3. Introducere în MFC

După cum sugerează şi denumirea, ***Microsoft Foundation Class*** (sau prescurtat ***MFC***) este o bibliotecă de clase, dezvoltată de firma Microsoft în vederea uşurării programării aplicațiilor Windows. Multe dintre clasele MFC încapsuleaza funcțiile Windows API, dar permit o utilizare mai "prietenoasă" a acestora.

Folosind clasele din biblioteca MFC, aplicațiile Windows se pot programa obiectual de o manieră mult mai simplă decât permite API. Acest fapt prezintă următoarele avantaje:

- creşte semnificativ lizibilitatea codului, o aplicație putând fi "descifrată" mult mai uşor;
- se simplifică programarea aplicației, prin folosirea claselor deja implementate de MFC;
- se micşorează semnificativ timpul de dezvoltare al unei aplicații Windows;
- programatorul are posibilitatea de a se concentra numai asupra aspectelor particulare ale aplicației, lăsând aspectele standard (clase de ferestre, proceduri fereastră etc.) în seama claselor MFC;
- se pot folosi în procesul de programare programele *vrăjitor* (***wizards***) precum ***AppWizard*** şi ***ClassWizard***, care uşurează munca programatorului;

Clasele incluse în biblioteca MFC sunt dispuse în mai multe categorii, dintre care putem aminti:

- ***clase de uz general*** - clase pentru manipularea fişierelor (`CFile`) şi şirurilor de caractere (`CString`), clase excepții (`CException`), clase pentru reprezentarea unor zone de pe ecran (`CRect`, `CPoint`);
- ***clase de obiecte vizuale*** - aceste clase manevrează aproape tot ce este vizibil într-un program Windows. În această categorie intră clasele de ferestre (`CWnd`, `CFrameWnd` şi clasele derivate din ele), meniuri (`CMenu`), controale (`CButton`, `CEdit`, `CListBox` etc. – derivate din clasa `CWnd`) şi obiecte de desenare, cum ar fi creioanele (`CPen`) şi pensulele (`CBrush`);
- ***clase de aplicație*** - în această categorie sunt incluse clasele care gestionează obiectele fir de execuție (`CWinThread`), aplicație (`CWinApp` – derivată din `CWinThread`), precum şi clasele care implementează arhitectura Document-Reprezentare (`CDocument`, `CView`, şi clasele derivate din ele). Arhitectura Document-Reprezentare este folosită de majoritatea programelor Windows;
- ***clase de tip colecție*** - aceste clase se folosesc pentru stocarea altor obiecte în structuri de date de tip tablou (`CArray`), listă (`CList`) sau hartă (`CMap`);
- ***clase suport OLE2*** - aceste clase simplifică scrierea aplicațiilor care beneficiază de facilitățile OLE2;
- ***clase pentru baze de date*** - aceste clase facilitează manipularea bazelor de date (`CDatabase`, `CRecordset`, `CDaoDatabase`, `CDaoRecordset`);
- ***clase pentru dezvoltarea aplicațiilor distribuite (în rețea)*** - câteva dintre aceste clase sunt: `CSocket` (pentru deschiderea unui canal de comunicație între două calculatoare), `CFTPConnection` şi `CHTTPConnection` (pentru deschiderea unei sesiuni de lucru FTP, respectiv HTTP);

Clasele implementate în biblioteca MFC folosesc moştenirea simplă. Majoritatea claselor MFC sunt derivate (direct sau indirect) din clasa `CObject`. De asemenea, toate clasele care reprezintă ferestre sau controale de diferite tipuri sunt derivate din clasa `CWnd`. Clasele `CObject` şi `CWnd` utilizează funcțiile virtuale pentru implementarea polimorfismului în clasele derivate. Aceasta permite obiectelor din program să acceseze funcții de uz general, prin intermediul unui pointer la clasa de bază.

3.1 Scrierea unei aplicații simple cu MFC

De obicei, la scrierea unui program MFC, se foloşeste ***AppWizard***, un program "vrăjitor" care generează automat scheletul programului, particularizat pentru aplicația respectivă. Totuşi, un program MFC se poate scrie şi fără a beneficia de asistența vreunui program vrăjitor. Chiar şi în acest caz, este mai uşor să se scrie un program utilizând biblioteca MFC, decât direct, folosind funcțiile Windows API.

Pentru a scrie un program MFC fără a folosi vreun program vrăjitor, trebuie parcurşi următorii paşi:

- se creaza un proiect de tip **Win32 Application**, care nu conține nici un fişier (**An empty project**);
- din meniul **Project->Settings**, se selectează **Settings for: all configurations**, iar de la tab-ul **General**, în caseta combinată **Microsoft Foundation Classes:** se selectează **Using MFC in a shared library**;
- se introduce codul programului, se compilează, se editează legăturile şi se lansează în execuție.

Să facem un prim exemplu. Vom crea un proiect după paşii descrişi mai sus, pe care-l vom numi ***PrimulMFC***. Vom adăuga proiectului un fişier header, numit *PrimulMFC.h*:

```
class CPrimaAplicatie: public CWinApp
{
public:
   BOOL InitInstance();
};

class CPrimaFereastra: public CFrameWnd
{
public:
   CPrimaFereastra();
};
```

Acum urmează să adăugăm la proiect fişierul *PrimulMFC.cpp*, în care vom defini funcțiile declarate anterior:

```
#include <afxwin.h>
#include "PrimulMFC.h"

BOOL CPrimaAplicatie::InitInstance()
{
   m_pMainWnd = new CPrimaFereastra;
   m_pMainWnd->ShowWindow(m_nCmdShow);
   m_pMainWnd->UpdateWindow();
   return TRUE;
```

```
}

CPrimaFereastra::CPrimaFereastra()
{
   Create(NULL, "Prima mea aplicatie MFC");
}

   CPrimaAplicatie primaApp;
```

Dacă vom compila şi executa programul, va apare o fereastră! Dar, nu avem nicăieri o funcţie `main()` şi totuşi programul se execută corect! De aceea, se impun câteva explicaţii pe marginea acestui program:

- pentru a putea folosi clasele MFC, este necesară editarea legăturilor între fişierele cu cod obiect ale bibliotecii MFC şi fişierele proiectului (selectarea opţiunii **Project->Settings** etc.). Declaraţiile claselor din biblioteca MFC este făcută în fişierul *afxwin.h*, deci acest fişier trebuie inclus în codul sursă al programului;
- orice program MFC este un program executabil. El trebuie să conţină o clasă derivată public din clasa `CwinApp`, care este o clasă care încapsulează programe Windows. Orice program care se execută sub Windows este un obiect de clasă `CWinApp` sau de o clasă derivată public din aceasta. De obicei, clasa derivată public redefineşte metoda `InitInstance()` a clasei `CWinApp`. Această metodă, după cum sugerează şi numele său, este apelată automat de cadrul aplicaţie la pornirea unei instanţe a aplicaţiei (la crearea unui obiect aplicaţie, pentru a-l iniţializa). Ea este declarată ca virtuală în cadrul clasei `CWinApp`, aşa încât se va apela varianta corectă (cea redefinită în noua clasă aplicaţie). Este neapărat necesară redefinirea funcţiei `InitInstance()` cu semnătura:

  ```
  BOOL InitInstance()
  ```

  pentru a nu se pierde caracterul de funcţie virtuală.

  De obicei, funcţia `InitInstance()` are rolul de a crea fereastra principală a aplicaţiei şi de o a afişa pe ecran. Pentru aceasta, în funcţia `InitInstance()` se utilizează pointerul `CWnd* m_pMainWnd`, care este o variabilă membru al clasei `CWinApp` şi moştenită de clasa aplicaţie (`CPrimaAplicatie` în cazul nostru), pentru a crea un nou obiect fereastră, asociat programului aplicaţie. Pentru aceasta, se scrie o linie de cod ca mai jos:

  ```
  m_pMainWnd = new CPrimaFereastra;
  ```

  unde în operatorul `new` se apelează constructorul unei clase derivată, direct sau indirect, în mod public, din clasa `CWnd` sau `CFrameWnd`. Aceasta permite conversia explicită de la un obiect de tip `CPrimaFereastra*` (returnat de operatorul `new`, în exemplu) la un obiect de tip `CWnd*` (respectiv `m_pMainWnd`).

  După crearea obiectului de tip fereastră, acesta este afişat pe ecran folosind funcţiile `ShowWindow()` şi `UpdateWindow()`, membre ale clasei `CWnd`. Funcţia `ShowWindow()` primeşte ca parametru un întreg (`m_nCmdShow`) care specifică modul de afişare iniţială a ferestrei aplicaţiei (normal, minimizat, maximizat).

  După succesiunea normală a operaţiilor sus-menţionate, funcţia `InitInstance()` returnează `TRUE`, semnalizând faptul că aplicaţia poate continua normal.

- folosirea unei clase derivate (direct sau indirect) din clasa `CWnd` nu este obligatorie. Se pot folosi una dintre clasele fereastră furnizate de MFC: `CWnd`, `CFrameWnd`, sau altă clasă fereastră. Totuşi, exemplul de mai sus defineşte o nouă clasă fereastră, bazată pe clasa `CFrameWnd`. Motivul utilizării acestei clase ca şi clasă de bază este prezentat în paragraful următor. Clasa `CFrameWnd` este derivată direct din clasa `CWnd`, şi reprezintă o fereastră cadru a unei aplicații. Constructorul clasei *`CFirstWnd`* trebuie să apeleze funcția `Create()` (membră a clasei `CFrameWnd`). Primul parametru transmis funcției se referă la clasa de ferestre pe care dorim să o folosim pentru fereastra cadru a aplicației. El este `NULL`, deci se va folosi o clasă de ferestre înregistrată de MFC. Cel de-al doilea parametru este titlul ferestrei principale a aplicației. Ceilalți parametri ai funcției `CFrameWnd::Create()` nu mai sunt transmişi explicit, ei luând valori implicite.
- orice program MFC posedă un obiect (şi numai unul) derivat din clasa aplicație (`CPrimaAplicatie primaApp`, în cazul exemplului de mai sus). Declararea unui asemenea obiect (sub forma unei variabile globale) duce la declanşarea mecanismului MFC de desfăşurare a unei aplicații (apelul funcției `InitInstance()` în constructorul clasei aplicație, care crează un nou obiect fereastră, apelându-se astfel constructorul clasei fereastră, care apelează funcția `Create()` pentru a crea fereastra, iar apoi aceasta este afişată, etc.). Putem considera acest obiect aplicație ca fiind echivalent funcției `main()` dintr-un program consolă, sau funcției `WinMain()` dintr-un program Windows scris folosind Windows API.

3.2 Manipularea mesajelor in MFC

Se poate observa cât de mult se simplifică scrierea unei aplicații Windows folosind MFC. Totuşi, făcând comparație cu aplicațiile Windows din capitolul precedent, această aplicație nu are o facilitate: ea nu tratează nici un mesaj Windows, ci lasă tratarea mesajelor în seama procedurilor implicite.

Să modificăm programul anterior, astfel încât să trateze mesajele Windows. Vom intercepta şi trata mesajele `WM_CLOSE` şi `WM_KEYDOWN`. În primul rând, va trebui să modificăm fişierul header:

```
class CPrimaAplicatie: public CWinApp
{
public:
   // redefineste functia InitInstance a clasei CWinApp
   BOOL InitInstance();
};
class CPrimaFereastra: public CFrameWnd
{
public:
   CPrimaFereastra();
protected:
   afx_msg void OnClose();
   afx_msg void OnKeyDown(UINT nChar);
   DECLARE_MESSAGE_MAP()
};
```

Pentru a putea trata mesajele adresate unei ferestre, este necesară definirea unei ***hărți de mesaje***. O hartă de mesaje este utilizată pentru a stabili legături între mesajele transmise unei ferestre şi funcțiile destinate procesării acestor mesaje. Declararea unei

hărți de mesaje se face în interiorul unei clase derivate din clasa `CWnd` (uzual în fişierul header), folosind macroul DECLARE_MESSAGE_MAP, astfel:

```
DECLARE_MESSAGE_MAP()
```

Observație: Nu se pune ";" după acest macrou!

Declararea hărții de mesaje se face în clasa derivată din clasa fereastră, deoarece, să ne reaminim, Windows trimite toate mesajele către fereastra asociată programului aplicație. Declararea hărții de mesaje şi a funcțiilor de tratare a mesajelor se face în cadrul unei secțiuni protejate a clasei fereastră. Funcțiile de tratare a mesajelor au semnături predefinite şi declararea lor începe întotdeauna cu cuvântul `afx_msg`. Acest cuvânt semnalizează compilatorului faptul că este vorba despre o funcție ce tratează mesaje. Ceea ce urmează după acest cuvânt cheie, este specific fiecărui mesaj tratat. Nu este necesară învățarea semnăturilor tuturor funcțiilor de tratare a mesajelor. Visual C++ posedă un program vrăjitor, numit ***ClassWizard***, care permite inserarea automată a funcțiilor de tratare a mesajelor în cadrul unei clase. Pentru folosirea *ClassWizard*, proiectele trebuie să fi fost generate cu ***AppWizard***, un alt program vrăjitor. Din păcate, înseamnă că nu putem folosi *ClassWizard* în exemplul de mai sus. Din fericire însă, uzual proiectele MFC se generează folosind *AppWizard*, tocmai în ideea de a beneficia de suportul oferit de *ClassWizard* (şi alte utilitare).

Observăm că funcția de tratare a mesajului `WM_CLOSE` are semnătura

```
afx_msg void OnClose();
```

şi este de fapt o funcție membră a clasei `CWnd`. Această funcție este redefinită de clasa `CFirstWnd`. Este necesar ca funcția `OnClose()` redefinită să apeleze versiunea sa de bază, prin intrucțiunea `CFrameWnd::OnClose()`, pentru că aceasta distruge obiectul fereastră şi dealocă memoria. Dacă nu am apela această funcție, ar trebui să avem noi grijă de aceste operații. Semnătura funcției de tratare a mesajului `WM_KEYDOWN` este

```
afx_msg void OnKeyDown(UINT nChar);
```

Este necesar ca această funcție să primească un parametru de intrare. Acest parametru este utilizat pentru transmiterea codului ASCII a tastei apăsate. De completarea lui se va ocupa biblioteca MFC.

Să vedem cum modificăm fişierul sursă:

```
#include <afxwin.h>
#include "PrimulMFC.h"

BOOL CPrimaAplicatie::InitInstance()
{
   …
   return TRUE;
}

BEGIN_MESSAGE_MAP(CPrimaFereastra, CFrameWnd)
   ON_WM_CLOSE()
   ON_WM_KEYDOWN()
END_MESSAGE_MAP()
```

```
CPrimaFereastra::CPrimaFereastra()
{
   Create(NULL, "Prima mea aplicatie MFC");
}

void CPrimaFereastra::OnClose()
{
   MessageBox("Aplicatie terminata", "Gata", MB_OK|MB_ICONSTOP);
   CFrameWnd::OnClose();
}

void CPrimaFereastra::OnKeyDown(UINT nChar)
{
   CString text;
   switch(nChar){
      case VK_BACK: text="Ati apasat tasta BackSpace"; break;
      case VK_TAB:text="Ati apasat tasta Tab"; break;
      case VK_LEFT:text="Ati apasat tasta SageataStinga";break;
      case VK_RIGHT:text="Ati apasat tasta SageataDreapta"; break;
      case VK_UP:text="Ati apasat tasta SageataSus"; break;
      case VK_DOWN:text="Ati apasat tasta SageataJos"; break;
      case VK_RETURN:text="Ati apasat tasta Enter"; break;
      case VK_ESCAPE:text="Ati apasat tasta Escape"; break;
      case VK_CONTROL:text="Ati apasat tasta Control"; break;
      default: text="Ati apasat alta tasta"; break;
   }
   MessageBox(text, "Mesaj",MB_OK|MB_ICONEXCLAMATION);
}

   CPrimaAplicatie primaApp;
```

Harta de mesaje declarată în fişierul header este definită în fişierul sursă. Definirea hărții de mesaje trebuie făcută în afara clasei. Se folosesc macrourile `BEGIN_MESSAGE_MAP` şi `END_MESSAGE_MAP`, care marchează începutul, respectiv sfârşitul hărții de mesaje.

Atenție! Harta de mesaje nu conține nici un caracter ";"!

Macroul `BEGIN_MESSAGE_MAP` primeşte doi parametri: numele clasei derivate şi numele clasei de bază. Macroul `END_MESSAGE_MAP` nu primeşte nici un parametru. Aceste două macrouri apar la începutul, respectiv sfârşitul oricărei hărți de mesaje. Iată la ce folosesc cei doi parametri transmişi macroului `BEGIN_MESSAGE_MAP`: când se recepționează un mesaj, se încearcă tratarea lui printr-o funcție definită în clasa derivată. Dacă clasa derivată nu furnizează o funcție de tratare a mesajului, tratarea mesajului este lăsată în seama clasei de bază, în care există implementate funcții de tratare pentru toate mesajele. Acest mecanism se repetă în cazul clasei de bază (dacă aceasta este la rândul ei derivată dintr-o altă clasă). Cu alte cuvinte, la recepționarea unui mesaj, de exemplu `WM_CLOSE`, programul va căuta funcția `OnClose()` (să ne reamintim, declarată ca şi funcție `afx_msg`) în clasa `CPrimaFereastra`. Dacă nu o va găsi, nu e nici o problemă, va trata mesajul prin intermediul funcției implicite din clasa `CFrameWnd`. Este un fapt oarecum aşteptat. Şi înainte de a intercepta mesajul `WM_CLOSE` explicit în program, acesta putea fi închis.

Implementarea funcțiilor este oarecum cunoscută. Este totuşi demn de remarcat că la implementare, cuvântul `afx_msg` nu mai apare. Funcția `OnClose()` apelează funcția `MessageBox()`, după care distruge fereastra.

Funcția `MessageBox()` apelată aici nu este funcția `MessageBox()` apelată în exemplul Windows API. Aici apelăm funcția `CWnd::MessageBox()`, care este o funcție membru a clasei `CWnd`. De altfel, cele două funcții au semnături diferite:

```
int MessageBox(HWND, LPCSTR, LPCSTR, UINT);
int MessageBox(LPCSTR, LPCSTR, UINT);
```

Prima semnătură aparține funcției API, iar a doua funcției `CWnd`. Dacă am fi dorit să apelăm funcția Windows API `MessageBox()`, ar fi trebuit să scriem:

```
::MessageBox(this->m_hWnd, "...", "...", MB_OK);
```

Funcția API `MessageBox()` solicită ca prim parametru o variabilă de tip `HWND`. De remarcat construcția `this->m_hWnd`. Aceasta se referă la o variabilă membru de tip `HWND`, numit `m_hWnd`, moştenit de clasa `CFirstWnd` din clasa `CWnd`.

De reținut faptul că orice funcție Windows API care este redefinită într-o clasă MFC se poate apela (în interiorul acelei clase, sau al unei clase derivate din ea) folosind construcția

```
::Nume_Functie_API(lista_parametri_actuali);
```

Funcția `OnKeyDown()` primeşte ca parametru valoarea `UINT nChar`. În funcție de valoarea transmisă de Windows acestui parametru, se afişează mesajul corespunzător.Tipul `UINT` este de fapt un întreg fără semn. Diferența dintre `UINT` şi `unsigned int`, sau `BYTE`, sau `WORD` sau `DWORD`, este că acestea au lungime de reprezentare fixă, în timp ce dimensiunea `UINT` depinde de mediul software în care se execută programul: în Windows 3.1, 3.11 este `WORD`, iar pentru platformele 9.x şi NT este `DWORD`.

Să revenim puțin la harta de mesaje. Aceasta conține între macrourile de început şi sfârşit, intrări care specifică ce mesaje sunt tratate de clasa respectivă. Intrările în harta de mesaje sunt de fapt tot nişte macrouri. Fiecare mesaj general Windows are asociat în MFC un astfel de macrou. Regula de numire a macroului asociat este următoarea: se adaugă la numele mesajului (scris cu litere mari) prefixul `ON_`. Rezultă astfel macrouri de forma:

```
ON_WM_CLOSE()
ON_WM_PAINT()
ON_WM_LBUTTONDOWN()
```

Pentru fiecare dintre aceste macrouri (folosite în harta de mesaje), clasa fereastră trebuie să declare şi să definească o funcție de tratare a mesajului respectiv (cu o semnătură predefinită). Pentru mesajele `WM_CLOSE`, `WM_PAINT` şi `WM_LBUTTONDOWN`, exemplificate mai sus, aceste funcții au semnăturile:

```
afx_msg void OnClose();
afx_msg void OnPaint();
afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
```

Toate aceste funcții sunt funcții membre ale clasei `CWnd`. Datorită faptului că ele sunt redefinite şi incluse în harta de mesaje a clasei `CFirstWnd`, se apelează versiunea corectă a lor (fără a fi nevoie de declararea lor ca funcții virtuale). Nu se foloseşte

mecanismul funcțiilor virtuale din următorul considerent: dacă o clasă de bază declară o funcție virtuală, toate clasele derivate din acea clasă de bază vor conține câte un tablou de pointeri la versiunea corectă de funcție virtuală care trebuie apelată. Având în vedere faptul că sunt câteva sute de mesaje Windows, dimensiunea acestui tablou de funcții virtuale ar fi fost foarte mare şi de aceea s-a recurs la soluția hărților de mesaje.

Din cele arătate mai sus, putem trage concluzia că MFC lucrează cu două tipuri de funcții:

- ***funcții obişnuite***, care pot fi fie funcții membre ale unei clase, fie funcții din afara clasei. Pentru aceste funcții sunt necesare următoarele operații:
  1. declararea lor, uzual în fişiere header;
  2. definirea lor, uzual în fişiere sursă;
  3. apelarea funcției în cadrul programului;
- ***funcții care răspund la mesaje***, care se execută la transmiterea unui mesaj de către sistemul de operare către program. Pentru aceste funcții, sunt necesare următoarele operații:
  1. declararea lor, uzual în fişiere header, obligatoriu ca şi funcții `afx_msg`;
  2. inserarea în harta de mesaje a macroului `ON_...` corespunzător, pentru a specifia programului ce mesaje sunt interceptate;
  3. definirea lor, uzual în fişiere sursă;

Funcțiile* `afx_msg`*, spre deosebire de funcțiile obişnuite, nu trebuie apelate în program pentru a fi lansate în execuție. Ele sunt lansate în execuție automat, la interceptarea mesajului asociat! Totuşi, nu este nici o greşeală dacă le apelăm din program. Ele se vor executa în acest caz normal, ca şi orice altă funcție.

Întrebări şi probleme propuse

1. Implementați şi executați exemplele propuse în capitolul 3;
2. Care este deosebirea între modul în care tratează mesajele API şi MFC?
3. Implementați o funcție care afişează coordonatele prompterului la apăsarea butonului drept al mouse-lui.
   Indicații:
   - inserați în harta de mesaje macroul `ON_WM_RBUTTONDOWN();`
   - declarați funcția

```
afx_msg void OnRButtonDown(UINT nFlags,CPoint point);
```

   - implementați funcția ca mai jos:

```
void CPrimaFereastra::OnRButtonDown(UINT nFlags,CPoint point)
{
   CString text;
   text.Format(" Butonul a fost apasat la coordonatele
      [ x=%d , y=%d ]",point.x,point.y);
   MessageBox(text);
}
```

`CPoint` este o clasă având structura de date formată din variabilele publice `long x` şi `long y`. Este utilizată pentru a încapsula coordonatele unui punct de pe ecran. `CString` este o clasă care încapsulează şiruri de caractere. Şirurile pot fi formatate respectând tehnica de formatare din funcția `printf()`. Aceste clase vor fi descrise mai pe larg în capitolele următoare.