

## 5. Butoane, etichete, casete de editare...

Pentru că permit utilizarea imediată a controalelor oferite de MFC, vom utiliza în continuare ca interfață pentru programe, caseta de dialog. Crearea casetelor de dialog implică două etape:

- crearea machetei pentru caseta de dialog și adăugarea de controale;
- stabilirea legăturilor între caseta de dialog și controalele sale și respectiv clasele și funcțiile din codul C++;

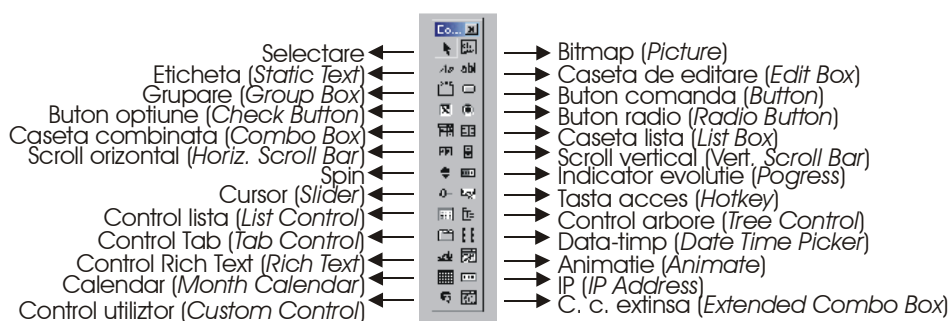


Figura 5.1 Controale disponibile

Pentru inserarea de controale în casetele de dialog, se utilizează bara cu instrumente de control, care poate fi afișată (dacă nu apare implicit) prin următoarele comenzi: **Tools -> Customize -> Toolbars -> Controls -> Close**. Bara de controale, împreună cu instrumentele oferite este prezentată în figura 5.1.

O altă bară de unelte pentru manipularea controalelor este bara **Dialog**, prin intermediul căreia controalele pot fi poziționate în interiorul casetei de dialog. Dacă bara nu apare implicit, ea poate fi afișată prin intermediul comenzilor: **Tools -> Customize -> Toolbars -> Dialog -> Close**. Această bară permite centrări, spațieri, alinieri, egalarea dimensiunilor controalelor, etc.

### 5.1 Macheta unei aplicații de tip casetă de dialog

La fel ca și în cursul precedent, crearea unei aplicații începe prin crearea unui proiect de tip **MFC AppWizard (exe)**. Fie *wiz2* numele acestui proiect. Se alege din nou opțiunea **Dialog Based**, iar la pasul 2 din 4 se poate realiza o primă personalizare a casetei de dialog, prin introducerea în caseta **Please a title for your dialog:** a titlului casetei. Fie acesta `Test Controale 1` (fig. 5.2).

Se parcurg apoi aceiași pași ca și în capitolul precedent, până la obținerea formei implicite a machetei:

- ștergerea etichetei statice `TODO: Place controls here`. Pentru aceasta se selectează controlul respectiv (prin apariția dreptunghiului de formatare și apăsarea tastei **Delete**).
- redenumirea butonului **OK** cu numele **Iesire**. Pentru aceasta, cu butonul selectat se activează meniul contextual, iar în caseta **Push Button Properties**, la **Caption** se scrie `Iesire`. Să ne reamintim, nu vom schimba identificatorul `IDOK`,

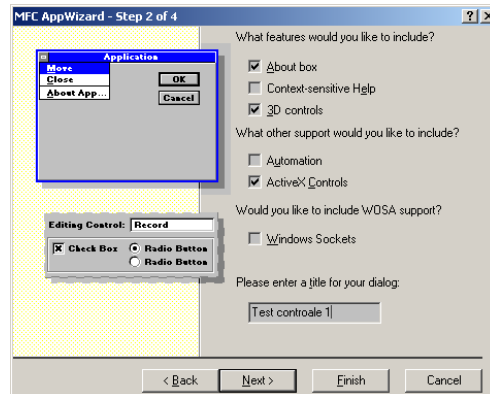


Figura 5.2 Se schimbă titlul...

pentru că altfel va trebui să scriem noi linia `CDialog::OnOK()` în funcția ce se execută la apăsarea butonului.

- se redenumesc butonul **Cancel** în **Anulare**.

În acest moment, se pot face modificări sau completări asupra casetei. Evident, pentru modificarea proprietăților casetei (sau componentelor sale), se lansează meniul contextual asociat casetei de dialog, alegându-se opțiunea **Properties** (click dreapta în interiorul casetei de dialog). Este util uneori, să se păstreze deschisă fereastra de opțiuni. Acest lucru se face prin schimbarea pictogramei din colțul stânga sus, prin click cu mouse-ul. Pictograma va fi schimbată într-o “pioneză”, iar caseta de dialog asociată va rămâne deschisă (figura 5.3).

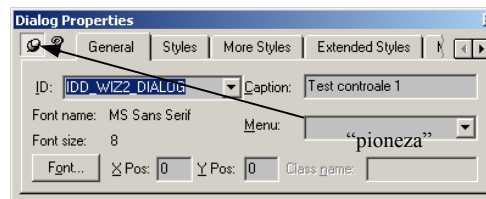


Figura 5.3 Proprietățile casetei de dialog

Caseta de proprietăți este caracteristică fiecărui tip de resursă în parte. Dacă tipul de resursă ar fi altul decât casetă de dialog, comenzile afișate de casetă ar fi altele. În cazul nostru, la opțiunea **General**, se pot schimba numele sub care resursa este identificată în program, denumirea casetei de dialog, aparența fonturilor, care implicit sunt MS Sans Serif 8. Schimbarea aparenței fonturilor se face prin apăsarea butonului **Font...** Dimensiunile casetei de dialog vor fi automat recalibrate în funcție de tipul fontului utilizat.

Casetele de editare **XPos** și **YPos** specifică poziția în care va apare inițial caseta de dialog, relativ la marginile ferestrei părinte. Valorile în aceste casete, vor avea ca efect afișarea casetei de dialog în mijlocul ferestrei.

Caseta de proprietăți conține de asemenea trei etichete (pagini) pentru definirea stilului casetei. Majoritatea opțiunilor de stil influențează alte opțiuni. De exemplu, deselectarea opțiunii **Title Bar** va avea ca efect nu numai înlăturarea barei de titlu, dar va dezactiva și opțiunile meniului de sistem și va elimina titlul. O reselectare ulterioară a acestei opțiuni de stil, va impune introducerea unui nou titlu. Ar fi bine să selectați și opțiunile **Minimize box** și **Maximize box** de la **Styles**, pentru a putea minimiza și respectiv maximiza caseta de dialog.

Pentru adăugarea de controale în caseta de dialog, este suficient să se realizeze click-and-drag asupra controlului dorit din bara **Controls**. Dimensionarea casetei se face prin selectarea ei (apariția dreptunghiului de formatare), “prinderea” cu mouse-ul a unuia din pătratele mici de pe margini (apariția unuia din simbolurile săgeată dublă), prin ținerea apăsată a butonului drept și deplasarea mouse-lui până la obținerea dimensiunii dorite. În mod similar se pot dimensiona și controalele introduse în caseta de dialog.

## 5.2 Adăugarea și alinierea controalelor

Să realizăm o casetă de dialog cu controalele din fig. 5.4, utilizând pentru controale identificatorii și etichetele din tabelul 5.1:

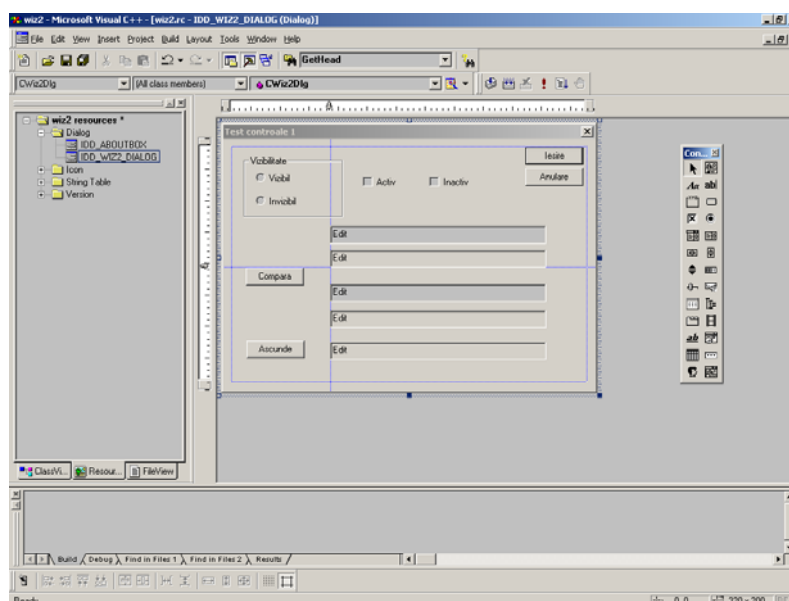


Figura 5.4 Alinierea controalelor în caseta de dialog

Tabelul 5.1

| Identificator  | Etichetă     |
|--|--------------|
| <b>Butoane de comandă (Button)</b>                     |              |
| IDC_COMPARA  | Compara      |
| IDC_ARATA  | Acunde       |
| <b>Butoane radio (Radio Button)</b>                    |              |
| IDC_VIZIBIL  | Vizibil      |
| IDC_INVIZIBIL  | Invizibil    |
| <b>Butoane de opțiune (Check Box)</b>                  |              |
| IDC_ACTIV  | Activ        |
| IDC_INACTIV  | Inactiv      |
| <b>Casete de editare (Edit Box) [.. de sus în jos]</b> |              |
| IDC_INTTEXT1   |              |
| IDC_OUTTEXT1   |              |
| IDC_INTTEXT2   |              |
| IDC_OUTTEXT2   |              |
| IDC_EGALITATE  |              |
| <b>Casete de grupare (GroupBox)</b>                    |              |
| IDC_STATIC   | Vizibilitate |

Nu trebuie să facem eforturi deosebite inițial pentru alinierea controalelor. Există mai multe facilități oferite de **WinApp** pentru alinierea acestora. În primul rând, putem afișa niște bare de ghidare (linii întrerupte de culoare albastră) pentru a marca poziționarea cursorilor. Acestea apar dacă apăsăm click stânga asupra barelor de măsură din exteriorul machetei. În poziția specificată, va apare automat o săgeată de la care pornește o bară de ghidare. Săgeata (și implicit bara de ghidare care pornește din ea) pot fi deplasate cu ajutorul mouse-ului, prin selectarea săgeții și deplasarea cu butonul din dreapta apăsător. Este o modalitate de poziționare, dar nu cea mai bună. În mod normal, pentru aranjarea controalelor, se utilizează opțiunea de meniu **L**ayout. Pentru utilizarea acestei opțiuni, se pot realiza următoarele:

- dimensionarea identică a mai multor controale - se selectează mai multe controale, prin selectarea unuia, ținerea apăsată a tastei **Shift** sau **Ctrl** și selectarea celorlalte controale. Apoi se utilizează opțiunea **M**ake **S**ame **S**ize și se precizează ce dimensiune se alege (lățime, înălțime sau ambele);
- alinierea – pentru aliniere se selectează grupul de controale ce dorim să fie aliniate și apoi cu **A**lign se alege tipul de aliniere dorită.

### 5.3 Stabilirea ordinii de selectare

Controalele dintr-o casetă pot fi accesate în serie. Această serie este cunoscută sub numele de *ordine de selectare*, reprezentând de fapt ordinea în care controalele primesc *input focus*-ul. La activarea unui control, pentru a selecta următorul control din serie, utilizatorul va apăsa tasta **Tab**, iar accesarea controlului precedent prin apăsarea combinației **Shift+Tab**. Un control care a primit *focus*-ul va fi marcat diferit. De exemplu, un buton va primi în interior un dreptunghi cu linie punctată, o casetă de editare va afișa în interior un prompter, etc. Etichetele statice nu pot primi focusul.

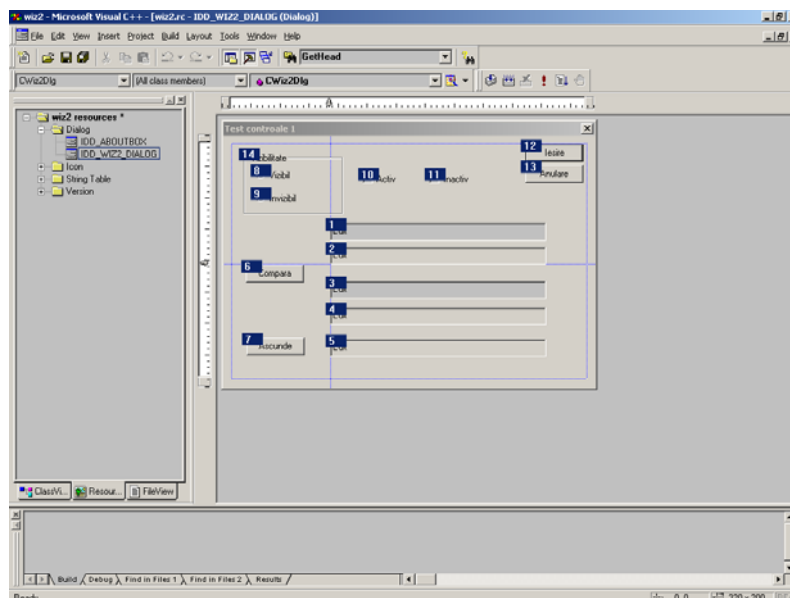


Figura 5.5 Stabilirea ordinii de selectare

Un control poate fi eliminat din ordinea de selectare, prin invalidarea proprietății **Tab Stop** din pagina **General** a casetei **Control Property**.

Stabilirea ordinii de selectare se face astfel:

- se selectează **L**ayout->**T**ab **O**rdor;
- controalele din caseta de dialog vor fi numerotate;

- se reface (dacă e nevoie) ordinea prin efectuarea de click pe fiecare control, în ordinea dorită;
- pentru testarea ordinii de selectare, se alege **Layout->Test**, sau se apasă **Ctrl+T**;  
Se propune stabilirea ordinii de selectare conform fig. 5.5.

## 5.4 Taste de acces

O tastă de acces (sau un *accelerator*) pentru un control se specifică prin inserarea simbolului & în eticheta controlului, înainte de caracterul care va avea rol de mnemonică. *Mnemonică* este un caracter subliniat, care poate fi utilizat ca o scurtătură pentru activarea controlului respectiv. De exemplu, un control cu eticheta `Test`, va avea scris în caseta **Caption** de la **Properties** eticheta `T&est`. El va putea fi activat, pe lângă mouse și de combinația de taste **Alt+E** sau **Alt+e**.

Există posibilitatea ca mai multe etichete să aibă aceeași mnemonică. În acest caz, la apăsarea mnemonice, va fi activat primul control considerând ordinea de selectare. Existența mnemonicelor multiple, se verifică cu opțiunea **Check Mnemonics** din meniul contextual. Cel mai frecvent, tastele de acces sunt utilizate pentru facilitarea accesului la intrările meniurilor.

## 5.5 Butoane, casete de editare...

Scopul în care sunt inserate controalele în caseta de dialog este de a realiza o anumită funcționalitate a interfeței. Înainte de a vedea cum se manipulează aceste butoane, să vedem ce a înscris **AppWizard** în fișierul `.rc` asociat proiectului, atunci când noi ne "jucăm" cu mouse-ul:

```
...
IDD_WIZ2_DIALOG DIALOGEX 0, 0, 320, 200
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_APPWINDOW
CAPTION "Test controale 1"
FONT 8, "MS Sans Serif"
BEGIN
    EDITTEXT IDC_INTXT1, 92, 69, 186, 14, ES_AUTOHSCROLL
    EDITTEXT IDC_OUTTXT1, 92, 88, 186, 14, ES_AUTOHSCROLL | ES_READONLY
    EDITTEXT IDC_INTXT2, 92, 115, 186, 14, ES_AUTOHSCROLL
    EDITTEXT IDC_OUTXT2, 92, 136, 186, 14, ES_AUTOHSCROLL | ES_READONLY
    EDITTEXT IDC_EGALITATE, 92, 161, 186, 14, ES_AUTOHSCROLL | ES_READONLY
    PUSHBUTTON "Compara", IDC_COMPARA, 19, 102, 50, 14
    PUSHBUTTON "Ascunde", IDC_ARATA, 20, 160, 50, 14
    CONTROL "Vizibil", IDC_VIZIBIL, "Button", BS_AUTORADIOBUTTON |
        WS_GROUP | WS_TABSTOP, 27, 26, 33, 10
    CONTROL "Invizibil", IDC_INVIZIBIL, "Button", BS_AUTORADIOBUTTON |
        WS_TABSTOP, 27, 44, 39, 10
    CONTROL "Activ", IDC_ACTIV, "Button", BS_AUTOCHECKBOX | WS_TABSTOP,
        120, 29, 32, 10
    CONTROL "Inactiv", IDC_INACTIV, "Button", BS_AUTOCHECKBOX |
        WS_TABSTOP, 177, 29, 37, 10
    DEFPUSHBUTTON "Iesire", IDOK, 260, 6, 50, 14
    PUSHBUTTON "Anulare", IDCANCEL, 260, 23, 50, 14
    GROUPBOX "Vizibilitate", IDC_STATIC, 17, 13, 86, 4986, 49
END
...
```

Vi se pare cunoscut, nu? Am scris și noi așa ceva, când ne “chinuiam” cu **WinAPI**. De fapt, **AppWizard** descrie resursele permanente în același fel cu **WinAPI**, dar o face el, nu trebuie să o facem noi!

Să revenim acum la controalele noastre. Orice control inserat într-o fereastră părinte, fie că aceasta este de clasă **CWnd**, fie că este de clasă **CDialog**, este de fapt un obiect de o anumită clasă. Marea majoritate a claselor încapsulate de **MFC** sunt derivate **public** direct sau indirect din clasa **CObject**. Câteva astfel de ierarhii sunt prezentate în fig. 5.6.

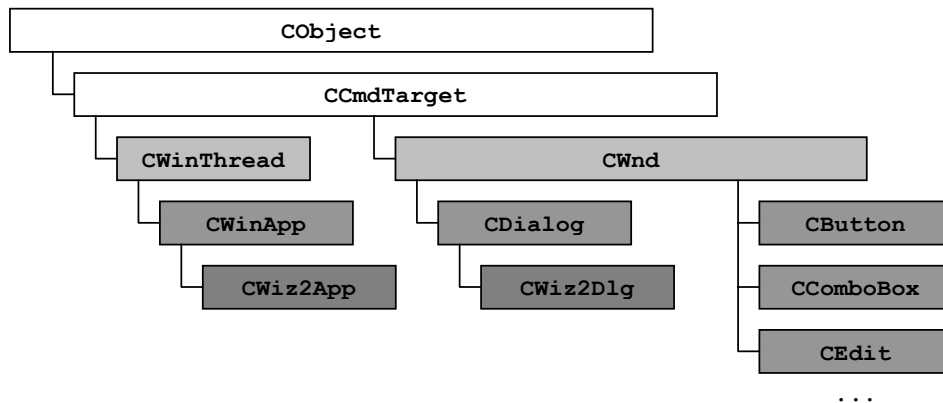


Figura 5.6 Câteva ierarhii de clase MFC

Pentru clasele proiectului nostru, arborele de derivare poate fi făcut vizibil astfel:

- în **ClassView** se selectează clasa dorită;
- se alege din meniul contextual **Base Classes...**;
- se acceptă reconstrucția proiectului, pentru a se putea construi ierarhia de clase; Astfel, arborele va fi făcut vizibil (fig. 5.7).

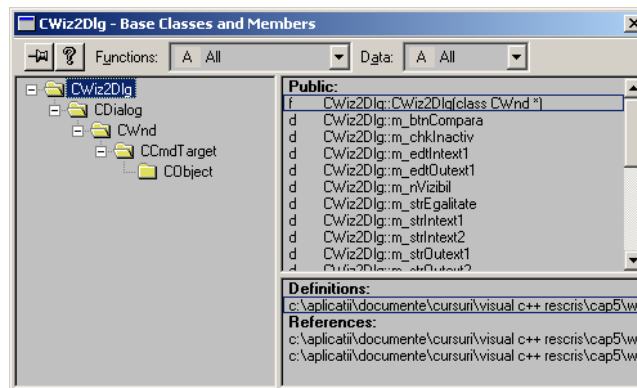


Figura 5.7 Ierarhia de clase pentru **CWiz2Dlg**

Cum putem manipula un obiect într-un program C++? Ca o “variabilă” de tipul clasei, sau prin intermediul unui pointer la obiect. Sunt tehnicile pe care le putem utiliza și pentru manipularea obiectelor noastre:

- pentru crearea unui pointer spre un obiect de interfață (control) se utilizează funcția

```
CWnd* CWnd::GetDlgItem(int nID) const;
```

Funcția primește ca parametru identificatorul obiectului spre care se construiește pointerul și returnează un pointer la clasa **CWnd**. Să ne reamintim că, un pointer la o clasă, poate pointa un obiect dintr-o clasă derivată public din aceasta. Deci, pentru

obiectele de clase derivate direct din `CWnd`, vom putea apela funcția fără o conversie explicită de tip. Pentru celelalte obiecte va fi nevoie însă de o conversie explicită de tip. Vom putea scrie de exemplu `GetDlgItem(IDC_ARATA)`; fără a avea eroare de compilare, dar dacă vom declara efectiv un pointer de clasa `CButton`, va trebui să scriem `CButton* pArata=(CButton*) GetDlgItem(IDC_ARATA)`; deoarece, să ne reamintim, conversia de la o clasă derivată public la clasa de bază se face explicit.

Oricărui control din caseta de dialog îi poate fi asociat un obiect de tipul clasei din care face parte controlul. **Prin intermediul acestui obiect vor putea fi apelate toate metodele puse la dispoziție de clasă și vor fi aplicate asupra controlului.** Obiectul va fi asociat ca și o variabilă ce este declarată în cadrul unei clase din proiect. Atunci când proiectul are mai multe clase din care sunt vizibili identificatorii controalelor (aceștia sunt mărimi globale) va trebui să fim atenți în ce clasă declarăm obiectul. În cazul nostru nu e o problemă, doar clasa `CWiz2Dlg` implementează o clasă fereastră. Asocierea unui obiect cu un control se face în **ClassWizard**:

- se apasă **Ctrl-W**, sau se alege **ClassWizard...** în meniul contextual din zona de cod a proiectului;
- se alege eticheta **Member Variables**; în acest moment, într-un proiect mai complex, se selectează clasa dorită în caseta combinată **Class name**;
- se observă că în lista **Control IDs**: sunt afișați identificatorii controalelor din caseta de dialog. Pentru a asocia unui control un obiect de tipul clasei, se selectează identificatorul în listă și se apasă butonul **Add Variable...**;

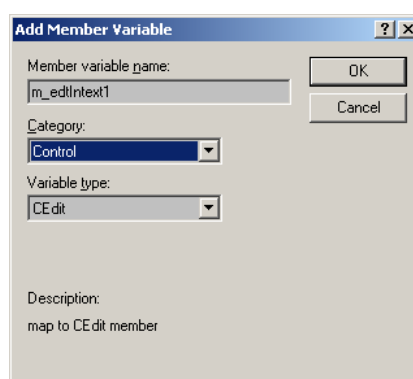


Figura 5. Variabilă **Control**

- va apare caseta de dialog **Add Member Variable**. Acum se selectează **Control** în caseta combinată **Category**: și se observă că în caseta combinată **Variable type**: a apărut numele clasei din care face parte controlul. În final, în caseta de editare **Member variable name**: se completează numele variabilei respective. Astfel, **am mapat peste control un obiect de aceeași clasă cu el**, prin intermediul căruia, după cum am mai spus, vom putea apela pentru control metodele clasei. Ca un sfat, marcați în numele variabilei tipul controlului pe care-l mapează, pentru a putea identifica ulterior mai ușor variabila respectivă. De exemplu, numiți butoanele `m_btn...`, casetele de editare `m_edt...`, casetele combinate `m_cb...` etc. Nu e obligatoriu, dar poate fi util. De fapt, puteți folosi orice nume pentru variabilele respective, cu condiția să fie un nume valid C++; De exemplu, în fig. 5.8 se vede că am mapat variabila `m_edtIntext1`. Această variabilă este mapată peste caseta de editare `IDC_INTEXT1`;
- se apasă **OK** pentru finalizarea mapării și se repetă operația pentru toate controalele peste care trebuie mapate obiecte. În exemplul nostru, vom face mapările prezentate în tabelul 5.2:

Tabelul 5.2

| Identificator | Categorie | Tip     | Nume variabilă |
|---------------|-----------|---------|----------------|
| IDC_ARATA     | Control   | CButton | m_btnArata     |
| IDC_INACTIV   | Control   | CButton | m_chkInactiv   |
| IDC_INTEXT1   | Control   | CEdit   | m_edtIntext1   |
| IDC_OUTTEXT1  | Control   | CEdit   | m_edtOutext1   |

Am văzut că la maparea unei variabile, în caseta combinată **Category**: opțiunea implicită este **Value**. Aceasta permite maparea peste control a unei alte variabile, de această dată de un tip predefinit sau de o clasă generală, *care va permite manipularea în interiorul programului a conținutului înscris în control*. De altfel, dacă veți deschide caseta combinată **Variable type**: veți vedea ce tipuri de variabile pot fi asociate controalelor. Nu toate controalele au această posibilitate, dar este evident spre exemplu, că în cazul unei casete de editare va trebui să preluăm în program valoarea pe care am tastat-o în aceasta, sau să afișăm în casetă un text din program. *Un control poate fi mapat simultan de un obiect de categorie Control și o variabilă Value, dar nu poate fi mapat de două obiecte sau două variabile!* Din nou, ca un sfat, specificați în numele variabilei tipul acesteia: `m_n...` pentru `int`, `long`, etc, `m_str...` pentru `CString`, etc.

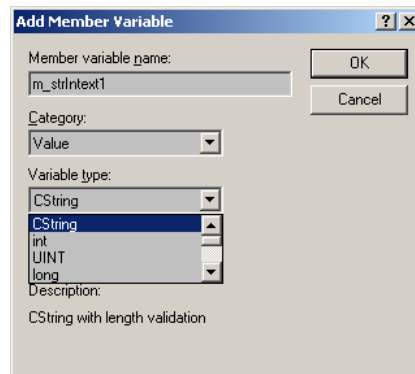


Figura 5.9 Variabilă Value

În fig. 5.9 este prezentat modul în care a fost mapată variabila `m_strIntext1` peste caseta de editare `IDC_INTEXT1`. Înainte de aface altceva, vom selecta pentru `IDC_VIZIBIL` opțiunile **Group** și **Tab stop** (meniu contextual, **Properties**, eticheta **General**, etc) iar pentru `IDC_INVIZIBIL` opțiunea **Tab stop**. Vom înțelege de ce am făcut asta în unul din paragrafele următoare. Se propune asocierea de variabile controalelor ca în tabelul de mai jos:

Tabelul 5.3

| Identificator | Categorie | Tip     | Nume variabilă |
|---------------|-----------|---------|----------------|
| IDC_EGALITATE | Value     | CString | m_strEgalitate |
| IDC_INTEXT2   | Value     | CString | m_strIntext2   |
| IDC_OUTTEXT2  | Value     | CString | m_strOutext2   |
| IDC_VIZIBIL   | Value     | int     | m_nVizibil     |

Vă întrebați probabil cum se face transferul de date între control și variabila care îi mapează conținutul? Acest mecanism va fi prezentat în detaliu într-unul din capitolele următoare. Ce trebuie să știm acum, este că transferul se face sub comanda funcției

```
BOOL CWnd::UpdateData(BOOL bSaveAndValidate=TRUE);
```



Dacă funcția este apelată cu parametrul `TRUE` (implicit), valoarea conținută în controale este încărcată în variabilele asociate. Pentru `FALSE`, valorile din variabile sunt afișate în controale (fig. 5.10).

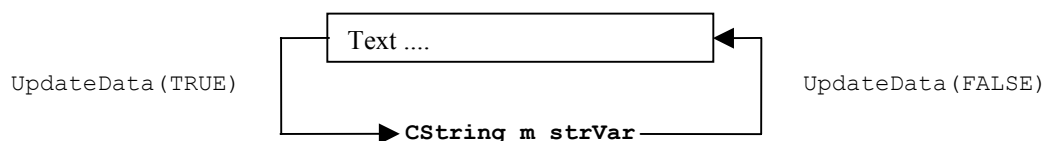


Figura 5.10 Transferul valorii între control și variabila mapată

Și acum, să recapitulăm:

- MFC manipulează două categorii de funcții: funcții obișnuite, adăugate claselor cu ajutorul meniului contextual, intrarea **Add Member Function...** și respectiv, funcții `afx_msg`, care răspund la mesaje, adăugate proiectului cu ajutorul **ClassWizard** (revezi capitolele 3 și 4);
- MFC manipulează 2 categorii de variabile:
  - **variabile obișnuite**, de un anumit tip sau clasă, adăugate clasei în care există cu ajutorul meniului contextual, intrarea **Add Member Variable...**;
  - **variabile mapate controalelor**, care se mapează cu **ClassWizard**. Acestea la rândul lor pot fi de două categorii:
    1. **variabile de categorie Value**, care preiau valoarea conținută în control pentru a putea fi manipulată de program și pentru care transferul datelor este controlat de funcția `UpdateData()`;
    2. **variabile de categorie Control**, care sunt obiecte de tipul clasei controlului și prin intermediul cărora se pot apela metodele clasei referitor la controlul respectiv;

## 5.6 Să vorbim despre butoane

MFC pune la dispoziția programatorilor 3 tipuri de butoane: butoane de comandă, butoane radio (Radio button) și casete de validare (Check box). Toate cele 3 tipuri sunt obiecte de clasa `CButton` și generează două mesaje: `BN_CLICKED` și `BN_DOUBLECLICKED`, acesta din urmă fiind generat de un dublu click cu mouse-ul asupra butonului. În realitate, acest ultim eveniment nu poate fi generat efectiv pentru butoanele de comandă. Cu toate că cele trei tipuri de butoane sunt obiecte de aceeași clasă, aspectul și funcționalitatea lor diferă:

- butoanele de comandă sunt acele butoane la a căror apăsare programul execută o singură comandă, implementată în cadrul programului prin intermediul unei funcții. Orice șablon casetă de dialog conține în forma inițială (și în marea majoritate a cazurilor și în forma finală) două butoane de comandă, respectiv **OK** și **Cancel**.
- butoanele radio sunt utilizate în marea majoritate a cazurilor pentru a marca în program acțiuni mutual exclusive. În general ele sunt utilizate ca un grup de butoane, un singur buton fiind validat (marcat cu punct în interior) la un moment dat. Gruparea butoanelor se face cum am arătat în paragraful 5.5: primul buton din grup are validate opțiunile **Group** și **Tab stop**, iar celelalte butoane vor avea validată doar opțiunea **Tab stop**. Grupul se termină când nu mai există butoane

radio, sau când apare un nou buton cu opțiunea **Group** validată, marcând începerea unui nou grup. Dacă ați făcut gruparea corect, veți observa în **ClassWizard**, că la eticheta **Member Variables**, în caseta **Control IDs**: apare doar identificatorul primului buton din grup. De asemenea, veți observa că dacă doriți să mapați acestui control (grup de controale) o variabilă de categorie *Value*, aceasta poate avea doar tipul *int*. Variabila (să presupunem că aceasta se numește *m\_var*) va prelua (sau va marca în grupul de controale) numărul de ordine al controlului marcat, considerat în ordinea de selectare, cu indicele primului control, din grup de valoarea 0 (fig. 5.11).

Implicit, la lansarea în execuție a programului, acest indice are valoarea -1,

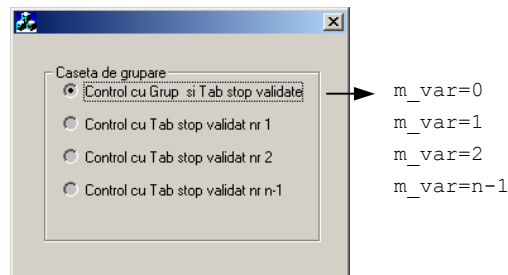


Figura 5.11 Variabila *m\_var* mapată la valori de la 0 la n-1

deci nici un buton nu va fi marcat. Pentru ca primul buton de exemplu să fie marcat, va trebui ca în constructorul clasei de dialog, sau mai bine, să ne reamintim, în funcția *OnInitDialog()* să scriem secvența:

```
m_var=0;
UpdateData(FALSE);
```

Ca o observație, caseta de grupare în care am inserat butoanele, are doar rol decorativ, ea nu implică automat definirea butoanelor ca un grup.

- casetele de validare, la rândul lor, marchează una sau mai multe opțiuni. Spre deosebire de butoanele radio, aceste opțiuni nu sunt mutual exclusive, putând fi validate simultan mai multe dintre ele. Utilizarea tipică a acestor butoane implică 2 funcții:

```
int CButton::GetCheck() const;
void CButton::SetCheck(int nCheck);
```

Prima funcție returnează 1 în situația în care caseta de validare pentru care este apelată este validată și 0 în caz contrar. A doua, validează caseta pentru care este apelată dacă primește ca parametru valoarea 1 și respectiv o invalidează dacă primește valoarea 0.

Să facem un exemplu. Vom utiliza următoarele funcții:

```
BOOL CWnd::ShowWindow(int nCmdShow);
BOOL CWnd::EnableWindow(BOOL bEnable=TRUE);
void CWnd::SetWindowText(LPCTSTR lpszString);
void CWnd::GetWindowText(CString& rString) const;
```

Prima funcție, să ne reamintim, stabilește starea de vizibilitate sau invizibilitate a unei ferestre. A doua activează sau inactivează o fereastră, a treia scrie textul

`lpString` în eticheta ferestrei (zona `Caption`), iar a patra preia textul din eticheta ferestrei și îl salvează în șirul `rString`. Oricare din aceste funcții apelate prin intermediul unui obiect ce mapează un control, sau a unui pointer la controlul respectiv, se vor aplica asupra controlului. Apelate simplu, se vor aplica asupra ferestrei principale a programului.

Să implementăm funcția care răspunde la mesajul `BN_CLICKED` generat de butonul `IDC_ARATA`, astfel încât, în funcție de valoarea variabilei `m_nVizibil`, să ascundă sau să afișeze casetele de editare. În același timp, vom schimba eticheta butonului, în concordanță cu acțiunea pe care o implementează. Cum facem asta? Sigur știți: apăsăm dublu click pe buton și acceptăm pentru funcție numele `OnArata()`:

```
void CWiz2Dlg::OnArata()
{
    // TODO: Add your control notification handler code here
    UpdateData();
    if (m_nVizibil)
    {
        m_edtIntext1.ShowWindow(FALSE);
        m_edtOutext1.ShowWindow(FALSE);
        GetDlgItem(IDC_INTEXT2)->ShowWindow(FALSE);
        GetDlgItem(IDC_OUTEXT2)->ShowWindow(FALSE);
        GetDlgItem(IDC_EGALITATE)->ShowWindow(FALSE);
        m_btnArata.SetWindowText("Arata");
    }
    else
    {
        m_edtIntext1.ShowWindow(TRUE);
        m_edtOutext1.ShowWindow(TRUE);
        GetDlgItem(IDC_INTEXT2)->ShowWindow(TRUE);
        GetDlgItem(IDC_OUTEXT2)->ShowWindow(TRUE);
        GetDlgItem(IDC_EGALITATE)->ShowWindow(TRUE);
        m_btnArata.SetWindowText("Ascunde");
    }
}
```

Întâi preluăm indicele butonului radio validat cu funcția `UpdateData()`. Dacă valoarea `m_nVizibil` este 1, înseamnă că am selectat butonul radio `IDC_INVIZIBIL` și ascundem controalele. De asemenea, schimbăm eticheta butonului `IDC_ARATA` în "Arata". În caz contrar (`m_nVizibil` poate lua în cazul nostru doar valorile 0 și 1. Dacă variabila asociată grupului de butoane radio poate lua mai multe valori, selecția acestora se face cu un selector `switch`), afișăm controalele și aplicăm butonului `IDC_ARATA` eticheta "Ascunde".

Observați că pentru unele controale funcțiile au fost apelate prin intermediul obiectelor mapate în **ClassWizard**, iar pentru unele prin intermediul pointerilor generați de funcția `GetDlgItem()`. Am ales această cale mixtă, din rațiuni pur didactice. Uzual, pentru păstrarea caracterului unitar al programului, toate controalele vor fi tratate la fel, fie prin intermediul obiectelor mapate, fie prin intermediul pointerilor.

Cum inițial controalele sunt vizibile, va trebui să marcăm implicit butonul radio `IDC_VIZIBIL`. Pentru aceasta, vom adăuga funcției `OnInitDialog()` liniile:

```

BOOL CWiz2Dlg::OnInitDialog()
{
    ...
    // TODO: Add extra initialization here
    m_nVizibil=0;
    UpdateData(FALSE);
    return TRUE; // return TRUE ...
}

```

După cum bănuiați, am dat valoarea 0 variabilei `m_nVizibil` și am transmis-o grupului de controale mapat de aceasta, prin apelul `UpdateData(FALSE)`.

Să implementăm acum activarea și inactivarea butoanelor de comandă, prin validarea sau invalidarea uneia din casetele de validare. Deoarece butoanele de comandă nu pot fi simultan active și inactive, la validarea uneia din casete va trebui să o invalidăm pe cealaltă.

Validarea unui buton de tip casetă de validare produce mesajul `BN_CLICKED`. Vom implementa deci funcțiile `OnActiv()` și `OnInactiv()`, care răspund mesajelor `BN_CLICKED` generate de butoanele `IDC_ACTIV` și respectiv `IDC_INACTIV`. Cum? Din nou dublu click pe buton și așa mai departe.

```

void CWiz2Dlg::OnActiv()
{
    // TODO: Add your control notification handler code here
    if (m_chkInactiv.GetCheck())
        m_chkInactiv.SetCheck(0);
    m_btnArata.EnableWindow(TRUE);
    GetDlgItem(IDC_COMPARA)->EnableWindow(TRUE);
}

void CWiz2Dlg::OnInactiv()
{
    // TODO: Add your control notification handler code here
    CButton *pActiv=(CButton*) GetDlgItem(IDC_ACTIV);
    if (pActiv->GetCheck())
        (pActiv->SetCheck(0));
    m_btnArata.EnableWindow(FALSE);
    GetDlgItem(IDC_COMPARA)->EnableWindow(FALSE);
}

```

În prima funcție, am testat prin intermediul obiectului mapat, starea de validare a butonului `IDC_INACTIV`. Dacă acesta este validat, îl invalidăm. Apoi, activăm cele 2 butoane de comandă. În a doua funcție, deoarece pentru butonul `IDC_ACTIV` nu există mapat nici un obiect, am fost nevoiți să creem un pointer spre acesta. Cum butonul este un obiect de clasă `CButton` și funcția `GetDlgItem()` returnează un pointer `CWnd*`, am fost nevoiți să facem o conversie explicită de tip. Apoi, am invalidat dacă e nevoie butonul `IDC_ACTIV` și am invalidat butoanele de comandă. Din nou, în scop didactic, am apelat controalele în mod diferit, prin obiecte mapate, respectiv prin pointeri.

Deoarece inițial butoanele de comandă sunt active, va trebui să validăm implicit butonul `IDC_ACTIV`. Pentru aceasta, în funcția `OnInitDialog()` vom scrie:

```

BOOL CWiz2Dlg::OnInitDialog()
{
    ...

```

```

UpdateData (FALSE);
CButton *pActiv=(CButton*) GetDlgItem(IDC_ACTIV);
pActiv->SetCheck(1);
return TRUE; // return TRUE ...
}

```

Ca o observație, puteam foarte bine să asociem funcții mesajului BN\_CLICKED produs de butoanele radio IDC\_VIZIBIL și respectiv IDC\_INVIZIBIL (dublu click pe butoane), situație în care butonul de comandă IDC\_ARATĂ este oarecum inutil. Conținutul acestor funcții ar fi:

```

void CWiz2Dlg::OnVizibil()
{
    // TODO: Add your control notification handler code here
    m_edtIntext1.ShowWindow(TRUE);
    m_edtOutext1.ShowWindow(TRUE);
    GetDlgItem(IDC_INTEXT2)->ShowWindow(TRUE);
    GetDlgItem(IDC_OUTEXT2)->ShowWindow(TRUE);
    GetDlgItem(IDC_EGALITATE)->ShowWindow(TRUE);
    m_btnArata.SetWindowText("Ascunde");
}

void CWiz2Dlg::OnInvizibil()
{
    // TODO: Add your control notification handler code here
    m_edtIntext1.ShowWindow(FALSE);
    m_edtOutext1.ShowWindow(FALSE);
    GetDlgItem(IDC_INTEXT2)->ShowWindow(FALSE);
    GetDlgItem(IDC_OUTEXT2)->ShowWindow(FALSE);
    GetDlgItem(IDC_EGALITATE)->ShowWindow(FALSE);
    m_btnArata.SetWindowText("Arata");
}

```

Dacă am dori și actualizarea conținutului variabilei `m_nVizibil` pentru utilizare în alte funcții, ar trebui să apelăm la începutul fiecărei funcții `UpdateData()`.

## 5.7 ... și câte ceva despre casete de editare

Interfața de intrare cel mai des utilizată pentru aplicațiile Windows este caseta de editare. Controalele de tip casetă de editare sunt utilizate ca și controale de intrare, pentru recepționarea textului introdus, precum și ca și controale de ieșire, pentru afișarea informațiilor. Prin efectuarea unui click cu mouse-ul în interiorul casetei, aceasta va prelua implicit focusul, putând fi introdus textul. Asupra textului introdus, se pot efectua prin intermediul meniului contextual (click butonul drept al mouse-lui) următoarele operații: **Undo**, **Cut**, **Copy**, **Paste**, **Delete** și **Select All**.

La definirea unei casete de dialog, sunt accesibile mai multe opțiuni de stil, cum ar fi:

- **Password**: maschează textul introdus, prin înlocuirea caracterelor cu “\*”;
- **Uppercase**, **Lowercase**: transformă automat textul introdus în majuscule, respectiv minuscule;
- **Number**: permite introducerea doar a caracterelor numerice;
- **Read-only**: utilizează caseta de editare doar pentru afișare. Ea este colorată la culoarea de fond a casetei de dialog în care apare;

La fel ca și în cazul butoanelor, în cazul casetelor de editare pot fi asociate funcții de tratare a mesajelor caracteristice. Mesajele generate de diferite evenimente produse asupra casetelor de editare sunt prezentate în tabelul 5.4:

Tabelul 5.4

| Mesaj        | Descriere   |
|--------------|---|
| EN_CHANGE    | trimis pentru a informa că textul s-a modificat după afișarea acestuia;                           |
| EN_UPDATE    | trimis pentru a specifica faptul că în caseta de editare urmează să fie afișat un text modificat; |
| EN_SETFOCUS  | trimis când controlul primește focusul;   |
| EN_KILLFOCUS | trimis atunci când controlul pierde focusul;  |
| EN_MAXTEXT   | trimis când textul depășește lungimea permisă și este trunchiat;                                  |
| EN_HSCROLL   | trimis când se efectuează click pe bara de derulare orizontală, pentru casetele multi-linie;      |
| EN_VSCROLL   | trimis când se efectuează click pe bara de derulare verticală, pentru casetele multi-linie;       |
| EN_ERRSPACE  | trimis în cazul în care controlul nu poate alocă memorie;   |

Pentru a obține interfața din fig. 5.4, vom seta opțiunile de stil **Read-only** (click dreapta, **Properties**, **Styles**, etc) pentru casetele IDC\_OUTEXT1, IDC\_OUTEXT2 și IDC\_EGALITATE și respectiv **Password** pentru IDC\_INTEXT1 și IDC\_INTEXT2.

Dorim ca la introducerea unui text în casetele IDC\_INTEXT1 și IDC\_INTEXT2, acesta să apară în clar, simultan cu tastarea fiecărei litere, în casetele IDC\_OUTEXT1, IDC\_OUTEXT2. Mesajul generat la tastarea fiecărui caracter în caseta de editare este EN\_CHANGE. Deci acestui mesaj va trebui să îi asociem funcții. Cum? Ca și la butoane: **ClassWizard** eticheta **MessageMaps**, se selectează IDC\_INTEXT1 în caseta **Object IDs**., mesajul EN\_CHANGE în caseta **Messages**, se apasă butoanele **Add Function** și **Edit Code**, ș.a.m.d. Funcțiile sunt:

```
void CWiz2Dlg::OnChangeIntext1()
{
    // TODO: If this is a RICHEDIT control...
    // TODO: Add your control notification handler code here
    CString strIntext1;
    m_edtIntext1.GetWindowText(strIntext1);
    m_edtOutext1.SetWindowText(strIntext1);
}

void CWiz2Dlg::OnChangeIntext2()
{
    // TODO: If this is a RICHEDIT control...
    // TODO: Add your control notification handler code here
    UpdateData();
    m_strOutext2=m_strIntext2;
    UpdateData(FALSE);
}
```

Deoarece primelor două casete de editare le-am mapat doar variabile de categorie Control, în funcția OnChangeIntext1() a trebuit să citim și să afișăm textul cu funcțiile GetWindowText() și respectiv SetWindowText(). În funcția OnChangeIntext2() am înscris întâi în variabilele de categorie Value conținutul casetelor (UpdateData()), am egalat variabilele și apoi le-am afișat conținutul în controale (UpdateData(FALSE)).

Vom implementa acum funcția care răspunde mesajului `BN_CLICKED` generat de butonul `IDC_COMPARA` astfel încât: să compare textele

- compară textele din casetele de editare `IDC_INTEXT1` și `IDC_INTEXT2`;
- dacă cele 2 casete sunt goale, afișează un mesaj în bara de titlu a casetei de dialog;
- dacă cel puțin una din casete conține text, afișează astfel:
  - textul mai mare lungime în bara de titlu a casetei de dialog;
  - dacă cele 2 texte sunt identice, afișează în caseta de editare `IDC_EGALITATE`;

Cum casetele conțin texte și acestea sunt manipulate în program de clasa `CString`, să amintim câteva metode puse la dispoziție de această clasă:

- operatori:
  - `=` atribuie o valoare (un text) șirului;
  - `+` returnează un nou obiect `CString` în care sunt concatenați cei doi operanzi `CString` implicați;
  - `+=` concatenează șirul din dreapta operatorului la sfârșitul șirului din stânga operatorului;
  - `<`, `>`, `<=`, `>=`, `!=`, `==` operatori de comparare;
- câteva metode mai des utilizate:
  - `BOOL IsEmpty()`; - returnează `TRUE` dacă obiectul `CString` este gol și `FALSE` în caz contrar;
  - `void Empty()`; - golește obiectul `CString`;
  - `int GetLength()`; - returnează lungimea șirului conținut de obiectul `CString`;
  - `CString Mid(int nFirst, int nCount)`; - returnează un subșir conținând `nCount` caractere din obiectul `CString`, începând cu caracterul de indice `nFirst` (primul caracter dintr-un `CString` are indicele 0);
  - `CString Left(int nCount)`; - returnează un subșir care conține primele `nCount` caractere din obiectul `CString`;
  - `CString Right(int nCount)`; - returnează un subșir care conține ultimele `nCount` caractere din obiectul `CString`;
  - `void MakeUpper()`; - transformă toate caracterele din obiectul `CString` în majuscule;
  - `void MakeLower()`; - transformă toate caracterele din obiectul `CString` în minuscule;
  - `void MakeReverse()`; - transformă toate caracterele majuscule din obiectul `CString` în minuscule, respectiv minusculele în majuscule;
  - `void Format(LPCTSTR lpszFormat, ...)`; - formatează un șir de caractere, după aceleași reguli de la funcția `printf()`;
  - `int Find(LPCTSTR lpszSub)`; - caută (de la început spre sfârșit) subșirul `lpszSub` în obiectul `CString`. Dacă îl găsește returnează indicele în obiectul `CString` la care a găsit prima dată subșirul. Dacă subșirul nu există, returnează `-1`;

Funcția `OnCompara()` va avea implementarea:

```
void CWiz2Dlg::OnCompara()
{
    // TODO: Add your control notification handler code here
    UpdateData();
    if (strIntext1.IsEmpty() && strIntext2.IsEmpty())
```

```

        SetWindowText(" Nu am ce scrie!");
    else
        if (strIntext1.GetLength()>m_strIntext2.GetLength())
            Scrie(IDC_INTEXT1);
        else
            if (strIntext1.GetLength()<m_strIntext2.GetLength())
                Scrie(IDC_INTEXT2);
            else
            {
                m_strEgalitate=strIntext1;
                UpdateData(FALSE);
            }
    }
}

```

Funcția nu necesită explicații suplimentare, nu utilizează decât tehnici descrise anterior. Trebuie totuși observat că variabila `CString strIntext1` declarată în funcția `OnChangeIntext1()` este o variabilă locală. Pentru a o putea utiliza și în funcția `OnCompara()` va trebui să îi ștergem declarația din `OnChangeIntext1` și să o adăugăm ca și o variabilă membru a clasei `CWiz2Dlg`. De asemenea, am apelat funcția `Scrie(UINT ID)`, care trebuie adăugată și ea clasei `CWiz2Dlg`, ca și o funcție membru. Funcția afișează în bara de titlu a casetei de dialog textul conținut de controlul al cărui identificator îl primește ca parametru de intrare:

```

void CWiz2Dlg::Scrie(UINT ID)
{
    CString strTemp;
    CEdit* pEdit=(CEdit*)GetDlgItem(ID);
    pEdit->GetWindowText(strTemp);
    SetWindowText(strTemp);
}

```

## 5.8 Să complicăm lucrurile... Subclasarea unei casete de editare

Există situații, în care textele introduse în casetele de editare, trebuie să satisfacă anumite condiții suplimentare. Astfel, în momentul introducerii lor, asupra textelor trebuie făcute o serie de validări. În acest sens, este nevoie de o extindere a funcționalității casetelor de editare, realizându-se o subclasare a clasei `CEdit`. Subclasarea este în esență construirea unei noi clase, derivată public sau privat din clasa `CEdit` și a cărei obiecte, pe lângă atributele moștenite de clasa de bază, să aibă asociate noi atribute, care să rezolve funcțiile pentru care au fost create.

Să adăugăm de exemplu o casetă de editare în caseta de dialog, cu identificatorul `IDC_DERIVATA`, ca în fig. 5.12. Să presupunem că această casetă acceptă doar caractere alfanumerice (litere) și că le afișează automat ca majuscule. De asemenea, fiecare caracter este urmat de un spațiu. Va trebui să implementăm o nouă clasă, numită `CEditNou`, derivată public din clasa `CEdit`, care va realiza introducerea caracterelor în forma dorită. Pentru aceasta, vom parcurge următorii pași:

- lansăm **ClassWizard** la eticheta **Message Maps**;
- selectăm clasa `CWiz2Dlg` în caseta combinată **Class name**;
- apăsăm butonul **Add Class...** și apoi **New...**;
- va apare caseta de dialog **NewClass** (fig. 5.13), care permite descrierea noii clase;
- în această casetă, completăm în caseta de editare **Name**: numele clasei: `CEditNou`;



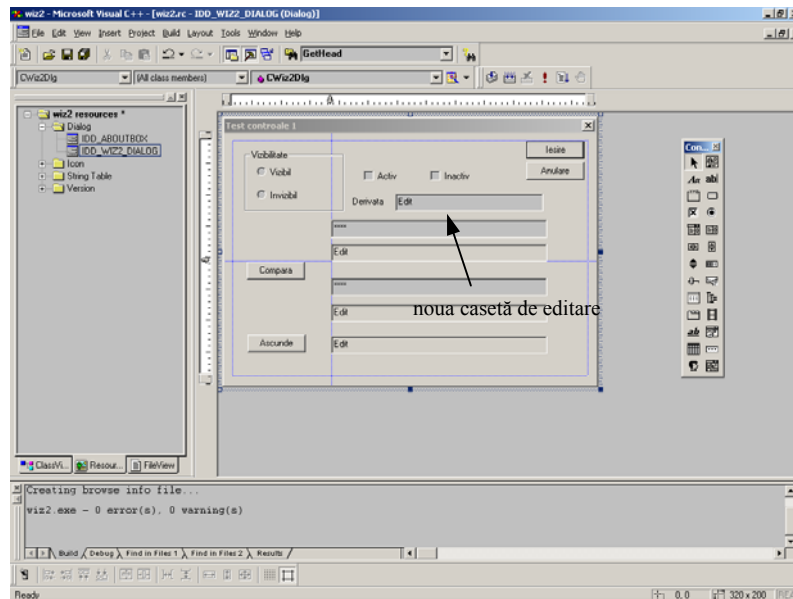


Figura 5.12 Casetă de editare de clasă derivată

- observăm că în caseta de editare **FileName:** a fost tipărit `CEditNou.cpp`. Aceasta înseamnă că proiectului îi sunt adăugate fișierele `EditNou.h` și `EditNou.cpp`, care declară și implementează noua clasă;
- va trebui să precizăm acum cine este superclasa noii clase create. Pentru aceasta, în caseta combinată **Base class:** alegem `CEdit`;
- apăsăm apoi **OK** și totul e gata!

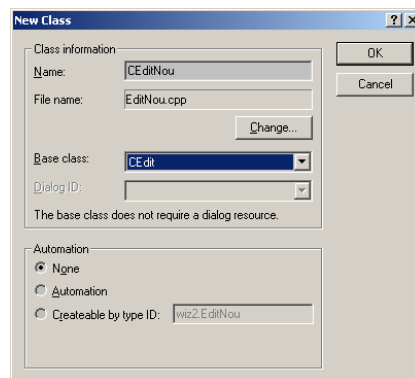
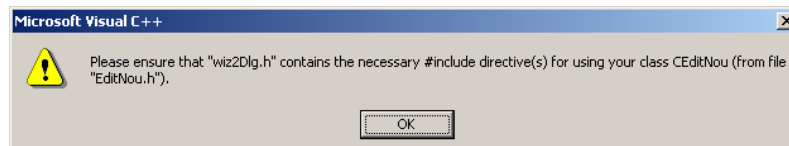


Figura 5.13 Definim noua clasă

Vom mapa acum casetei de editare `IDC_DERIVATA` un obiect de clasă `CEditNou`. Tot în **ClassWizard**, la eticheta **MemberVariables** va trebui să executăm următoarele:

Figura 5.14 Să nu uităm să includem `EditNou.h`!

- în caseta de editare **Class name:** selectăm `CWiz2Dlg`;
- în lista **Object IDs:** selectăm `IDC_DERIVATA`;
- apăsăm butonul **Add Variable**;

- în caseta de dialog **Add Member Variable** definim variabila `m_edtNou`, de categorie `Control` și tip `CEditNou`;
- apăsăm **OK**. Totul e gata, dar **AppWizard** sesizează că a fost mapat un obiect de o clasă nouă, pentru care el nu a inserat o directivă `#include` în fișierul *CWiz2Dlg.cpp* la crearea proiectului. Ca să ne atenționeze, va afișa mesajul din fig. 5.14.
- apăsăm **OK** pentru a ieși din **ClassWizard**;

Va trebui deci să includem fișierul care declară noua clasă. O facem la începutul fișierului *wiz2Dlg.h*, după cum am fost atenționați de **AppWizard**:

```
// CWiz2Dlg dialog
#include "EditNou.h"
class CWiz2Dlg : public CDialog
{
// Construction
public:
    void Scribe(UINT ID);...
```

Urmează să implementăm noua funcționalitate a clasei. Deoarece orice caracter tastat va trebui testat și modificat, va trebui să interceptăm pentru noua clasă mesajul Windows `WM_CHAR`. Cum facem asta? Să ne reamintim:

- în **ClassWizard**, selectăm eticheta **Message Maps**;
- în caseta combinată **Class name**: se selectează `CEditNou`. Astfel, în lista **Object IDs**: va apare doar numele clasei, fără alți identificatori;
- se selectează numele clasei în **Object IDs**, iar în lista **Messages**: se selectează `WM_CHAR`;
- se apasă butoanele **Add Function** și **Edit Code**;

Se poate observa că **ClassWizard** a creat funcția `CEditNou::OnChar()` astfel încât aceasta primește următorii parametri:

- `UINT nChar` – conține codul UNICODE sau virtual (în cazul nostru ASCII) al tastei apăsate;
- `UINT nReptCnt` – conține numărul de caractere transmise în timpul în care tasta a fost apăsată;
- `UINT nFlags` – informații asupra tastei apăsate, pe 32 de biți, dintre care cea mai importantă ar fi codul de scanare a tastei; Vom folosi următoarele funcții:
- `int isalpha(int c)` – returnează 0 dacă `c` nu este codul ASCII al unui caracter alfanumeric (literă) și respectiv o valoare diferită de 0 în caz contrar;
- `int islower(int c)` – returnează 0 dacă `c` nu este un caracter literă minusculă și respectiv o valoare diferită de 0 în caz contrar;
- `CWnd::DefWindowProc(UINT message, WPARAM wParam, LPARAM lParam)` – apelează procedura fereastră implicită, transmițându-i mesajul `message` și parametrii asociați `wParam` și `lParam` (revezi eventual `WndProc`, cap. 2);

Implementarea funcției `OnChar()` va fi:

```
void CEditNou::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO: Add your message handler code here and/or call default
```

```

if (isalpha(nChar))
{
    if(islower(nChar)) nChar -=32;
    DefWindowProc(WM_CHAR, nChar,0);
    nChar=' ';
    DefWindowProc(WM_CHAR, nChar,0);
}
if (nChar == VK_BACK) {
    DefWindowProc(WM_CHAR, nChar,0);
    DefWindowProc(WM_CHAR, nChar,0);
}
}

```

Funcția testează dacă tasta apăsată a transmis un caracter alfanumeric. Dacă da, dacă este cazul, transformă minuscula în majusculă. Să ne amintim că 'a'-'A'=32. Este apoi generat mesajul WM\_CHAR și trimis procedurii fereastră implicite, pentru a afișa caracterul. Să ne reamănim că mesajul WM\_CHAR transmite codul ASCII în parametrul wParam (vezi cap. 2). După afișarea fiecărei majuscule, mai apelăm o dată procedura fereastră, pentru a afișa în spațiu.

Un caz special este apăsarea tastei **Backspace**, pentru care va trebui să ștergem 2 caractere: spațiul și litera. De aceea este apelată de 2 ori procedura fereastră implicită, transmițându-i-se caracterul virtual VK\_BACK.

Gata! Am terminat! Caseta noastră de editare va afișa textul ca în fig. 5.15.

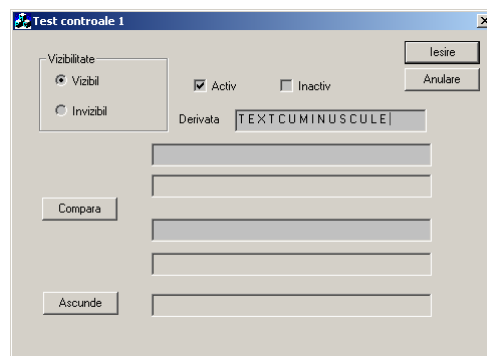


Figura 5.15 Caseta de clasa **CEditNou**

## Întrebări și probleme propuse

1. Implementați și executați toate exemplele propuse în capitolul 5;
2. Care sunt categoriile de variabile mapate controalelor? Cum se mapează aceste variabile?
3. Ce efect are anularea proprietății **Tab stop** pentru un control?
4. Cum se definește un grup de controale? Cât ține un astfel de grup?
5. Modificați programul astfel încât acțiunea implementată de butonul IDC\_COMPARA să se facă la terminarea tastării în casetele IDC\_INTEXT1 și IDC\_INTEXT2 (la producerea evenimentului EN\_KILLFOCUS), doar dacă există text tastat în ambele casete de editare.
6. Modificați funcția OnChar() pentru clasa CEditNou ca mai jos:

```

void CEditNou::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{
    // TODO: Add your message handler code here and/or call default
    if (isalpha(nChar))

```

```
{
    if(islower(nChar)) nChar -=32;
    DefWindowProc(WM_CHAR, nChar,0);
    nChar=' ';
    DefWindowProc(WM_CHAR, nChar,0);
    nChar='_';
    DefWindowProc(WM_CHAR, nChar,0);
}
if (nChar == VK_BACK) {
    DefWindowProc(WM_CHAR, nChar,0);
    DefWindowProc(WM_CHAR, nChar,0);
}
}
```

Ce efect au asupra textului noile instrucțiuni introduse? Există o mică problemă. Care a aceasta?