2. Laborator

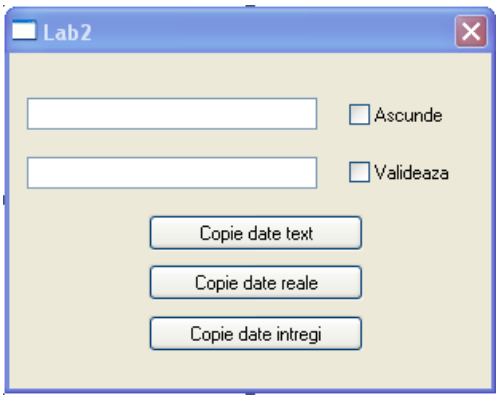

Figura 2.1. Interfața grafică a programului

1.1. Scopul laboratorului

În cadrul acestui laborator veți învăța elementele fundamentale ale realizării unui proiect în mediul Visual Studio. **Elemente pe care le veți utiliza în toate laboratoarele ce vor urma și care vor fi testate în cadrul examenului**. Astfel, aceste elemente sunt:

1. **Realizarea unui proiect**;
2. Modalitatea de programare vizuală – construirea interfeței cu utilizatorul;
3. **Asocierea unei funcții, diferitelor elemente de pe interfața grafică a programului, care să trateze diferitele mesaje (acțiuni) realizate de utilizatorul interfeței** (click, dublu click, click dreapta, etc.);
4. **Interogarea sau influențarea stării unui element de pe interfața grafică** – prin intermediul unei variabile (mai exact a unei clase) asociată cu respectivul element.

1.2. Cerințe

Realizați un program care sa aibă o interfață grafică similară cu cea prezentată în **Figura 2.1**. Programul:

1. Va avea 2 elemente de tip EDIT BOX, trei butoane (cu următoarele mesaje pe ele: „Copie date text”, „Copie date reale” și „Copie date întregi”) și două CHECK BOX-uri;
2. CHECK BOX-urile vor avea funcțiile de ascundere/afișare (elementul va dispare și va apare pe interfața grafică funcție de starea elementului – bifat/nebifat) și, respectiv, de validare/invalidare (atunci când butonul este invalidat este prezent pe interfața grafică dar utilizatorul acesteia nu poate să îl utilizeze) a butonul „Copie date text”.
3. La apăsarea butonului „Copie date text” valoarea de tip text din primul *edit box* va apare in cel de al doilea însoțită de un text oarecare (gen: „Textul este:”);

4. La apăsarea butonului „Copie date reale” valoarea de tip real din primul *edit box* va fi înmulțită cu 2 iar rezultatul va apare in cel de al doilea *edit box* însoțită de un text oarecare (gen: „Dublul valorii reale este:”);
5. La apăsarea butonului „Copie date întregi” valoarea întreagă din primul *edit box* va fi înmulțită cu 3 iar rezultatul va apare in cel de al doilea *edit box* însoțită de un text oarecare (gen: „Triplul valorii întregi este:”);

1.3. Concepte de abordare a dezvoltării de programe în mediul Visual Studio

Pentru o scriere corectă a programelor (în orice tip de mediu de dezvoltare și în orice limbaj) vă rog țineți cont de următoarele aspecte:

1. Obiectivul principal este cel al funcționalității corecte a programelor și ulterior a obținerii unei interfețe grafice frumoase și ergonomice. Din acest motiv, mai ales acum că sunteți le început, verificați în fiecare pas (cu fiecare nouă funcționalitate introdusă) rularea corectă a programelor. scrise de dvs. prin compilarea, *link*-editare și rularea acestora pe sistemul *embedded*.
2. În momentul existenței a 2 sau mai multe greșeli simultane rezolvați întotdeauna prima eroare apărută. O singură greșeală poate genera și 17 erori de compilare – tratați întotdeauna prima eroare și le veți rezolva și pe celelalte legate/corelate cu ea.
3. Utilizați *help*-ul mediului de dezvoltare întotdeauna când aveți nevoie (de exemplu, aveți dubii sau nu înțelegeți argumentele unei anumite funcții). Pentru lansarea *help*-ului mediului Visual Studio urmații pașii: **Help → Search**.
4. În cazul dificultăților de înțelegere a documentației oficiale, utilizați informația existentă pe internet dar nu fără a o trece prin filtrul personal și a o implementa și testa în diferite abordări – în acest mod atunci când o veți aplica în alt context să o puteți face fără probleme.
5. Creați-vă un fișier text in care să puneți bucăți de cod specifice, care rezolvă anumite probleme (de exemplu, cum se preia o valoare reală dintr-un EDIT BOX și cum se salvează într-o variabilă de tip double) în acest mod la scrierea următorului program vă va fi foarte ușor data viitoare când vă întâlniți cu o problemă similară.

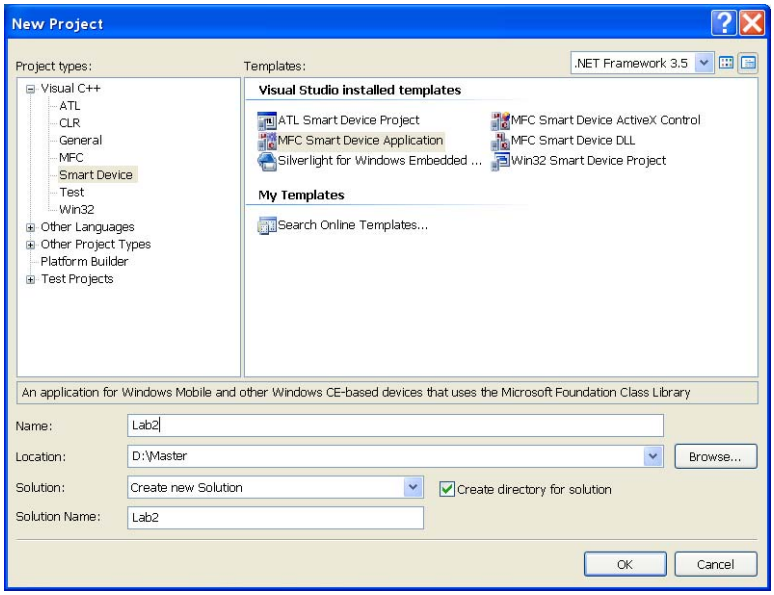

Figura 2.2. Prima fereastră de configurare a viitorului program ce se va dezvolta

1.4. Pașii de bază necesari pentru dezvoltarea unei aplicații

Pentru dezvoltarea unei aplicații generice (în cazul nostru particular, această aplicație este particularizată pe cerințele laboratorului actual – cerințe prezentate anterior) în mediul Visual Studio urmații pașii:

1. Încărcați imaginea SO pe sistemul *embedded* de dezvoltare – printr-o **primă instanță** a mediului Visual Studio;
2. Lansați în execuție mediul de dezvoltare Microsoft Visual Studio, **cea de a doua instanță**, în care veți dezvolta programul;
3. Creați structura cadru a programului dumneavoastră:
   a. **File → New → Project**

      sau apăsați combinația de taste

      Ctrl + **Shift** + **N**

   b. În noua fereastră deschisă se selectează:

      - În câmpul **Project Type** se alege limbajul ce va fi folosit **Visual C++** cu subopțiunea **Smart Devices**;
      - iar din potențialele *template*-uri existente se alege: **MFC Smart Device Application**; Librăria MFC conține o colecție de clase prin intermediul cărora se oferă utilizatorilor capacitatea de a interacționa într-un mod simplificat cu diferite elemente (precum ferestre, meniuri, butoane etc.) într-un mod cât mai simplu utilizatorului – scriind o cantitate cât mai mică de cod pentru respectivele funcționalități;
      - Puneți în câmpul **Name** numele proiectului (de exemplu: Lab2);
      - Directorul de lucru va fi pe discul D și se numește **Embedded**;
      - Apăsați **OK**.

   c. În următoarea fereastră de configurare, **Figura 2.3**, va sunt prezentate setările implicite – dacă sunt satisfăcătoare se poate apăsa **Finish**, nu este cazul nostru, deci apăsăm butonul **Next**.

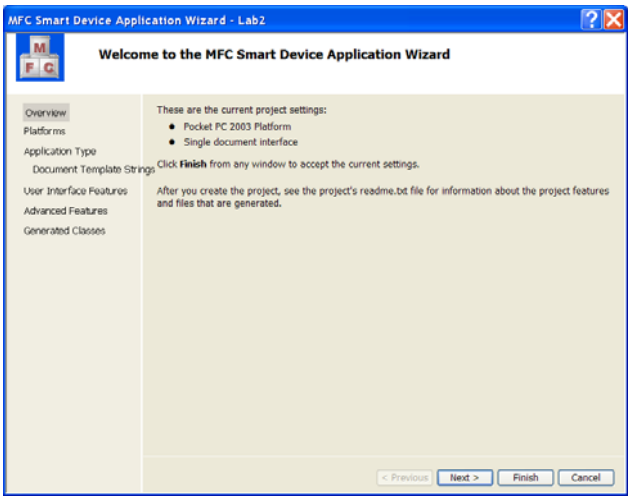

Figura 2.3. Cea de a doua fereastră a wizard-ului

d. În acest pas selectați SDK-ul (*Software Development Kit*) specific plăcii dumneavoastră (cea pe care veți urma să dezvoltați aplicația). Pentru cele 3 sisteme de dezvoltare existente în laborator alegeți: **MyWEC7_SDK** (pentru sistemul eBOX-3300), **OMAP3530_EVM** (pentru placa de dezvoltare OMAP3530 EVM) și **BeagleBoard-XM – WEC7** (pentru sistemul BeagleBoard-XM). În cazul prezentat în **Figura 2.4** alegeți **OMAP3530_EVM** SDK și înlăturați SDK-ul existent (Pocket PC 2003 Platform sau orice alt SDK existent implicit) din câmpul **Selected SDKs**.

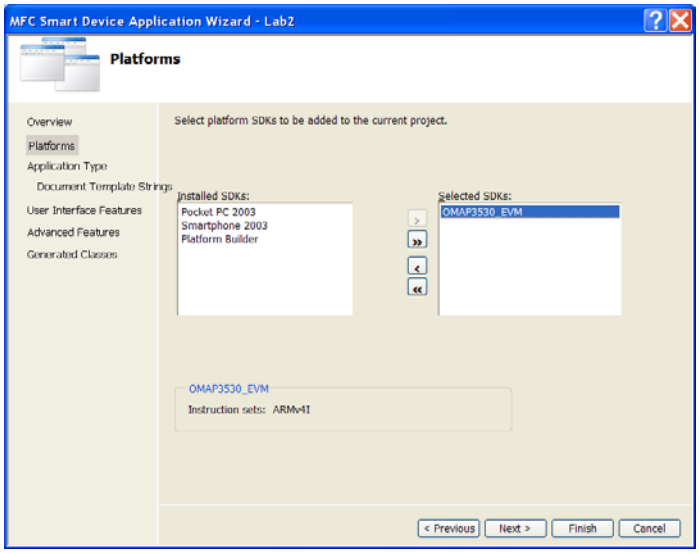

Figura 2.4. Selectarea SDK-ului

În momentul când lucrați fără existența vreunei plăci de dezvoltare, puteți utiliza simulatoarele oferite în cadrul mediului Visual Studio. Într-o astfel de situație alegeți SDK-ul **Poket PC 2003**.

O aplicație se execută și rulează susținută de un SO, Windows Embedded Compact în cazul nostru, creat special pentru un echipament *hardware* (un anumit sistem *embedded* ce are înglobate anumite periferice și componente suport) și pentru un SoC specific dezvoltat de o anumită companie – *Original Equipment Manufacturers* (OEMs).

Imaginea specifică a SO Windows Embedded Compact este creată cu ajutorul **Platform Builder** (o „unealtă" existent în mediul Microsoft Visual Studio) prin selectarea unor drivere și a unor componente specifice în concordanță cu necesitățile aplicației ce va rula și a dispozitivelor *hardware* ce formează sistemul *embbedded*.

Aplicația *software*, ce va fi executată pe dispozitivul *embbedded*, va fi dezvoltată în mediul integrat Microsoft Visual Studio. Această aplicație, specifică sistemului *embedded*, poate fi dezvoltată în două modalități specifice:

1. Prin crearea aplicației ca un subproiect în Microsoft Visual Studio **Platform Builder OS Design** ce conține toate acele componente specifice dispozitivului *embedded* – nu utilizăm aici această abordare – ea va fi prezentată la curs, și
2. Crearea aplicației ca un proiect separat în mediul Microsoft Visual Studio, dar utilizând un SDK specific sistemului *embedded*, SDK care a fost creat în **Platform**

Builder OS Design project și este direct dependent de dispozitivul *embedded* pe care aplicația va rula.

În cazul nostru SDK-ul OMAP3530_EVM a fost creat anterior, în proiectul imaginii SO dezvoltat cu ajutorul **Platform Builder**-ului și, ulterior, a fost instalat.

Apăsați butonul **Next**.

e. În următorul dialog de configurare aveți posibilitatea să alegeți tipul aplicației, limba în care se vor afișa diferitele mesaje și modalitatea de *link*-editare a bibliotecilor MFC, vezi **Figura 2.5**.

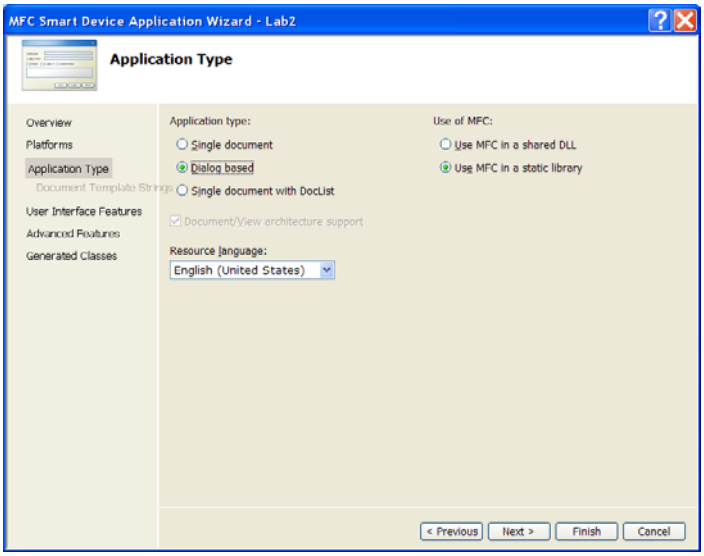

Figura 2.5. Dialogul de configurare a tipului de aplicație

În cadrul acestui laborator și în cea mai mare parte a celor ce vor veni, vom avea aplicații de tip **Dialog based** (interfața cu utilizatorul va fi formată dintr-o fereastră pe care vom plasa toate elementele de comandă, control și de prezentare a informațiilor). Aplicațiile de tip **Single document** sunt similare cu aplicația ***Notepad*** – fiind aplicații destinate gestionării unei foi de lucru și conțin toate acele meniuri predefinite de tip: nou document, salvare sau tipărire a documentului asupra căruia se lucrează etc.

Bibliotecile MFC pot fi utilizate de către un program în două modalități *share* sau *static*. Atunci când se alegea opțiunea *Use MFC in a static library* toate funcțiile, clasele, structurile de date etc. pe care programul nostru le utilizează din biblioteca MFC se vor găsi înglobate chiar în corpul programului, a fișierului exe rezultat în urma *link*-editării. Astfel, fișierul va fi mai mare (va necesita mai mult spațiu la stocare și mai multă memorie în timpul rulării) iar obținerea executabilului va dura mai mult (procesul de compilare și *link*-editare). Prin selectarea *Use MFC in a Shared DLL* programul rezultat va necesita utilizarea resurselor (a funcțiilor, claselor, structurilor de date și a oricărei alte resurse) de pe sistemul *embedded* (unde va rula), a **dll**-urilor specifice, sau, în cazul în care nu există, va necesita instalarea simultană cu el a fișierelor ***mfcxx.dll*** și ***msvcrxx.dll***.

Selectați opțiunile *Dialog Based* și *Use MFC in a static library* (această ultimă opțiune pentru a obține independența maximă față de sistemul pe care va rula aplicația) și apăsați butonul *Next*.

f. În noua fereastră dați un titlu semnificativ ferestrei de dialog și apăsați butonul *Next* – vezi **Figura 2.6**.

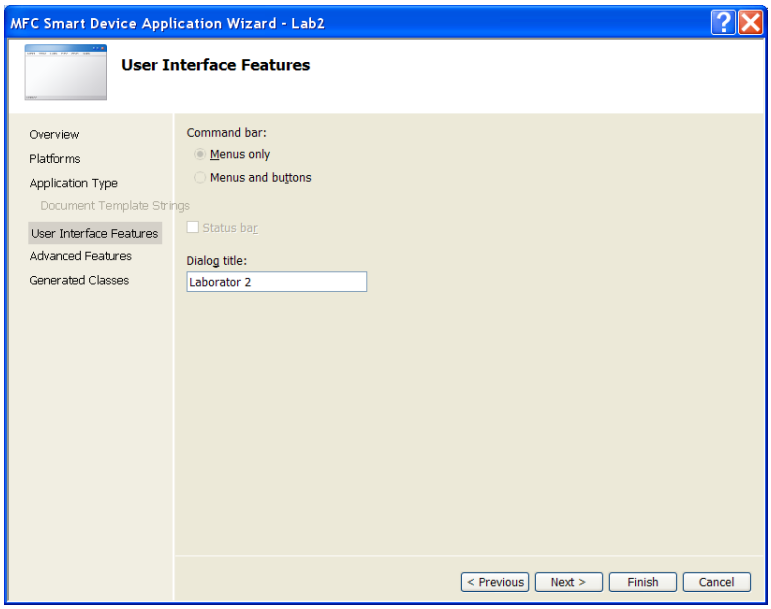

Figura 2.6. Selectarea caracteristicilor interfeței cu utilizatorul

g. În următoarea fereastră nu se selectează nici o opțiune (programele noastre nu vor utiliza nici ***ActiveX*** și nici ***sockets***), vezi **Figura 2.7**.

Apăsați butonul **Next**.

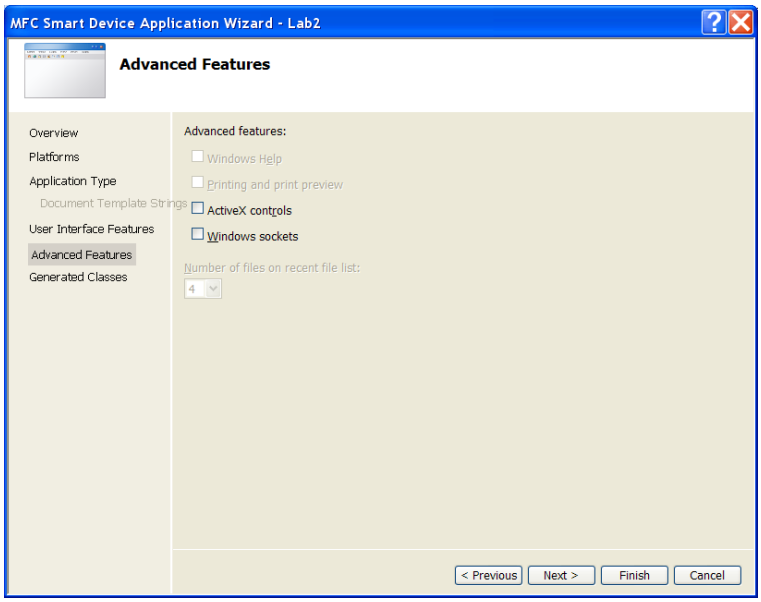

Figura 2.7. Alegerea opțiunilor avansate

h. În ultima fereastră de configurare vă apar clasele pe care mediul de dezvoltare le va genera în mod automat pentru dvs.

Visual C++ utilizează noțiunea de clasă drept noțiune fundamentală ce va fi întâlnită în tratarea tuturor elementelor utilizate. **Clasa este similară cu o structură de date din C, dar**

<u>față de aceasta grupează la un loc atât datele ce vor fi procesate cât și funcțiile care le procesează</u>.

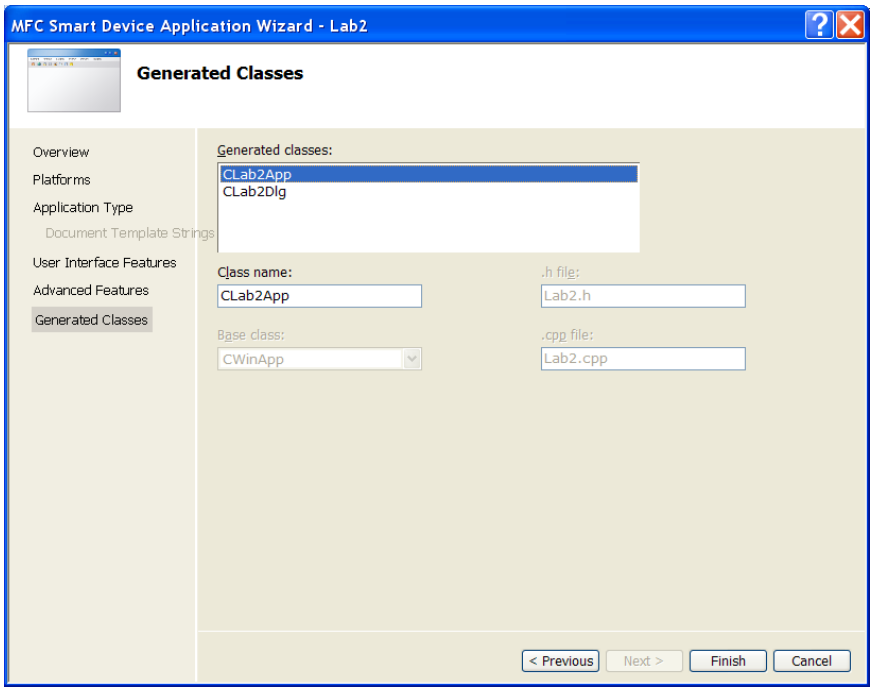

Figura 2.8. Ultima fereastră de configurare a *wizard*-ului

Astfel, tot programul nostru va îngloba 2 clase fundamentale. **CLab2App** este clasa similară cu funcția *main* din mediul ANSI C. În timp ce clasa **CLab2Dlg** va fi clasa asociată cu interfața grafică a utilizatorului – cu fereastra prin intermediul căreia utilizatorul interacționează cu codul.

Orice element (buton, *Check Box* etc.) care va fi plasat pe interfața grafică, care poate să aibă atașată o funcție (ce va fi executată atunci când starea lui se modifică), are atașată o clasă (prin intermediul căruia programul îi influențează comportamentul). Acest element grafic va avea atât funcțiile cât și clasele asociate drept membrii ai clasei CLab2Dlg.

i. Apăsați butonul Finish.

j. Configurați mediul și sistemul *embedded* pentru a permite rularea aplicației tocmai dezvoltată, verificați realizarea conexiunii – urmați pașii prezentați în prima lucrare de laborator.

k. Compilați, *link*-editați și executați programul pe placa de dezvoltare – urmați pașii prezentați în lucrarea de laborator anterioară.

l. Dacă obțineți *warning*-uri și erori similare cu cele de mai jos:

```
warning C4985: '_wcstoi64': attributes not present on previous declaration
error C2039: 'lstrlenW' : is not a member of 'ATL'
```

problema este dată de lipsa instalării *update*-ului **Windows Embedded Compact 7 ATL Update for Visual Studio 2008 SP1** sau a neconfigurării căilor de căutare pentru fișierele de tip *header* care trebuie incluse în proiect.

Instalați acest *update*.

Dacă după instalarea *update*-ului problema persistă urmații pașii: **Solution Tab** → Click dreapta pe proiect și alegeți **Properties** → **C/C++** → **General** → **Additional Include Directories** → și **selectați cele 2 căi** în conformitate cu figura de mai jos.

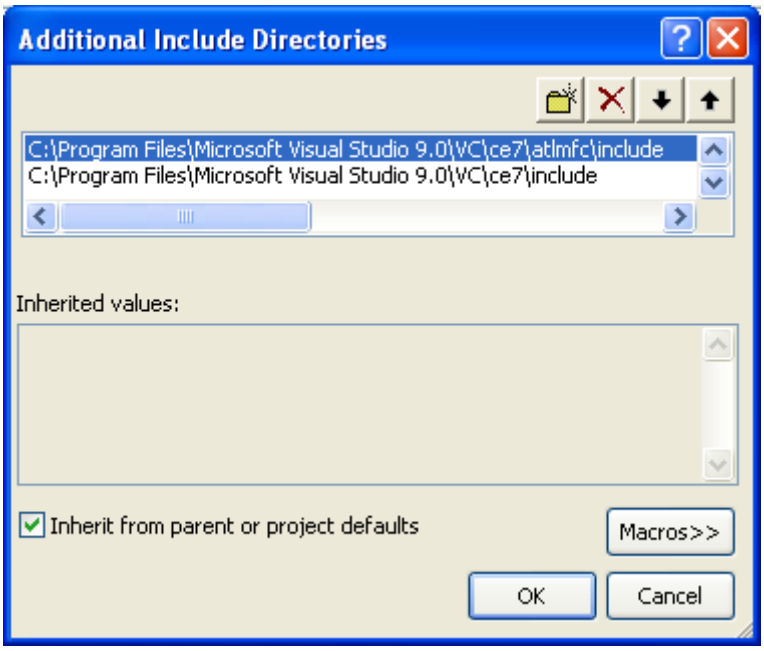

Figura 2.9. Căile ce trebuie incluse pentru link-editarea programului

1.5. Dezvoltarea aplicației

După finalizarea tuturor pașilor anterior menționați, s-a obținut o aplicație funcțională dar care încă nu are implementată nici o funcționalitate dorită de utilizator. Pentru atingerea obiectivelor propuse la începutul laboratorului urmați pașii:

4. Începeți construcția interfeței cu utilizatorul.
   - **a.** În tabul ***Resource View*** selectați numele proiectului, **Lab2** în cazul nostru (nume pe care dumneavoastră l-ați dat în prima fereastră de configurare a *wizard*-ului, **Figura 2.2**, în câmpul ***Name***), desfășurați *folder*-ul resurselor (Lab2.rc) și pe cel al ferestrelor de tip Dialog, dând dublu *click* pe IDD_LAB2_DIALOG. În acest moment fereastra principală de interfațare cu utilizatorul se va afișa, **Figura 2.10**.

     Dacă *tab*-ul ***Resource View*** nu este prezent afișați-l urmând secvența: View → Other Windows → Resource View sau apăsând combinația de taste Ctrl + Shift + E.

     ATENȚIE ! Mediul Visual Studio fiind unul de programare vizuală foarte mult cod este introdus în mod automat în diferite zone ale programului dvs., în fișiere diferite – în situația realizării unei singure acțiuni. Din acest motiv lucrați doar cu ferestrele Class View și Resource View (Figura 2.10 – stânga jos) ! Nu interacționați cu codul la nivelul fișierelor *cpp și *.h din intermediul ferestrei Solution Explorer !

   - **b.** Din panoul de instrumente (***Toolbox***) selectați și poziționați pe interfața grafică toate acele elemente necesare îndeplinirii funcționalităților programului, **Figura 2.10**.

Dacă *tab*-ul (fereastra) ***Toolbox*** nu este prezent afișați-l urmând secvența: View → ToolBox sau apăsând combinația de taste Ctrl + Alt + X.

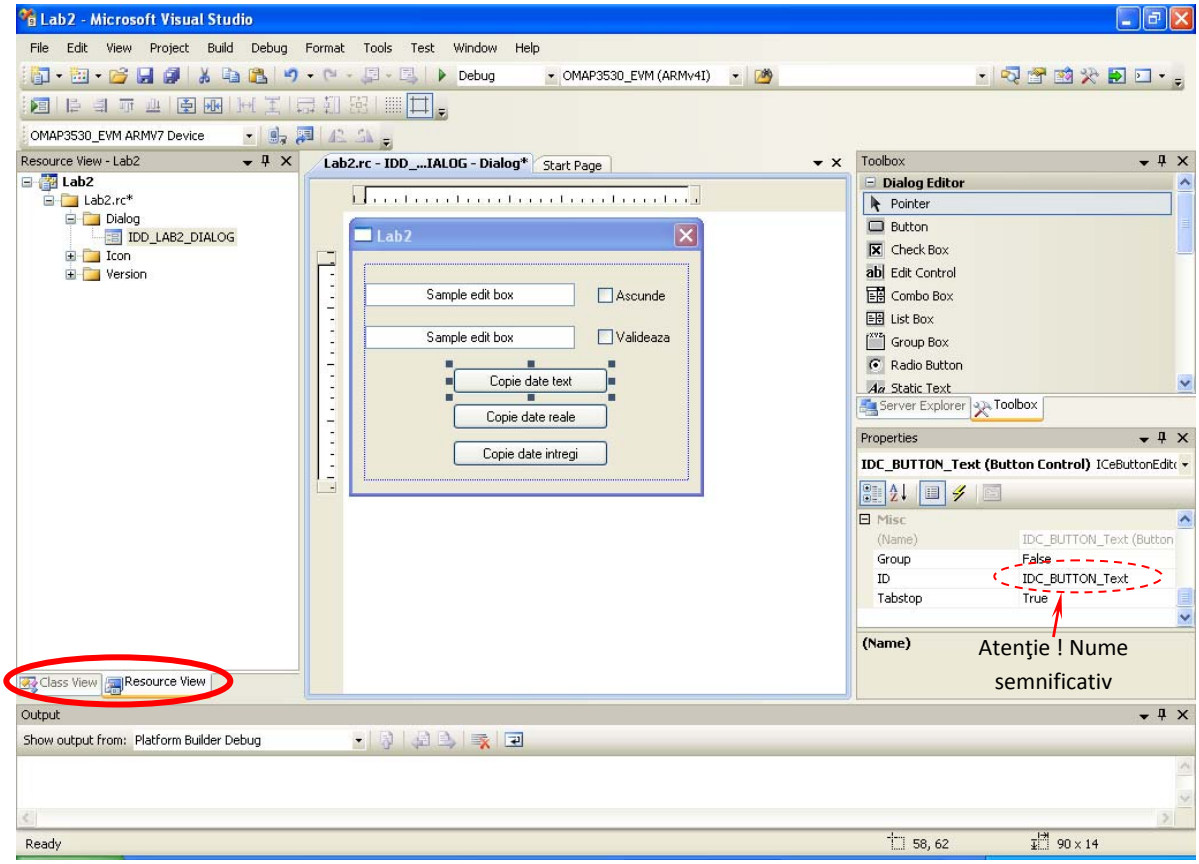

Figura 2.10. Interfața grafică a programului dvs.

c. Selectați fiecare element în parte (butoanele, *Check Box*-urile și elementele de tip *Edit Control*) și modificați în fereastra ***Properties*** diferitele câmpuri astfel încât informațiile text, ID unice de identificare să corespundă cerințelor și să fie cât mai relevante. De exemplu, în câmpul de identificare unic, ID din fereastra ***Properties***, alegeți numele de o așa natură astfel încât să fie cât mai sugestiv, indicând tipul elementului (*Button*) cât și funcția acestuia (de ex. *Text* – va copia informația dintr-un *Edit Control* în altul tratând-o ca un șir de caractere).

Dacă fereastra ***Properties*** nu este vizibilă dați click dreapta pe elementul a cărui proprietăți doriți să le modificați/vizualizați iar în fereastra deschisă alegeți opțiunea *Properties*.

d. Compilați programul și verificați funcționarea corectă a lui până în acest punct.

5. **<u>Asocierea de funcții elementelor de pe interfața grafică a programului</u>** – **cerință fundamentală și de bază necesară realizării oricărui program !**

Prin apăsarea unui buton sau prin bifarea unui *Check Box* dorim să lansăm în execuție o anumită bucată de cod care să realizeze anumite acțiuni prestabilite de noi anterior. Pentru a realiza acest lucru trebuie să asociem o funcție elementului de pe interfața grafică (codul acestei funcții va fi executat de fiecare dată când dăm, de exemplu, *click* sau *dublu click*), iar în corpul funcției să scriem un cod specific.

Pentru asocierea unei funcții cu un anumit element (*Button*, *Check Box* etc.) avem două modalități

de a o face:

a. Prima abordare, dați dublu *click* pe elementul respectiv – este cea mai simplă metodă dar care nu ne pezzrmite să configurăm vreo opțiune. Saltul în funcția asociată butonului se va realiza doar atunci când dăm *click* pe element. Această funcție va fi plasată în clasa care este în directă legătură și gestionează interfața grafică (Clab2Dlg – în cazul nostru), iar numele funcției va fi generat automat funcție de informația din câmpul ID asociat cu elementul nostru și de evenimentul tratat (*mouse click*).

b. Cea de a doua opțiune presupune să dați *click* dreapta pe elementul dorit și din fereastra care apare alegeți Add Event Handler …, rezultând deschiderea unei noi ferestre similară cu cea din **Figura 2.11**.

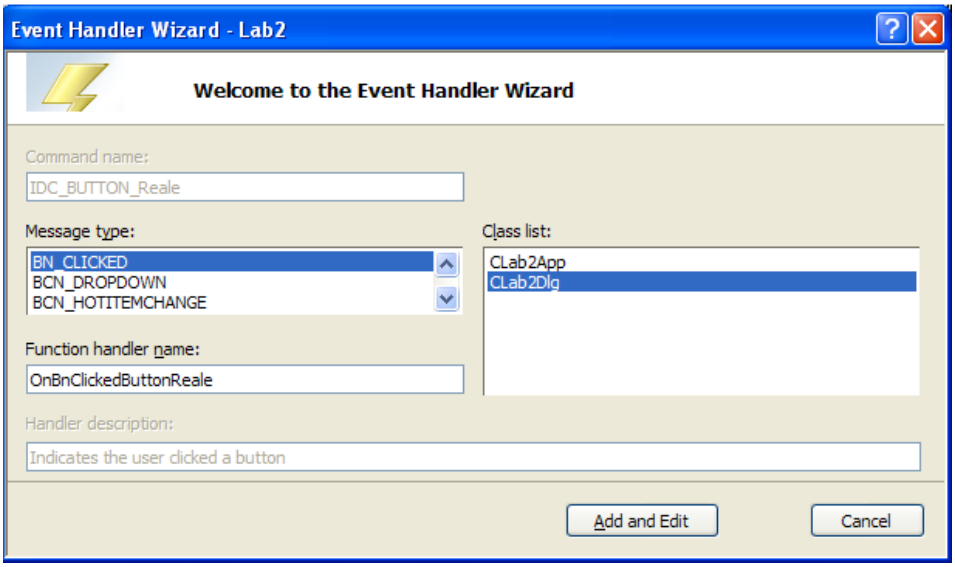

Figura 2.11. Fereastră de configurare a elementelor necesare tratării unui anumit eveniment

Se observă că pentru elementul cu ID-ul IDC_BUTTON_Reale (butonul care va copia un număr real dintr-un *Edit Control* în altul) putem selecta evenimentul ce va fi tratat în cadrul funcției (funcția va fi executată doar dacă utilizatorul va realiza acea acțiune specifică) din lista *Message type* (de ex. BN_CLICKED – buton apăsat o singură dată prin intermediul unui click stânga *mouse* sau, de ce nu, BN_DOUBLECLICKED – buton apăsat de două ori prin intermediul unui dublu *click* stânga mouse), numele funcției ce va trata mesajul (se observă că numele va include informațiile din câmpul ID plus informații despre evenimentul care a determinat apelarea funcției) și, putem alege, clasa care va conține funcția de tratare a evenimentului – în cazul nostru clasa CLab2Dlg (clasa care conține toate funcțiile și structurile de date ce au legătură cu interfața cu utilizatorul).

În situația în care avem nedumeriri în legătură semnificația unui anumit mesaj în câmpul Handler description avem o descriere sumara a mesajului ales.

În final apăsați butonul Add and Edit.

Asociați funcții prin una din cele două metode, prezentate anterior, tuturor butoanelor și *Check Box*-urilor existente pe interfața grafică. În final componența clasei CLab2Dlg va arăta similar ca în **Figura**

2.12**(b)**.

Figura 2.12. Componența clasei **CLab2Dlg**: (a) înainte și (b) după asocierea de funcții cu elementele de pe interfața grafică

6. **Interogarea sau influențarea stării unui element de pe interfața grafică** (**element fundamental în cadrul dezvoltării oricărui program !**) **este realizată prin intermediul unei variabile asociate (mai exact a unei clase specifice elementului respectiv)**. Pentru a asocia elementul cu o variabilă dați click dreapta pe element, alegându-se opțiunea Add Variable ...; în urma acestei acțiuni se va deschide o nouă fereastră similară cu cea din **Figura 2.13**.

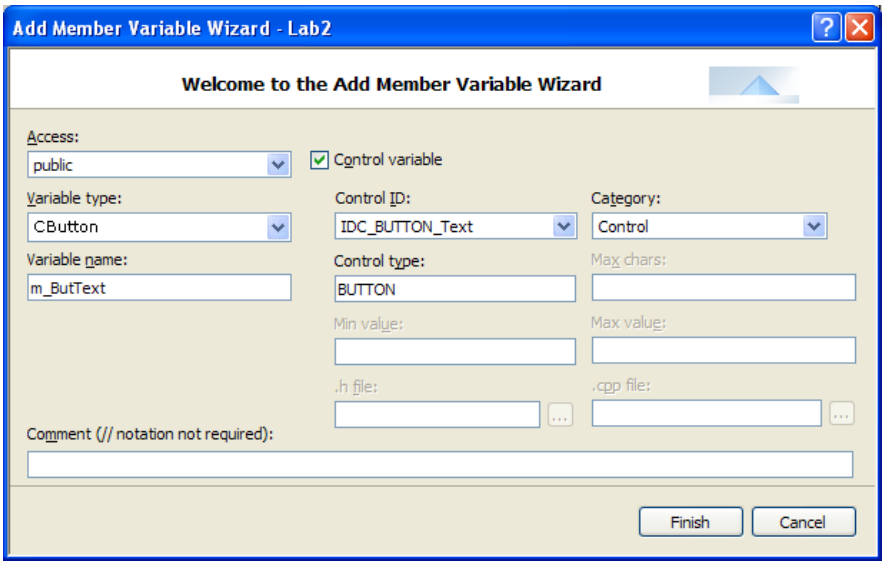

Figura 2.13. Fereastra de configurare a setărilor necesare asocierii unei variabile cu un element de pe interfața grafică

Este indicat ca această variabilă să fie de tip *public* în cadrul clasei CLab2Dlg (permițându-se accesul neîngrădit la ea de către toate funcțiile clasei) și va fi de tip **CButton** – o clasă de tip *Button*. O clasă de tip **CButton** va conține toate funcțiile care pot modifica, influența, interoga etc. starea unui

buton. Deoarece această variabilă este asociată cu elementul de tip IDC_BUTTON_TEXT prin apelarea funcțiilor din cadrul variabilei m_ButText (care este de tip clasă **CButton**) se va putea acționa numai asupra acestui buton.

Recomandare: *în momentul când dați diferite nume funcțiilor sau variabilelor pe care dumneavoastră le declarați și utilizați, dați-le nume cât mai sugestive astfel încât:*

1. *să le puteți deosebi de variabilele și funcțiile generate în mod implicit de către mediul de dezvoltare (în acest caz particular prin "m_" se codifică că este o variabilă declarată de către mine – **m**y variable),*
2. *să puteți să vă dați seama de tipul elementului căruia îi este asociată, în acest caz unui buton, de aici secvența de text "But", din numele m_ButText, semnifică că este asociată unui element de tip buton,*
3. *să puteți identifica din multitudinea de butoane pe care le aveți pe interfața grafică cărui buton îi este asociată, de aici ultima secțiune "Text" din numele ei – este butonul care copie o informație de tip text.*

Variabila trebuie să fie în mod obligatoriu de tipul *Control* (din câmpul *Category* – permite controlarea modului de comportament a elementului) pentru a-și putea face funcțiile anterior menționate. Deci, în câmpul *Variable Name* introduceți un nume semnificativ, ușor de identificat și asociat cu butonul cu care este asociată variabila.

Prin asocierea unei variabile de tip clasă cu elementele de pe interfața grafică, fiecare element va avea o clasă specifică ce conține funcții specifice de interacțiune cu elementul respectiv. Astfel:

a. variabila *m_ButText* este o clasa de tip CButton care conține în mod implicit un număr mare de funcții, printre care și:

- SetWindowTextW(_T("Bla")) - funcție ce permite afișarea unui text în cadrul unei anumite ferestre (Atenție ! și un buton este tratat drept o fereastră). Apelați această funcție sub forma:

  m_ButText.SetWindowTextW(_T("Bla"));

- EnableWindow – validează/invalidează (în cazul nostru particular vorbim despre un buton, dar poate fi orice altceva – *check box*, *edit box* etc.). Apelați această funcție astfel:

  m_ButText.EnableWindow (...);

- ShowWindow – afișare sau nu a elementului (comentariu identic ca mai sus). Apelați această funcție astfel:

  m_ButText.ShowWindow (...)

b. Pentru *check box*-uri: GetCheck (preiau starea lui – bifat sau nu) etc.

c. Pentru *Edit Control*-ul:

- GetWindowTextW – pentru preluare de informații. Evident că și *Edit Control*-ului îi asociez o variabilă (de tip CEdit), fie aceasta m_EditSursa, pentru a prelua textul utilizați linia:

m_EditSursa.GetWindowText(szEdit)

Textul preluat poate fi de orice tip (un număr real, un întreg, text etc.), szEdit este o clasă de tip *string* (CString szEdit) care va conține în interiorul ei structura de date ce stochează textul preluat din *Edit Control* dar și multe funcții ce pot procesa acest text.

- LineLength – determină numărul de caractere existent în *Edit Control*.
- SetWindowTextW – pentru afișarea unui text oarecare.

7. Filozofia preluării unei valori de pe un element de tip EDIT BOX, a prelucrării acestei valori și, ulterior, a afișării ei pe un alt element de tip EDIT BOX este următoarea:
   a. Se preia informația de pe elementul poziționat pe interfața grafică și se salvează într-o variabilă locală.

      În cazul nostru particular avem un element de tip **EDIT BOX**, iar funcția pe care o vom folosi pentru preluarea informației și salvarea locală a ei este GetWindowText. **Atenție!** Funcția preia un șir de caractere, deci indiferent dacă în **EDIT BOX** avem valoarea 3.1415926 ea va fi tratată ca șirul de caractere "3.1415926", deci ca un vector în care în primul element vom avea codul ASCII al cifrei "3", în cel de al doilea element avem codul ASCII a elementului "." etc.

   b. Șirul de elemente ASCII trebuie convertit către o valoare reală și stocat într-o variabilă de tip float sau double.

   c. Se realizează înmulțirea cu 2.

   d. Se convertește înapoi, rezultatul real stocat într-o variabilă de tip float sau double, către un șir de caractere ce va fi stocat local în altă variabilă (sau în aceeași variabilă inițială dacă conținutul ei nu ne mai trebuie ulterior).

   e. Se afișează această ultimă valoare pe elementul **EDIT BOX** în care se prezintă toate rezultatele. Funcția prin care se realizează aceasta (preluarea șirului din variabila locală și afișarea pe elementul grafic de pe interfața grafică) este SetWindowTextW.

      Dacă în mod clasic se lucrează cu un vector (*buffer*) de elemente de tip char în care se stochează o informație de tip șir de caractere ambele funcții GetWindowTextW sau SetWindowTextW lucrează cu variabile de tip clasă CString. Într-o astfel de clasă, CString, se stochează atât textul cât și toate funcțiile necesare manipulării acestui text.

8. Pentru a realiza conversia de la CSting la *float* (număr în virgulă mobilă) sau *integer* (întreg) puteți utiliza funcțiile:

   _wtof și wcstoul

9. În situația în care funcția _wtof nu funcționează (compilatorul nu găsește biblioteca asociată funcției) utilizați funcția wcstombs pentru realizarea conversiei de la un CString la un vector char (funcție utilizată pentru a extrage din clasa CString datele de tip text, codate unicode, și convertirea lor către un vector de tip char) și ulterior funcția atof pentru convertirea lor în format float.
10. Pentru preluarea informației reale, dublarea valorii și afișare rezultatului puteți folosi următoarea secvență de cod:

```
// în variabilele myInputText și myOutputText de tip
// CString se stochează informația ce va fi preluata și
// cea care va fi afișată de pe/către interfața grafică
CString myInputText, myOutputText;
double rezultat;

// m_EditInput este variabila asociată cu primul element
// de tip CEdit din care se citesc informațiile
m_EditInput.GetWindowTextW(myInputText);

// funcția GetBuffer este o funcție internă clasei CString
// care accesează zona de date unde sunt stocate informațiile
// de tip text și le preia
rezultat = _wtof ( myInputText.GetBuffer());

rezultat = rezultat * 2;

// pentru formatarea datelor și stocarea într-o variabilă de
// tip CString utilizați funcția Format (internă și ea clasei
// CString) – funcție similară  ca mod de funcționare cu
// printf
myOutputText.Format (_T("Dublul valorii reale este %f"), rezultat);

// m_EditOutput este variabila asociată cu cel de al doilea
// element de tip CEdit în care se afișează informațiile
m_EditOutput.SetWindowTextW ( myOutputText );
```

11. Puneți *brakepoint*-uri, rulați codul anterior prezentat pas cu pas și vizualizați starea diferitelor variabile în timpul rulării pas cu pas.

12. Dezvoltați întreg codul, conform cerințelor de la începutul laboratorului.

13. De multe ori se poate lucra și cu un cast implicit, așa cum este prezentat în exemplul de mai jos:

```
CString szResult, szEdit;

szResult.Format(_T(„%s are %d caractere”), szEdit.GetBuffer(), 14);

// la fel de bine linia anterioară poate fi înlocuită cu:
//
//        szResult.Format(_T(„%s are %d caractere”), szEdit, 14);
//
// cu toate că szEdit este o variabilă de tip clasă iar funcția așteaptă un
// string (%s), mediul Visual Studio face o conversie de tip cast către
// sting din clasa CString

//Pentru afișare utilizați codul:
m_EditOutput.SetWindowTextW(szResult);
```