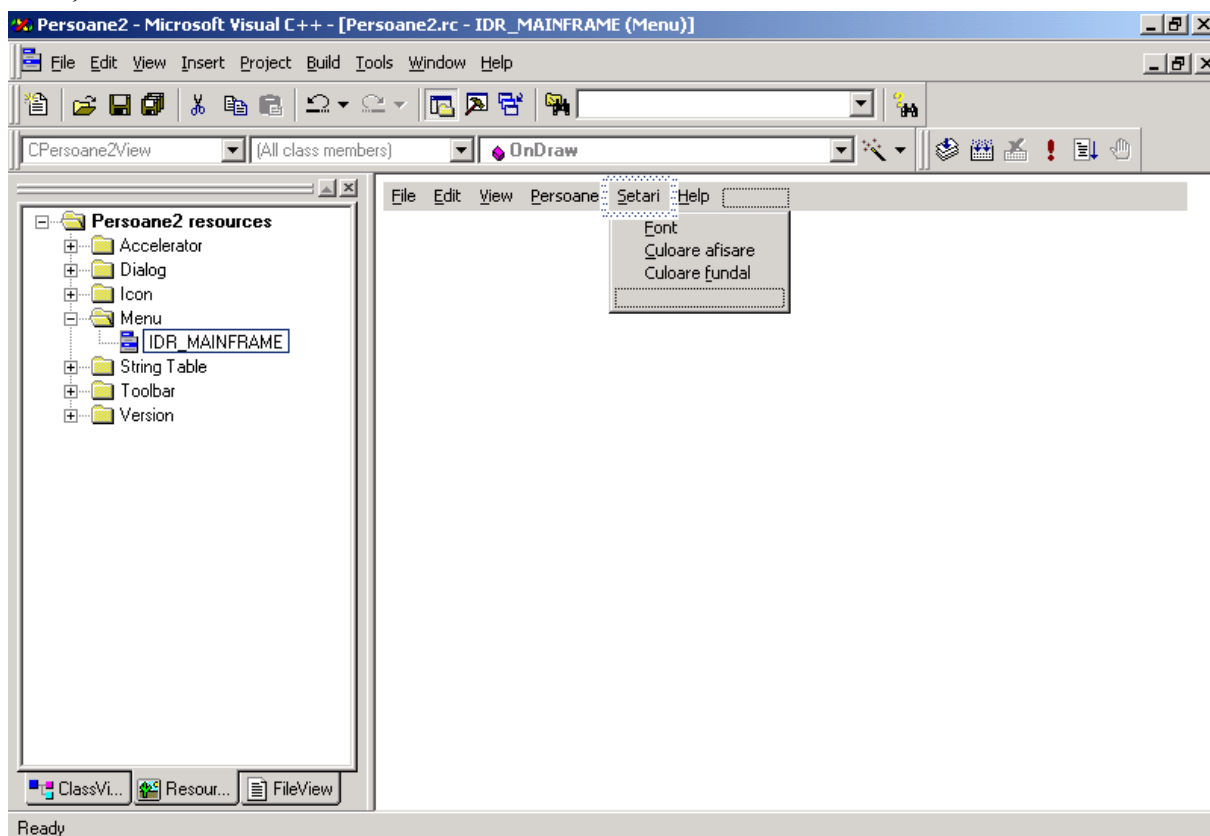
```

Laborator C++

```
26) CSize sizeTotal;
27) sizeTotal.cx =100;
28) sizeTotal.cy = inaltime_text*(n+1);
29) SetScrollSizes(MM_TEXT, sizeTotal);
30)
31)
32)
33) for(int i=0;i<n;i++)
34) {
35)     str.Format("%s  %s, varsta: %d ",pDoc->persoanele[i].GetNume(),
        pDoc->persoanele[i].GetPrenume(),pDoc->persoanele[i].GetVarsta());
36)     pDC->TextOut(10,inaltime_text*(i+1),str);
37) }
38)
39) }
```

În liniile 9-10 setăm culoarea de scris și cea de fundal pe care le-am ales (altfel rămân cele implicite). În liniile 12-17 schimbăm fontul de scriere (asta doar în cazul în care utilizatorul a ales vreun font). În liniile 26-29 vom crea o bară de derulare verticală atunci când este cazul (adică atunci când avem mai multe linii de afișat decât se poate afișa între dimensiunile curente). Iar în liniile 33-37 se afișează toate persoanele pe care le-am introdus în vectorul **persoanele**.

Pasul 8: Vom adăuga acum opțiunile de meniu necesare schimbării culorii de fundal, culorii de scriere și a fontului. Pentru aceasta veți lucra analog cu **pasul 6** (vezi imaginea următoare pentru mai multe detalii). Astfel că veți crea un meniu nou **Setari culoare si font**, unde veți crea trei opțiuni de meniu: **Setare Font, Culoare afisare** și **Culoare fundal**.



Laborator C++

Cu ajutorul lui **ClassWizard** creați cele trei funcții de tratare a acestor trei opțiuni de meniu (nu uitați să alegeți astfel încât clasa de bază să fie **CPersoane2View**!).

După ce apăsați pe butonul **Edit Code** inserați liniile de cod de mai jos:

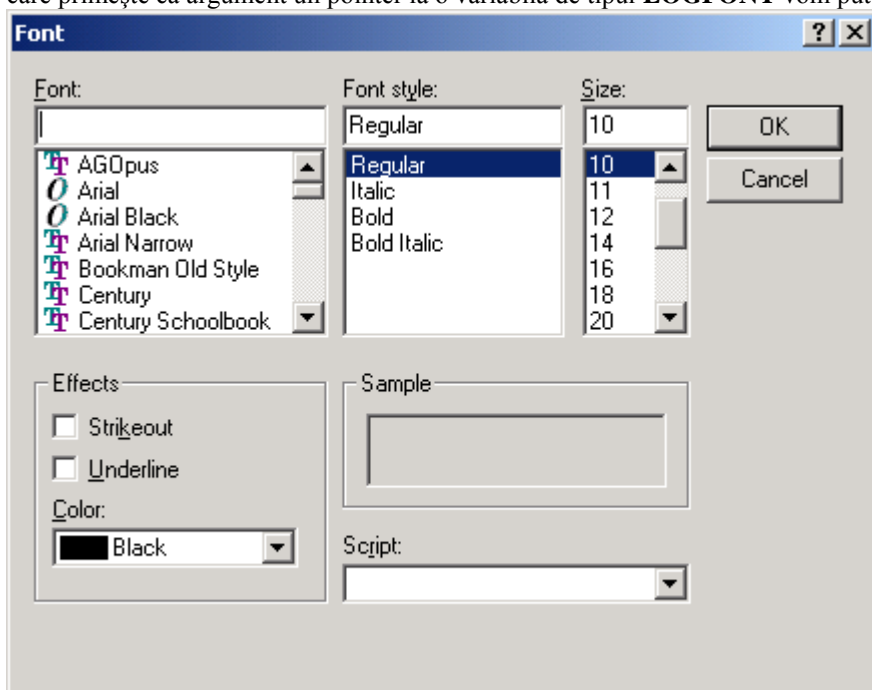
```
1) void CPersoane2View::OnFont()
2) {
3)     // TODO: Add your command handler code here
4)     //schimbarea fontului folosit
5)
6)     CFontDialog dlg;
7)     int ret = dlg.DoModal();
8)     if (ret == IDOK)
9)     {
10)         schimbare_font=1;
11)         dlg.GetCurrentFont(&lf);
12)         Invalidate();
13)     }
14) }
15)
16) void CPersoane2View::OnCuloareAfisare()
17) {
18)     // TODO: Add your command handler code here
19)
20)     CColorDialog dlg;
21)
22)     int valoare_return;
23)     valoare_return=dlg.DoModal();
24)     if(valoare_return==IDOK)
25)     {
26)         culoare_scris=dlg.GetColor();
27)     }
28)     Invalidate();
29)
30) }
31)
32) void CPersoane2View::OnCuloareFundal()
33) {
34)     // TODO: Add your command handler code here
35)     CColorDialog dlg;
36)
37)     int valoare_return;
38)     valoare_return=dlg.DoModal();
39)     if(valoare_return==IDOK)
40)     {
41)         culoare_fundal=dlg.GetColor();
42)     }
43)     Invalidate();
44) }
```

În liniile 20 și respectiv 35 am declarat un obiect (**dlg**) de tip **CColorDialog**. Cu ajutorul funcției membru **DoModal()** deschidem această casetă de dialog. Dacă vom apăsa pe butonul **OK** această casetă de dialog va returna constanta **IDOK**. În acest caz, cu ajutorul funcției membru **GetColor()**, vom obține culoarea selectată și o vom copia în variabila membru a clasei reprezentare **culoare_scris** respectiv **culoare_fundal**. Analog se operează și în cazul celeilalte funcții (folosind însă **CFontDialog**). Vom folosi funcția **Invalidate()** pentru a se redesena zona **client** de fiecare dată când schimbăm vreo culoare sau fontul.

Sistemul de operare **Windows** conține o serie de casete de dialog. Acestea se folosesc pentru alegerea culorii, a fontului, pentru alegerea unui fișier ce trebuie deschis, etc. Mai sus noi am folosit două astfel de casete de dialog: cea pentru alegerea culorii și cea pentru selectarea fontului. Pentru a folosi caseta de dialog pentru alegerea culorii trebuie doar să declarăm un obiect de tipul **CColorDialog** și să folosim funcția membru **DoModal**. Imaginea de mai jos ne arată caseta de dialog ce se folosește pentru alegerea culorii. Funcția membru **GetColor** ne returnează culoarea selectată (de fapt această funcție returnează o valoare de tipul **COLORREF**).



Analog și pentru alegerea fontului folosind însă clasa **CFontDialog**. Imaginea de mai jos ne arată caseta de dialog ce se folosește pentru alegerea fontului. Cu ajutorul funcției membru (a acesetei clase) **GetCurrentFont** care primește ca argument un pointer la o variabilă de tipul **LOGFONT** vom putea obține fontul ales.



Laborator C++

În ambele cazuri funcția membru **DoModal** returnează una din următoarele constante predefinite: **IDOK** sau **IDCANCEL** în funcție de butonul pe care s-a apăsător pentru închiderea casetei de dialog.
Constructorul clasei **CColorDialog** este:

CColorDialog(COLORREF clrInit = 0, DWORD dwFlags = 0, CWnd* pParentWnd = NULL);

unde:

clrInit	Este culoarea implicită de selecție. Dacă nu se alege nici o culoare valoarea implicită este RGB(0,0,0) (adica negru).
dwFlags	O mulțime de indicatori care personalizează funcția și reprezentarea casetei de dialog.
pParentWnd	Un pointer către fereastra părinte a acestei casete de dialog.

Această clasă are, printre altele, următoarele funcții membru:

DoModal	Afișează o casetă de dialog care permite utilizatorului să selecteze culoarea dorită
GetColor	Returnează o structură de tip COLORREF care conține valorile culorii selectate.
GetSavedCustomColors	Retrieves custom colors created by the user.
SetCurrentColor	Setează culoarea curentă selectată să fie cea specificată de această funcție.

Constructorul clasei **CFontDialog** este

CFontDialog(LPLOGFONT lplfInitial = NULL, DWORD dwFlags = CF_EFFECTS | CF_SCREENFONTS, CDC* pdcPrinter = NULL, CWnd* pParentWnd = NULL);

unde:

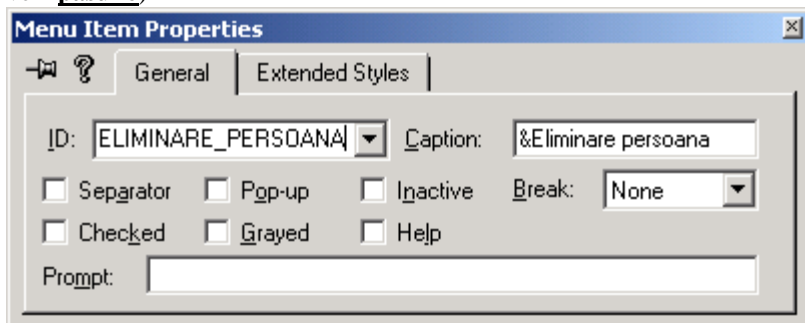
lplfInitial	Un pointer la o structură de date de tip LOGFONT ce vă permite setarea unor caracteristici ale fontului.
dwFlags	Specifică unul sau mai mulți indicatori de font aleși. Una sau mai multe valori predefinite se pot combina utilizând operatorul binar OR .
pdcPrinter	Un pointer către un dispozitiv de context pentru imprimantă.
pParentWnd	Un pointer către fereastra părinte a acestei casete de dialog.

Observați că această clasă are toți parametri implicați. Veți modifica acești parametri doar în cazul în care doriți o casetă de dialog pentru selectarea fontului diferită de cea implicită.

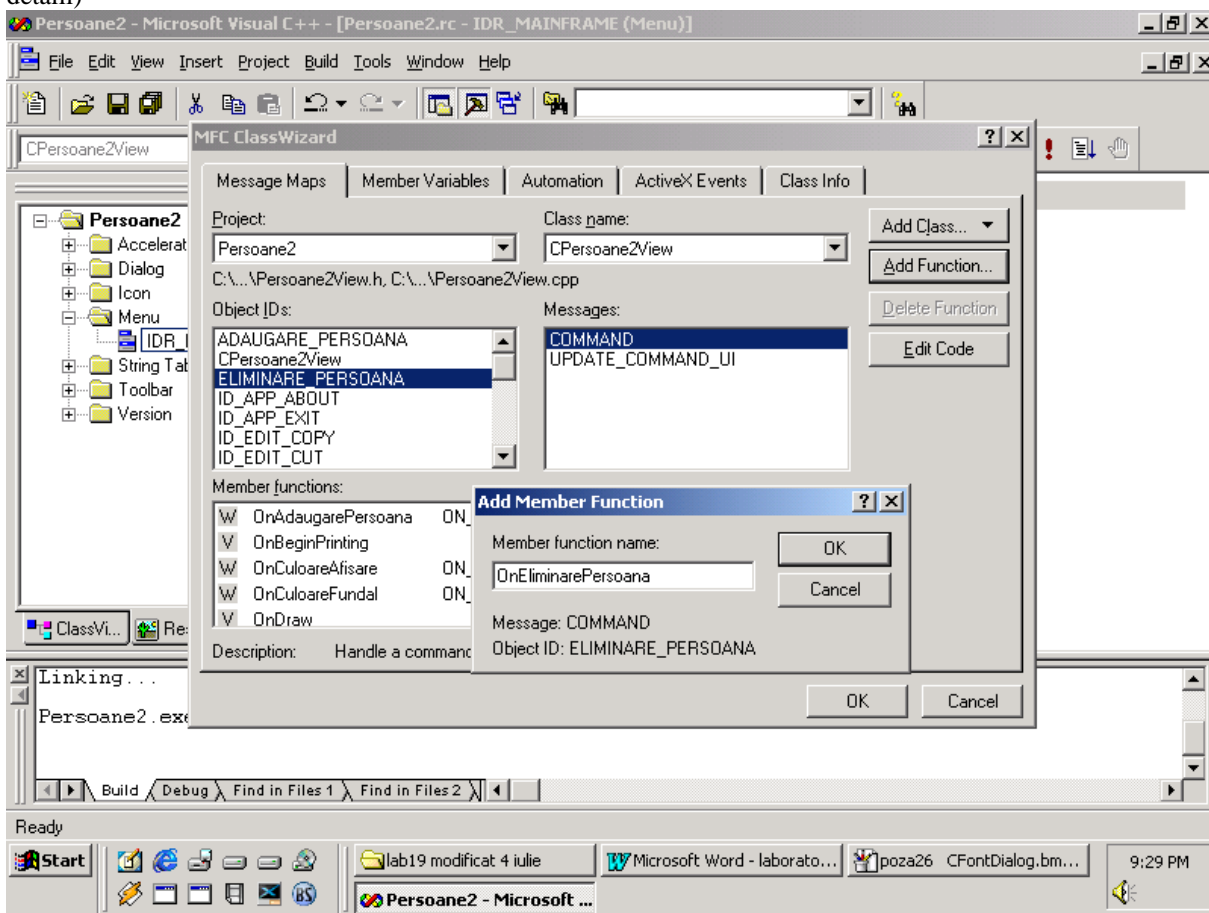
Această clasă are, printre altele, următoarele funcții membru:

DoModal	Afișează o casetă de dialog care permite utilizatorului să selecteze fontul dorit
GetCurrentFont	Transmite numele fontului curent selectat.
GetStyleName	Returnează numele stilului fontului selectat
GetSize	Returnează mărimea punctului al fontului selectat.
GetColor	Returnează culoarea fontului selectat
GetWeight	Returnează mărimea fontului selectat.
IsStrikeOut	Determină dacă fontul este afișat sau nu tăiat de o linie.
IsUnderline	Determină dacă fontul este afișat sau nu subliniat.
IsBold	Determină dacă fontul este sau nu îngroșat (bold).
IsItalic	Determină dacă fontul este sau nu înclinat (italic).

Pasul 9: Vom crea acum încă o opțiune de meniu pentru ștergerea unei persoane din vector (și implicit ștergerea acelei persoane din persoanele afișate în zona **Client** a ferestrei). Pentru aceasta creați încă o opțiune de meniu numită **Eliminare persoana** în meniul **Persoane**, sub opțiunea de meniu **Adaugare persoana**. (pentru detalii vezi **pasul 6**)



Asociați acum o funcție de tratare a acestei opțiuni de meniu. (vezi imaginea următoare sau **pasul 6** pentru detalii)



Inserați pe urma liniile de cod de mai jos:

- 1) void CPersoane2View::OnEliminarePersoana()
- 2) {
- 3) // TODO: Add your command handler code here
- 4) CPersoane2Doc* pDoc = GetDocument();

Laborator C++

```
5)    ASSERT_VALID(pDoc);
6)
7)    CDlgPers dlg;
8)    int valoare_returnata=dlg.DoModal();
9)    if(valoare_returnata==IDOK)
10)   {
11)       for(int i=0;i<pDoc->persoanele.size();i++)
12)           if(dlg.m_Nume==pDoc->persoanele[i].GetNume() )
13)               if(dlg.m_Prenume==pDoc->persoanele[i].GetPrenume() )
14)                   if(dlg.m_Varsta==pDoc->persoanele[i].GetVarsta() )
15)                       {
16)                           pDoc->persoanele.erase(&pDoc->persoanele[i]);
17)                           break;
18)                       }
19)       Invalidate();
20)   }
21) }
```

În liniile 4-5 vom crea un pointer către obiectul clasei document pentru a avea acces la membri acesteia. În liniile 7-8 (folosind același tip de casetă de dialog ca și pentru adăugarea datelor unei persoane) vom afișa o casetă de dialog pentru obținerea datelor despre persoana pe care dorim să o ștergem din listă. În liniile 9-20 căutăm în lista **persoanele** persoana pe care am introdus-o mai înainte și în cazul în care găsim o persoană care să corespundă cu aceste date o ștergem din listă. (Observație: se va șterge doar prima persoană din listă care corespunde cu datele introduse. Dacă ștergem linia a 17-a se vor șterge toate persoanele care corespund cu datele introduse).

Acum compilați și rulați programul.

Laborator 20

Scrieți un program care desenează dreptunghiuri în zona client a ferestrei cu ajutorul mouse-ului. Un colț al dreptunghiului va fi punctul în care se apasă butonul din stânga al mouse-ului iar colțul opus va fi punctul în care se eliberează butonul din stânga al mouse-ului. Se va putea modifica grosimea liniei cu care se desenează dreptunghiul printr-o casetă de dialog personalizată, culoarea liniei și culoarea de umplere a dreptunghiului.

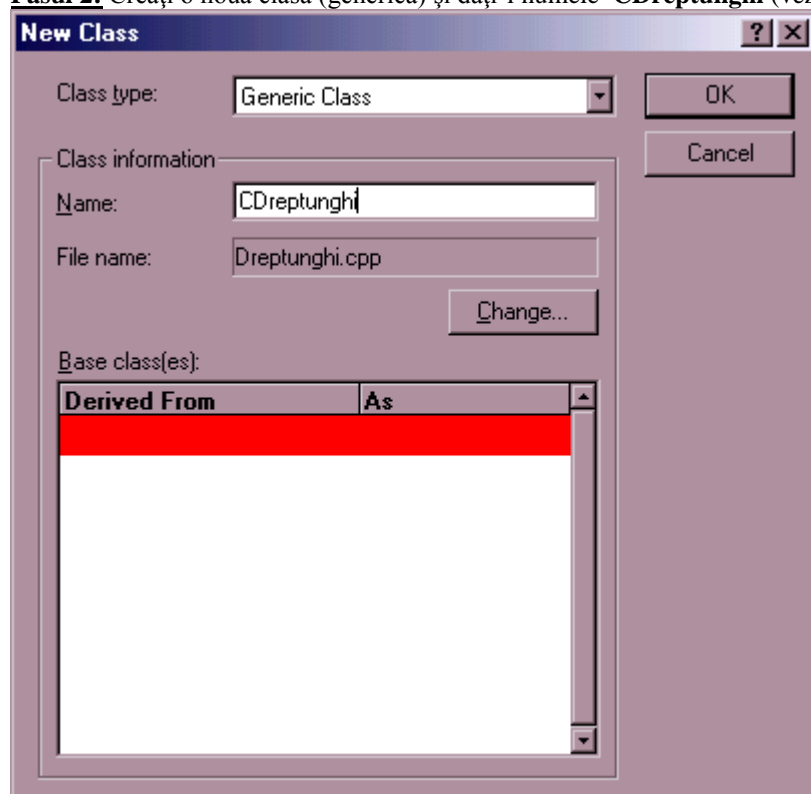
Cuvinte cheie: mesaje, WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_LBUTTONUP

În funcția OnMouseMove() se verifică dacă se mișcă mouse-ul cu butonul din stânga apăsat. (constanta MK_LBUTTON). CBrush, CPen, Invalidate(), InvalidateRect().

Rezolvare:

Pasul 1: Creați o nouă aplicație SDI și dați-i numele **DesenareDreptunghiuri**.

Pasul 2: Creați o nouă clasă (generică) și dați-i numele **CDreptunghi** (vezi imaginea de mai jos).



În constructorul acestei clase introduceți *variabilele membru*: **punct_initial** și **punct_final** ambele de tipul **CPoint**, *funcția membru*: **Rect()** (va returna un obiect de tip **CRect**) și *constructorul*: **CDreptunghi(CPoint initial, CPoint final)**; astfel declarația clase va arăta ca mai jos:

```

1) class CDreptunghi
2) {
3) public:
4)     CDreptunghi();
5)     CDreptunghi(CPoint initial, CPoint final);
6)     virtual ~CDreptunghi();
7)     CRect Rect();
8) public:
9)     CPoint punct_initial, punct_final;
10) };
    
```

Acum vom codul sursă pentru fiecare funcție membru. Pentru aceasta introduceți liniile de cod sursă de mai jos:

Laborator C++

```
1)  //////////////////////////////////////
2)  // Construction/Destruction
3)  //////////////////////////////////////
4)
5)  CDreptunghi::CDreptunghi()
6)  {
7)
8)  }
9)
10) CDreptunghi::~~CDreptunghi()
11) {
12)
13) }
14)
15) CDreptunghi::CDreptunghi(CPoint initial, CPoint final)
16) {
17)     punct_initial=initial;
18)     punct_final=final;
19) }
20)
21) CRect CDreptunghi::Rect()
22) {
23)     return CRect(punct_initial, punct_final);
24) }
```

Liniile de cod **1-13** sunt introduse implicit, deci nu trebuie să mai introducem decât liniile **14-24**. Liniile **15-19** reprezintă constructorul clasei pe care tocmai am creat-o. Acest constructor va accepta ca parametri două variabile de tipul **CPoint**. **punct_initial** și **punct_final** vor reprezenta cele două vârfuri opuse (nealăturate) ale unui dreptunghi desenat. Atunci când vom desena dreptunghiul, vom avea nevoie de un obiect de tipul **CRect**. Funcția membru **Rect()** (vezi liniile **21-24**) ne returnează tocmai un obiect de tipul **CRect** corespunzător punctelor **punct_initial** și **punct_final**.

Pasul 3: Acum vom declara un vector de obiecte de tipul **CDreptunghi**. Scopul este de a putea memora toate dreptunghiurile pe care le desenăm astfel că la o eventuală redimensionare sau doar la o redesenare a **zonei client** dreptunghiurile desenate să nu se șteargă. Pentru acesta efectuați un dublu-clic pe numele clasei document **CDesenareDreptunghiuriDoc** și introduceți liniile de cod **3,4** și **17** ca mai jos:

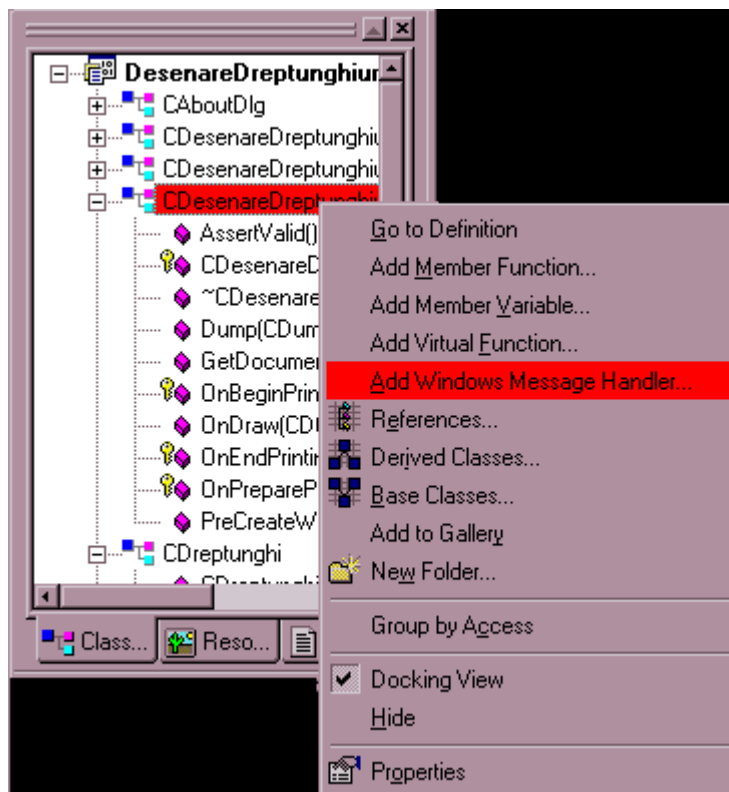
```
1)  #pragma once
2)  #endif // _MSC_VER > 1000
3)  #include<vector>
4)  #include"Dreptunghi.h"
5)
6)  class CDesenareDreptunghiDoc : public CDocument
7)  {
8)  protected: // create from serialization only
9)      CDesenareDreptunghiDoc();
10)     DECLARE_DYNCREATE(CDesenareDreptunghiDoc)
11)
12) // Attributes
13) public:
14)
15) // Operations
16) public:
17)     std::vector<CDreptunghi> toate_dreptunghiurile;
18) // Overrides
```

Laborator C++

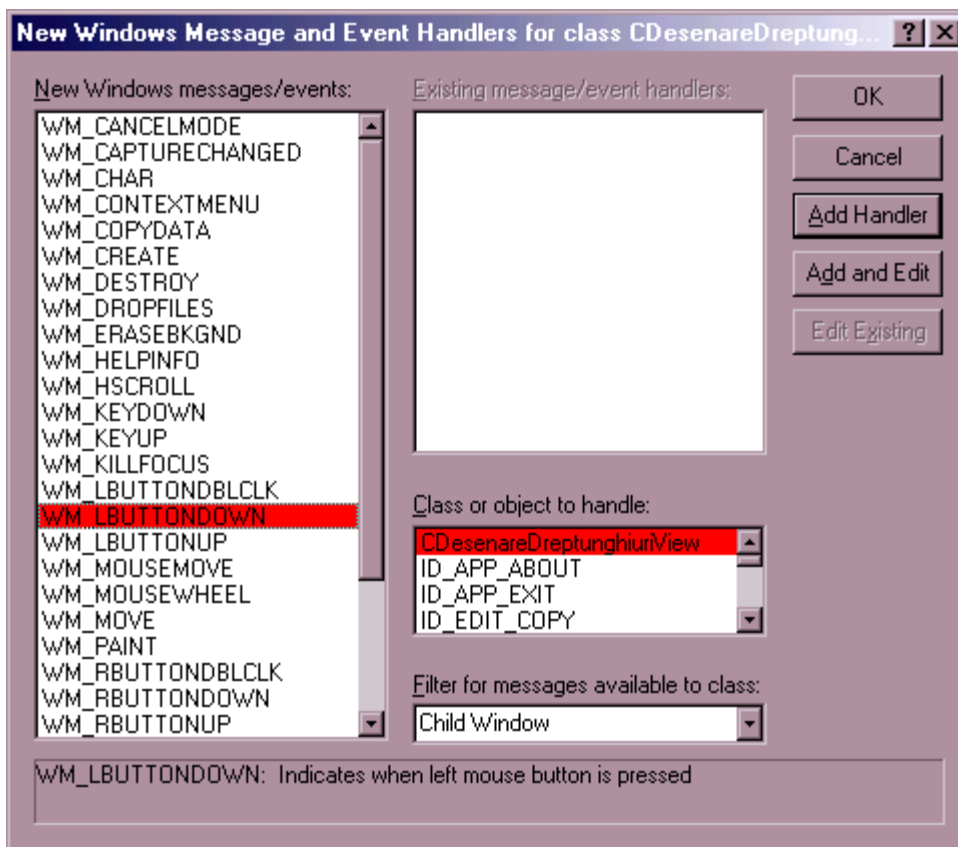
Astfel am declarat în linia **17** un vector: **toate_dreptunghiurile** de obiecte de tipul **CDreptunghi**.

Pasul 4: În acest pas vom introduce în clasa reprezentare codul sursă necesar desenării unui dreptunghi și memorării acestuia în vectorul **toate_dreptunghiurile**.

Pentru început vom avea nevoie de două variabile de tipul **CPoint** pe care le vom numi **punct_initial_apasat** și **punct_tinut_apasat**. Cu ajutorul acestora vom ști pentru fiecare dreptunghi desenat care este punctul unde s-a apăsăat pe butonul stâng al mouse-ului și respectiv unde a fost eliberat ultima dată butonul stâng. Pentru a ști unde s-a apăsăat butonul stâng al mouse-ului vom trata mesajul **WM_LBUTTONDOWN**. Pentru aceasta pe numele clasei reprezentare efectuați un clic-dreapta și selectați opțiunea **Add Windows Message Handler...** (vezi imaginea de mai jos).



Selectați apoi opțiunea **WM_LBUTTONDOWN** după care efectuați un clic pe butonul **Add and Edit**. (vezi imaginea următoare).



Acum introduceți liniile de cod sursă de mai jos:

```

1) void CDesenareDreptunghiurView::OnLButtonDown(UINT nFlags, CPoint point)
2) {
3)     // TODO: Add your message handler code here and/or call default
4)     punct_initial_apasat=point;
5)     punct_tinut_apasat=point;
6)     CView::OnLButtonDown(nFlags, point);
7) }
```

Astfel în linia a patra introducem în **punct_initial_apasat** poziția unde s-a apăsut butonul stâng al mouse-ului. Analog tratați mesajele **WM_MOUSEMOVE** și **WM_LBUTTONUP**.

Funcția **OnMouseMove()** va desena dreptunghiul ce are vârfurile opuse: punctul unde s-a apăsut butonul stâng al mouse-ului și punctul curent obținut din mișcarea mouse-ului (butonul stâng rămânând în continuare apăsut). Pentru a determina dacă butonul stâng este apăsut trebuie verificat parametrul **nFlags**. Acest parametru poate fi orice combinație a următoarelor valori:

MK_CONTROL	setat dacă este apăsată tasta Ctrl .
MK_LBUTTON	setat dacă este apăsut butonul stâng al mouse-ului
MK_MBUTTON	setat dacă este apăsut butonul din mijloc al mouse-ului
MK_RBUTTON	setat dacă este apăsut butonul drept al mouse-ului
MK_SHIFT	setat dacă este apăsată tasta SHIFT

Scrieți liniile de cod de mai jos pentru funcțiile de tratare a mesajelor **WM_MOUSEMOVE** și **WM_LBUTTONUP**.

```

1) void CDesenareDreptunghiurView::OnMouseMove(UINT nFlags, CPoint point)
2) {
3)     // TODO: Add your message handler code here and/or call default
4)     CDC *pdc;
```

Laborator C++

```
5)     pdc=GetDC();
6)     if(nFlags==MK_LBUTTON)
7)     {
8)         //sterge ultimul dreptunghi desenat
9)         InvalidateRect(CRect(punct_initial_apasat,punct_tinut_apasat));
10)
11)        pdc->Rectangle(CRect(punct_initial_apasat, point));
12)        punct_tinut_apasat=point;
13)    }
14)    CView::OnMouseMove(nFlags, point);
15) }
16)
17) void CDesenareDreptunghiView::OnLButtonUp(UINT nFlags, CPoint point)
18) {
19)     // TODO: Add your message handler code here and/or call default
20)     CDesenareDreptunghiDoc* pDoc = GetDocument();
21)     ASSERT_VALID(pDoc);
22)     pDoc->toate_dreptunghiurile.push_back(CDreptunghi(punct_initial_apasat,point));
23)     Invalidate();
24)     CView::OnLButtonUp(nFlags, point);
24) }
```

În linia 9 funcția **InvalidateRect()** redesenează **zona client** dreptunghiulară determinată de parametrii **punct_initial_apasat** și **punct_tinut_apasat** (de aceea, în funcția de tratare a mesajului **WM_LBUTTONDOWN** l-am inițializat și pe **punct_tinut_apasat**). În linia 11 desenăm dreptunghiul și actualizăm în linia 12 poziția punctului **punct_tinut_apasat**. În linia 22, atunci când am eliberat butonul stâng al mouse-ului, introduceti în vectorul **toate_dreptunghiurile** ultimul dreptunghi desenat după care, cu ajutorul funcției **Invalidate()** redesenăm toată **zona client** a ferestrei de lucru.

Pașul 5: Vom desena toate dreptunghiurile din vectorul **toate_dreptunghiurile** folosind grosimea liniei și culorile (pentru umplere respectiv pentru culoarea liniei) cu valori implicite. Pentru aceasta declarați încă două variabile membru **culoare_penita**, **culoare_brush** (de tip **COLORREF**) și **grosime_linie** (de tipul **int**) după care în constructorul clasei reprezentare le vom inițializa:

```
1) CDesenareDreptunghiView::CDesenareDreptunghiView()
2) {
3)     // TODO: add construction code here
4)     culoare_penita=RGB(0,0,0);
5)     culoare_brush=RGB(0, 0, 255);
6)     grosime_linie=3;
7) }
```

Pentru desenarea tuturor dreptunghiurilor de vector introduceți în funcția **OnDraw()** următoarele linii de cod:

```
1) void CDesenareDreptunghiView::OnDraw(CDC* pDC)
2) {
3)     CDesenareDreptunghiDoc* pDoc = GetDocument();
4)     ASSERT_VALID(pDoc);
5)     // TODO: add draw code for native data here
6)     CBrush brush(culoare_brush);
7)     CBrush* pOldBrush = pDC->SelectObject(&brush);
8)
9)     CPen penita;
10)    penita.CreatePen(PS_SOLID, grosime_linie, culoare_penita);
11)    CPen* pOldPen = pDC->SelectObject(&penita);
```

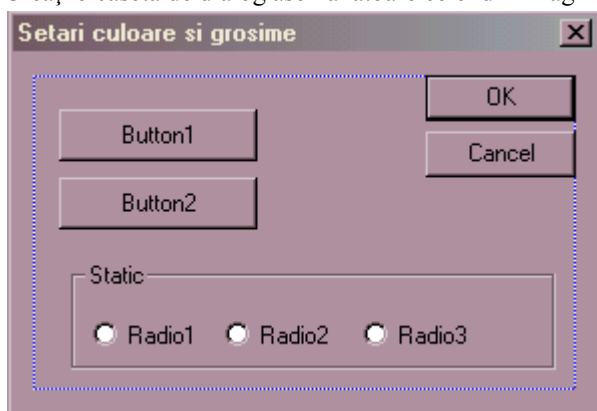

Laborator C++

```
12)
13)     for(int i=0; i<pDoc->toate_dreptunghiurile.size();i++)
14)         pDC->Rectangle(pDoc->toate_dreptunghiurile[i].Rect());
15)
16)     pDC->SelectObject(pOldBrush);
17)     pDC->SelectObject(pOldPen);
18) }
```

În liniile **6-11** facem ca penița și pensula să aibă folosească culorile din variabilele membru **culoare_brush**, **culoare_penita** și **grosime_linie**.

Pasul 6: În acest pas vom crea o opțiune de meniu pentru schimbarea culorilor și a grosimei liniei de desenare a dreptunghiurilor. Pentru aceasta vom crea o casetă de dialog căreia îi vom asocia apoi o clasă.

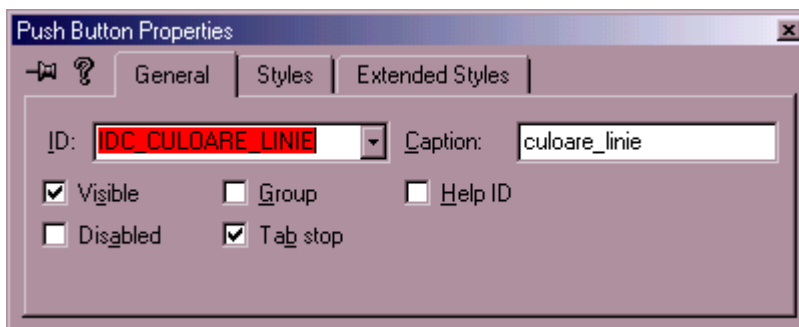
Creați o casetă de dialog asemănătoare celei din imaginea de mai jos:



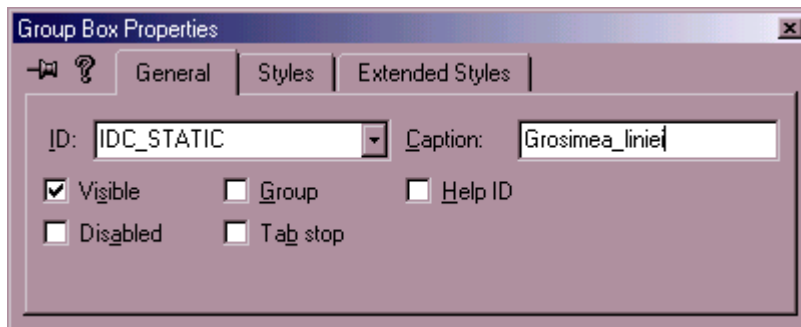
Schimbați numele casetei de dialog și **ID**-ul acestuia astfel: **Setari culoare si grosime** și respectiv

IDD_SETARI.

Apăsăți un clic-dreapta pe primul buton și selectați opțiunea **Properties**. Schimbați numele **ID** și **Caption** ca în imaginea de mai jos.

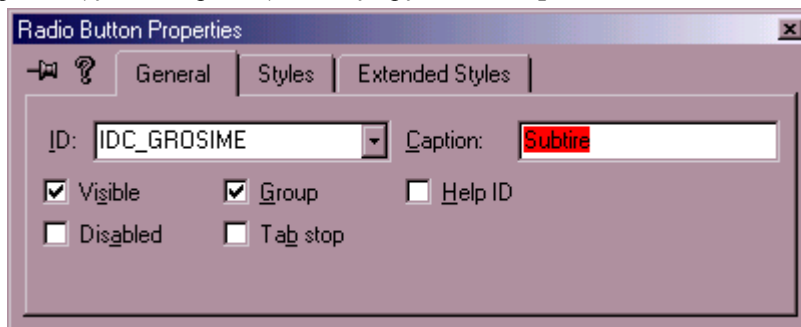


Operați analog în cazul celui de-al doilea buton cu denumirile: **IDC_CULOARE_UMPLERE** și **culoare_umplere**. În cazul **etichetei de grup** setați eticheta **Caption** cu **Grosimea liniei**. (vezi imaginea de mai jos).

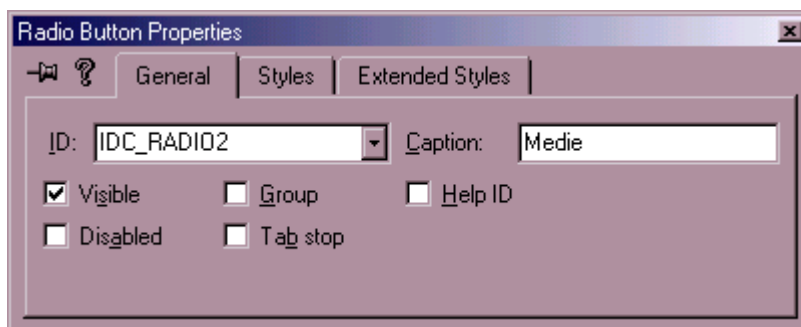


În cazul celor trei **butoane de opțiune** setați numele identificatorului și al celui de la **Caption** ca în imaginile de mai jos:

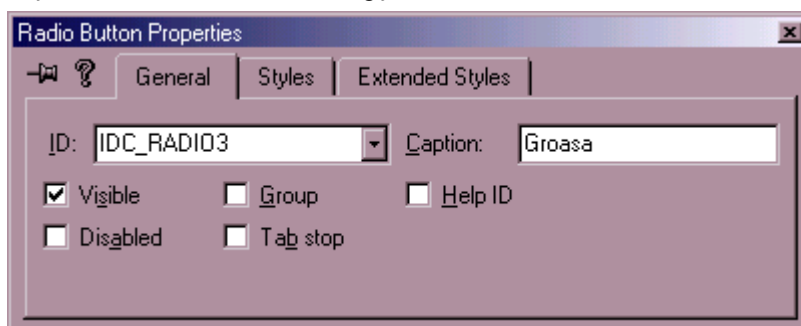
La primul (și doar la primul) selectați opțiunea **Group**.



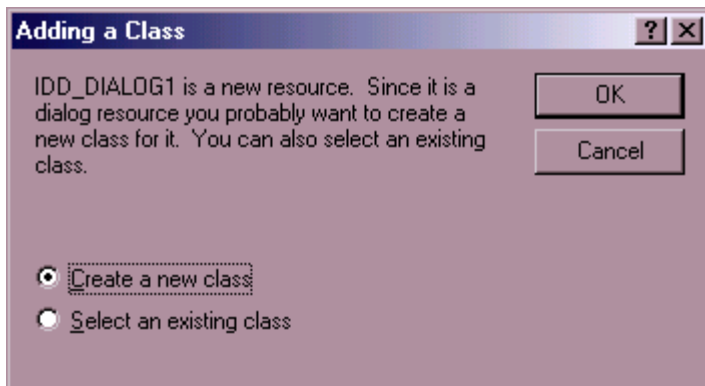
În cazul celui de-al doilea buton asigurați-vă că nu este selectat opțiunea **Group**.



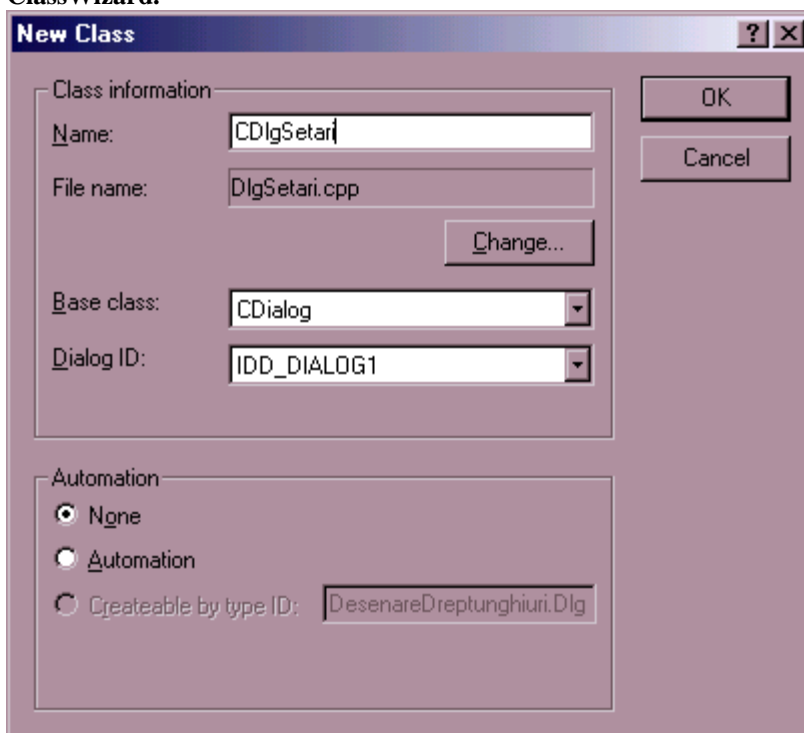
La fel și în cazul ultimului buton de opțiune.



Acum, pentru a crea o nouă clasă (pentru caseta de dialog pe care ați creat-o) apăsați un clic oriunde pe suprafața casetei de dialog și apăsați **Ctrl+W**. Se va deschide următoarea casetă de dialog:



Apăsați pe butonul **OK** după care se va deschide caseta de dialog **New Class**. Introduceți numele **CDlgSetari**. Apăsați apoi pe **OK** pentru a închide această casetă de dialog și apoi încă o dată pe **OK** pentru a închide **ClassWizard**.



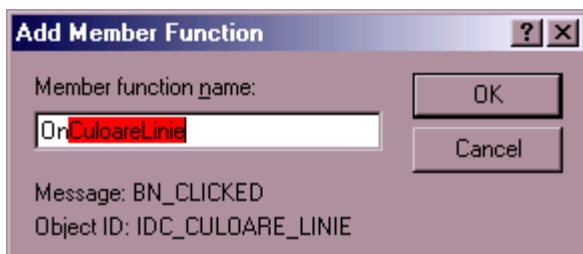
Aici introduceți două noi variabile membre ale acestei noi clase: **clr_linie**, **clr_umplere** (ambele de tipul **COLORREF**) ca mai jos:

- 1) class CDlgSetari : public CDialog
- 2) {
- 3) // Construction
- 4) public:
- 5) COLORREF clr_linie, clr_umplere;
- 6) CDlgSetari(CWnd* pParent = NULL); // standard constructor
- 7)
- 8)
- 9) // Dialog Data

Vom folosi aceste variabile pentru a determina care este noua culoare aleasă/noua grosime aleasă. Casetă de dialog va fi inițializată implicit la valorile anterior setate.

Apăsați un dublu-clic pe butonul **culoare_linie** și se va deschide caseta de dialog **Add Member Function**.

Laborator C++



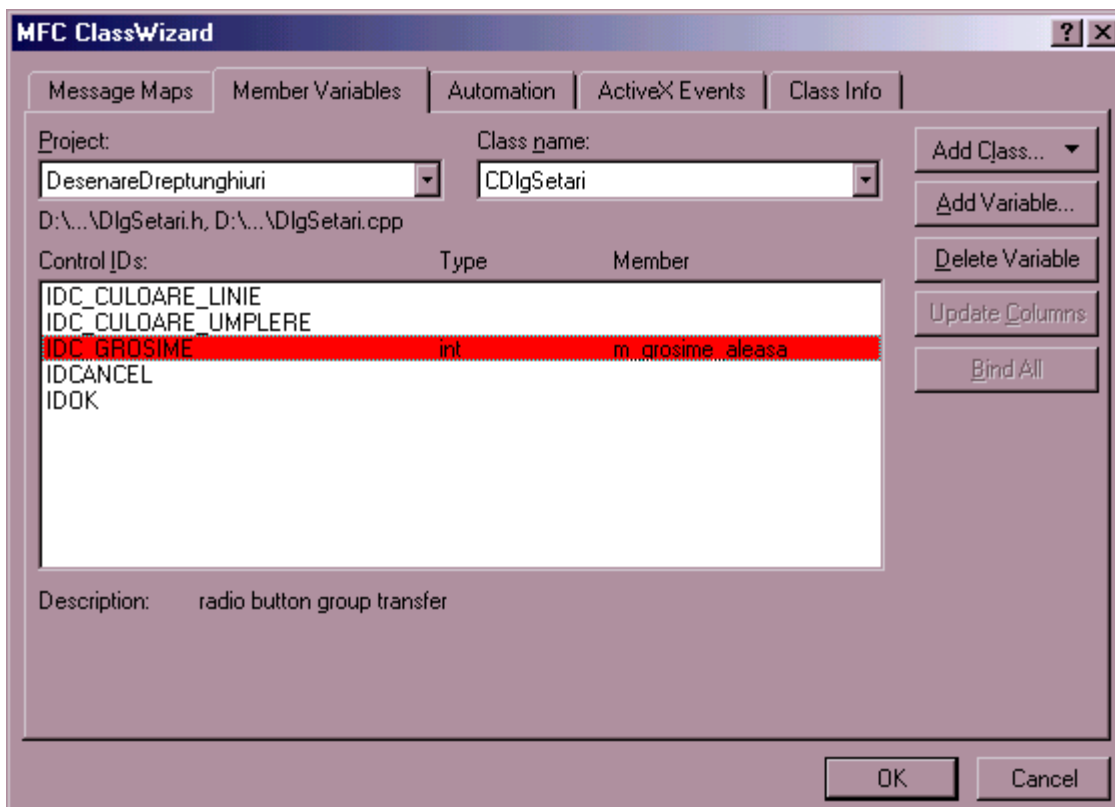
Apăsați pe butonul **OK** și inserați următoarele linii de cod:

```
1) void CDlgSetari::OnCuloareLinie()
2) {
3)     // TODO: Add your control notification handler code here
4)     CColorDialog dlg;
5)     int ret=dlg.DoModal();
6)     if(ret==IDOK)
7)     {
8)         clr_linie=dlg.GetColor();
9)     }
10) }
```

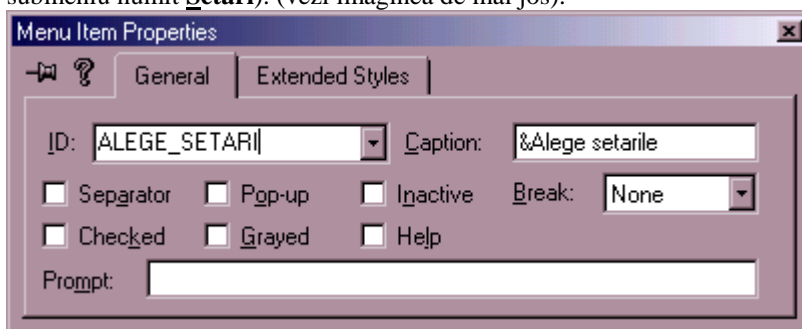
Această funcție setează valorarea lui **clr_linie** la noua valoare aleasă (doar în cazul apăsării butonului **OK** a casetei de dialog **dlg**). În mod analog se efectuează în cazul celui de-al doilea buton. Acceptați numele implicit **OnCuloareUmplere** după care apăsați pe butonul **OK** și introduceți liniile de cod următoare:

```
1) void CDlgSetari::OnCuloareUmplere()
2) {
3)     // TODO: Add your control notification handler code here
4)     CColorDialog dlg;
5)     int ret=dlg.DoModal();
6)     if(ret==IDOK)
7)     {
8)         clr_umplere=dlg.GetColor();
9)     }
10) }
```

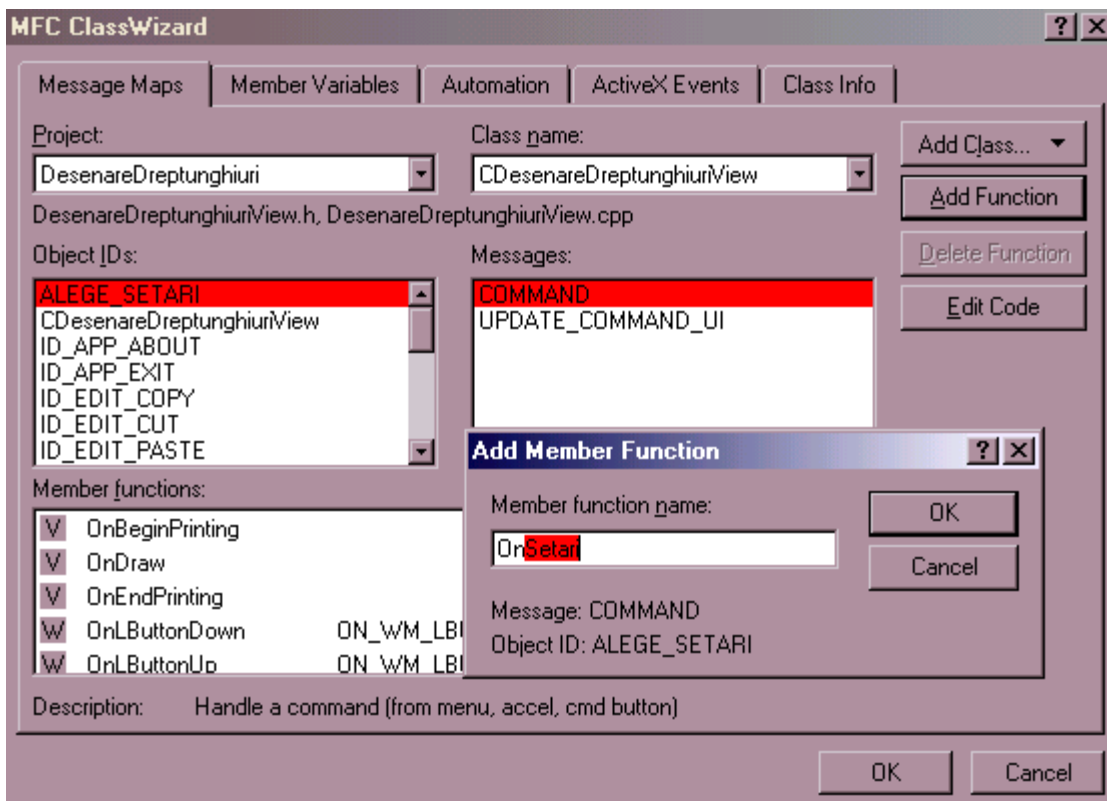
Pentru grosimea liniei va trebui să mapăm o variabilă peste primul buton de opțiune. Pentru aceasta deschideți **Class Wizard** (apasați pentru aceasta **Ctrl+W**). Noi am numit această variabilă **m_grosime_aleasa** (de tipul **int**). Acum **Class Wizard** trebuie să arate ca în imaginea de mai jos:



Pasul 7: Până acum am determinat ce anume să facă caseta de dialog. Mai trebuie doar să apelăm, printr-o opțiune de meniu această fereastră și să o determinăm să interacționeze cu clasa reprezentare. Inserați o nouă opțiune de meniu. Noi am denumit această nouă opțiune de meniu **Alege setarile** (am creat inițial un nou submeniu numit **Setari**). (vezi imaginea de mai jos).



Pentru a scrie codul sursă necesar acestei noi opțiuni de meniu deschideți **Class Wizard**, selectați ca numele clasei să fie clasa reprezentare, identificatorul obiectului să fie **ALEGE_SETARI**, selectați mesajul **COMMAND** și apăsați butonul **Add Function**. Acceptați numele implicit de funcție după care apăsați pe butonul **OK** pentru a închide această casetă de dialog și apoi apăsați pe butonul **Edit Code**.



Înserați acolo următoarele linii de cod:

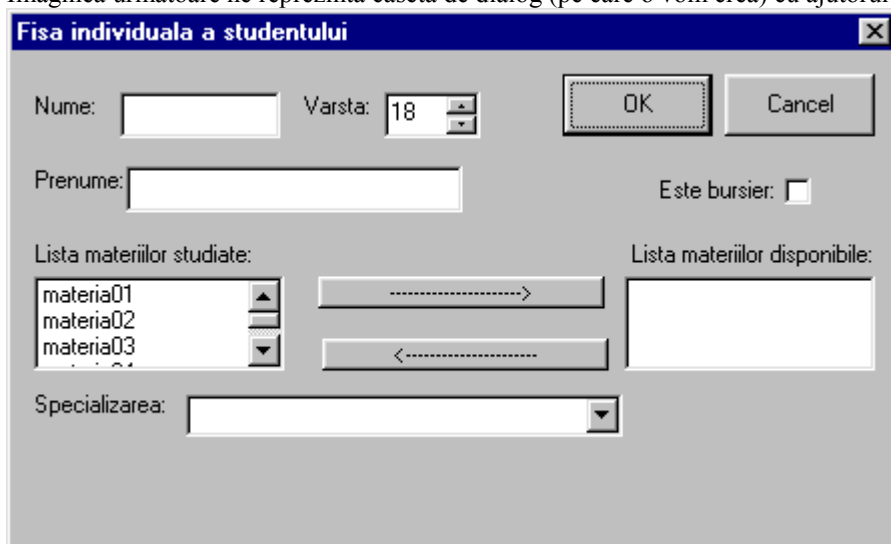
```

1) #include"DlgSetari.h"
2) void CDesenareDreptunghiuriView::OnSetari()
3) {
4)     // TODO: Add your command handler code here
5)     CDlgSetari dlg;
6)     //dam valori implicite variabilelor
7)     dlg.clr_linie=culoare_penita;
8)     dlg.clr_umplere=culoare_brush;
9)     dlg.m_grosime_aleasa=grosime_linie;
10)
11)    int ret=dlg.DoModal();
12)    //daca vom apasa butonul OK
13)    if(ret==IDOK)
14)    {
15)        culoare_penita=dlg.clr_linie;
16)        culoare_brush=dlg.clr_umplere;
17)        grosime_linie=dlg.m_grosime_aleasa;
18)    }
19)    Invalidate();
20) }
```

În liniile 7-9 dăm valori implicite variabilelor casetei de dialog după care, în linia 11 afișăm caseta de dialog. În cazul în care se apasă pe butonul **OK** se vor actualiza variabilele clasei reprezentare cu valorile noi alese după care reafișăm toată zona client pentru a se vedea eventualele schimbări ale setărilor culorilor și/sau grosimii liniei.

Laborator 21

Scrieți un program MFC care să permită introducerea unor date despre studenți: *nume*, *prenume*, *vârsta*, este sau nu *bursier*, *materiile studiate* precum și *specializarea* la care studiază. *Numele* și *prenumele* se vor introduce printr-o caseta de editare (**EditBox**), *vârsta*, care poate lua valori între 18 și 100, se va modifica cu ajutorul unui **Spinner**. Cu ajutorul unei casete de validare (**CheckBox**) se va determina dacă studentul este sau nu bursier. De asemenea se va putea selecta dintr-o listă (**ListBox**) cu toate materiile disponibile acele materii pe care studentul le-a studiat până în prezent iar cu ajutorul unui buton (**Button**) le vom introduce pe acestea din urmă într-o altă listă. În final vom putea alege, dintr-o casetă combinată (**ComboBox**), specializarea studentului. Imaginea următoare ne reprezintă caseta de dialog (pe care o vom crea) cu ajutorul căreia vom introduce datele:



Vă propunem în final extinderea acestui proiect cu următoarele:

- realizați serializarea;
- realizați o opțiune de meniu pentru stergerea unui student
- realizați o opțiune de meniu pentru stergerea unui grup de studenți (aflați într-un interval)
- realizați o opțiune de meniu pentru aranjarea alfabetică a studenților
- realizați o opțiune de meniu pentru schimbarea fontului și/sau culorii folosite pentru de afișare
- realizați o bară de derulare orizontală.
- realizați o opțiune de meniu pentru introducerea de (popularea cu) elemente în cele prima lista (de tip **ListBox**) și în lista combinată (cea de tip **ComboBox**).

Rezolvare:

Pasul 1: Creați o nouă aplicație MFC de tip **SDI** pe care să o denumiți **DateStudenti** și a cărei clasă de bază să fie **CScrollView**. La acest pas vom crea caseta de dialog (pe care o vom utiliza pentru introducerea datelor) și vom crea pentru aceasta o nouă clasă.

Mai întâi inserați o nouă casetă de dialog. Efectuați un clic-dreapta pe suprafața acesteia și selectați opțiunea **Properties**. Se va deschide caseta de dialog **Dialog Properties**. Introduceți **IDD_STUDENT** în dreapta etichetei **ID** și **Fisa individuala a studentului** în dreapta etichetei **Caption** (vezi imaginea de mai jos).



Creați acum o nouă clasă care să corespundă acestei casete de dialog. Pentru aceasta efectuați un clic pe suprafața acesteia și apăsați **Ctrl+W** pentru a deschide **ClassWizard**. Se va deschide caseta de dialog **Adding a Class** care are selectat implicit butonul radio **Create a new class**. Efectuați acum un clic pe butonul **OK** pentru a crea o nouă clasă. Se va deschide acum automat caseta de dialog **New Class**. Introduceți un nume pentru această nouă clasă, de exemplu **CdlgStudent** după care apăsați de două ori pe butonul **OK** pentru a închide această casetă de dialog și caseta de dialog **MFC ClassWizard**.

Acum redimensionați caseta de dialog pentru a fi suficient de mare. Vom adăuga elementele necesare pentru ca această casetă de dialog să arate ca în imaginea ce se află imediat după enunțul acestui laborator (de aceea, ghidați-vă după acea imagine pentru poziționarea lor). Adăugați următoarele controale:

- șase *etichete statice* (**static text**) care să indice **Nume:**, **Prenume:**, **Varsta:**, **Lista materiilor disponibile:**, **Lista materiilor studiate:** și respectiv **Specializarea:**.

- trei *casete de editare* (**edit box**) câte una în dreapta etichetelor **Nume:**, **Prenume:**, respectiv **Varsta:**, a căror **ID** să fie respectiv **IDC_NUME**, **IDC_PRENUME** și **IDC_INT_VARSTA**

- un *control de modificare incrementală* (**Spin**) în dreapta casetei de editare care se află în dreapta etichetei **Varsta:**. Deschideți caseta de dialog **Spin Properties** (pentru aceasta efectuați un clic-dreapta pe suprafața acestuia și selectați opțiunea **Properties**). Introduceți **IDC_VARSTA** în dreapta etichete **ID**. Selectați opțiunile **Auto buddy** și **Set buddy integer**, apoi opțiunea **Right** din lista **Alignment**.

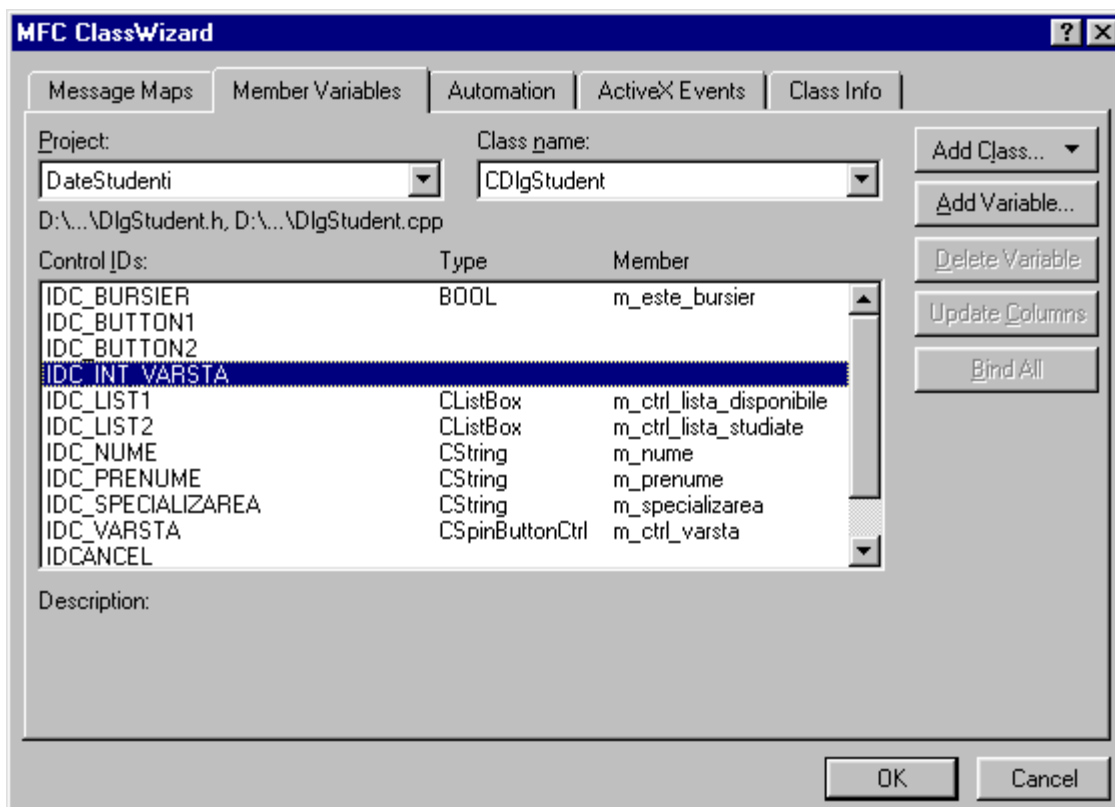
- o *casetă de validare* (**Check Box**) a cărei **ID** să fie **IDC_BURSIER**, iar în dreapta etichetei **Caption** introduceți **Este bursier:**. Selectați apoi opțiunea **Right aligned text** pentru ca textul să apară scris în stânga dreptunghiului pentru validare.

- două *casete cu listă* (**List Box**), câte una sub fiecare din etichetele **Lista materiilor disponibile:** și **Lista materiilor studiate** a căror identificatori să fie **IDC_LIST1** și **IDC_LIST2**.

- două *butoane* (**Button**) cu ajutorul cărora vom putea modifica cele două liste “traversând” câte un element dintr-o listă în cealaltă. Noi le-am “etichetat” cu ----- > respectiv cu < ----- .

- ultimul element adăugat va fi o *casetă combinată* (**Combo Box**) cu ajutorul căreia vom selecta specializarea studentului. Schimbați “**ID**”-ul său cu **IDC_SPECIALIZAREA** și selectați opțiunea **Drop List** al listei de opțiuni **Type** din pagina **Styles**. Pentru a “popula” această casetă combinată procedați astfel: selectați pagina **Data** și introduceți câte un element după care, pentru a scrie pe rând nou, apăsați **Ctrl+W**. Introduceți astfel elementele următoare: **Matematica-Informatica**, **Matematica-Fizica**, **Fizica-Chimie**, **Informatica-colegiu**, **Fizica**, **Chimie**, **Chimie-Biologie** și **Biologie**.

Pentru a putea lucra cu această casetă de dialog vom mapa câteva controale și variabile așa cum se arată în imaginea de mai jos:



Mai adăugați două variabile membru, **selecția1** și **selecția2** de tip **int** acestei clase (**CDlgStudent**) . Acestea vor reprezenta indexul elementului curent selectat în fiecare din cele două casete cu listă.

Pasul 2: La acest pas vom popula prima lista cu materiile de studiu, vom stabili vârsta implicită care să fie selectată la momentul apariției caseta de dialog. Pentru a putea inițializa această casetă de dialog vom trata mesajul Windows numit **OnInitDialog**. Pentru aceasta efectuați un clic dreapta pe numele clasei asociate casetei de dialog **CDlgStudent** și selectați opțiunea **Add Windows Message Handler...** . Se va deschide astfel caseta de dialog **New Windows Message and Event Handlers for class CDlgStudent**. Selectați **WN_INITDIALOG** după care apăsați pe butonul **Add and Edit**.

Aici introduceți liniile de cod de mai jos pentru a inițializa caseta de dialog.

```

1)  BOOL CDlgStudent::OnInitDialog()
2)  {
3)      CDialog::OnInitDialog();
4)      // TODO: Add extra initialization here
5)      m_ctrl_varsta.SetRange(18,100);
6)      m_ctrl_varsta.SetPos(18);
7)      m_ctrl_lista_disponibile.ResetContent();
8)      m_ctrl_lista_studiate.ResetContent();
9)      m_ctrl_lista_disponibile.AddString("materia01");
10)     m_ctrl_lista_disponibile.AddString("materia02");
11)     m_ctrl_lista_disponibile.AddString("materia03");
12)     m_ctrl_lista_disponibile.AddString("materia04");
13)     m_ctrl_lista_disponibile.AddString("materia05");
14)     m_ctrl_lista_disponibile.AddString("materia06");
15)     m_ctrl_lista_disponibile.AddString("materia07");
16)     m_ctrl_lista_disponibile.AddString("materia08");
17)     m_ctrl_lista_disponibile.AddString("materia09");
18)     m_ctrl_lista_disponibile.AddString("materia10");

```

Laborator C++

```

19)    m_ctrl_lista_disponibile.AddString("material1");
20)    m_ctrl_lista_disponibile.AddString("material2");
21)    m_ctrl_lista_disponibile.AddString("material3");
22)    m_ctrl_lista_disponibile.AddString("material4");
23)    m_ctrl_lista_disponibile.AddString("material5");
24)    m_ctrl_lista_disponibile.AddString("material6");
25)    selectia1=-1;
26)    selectia2=-1;
27)    return TRUE; // return TRUE unless you set the focus to a control
28)    // EXCEPTION: OCX Property Pages should return FALSE
29)    }

```

În linia **5)** stabilim limitele pentru controlul de modificare incrementală **m_CtrlVarsta** iar în linia **6)** stabilim poziția sa inițială (implicită).

Un control de modificare incrementală are următoarele funcții membre:

SetAccel	Modifică pasul cu care să se modifice valoarea controlului.
GetAccel	Returnează valoarea pasului cu care se modifică valoarea controlului.
<i>SetBase</i>	Modifică baza în care să opereze controlul.
GetBase	Returnează baza în care operează controlul.
SetPos	Modifică poziția curentă a controlului.
GetPos	Returnează poziția curentă a controlului.
SetRange	Modifică limitele superioară și inferioară (intervalul) pentru control.
GetRange	Returnează limitele superioară și inferioară (intervalul) ale controlului.
SetRange32	Modifică limitele superioară și inferioară (intervalul) folosind întregi pe 32 de biți pentru control.
GetRange3	Returnează limitele superioară și inferioară (intervalul) folosind întregi pe 32 de biți ale controlului.

În liniile **7), 8)** ștergem orice toate elementele din cele două liste cu materii., iar în liniile **9) - 24)** populăm prima listă cu 16 elemente. O parte din funcțiile membre ale unui control de tip listă combinată este scrisă mai jos:

GetCount	Returnează numărul de șiruri din caseta cu listă.
GetTopIndex	Returnează indexul primului șir afișat în caseta cu listă.
SetTopIndex	Setează indexul (care pornește de la 0) de unde începe afișarea elementelor din caseta cu listă.
GetItemData	Returnează valoarea pe 32 de biți asociat elementului casetei cu listă.
GetItemDataPtr	Returnează un pointer la elementul casetei cu listă.
SetItemData	Modifică valoarea pe 32 de biți asociată elementului casetei cu listă.
GetCurSel	Returnează indexul elementului curent care este selectat
SetCurSel	Selectează un șir dintr-o casetă cu listă.
SetSel	Selectează/deselectează un element dintr-o casetă cu listă cu selecție multiplă.
GetSelCount	Returnează numărul șirurilor care sunt selectate la un moment dat într-o casetă cu listă cu selecție multiplă.
GetSelItems	Returnează indicii șirurilor care sunt selectate la un moment dat.
SetItemRange	Selectează/deselectează un interval de șiruri într-o casetă cu listă cu selecție multiplă.
AddString	Adaugă un șir în caseta cu listă.
DeleteString	Șterge un șir din caseta cu listă.

InsertString	Inserează un șir într-o loc anume în caseta cu listă.
ResetContent	Șterge toate elementele din caseta cu listă.
Dir	Adaugă numele fișierelor din directorul curent.
FindString	Caută un șir în caseta cu listă.
SelectString	Caută și selectează un șir într-o casetă cu listă cu o singură selecție.

În liniile **25), 26)** setăm valorile implicite pentru variabilele **selectia1** și **selectia2** la **-1**. Aceste valori reprezintă indexul elementului care este selectat curent în fiecare din cele două liste. Inițial nu vom avea selectat nici un element așa că vom stabili valorile acestora la **-1**.

Pașul 3: La acest pas vom realiza funcționalitatea celor două butoane: **----- >** respectiv **< -----**. Mai întâi mai adăugați o variabilă membru acestei clase (**CDlgStudent**), un vector de lungime 16 (acest număr reprezintă număr maxim de materii pe care am considerat că vom introduce în oricare din cele două liste!) numită **materiile_alese** de tipul **CString** (**CString materiile_alese[16]**). În acest vector avea întotdeauna numele materiilor alese (numele elementelor din caseta a doua cu listă).

Deschideți caseta de dialog **MFC ClassWizard**. Selectați clasa **CDlgStudent**, identificatorul obiectului să fie **IDC_LIST1** și mesajul **LBN_SELCHANGE**. Acest mesaj este transmis ori de câte ori se schimbă elementul selectat (al casetei cu listă). Apăsați apoi pe butonul **Add Function...**, acceptați numele implicit **OnSelchangeList1** apăsând pe butonul **OK** după care apăsați pe butonul **Edit Code** pentru a introduce liniile de cod de mai jos. Cu ajutorul acestei funcții vom introduce în variabila **selectia1** indexul elementului din prima listă care este selectat la un moment dat (în mod curent).

```

1) void CDlgStudent::OnSelchangeList1()
2) {
3)     // TODO: Add your control notification handler code here
4)     //GetSel returneaza 0 daca nu este selectat, LB_ERR daca apare eroare, un nr pozitiv daca e selectat
5)     for(int i=0;i<m_ctrl_lista_disponibile.GetCount();i++)
6)         if(m_ctrl_lista_disponibile.GetSel(i)!=0 && m_ctrl_lista_disponibile.GetSel(i)!=LB_ERR)
7)             selectia1=i;
8)     Invalidate();
9) }
```

Acum apăsați un dublu-clic pe butonul **----- >** pentru a crea funcția corespunzătoare acestuia (și care ca răspunde mesajului **BN_CLICKED**). Acceptați numele implicit **OnButton1** (sau puteți să-l schimbați cu altul mai convenabil dacă doriți) și introduceți liniile de cod de mai jos. Această funcție va face ca, în cazul în care este selectat un element din prima listă (adică **selectia1** reprezintă un număr de indice valid pentru numărul de ordine al vreunui element din prima listă), acesta să fie inserat în lista a doua.

```

1) void CDlgStudent::OnButton1()
2) {
3)     // TODO: Add your control notification handler code here
4)     CString str;
5)     if(selectia1>=0)
6)     {
7)         int lungime_cuvant;
8)         lungime_cuvant = m_ctrl_lista_disponibile.GetTextLen(selectia1);
9)         m_ctrl_lista_disponibile.GetText(selectia1,str.GetBuffer(lungime_cuvant));
10)        m_ctrl_lista_studiate.AddString(str);
11)        m_ctrl_lista_disponibile.DeleteString(selectia1);
12)        selectia1=-1;
13)        str.ReleaseBuffer();
14)    }
15)    //salvam in materiile_alese ce avem in lista a doua
16)    int n=m_ctrl_lista_studiate.GetCount();
17)    for(int i=0;i<n;i++)
18)    {
```

Laborator C++

```
19)         int lungime_cuvant=m_ctrl_lista_studiate.GetTextLen(i);
20)         m_ctrl_lista_studiate.GetText(i, materiile_alese[i].GetBuffer(lungime_cuvant));
21)     }
22)     for(i=n;i<16;i++)
23)         materiile_alese[i]="";
24)     Invalidate();
25) }
```

În liniile **7-11** inserăm elementul selectat din prima listă în lista a doua după care îl ștergem din prima listă. În linia **12** reinițializăm variabila **selectia1** cu valoarea -1. În liniile **16-21** salvăm aceste elemente în vectorul **materiile_alese**. Liniile **22-23** ne asigură că restul elementelor din vector sunt nule.

Absolut analog vom proceda pentru celălalt buton și pentru cea de-a doua listă. Deosebiri sunt mici în ceea ce privește codul sursă:

```
1) void CDlgStudent::OnSelchangeList2()
2) {
3)     // TODO: Add your control notification handler code here
4)     for(int i=0;i<m_ctrl_lista_studiate.GetCount();i++)
5)         if(m_ctrl_lista_studiate.GetSel(i)!=0 && m_ctrl_lista_studiate.GetSel(i)!=LB_ERR)
6)             selectia2=i;
7)     Invalidate();
8) }
9)
10) void CDlgStudent::OnButton2()
11) {
12)     // TODO: Add your control notification handler code here
13)     CString str;
14)     if(selectia2>=0)
15)     {
16)         int lungime_cuvant;
17)         lungime_cuvant = m_ctrl_lista_studiate.GetTextLen(selectia2);
18)         m_ctrl_lista_studiate.GetText(selectia2,str.GetBuffer(lungime_cuvant));
19)         m_ctrl_lista_disponibile.AddString(str);
20)         m_ctrl_lista_studiate.DeleteString(selectia2);
21)         selectia2=-1;
22)         str.ReleaseBuffer();
23)     }
24)     //salvam in materiile_alese ce avem in lista2
25)     int n=m_ctrl_lista_studiate.GetCount();
26)     for(int i=0;i<n;i++)
27)     {
28)         int lungime_cuvant=m_ctrl_lista_studiate.GetTextLen(i);
29)         m_ctrl_lista_studiate.GetText(i, materiile_alese[i].GetBuffer(lungime_cuvant));
30)     }
31)     for(i=n;i<16;i++)
32)         materiile_alese[i]="";
33)     Invalidate();
34) }
35) }
```

Pasul 4: Vom crea acum o nouă clasă (generică), **CFisaIndividualaStudent**, clasă care va avea următoarele variabile membre: **Nume**, **Prenume**, **Specializarea**, **MateriiStudiate** (ambele de tipul **CString**), **Varsta** (de tipul **int**) și **Bursier** (de tipul **BOOL**).

Pentru această clasă vom crea următorul constructor:

Laborator C++

CFisaIndividualaStudent(CString nume, CString prenume, int varsta, CString materiistudiate, BOOL bursier, CString specializarea);

Completați definiția acestei clase cu următoarele linii de cod:

```
1) class CFisaIndividualaStudent
2) {
3) public:
4)     CFisaIndividualaStudent(CString nume, CString prenume, int varsta,
5)                             CString materiistudiate, BOOL bursier, CString specializarea);
6)     virtual ~CFisaIndividualaStudent();
7) public:
8)     CString Nume, Prenume, MateriiStudiate, Specializarea;
9)     int Varsta;
10)    BOOL Bursier;
11) };
```

Constructorul acestei clase trebuie completat cu următoarele linii de cod:

Acum în clasa document vom introduce o variabilă-membru numită **lista_studenti** și care va reprezenta un vector cu toți studenții pe care îi introducem. Pentru aceasta urmați pașii de mai jos.

Pe pagina **ClassView** efectuați un dublu-clic pe numele clase reprezentare și completați cu liniile de cod necesare (liniile **2**, **3** și **12**) pentru a arăta ca mai jos:

```
1) #endif // _MSC_VER > 1000
2) #include "FisaIndividualaStudent.h"
3) #include<vector>
4)
5) class CDateStudentiDoc : public CDocument
6) {
7) protected: // create from serialization only
8) CDinNou22Doc();
9) DECLARE_DYNCREATE(CDinNou22Doc)
10) // Attributes
11) public:
12)     std::vector<CFisaIndividualaStudent> lista_studenti;
...

```

Astfel în linia **12**) am declarat o variabilă membru pentru clasa document numită **lista_studenti** și în care vom stoca mai multe obiecte de tip **CFisaIndividualaStudent**. (motiv pentru care am și scris comanda din linia **2**))

Pasul 5: Creați un nou submeniu numit **Student** și o opțiune de meniu numită **Adaugare** după care apăsați un clic dreapta și selectați opțiunea **Properties**. Introduceți identificatorul **ID_ADAUGARE** apoi apăsați enter pentru a valida. Deschideți **MFC ClassWizard**. Selectați numele clasei **CDateStudentiDoc**, numele identificatorului selectat să fie **ID_ADAUGARE**, selectați mesajul **COMMAND** după care selectați butonul **Add Function** Acceptați numele implicit **OnAdaugare** apăsând pe butonul **OK** după care apăsați pe butonul **Edit Code**. Apoi completați cu liniile de cod de mai jos:

```
1) #include "DlgStudent.h"
2) void CDateStudentiDoc::OnAdaugare()
3) {
4)     // TODO: Add your command handler code here
5)     CDlgStudent dlg;
6)     int ret=dlg.DoModal();
7)     if(ret==IDOK)
8)     {
9)         CString materii="", str1;
10)        for(int i=0;i<16;i++)

```

Laborator C++

```
11)         if(dlg.materiile_alese[i]!="")
12)         {
13)             str1.Format("%s, ",dlg.materiile_alese[i]);
14)             materii=materii+str1;
15)         }
16)         CFisaIndividualaStudent student(dlg.m_nume, dlg.m_prenume, dlg.m_varsta,
17)             materii,dlg.m_este_bursier, dlg.m_specializarea);
18)         lista_studenti.push_back(student);
19)     }
20)     UpdateAllViews(NULL);
21) }
```

În linia **6)** se va deschide caseta de dialog **dlg**. În cazul apăsării pe butonul **OK** (deci a validării opțiunilor făcute) introducem în șirul **materii** numele tuturor materiilor alese (adică cele care vor fi în lista a doua) după care vom construi un obiect de tipul **CFisaIndividualaStudent** pe care îl vom adăuga în vectorul **lista_studenti**. **Obs.** Variabila **m_varsta** este variabila de tipul **int** pe care am mapat-o peste caseta de editare unde se introduce varsta. Am mapat această variabilă deoarece este mult mai ușor de folosit. Deschideți deci caseta de dialog **MFC ClassWizard** și în pagina **Member Variables** selectați clasa **CDlgStudent**. Acum adăugați o variabilă de tipul **int** peste caseta de editare al cărui identificator **IDC_INT_VARSTA**. Astfel de fiecare dată când veți dori să aflați care este valoarea înscrisă în această casetă de editare va trebui doar să consultați această variabilă de tipul **int**.

Pasul 6: În funcția **OnDraw()** a clasei reprezentare introduceți următoarele linii de cod:

```
1) void CDateStudentiView::OnDraw(CDC* pDC)
2) {
3)     CDateStudentiDoc* pDoc = GetDocument();
4)     ASSERT_VALID(pDoc);
5)     // TODO: add draw code for native data here
6)     int n=pDoc->lista_studenti.size();
7)     CSize sizeTotal;
8)     sizeTotal.cx =100; sizeTotal.cy =60*n;
9)     SetScrollSizes(MM_TEXT, sizeTotal);
10)    CString str;
11)    for(int i=0;i<n;i++)
12)    {
13)        str.Format("%d) %s %s, %d ani, %s; ",i+1,pDoc->lista_studenti[i].Nume,
14)            pDoc->lista_studenti[i].Prenume,pDoc->lista_studenti[i].Varsta,
15)            pDoc->lista_studenti[i].Specializarea);
16)        if(pDoc->lista_studenti[i].Bursier)
17)            str=str+" (Este bursier)";
18)        else
19)            str=str+" (Nu este bursier)";
20)        pDC->TextOut(10, 60*i, str);
21)        str.Format("        Specializarea: %s", pDoc->lista_studenti[i].Specializarea);
22)        pDC->TextOut(10, 60*i+20, str);
23)        str.Format("        Materiile studiate: %s",pDoc->lista_studenti[i].MateriiStudiate);
24)        pDC->TextOut(10, 60*i+40, str);
25)    }
26) }
```

În liniile **11)-25)** afișăm fiecare student din listă împreună cu datele sale personale.

Laborator 22

Problema:

Scrieți un program MFC de tip SDI care permite utilizatorului să deseneze cercuri și pătrate în zona client a ferestrei prin apăsarea butonului stâng al mouse-ului. Instrumentul de desenare se selectează dintr-o caseta de dialog în care sunt două butoane radio (cerc respectiv pătrat). Grosimea și culoarea liniilor se stabilește fiecare cu o caseta de dialog personalizată. Datele se vor salva pe disc.

Observatii:

/* Crearea unei funcții mesaj răspuns la mesajul WM_LBUTTONDOWN, care desenează un patrat.

Crearea unei funcții mesaj răspuns la mesajul WM_MOUSEMOVE, care desenează un cerc la fiecare punct prin care trece mouse-ul. Atenție !!! funcțiile mesaj răspuns vor face parte din clasa View.

În aceste funcții se va crea un context de dispozitiv cu CClientDC dc(this). (Codul pentru desenare nu va apărea în OnDraw ci în funcția care tratează mesajele)

Adăugarea unor variabile membre în clasa vizualizare. Ultimul punct în care s-a dat click, culoare folosită pt. desenare și unealta care se folosește (cerc, patrat).

Mutarea codului pt. desenare în funcția OnDraw(). Apelarea funcției Invalidate() din funcțiile care trebuie să modifice vizualizarea.

Adăugarea unei opțiuni de meniu pt. Stabilirea culorii de desenare care se alege folosind clasa CColorDialog. Funcția doModal().

Adăugarea unei opțiuni de meniu pt. Stabilirea uneltei de desenare. Unelte disponibile se vor defini într-o enumerare enum Tool {Cerc, Patrat}.

Editarea constructorului clasei vizualizare pt. a inițializa culoarea curentă, unealta și punctul curent.

*/

Rezolvare:

Se crează un proiect de tip **SDI** cu numele **Desenare**, după care se adaugă o clasă de tip generic cu numele **Forma** de tip generic care va fi clasă de bază pentru clasele **CCerc** și **CPătrat** pe care le vom implementa pe urmă.

Dar să revenim la clasa **Forme** după ce se va implementa și se va adăuga un constructor care va arăta astfel:

```
Forme::Forme(CPoint c, unsigned d, COLORREF u, COLORREF l, UINT g)
{
    centru=c;
    x=d;
    fillcolor=u;
    linecolor=l;
    grosime=g;
}
```

variabilele care apar vor fi declarate tot în clasa **Forme**

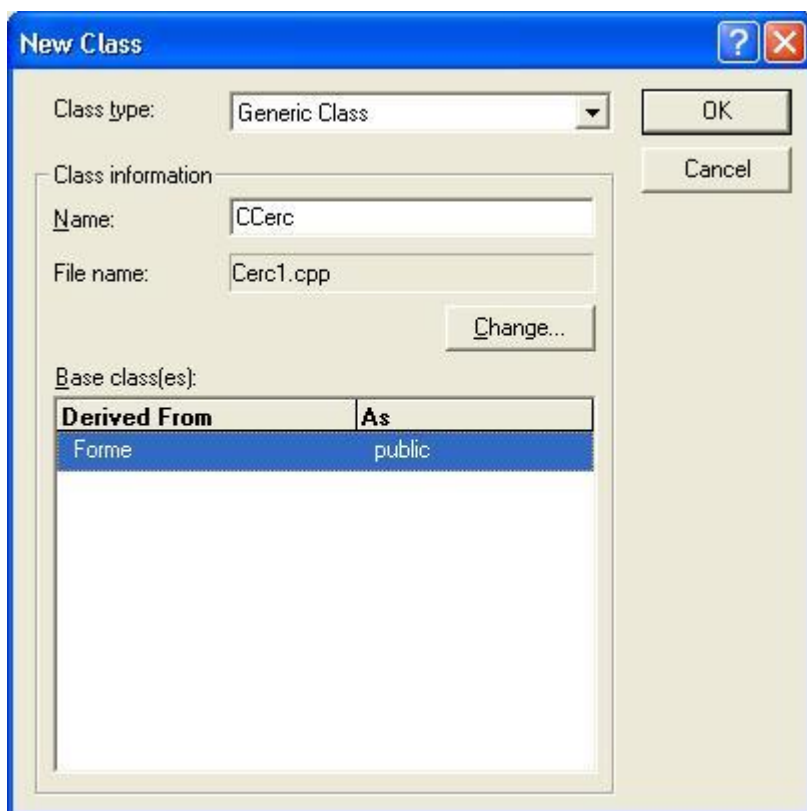
```
UINT grosime;
COLORREF fillcolor;
COLORREF linecolor;
unsigned x;
CPoint centru;
```

De asemenea clasei **Forme** i se adaugă și o funcție membru virtual pură cu numele **Draw**, adică în declarația clasei se introduce următorul cod în secțiunea public:

```
virtual void Draw(CDC *) =0;
```

Clasele **CCerc** și **CPatrat** vor fi implementate la fel.

Se apasă click dreapta pe **Desenare Class** din **ClassView** se alege **New Class** și se scrie numele clasei, tipul acestuia în cazul nostru **Generic Class** și clasa din care se va deriva în cazul nostru **Forme**



Clasei **CCerc** i se va adăuga un constructor și o funcție membră **Draw**.
Constructorul va arăta astfel:

```
CCerc::CCerc(CPoint p, unsigned c, COLORREF u, COLORREF l, UINT  
g) :Forme(p,c,u,l,g)  
{  
}
```

și are ca și clasă de bază clasa **Forme**.
Funcția membru va arăta astfel:

```
void CCerc::Draw(CDC *pdc)  
{  
    pdc->Ellipse(centru.x,centru.y,centru.x-x,centru.y-x);  
}
```

La fel se procedează și în cazul clasei **CPatrat** doar funcția **Draw** va fi puțin schimbată în sensul că în loc de **Ellipse** se va scrie **Rectangle**.

Laborator C++

```
void CPatrat::Draw(CDC *pdc)
{
    pdc->Rectangle(centru.x,centru.y,centru.x-x,centru.y-x);
}
```

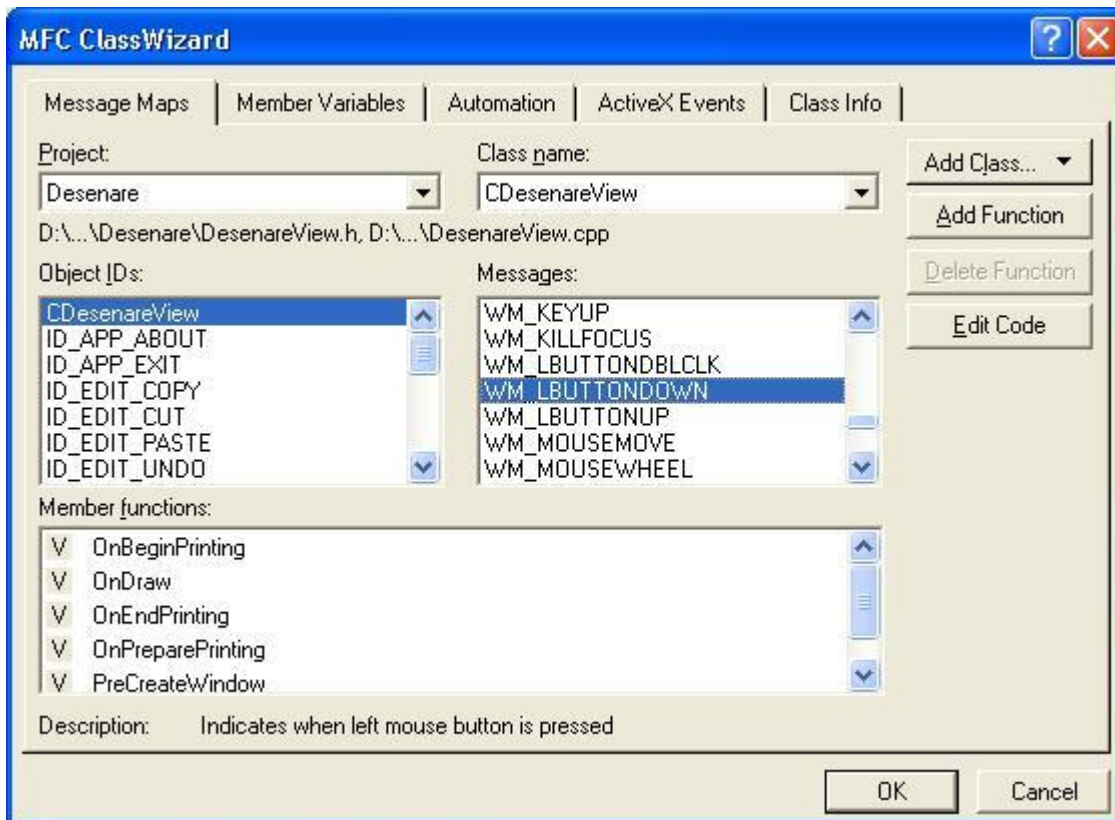
Următorul pas în rezolvarea problemei va fi adăugarea unei funcții care tratează apăsarea butonul stîng al mouse-ului adică atunci când vom apăsa butonul stîng se va desena în zona client un cerc sau în pătrat în funcție de opțiunea aleasă. Inițial se va desena cercuri, pentru a realiza acest lucru se va introduce următorul cod în constructorul clasei **CDesenareView**

```
raza=30;
formacurenta=0;
```

și se vor declara variabilele:

```
int formacurenta;
UINT raza;
```

forma curentă se va folosi pentru a seta forma patrat sau cerc, iar raza va fi raza cercului respectiv a pătratului. Se deschide **ClassWizard** din meniul **View** și se adaugă funcția **WM_LBUTTONDOWN** apăsând butonul **Add**



Function după care **Edit Code** pentru a adăuga cod în noua funcție creată.

După apăsarea butonului **Edit Code** funcția va fi modificată ca să arate astfel:

```
void CDesenareView::OnLButtonDown(UINT nFlags, CPoint point)
{
    // TODO: Add your message handler code here and/or call default
    CDesenareDoc* pDoc = GetDocument();
    Shack *r;
```

Laborator C++

```
switch (curentshack)
{
    case 0: r=new CCerc(point,raza,0,0,1);break;
    case 1: r=new CPatrat(point,raza,0,0,1);break;
}
pDoc->forme.push_back(r);
Invalidate();
CScrollView::OnLButtonDown(nFlags, point);
}
```

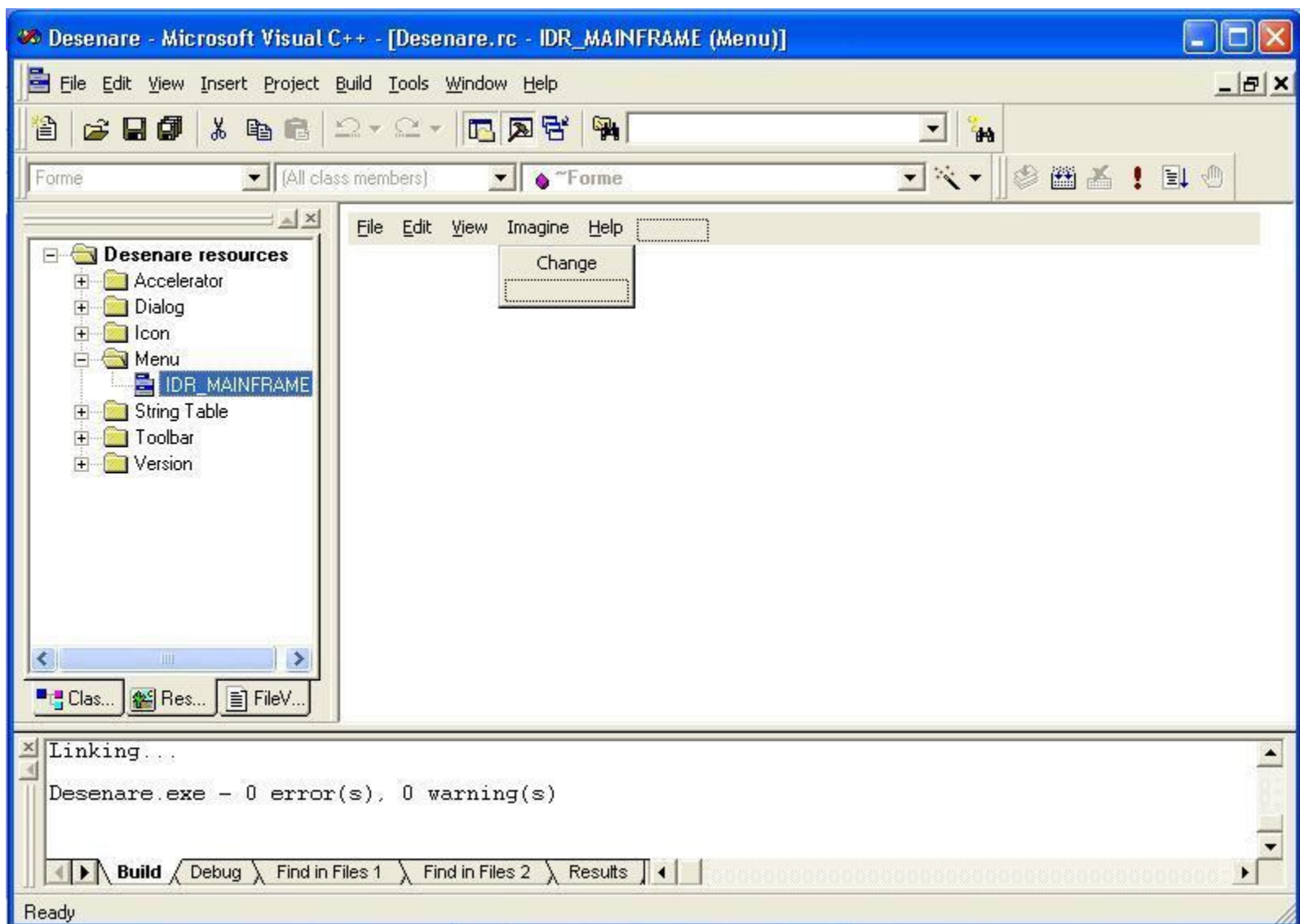
Pentru a se realiza serializarea se introduce următorul cod în declarația clasei **CdesenareDoc** în secțiunea public

```
std::vector<Forme *> forme;
```

și tot în acest fișier în secțiunea **#include** se scrie codul de mai jos:

```
#include <vector>
#include "Forme.h"
```

Pentru a se schimba forma cu care se desenează adică cercuri și pătrate se introduce o opțiune de meniu cu numele **Imagine** și un submeniu cu numele **Change**. Acest lucru se realizează apăsând un click dreapta pe opțiunea **ResourceView** și se deschide directorul **Menu** și în cadrul lui opțiunea **IDR_MAINFRAME**



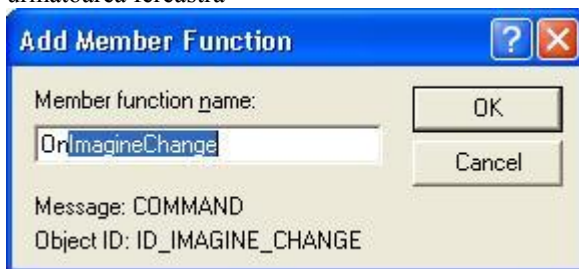
Laborator C++

după care se va adăuga meniu dorit apăsându-se un dublu click în locul destinat scrieri de noi meniuri după meniul **Help** și pe urmă se va muta în fața meniului **Help** prin apăsarea butonului stâng al mouse-ului și drenarea (adică se ține apăsător în timpul deplasării la stânga) acestuia în stânga. După realizarea meniului se apasă dublu click pe opțiunea **Change** și ne va apărea următoarea fereastră



în care putem observa ID-ul opțiunii **Change** în cazul nostru **ID_IMAGINE_CHANGE**.

În **ClassWizard** se va adăuga funcția **COMMAND** cu ajutorul opțiunii **Add Function** și va fi afișată următoarea fereastră



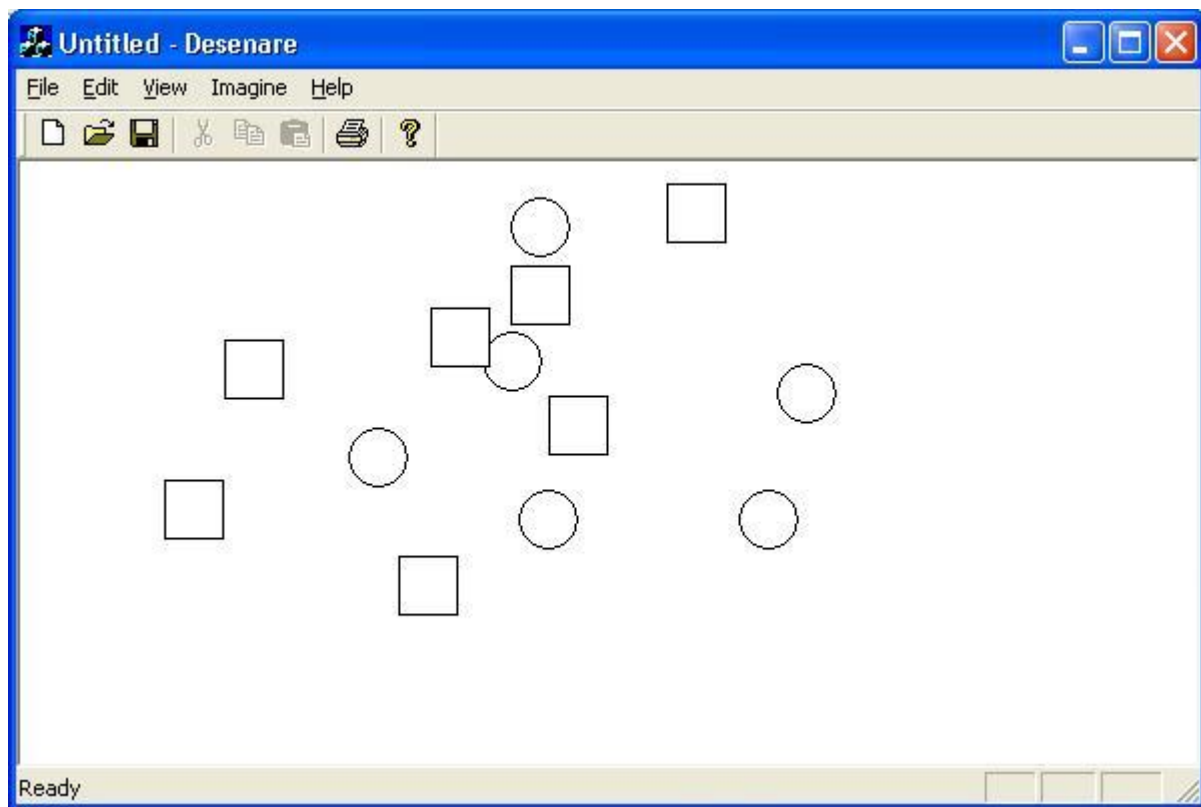
în care se va putea schimba numele funcției care tratează apăsarea acestui meniu adică **Change**.

După apăsarea butonului **OK** se va introduce în această funcție următorul cod:

```
if (formacurenta==1) formacurenta=0;  
else formacurenta=1;
```

în care se schimbă efectiv forma care se va desena în cadrul zonei client a ecranului.

După compilarea și execuția programului vom putea desena cercuri și pătrate cu ajutorul mouse-ului.



Laborator 23

Scrieti un program MFC pentru testarea functiei Message Box. Aplicatie de tip SDI. Se va crea o casuta de dialog personalizata cu urmatoarea forma:

La apasarea butonului OK se va afisa un MessageBox care va avea prop. date de caseta de dialog de mai sus. Dupa apasarea unui buton din Message Box va apare un alt Message Box in care va apare icoana Information butonul OK iar textul va indica butonul apasat.

Laborator 24

Scrieti un program MFC care cere utilizatorului introducerea unor date despre persoane nume, prenume, varsta sau data nasteri, locul de munca, starea civila. Sa se introduca intr-o caseta de dialog. Datele care vor fi introduse intr-o colectie numai daca sunt validate. Exemplu: Numele sa fie fara spatiu. Daca datele nu sunt vald atunci se va afisa un MessageBox prin care utilizatorul va fi informat ca a gresit ceva. Sa se incrementeze serializarea datelor (incarcarea din fisier).

Laborator 25

Scrieti un program MFC care permite introducerea unor siruri de caractere prin intermediul unei casete de dialog personalizata, siruri care vor fi afisate in zona client a ferestrei unul sub altul. Primul sir care se afiseaza este un antet care se va putea modifica oricand cu ajutorul unei alte casete de dialog personalizata. Sirurile vor fi salvate intr-o colectie de stringuri. CStringArray. Se va salva starea obiectului pe disc in fisier. Se va putea si incarca de pe disc acel obiect.

Laborator 26

Scrieti un program care sa permita desenarea in partea client cercuri prin apasarea butonului stang al mousului. Se vor putea parametriza prin casete de dialog: raza cercului, culoarea cercului, culoarea de umplere. Se va folosi o colectie de cercuri care se va putea salva pe si de pe disc.

Laborator 27

Lucrul direct cu fișiere de pe disc.

Problemă:

Scrieți un program care permite deschiderea unui fișier text de pe disc prin intermediul unei casete de dialog **File** -> **Open**. Se folosește **CFileDialog**. Și afișarea conținutului fișierului în zona client a ferestrei. Pentru lucru cu fișiere se folosește **CFile** (nume fișier, mod de deschidere, ... etc).

Rezolvare:

Se crează un proiect de tip **SDI** cu numele **Fișiere**. Clasa de bază a acestui proiect va fi **CScrollView**, deci la pasul 6 al aplicației se va înlocui în caseta **Base Class** clasa **CView** cu clasa **CScrollView**.

În clasa **CFisiereDoc** se declară o variabilă de tip **CStringArray** cu numele **linii**:

```
CStringArray linii;
```

Această variabilă se va folosi pentru a citi linii din fișierul ce urmează a fi descris.

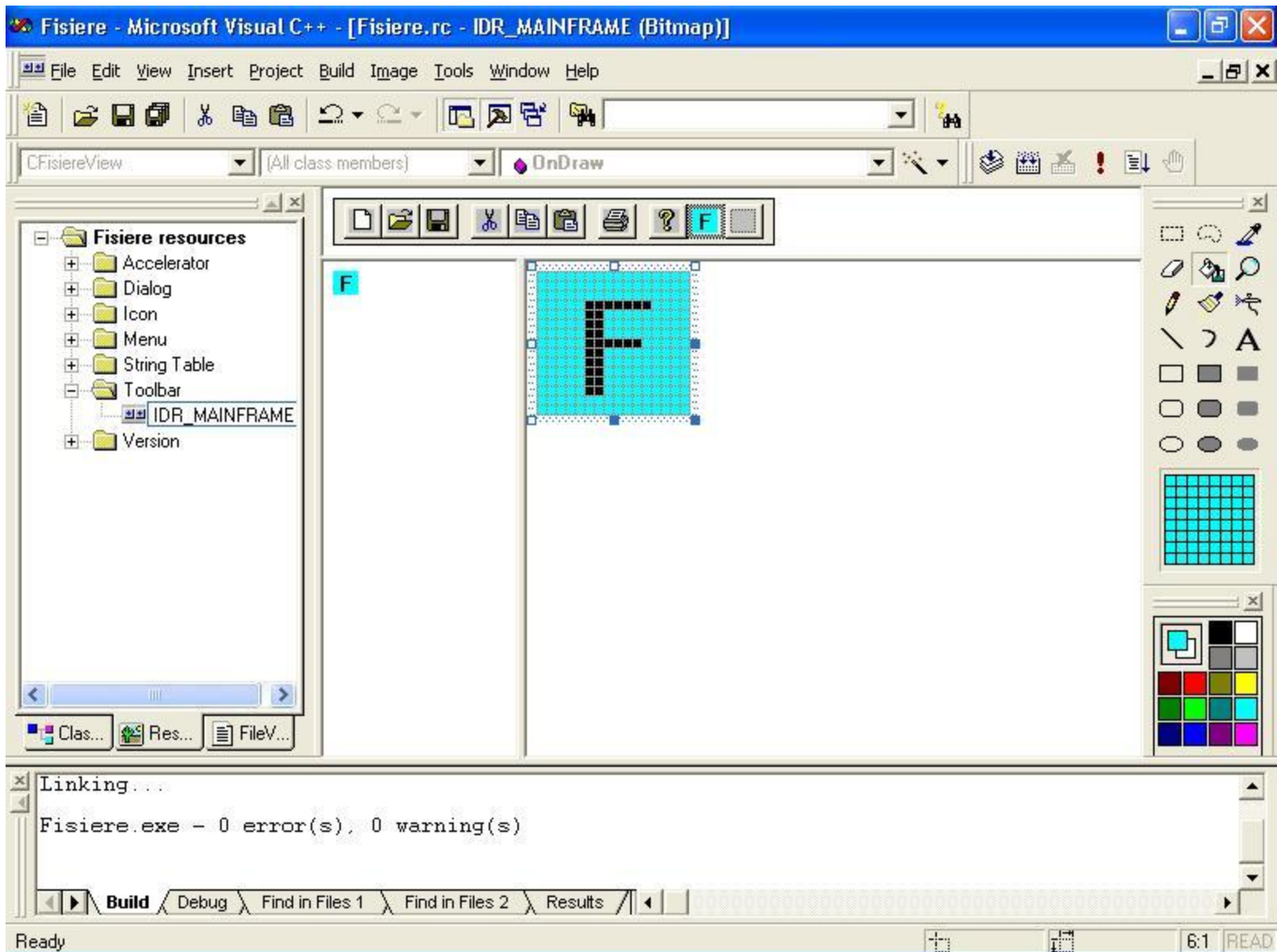
Funcția **OnDraw** a clasei **CFisiereView** se va modifica astfel încât să arate astfel:

```
1 void CFisiereView::OnDraw(CDC* pDC)
2 {
3     CFisiereDoc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5     // TODO: add draw code for native data here
6     int height = 20;
7     TEXTMETRIC tm;
8     pDC->GetTextMetrics(&tm);
9     height = tm.tmHeight;
10    int n = pDoc->linii.GetSize();
11    CSize sizeTotal;
12    // TODO: calculate the total size of this view
13    sizeTotal.cx = 500;
14    sizeTotal.cy = n * height;
15    SetScrollSizes(MM_TEXT, sizeTotal);
16    for (int i = 0; i < n; i++)
17        pDC->TextOut(20, height * i, pDoc->linii.GetAt(i));
18 }
```

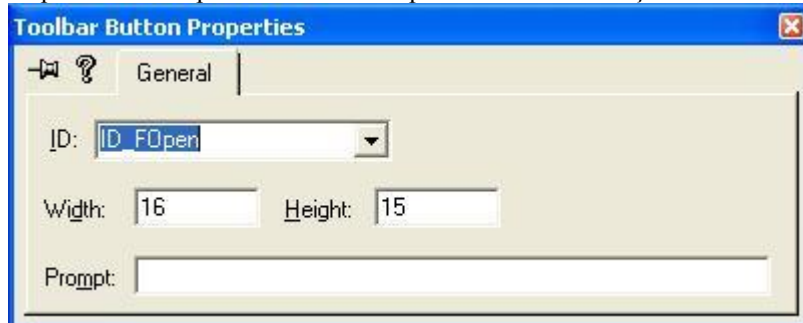
În linia 6 se declară o variabilă de tip întreg cu numele **height** și se inițializează cu valoarea 20 care se va folosi pentru a seta înălțimea textului afișat.

Se deschide **ResourceView**, directorul **Toolbar** și pe urmă **IDR_MAINFRAME** și se adaugă un buton astfel:

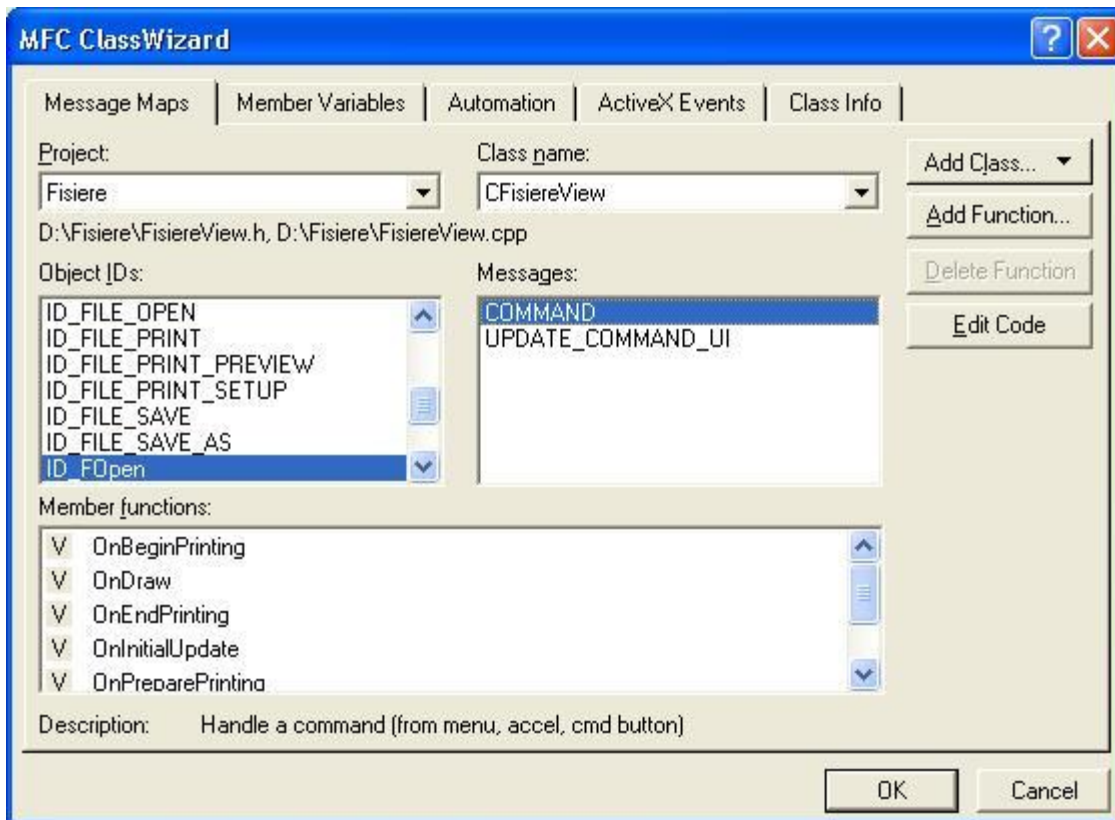
Laborator C++



După care se va apăsa un dublu click pe butonul nou creat și se va modifica ID-ul butonului astfel:



La următorul pas se va deschide **ClassWizard** și se va alege pagina **Message Maps**, **Object ID**-ul ales va fi **ID_FOpen** adică cel creat de noi pentru noul butonul. În fereastra **Messages** va fi aleasă opțiunea **COMMAND**



și pe urmă va fi apăsat butonul **Add Function** și se va adăuga o funcție cu numele **OnFOpen** astfel:



După care se va apăsa butonul **OK**.

Funcția pe care am creat-o va fi modificată astfel:

```

1 void CFisierView::OnFOpen()
2 {
3     // TODO: Add your command handler code here
4     CFileDialog dlg(TRUE, "txt", "Untitled");
5     int ret = dlg.DoModal();
6     ifstream file;
7     char buffer[501];
8     CString temp;
9     CFisierDoc* pDoc = GetDocument();
10    ASSERT_VALID(pDoc);
11    if (ret == IDOK)
12    {
13        file.open(dlg.GetPathName());
14        if (!file)    MessageBox("Nu am reusit sa deschid fisierul");
15        else

```

Laborator C++

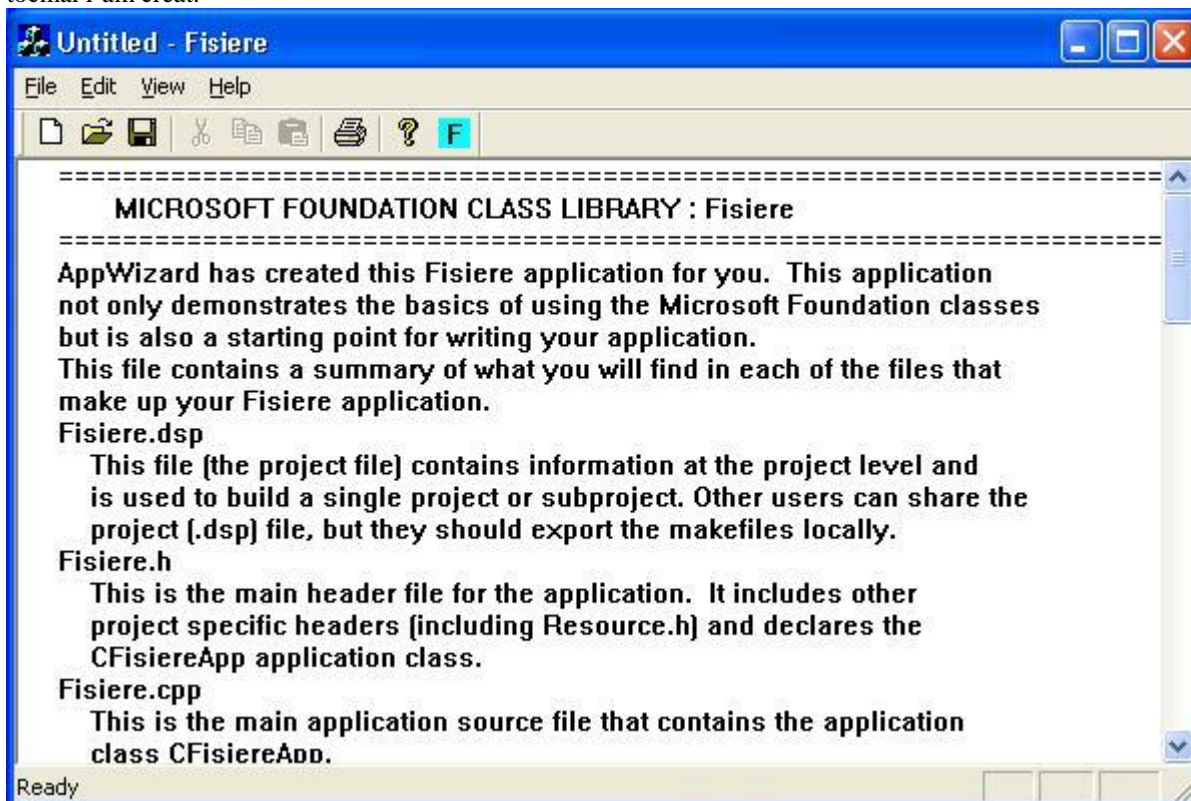
```
16     {
17         pDoc->linii.RemoveAll();
18         while(! file.eof())
19             {
20                 file .getline(buffer, 500);
21                 if (buffer[0] != '\0')
22                     {
23                         temp.Format("%s", buffer);
24                         pDoc->linii.Add(temp);
25                     }
26     Invalidate();
27     }
28 }
29 }
```

În fișierul **FisiereView.cpp** se va adăuga la începutul fișierului după secțiunea unde se include fișierele următoarea linie:

```
#include <fstream.h>
```

pentru a putea folosi clasa ifstream.

După compilarea și executarea programului se vor putea citi fișiere text de pe disc prin apăsarea butonului creat din cadrul meniului **Toolbar**. În exemplul de mai jos am deschis fișierul **ReadMe.txt** din proiectul pe care tocmai l-am creat.



Laborator 28

L26 Lucru cu bitmap.

L28 Lucru cu pagini de proprietăți.