

## **Prefață**

*Disciplina “Interfață Om-Mașină” studiază, așa cum reiese și din numele său, comunicarea între om și diferite aparate sau mașini. Noțiunea “mașină” implică o gamă largă de produse, de la un ceas de mână și până la instalații automatizate de mare capacitate. Comunicația reprezintă procesul prin care omul interacționează cu respectivul aparat. Nu de puține ori un utilizator și-a pus problema “Ce buton trebuie să apăs? Ce se întâmplă mai departe? De ce a apărut eroare?”. Studiul “Interfeței Om-Mașină” este util în a proiecta programe ușor de folosit, de la simple aplicații software cu caracter local și până la cele cu care se pot controla procesele industriale. De aici rezultă o caracteristică importantă a acestora, ce poartă numele de utilizabilitate. În domeniul software, utilizabilitatea presupune rezolvarea unei probleme aparent simplă: cum se poate proiecta un software astfel încât un începător să îl poată utiliza fără a fi nevoit să citească un manual de sute de pagini sau să urmărească un tutorial foarte complicat, în timp ce un utilizator cu experiență își poate derula munca fără a fi întârziat de pași inutili.*

*Autorii prezentei lucrări consideră că pentru a realiza o aplicație vizuală este necesară cunoașterea unor elemente de bază ale sistemului de operare Windows, precum și modalitățile de acces ale programatorului la acestea. Scopul principal al acestei lucrări este aprofundarea principiilor teoretice dobândite la orele aferente cursului “Interfață Om-Mașină”.*

*Lucrarea se adresează studenților Facultății de Automatică și Calculatoare din Universitatea “POLITEHNICA” din București, profilul Automatică, precum și inginerilor sau programatorilor ce doresc să-și consolideze gradual cunoștințele despre proiectarea și realizarea unei aplicații Windows. În consecință, problematica prezentată în lucrările de laborator are ca scop inițierea în munca de programare în mediul Windows SDK.*

*Aplicațiile aferente acestui îndrumar de laborator se găsesc pe Internet, la adresa [www.iom.aii.pub.ro](http://www.iom.aii.pub.ro).*

*Autorii țin de asemenea să precizeze că orice sugestii și propuneri din partea cititorilor sunt bine venite. Lucrarea de față se dorește a fi o lucrare deschisă, problematica prezentată putând fi, în mod evident, lărgită în cadrul unei apariții viitoare.*

*Autorii.*



## Laborator 1. Mediul Visual C++

Pentru desfășurarea activităților laboratorului aferent cursului “Interfață Om-Mașină” este necesară instalarea pachetului Microsoft Visual Studio și a MSDN-ului. Laboratorul 1 își propune familiarizarea studenților cu mediul Visual C++, precum și analiza unei aplicații simple în Visual C++.

### 1.1 Instalarea mediului de dezvoltare Microsoft Visual Studio 6.0

Mediul de dezvoltare Microsoft Visual Studio se instalează ca orice altă aplicație Windows, fără a ridica probleme deosebite.

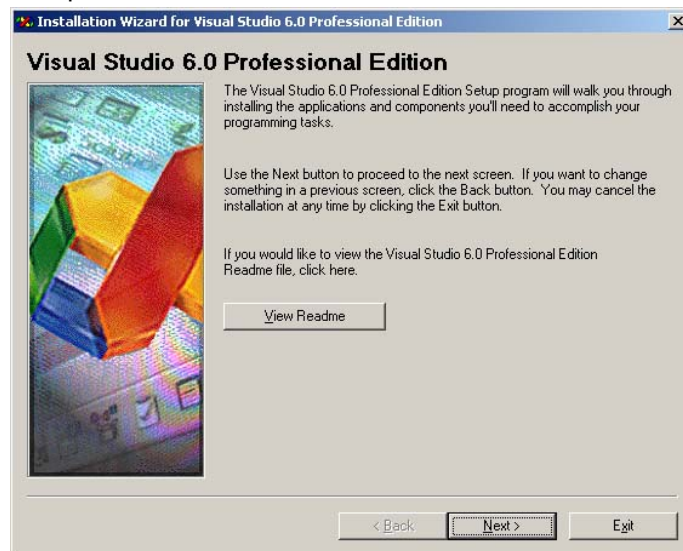


Figura 1.1. Ecranul de întâmpinare la instalarea Visual Studio.

Ecranul de mai sus îl întâlniți fie la câteva secunde după introducerea CD-ului de instalare în CD-ROM, dacă aveți activată opțiunea AutoRun Data CD în Windows, fie dacă lansați manual aplicația “Setup.exe” de pe respectivul CD.

Cu ajutorul butonului Next, veți parcurge diverse ferestre, unde vi se cere Serial Number, să acceptați EULA, etc. În fereastra din figura 1.2, se selectează instalarea Microsoft Visual Studio 6.0.

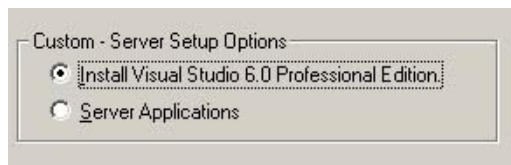


Figura 1.2.

Ulterior, vi se cere să stabiliți unde să fie copiate “Common Files”. Este de preferat să nu modificați această cale. După ce ați stabilit calea, se mai deschid o serie de ferestre de informare, până ajungeți la cea din figura 1.3, fereastra din care puteți stabili locația unde va fi instalat Microsoft Visual Studio, precum și componentele sale.



Figura 1.3.

Directorul implicit al lui Microsoft Visual Studio este Program Files, de pe partiția ce conține sistemul de operare. O mare parte a utilizatorilor Microsoft Windows preferă să instaleze aplicațiile uzuale pe o altă partiție, din motive de spațiu sau din alte motive (reunirea aplicațiilor, evitarea blocării programelor). În acest pas veți stabili locația pentru Microsoft Visual Studio. Pentru a modifica directorul de instalare acționați butonul “Change Folder” și selectați noua destinație.

Un alt pas important din instalarea lui Microsoft Visual Studio este selecția componentelor ce urmează a fi folosite de către utilizator. Pentru aceasta, efectuați click pe butonul “Custom”, și se va deschide fereastra din figura următoare:

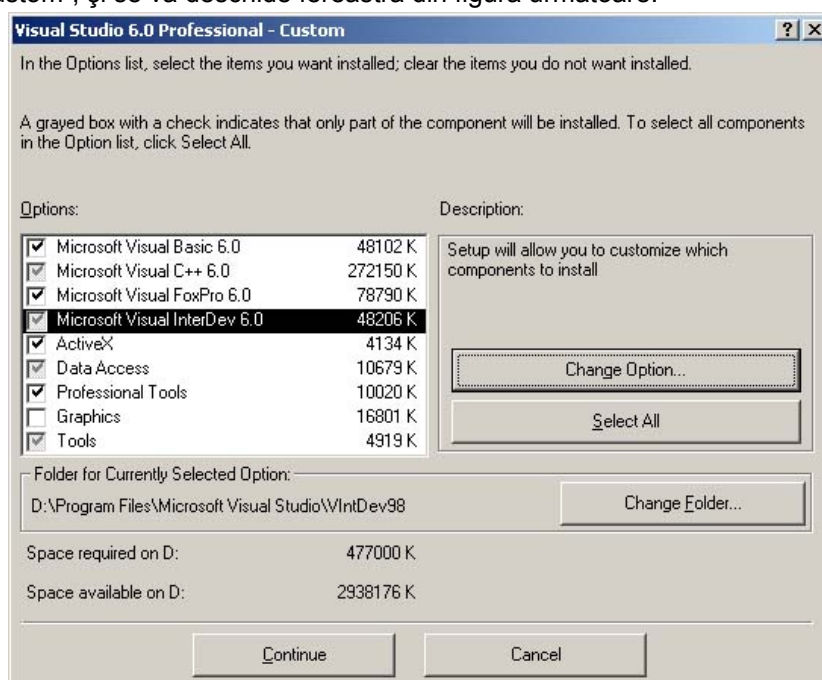


Figura 1.4. Componentele Microsoft Visual Studio

Pentru a putea rula aplicațiile aferente laboratoarelor cursului “Interfață Om - Mașină”, este necesar să instalați doar Visual Basic 6.0 și Visual C++ 6.0. Totuși, dacă plănuți să folosiți Visual Studio pentru a dezvolta și alte proiecte, atunci trebuie să vă gândiți ce alte componente mai instalați, în funcție de necesități și de spațiul disponibil. Pentru a adăuga ulterior alte componente, sau a șterge o parte din cele de care nu mai aveți nevoie, se repetă pașii de instalare descriși până în acest moment.

Help-ul furnizat de programul de instalare al Microsoft Visual Studio în ceea ce privește componentele este destul de slab. Ca o recomandare, dacă nu aveți nevoie în mod special de alte componente, deselectați totul în afară de Visual Basic și Visual C++ și treceți la pasul următor.

Se poate vedea în figura 1.4, că unele check-uri (bife) din dreptul programelor sunt negre pe fond alb, în timp ce altele sunt negre pe fond gri.

Bifele negre pe fond alb semnifică faptul că toate opțiunile din ierarhia acelei componente au fost selectate, în timp ce bifele negre pe fond gri implică selecția unor anumite opțiuni ale unei componente. Spre exemplu, în mod implicit, Visual Basic se va instala cu toate sub-opțiunile sale, în timp ce Visual C++ are selectate doar o parte din sub-opțiuni. Dacă deselectați o componentă cu subcomponente selectate parțial și apoi o selectați din nou, ea va fi instalată toate sub-opțiunile. Nu este însă recomandabil să faceți acest lucru în cazul lui Visual C++, deoarece o să descoperiți că spațiul de instalare necesar se dublează, pentru că vor fi selectate o serie de componente care nu sunt neapărat necesare, decât în cazuri speciale.

În pasul următor are loc copierea efectivă a fișierelor pe hard disk, în locația specificată. La sfârșitul procesului de copiere se mai deschid o serie de ferestre de informare, în funcție de opțiunile selectate, după care vi se cere restartarea calculatorului.

După restart și configurări asupra cărora nu puteți interveni, programul de instalare vă întreabă dacă doriți să instalați MSDN precum și alte componente ale lui Visual Studio. MSDN-ul se poate instala separat, așa cum se poate vedea în capitolul următor, iar celelalte componente nu sunt necesare. Treceți peste aceste ferestre cu Next, fără a selecta nimic și terminați instalarea. La sfârșitul procesului de instalare se va afișa o fereastră ce vă va informa despre succesul instalării Microsoft Visual Studio.

Următorul pas este instalarea MSDN-ului, fără de care este foarte greu să se dezvolte un program.

## 1.2 Instalarea MSDN Library pentru Microsoft Visual Studio 6

MSDN (Microsoft Developer Network) Library sau mai pe scurt, MSDN reprezintă help-ul Microsoft Visual Studio, fiind o colecție de descrieri ale funcțiilor, exemple, aplicații și altele. Instalarea sa este foarte simplă, pornirea realizându-se la fel ca la Visual Studio, fie automat, fie manual prin intermediul "Setup.exe".

După ce parcurgeți o serie de ferestre informale, ajungeți la fereastra următoare, unde va trebui să selectați tipul instalării:

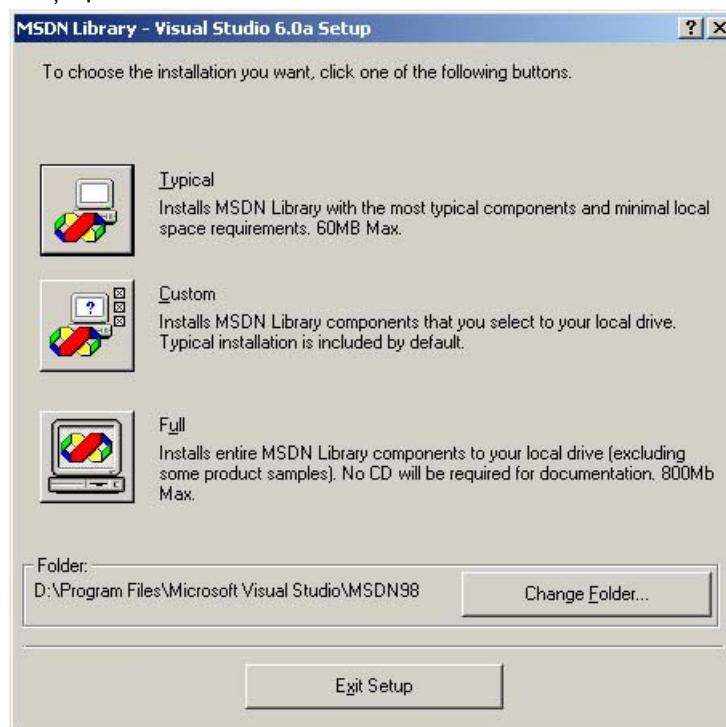


Figura 1.5. Moduri de instalare a MSDN-ului

După cum se observă, sunt disponibile trei moduri de instalare: Typical, Custom și Full. În cazul instalării Custom sunt selectate în mod implicit toate opțiunile – cu subcomponentele aferente, din modul Typical.

Pentru accesul facil la resurse, este bine să se instaleze MSDN-ul cu opțiunea Full. În acest mod nu mai este cerut nici unul din cele două CD-uri cu care vine MSDN. Instalarea în

modul Full, cu toate că are dezavantajul că ocupă mult spațiu, permite economisirea unui timp destul de însemnat.

În cazul în care nu se dispune de spațiu, pentru aplicațiile de la laborator sunt necesare următoarele componente: VB Documentation, VB Product Samples, VC++ Documentation, Platform SDK Documentation.

După copierea fișierelor și parcurgerea unor ferestre de informare, aveți instalat MSDN-ul pe calculator, alături de Visual Studio.

### 1.3 Visual C++ ca mediu de dezvoltare

Visual C++ face parte din colecția de programe Microsoft Developer's Workshop, colecție ce mai include prin altele Visual Basic, Visual Fox, Visual Java și altele, fiind, așa cum am văzut mai sus, o parte integrantă a Microsoft Visual Studio. Visual C++ este o unealtă de dezvoltare de tip IDE – Integrated Development Environment, ce permite atât scrierea de cod sursă și generarea unor aplicații, cât și proiectarea grafică a ferestrelor aplicațiilor sau desenarea unor icoane. Majoritatea aplicațiilor scrise pentru Microsoft Windows, indiferent de versiune, sunt dezvoltate fie în Visual Basic, fie în Visual C++

### 1.4 Programarea în Visual C++

Amploarea proiectelor software de astăzi, coroborată cu dezvoltarea în echipă implică noțiunea de **proiectare și programare modularizată**, ce aduce o serie de avantaje. Dintre acestea, cele mai importante sunt:

- identificarea rapidă a erorilor;
- programatorii implicați într-un proiect se pot ocupa doar de partea lor, fără a fi necesar să acumuleze informații suplimentare despre porțiunile dezvoltate de ceilalți membrii ai echipei;
- posibilitatea schimbării unor funcții, fără a fi afectat întregul proiect;
- posibilitatea adăugării unor funcții noi.

Programarea modularizată poate fi privită în două sensuri:

- împărțirea codului sursă în funcții sau proceduri pentru evitarea redundanței;
- împărțirea funcțiilor în diverse fișiere, în funcție de caracteristicile comune.

După cum se știe, un obiect, reprezentat la nivel de program printr-o clasă, conține variabile și metode (funcții) proprii. De aici rezultă imediat o modalitate de împărțire în fișiere a unui program, și anume alocarea unui fișier sursă separat fiecărei clase din proiect. Această modalitate se folosește foarte des, în special în programarea tip MFC.

În programarea în limbajul C sau C++ s-a impus de-a lungul timpului împărțirea codului sursă în fișiere header și fișiere sursă. În header găsim de obicei definiții de constante sau chiar definiții de clase, precum și prototipurile unor funcții, în timp ce în fișierele sursă exista implementările datelor din header.

Majoritatea limbajelor de programare permit dezvoltarea modularizată a programelor. Visual C++ se încadrează în aceste linii, însă noțiunea de workspace cu care lucrează permite facilități suplimentare, dintre care cea mai importantă este reunirea în aceeași fereastră a tuturor variabilelor și funcțiilor din cadrul unui proiect, indiferent că acestea se află în fișiere diferite (modul ClassView din workspace).

Un al doilea aspect al programării în Visual C++, pe lângă programarea modularizată este reprezentat de posibilitatea de Debug al programelor. În acest sens, un utilizator poate parcurge un program linie cu linie sau la nivel de funcție. Totodată, este disponibilă o fereastră Watch, în care se pot urmări valorile unor variabile din program.

Un alt aspect al programării în modul vizual este modul de dezvoltare al programelor. În Visual C++ sunt disponibile două tipuri de resurse: fișiere cod sursă și grafice.

Fișierele grafice sunt icoane, imagini, cursoare, care au un format special, impus de Windows.

Fișierele sursă sunt de două tipuri:

- fișiere ce conțin doar cod sursă
- fișiere tip resursă grafică, care sunt fișiere text, ce pot fi citite de un utilizator. Însă în mediul Visual C++ acestea sunt interpretate ca fiind o interfață grafică și afișate ca atare. O resursă poate conține o fereastră cu butoane, un grup de ferestre, meniuri sau alte informații legate de partea grafică a aplicației.

### 1.5 Diferențe între programarea Win API (SDK) și MFC

Win API (Application Program Interface) cunoscut și sub numele de Win SDK – Software Development Kit, reprezintă o colecție de funcții și proceduri, implementate în fișiere de tip DLL (Dinamic Link Library), în special în cele de bază: kernel32.dll, user32.dll și gdi32.dll. Win API permite dezvoltarea orientată pe obiecte, dar și programarea simplă, în C normal și este considerat un limbaj de programare tip low-level. Avantajul său este rapiditatea execuției unui program, datorită accesării directe a unor resurse deja existente în memoria RAM a calculatorului.

MFC – Microsoft Foundation Classes a fost introdus destul de devreme în programare și anume în 1992, în perioada în care C++ începuse să ia locul C-ului normal. MFC reprezintă o “împachetare” a Win API, în sensul că ajută foarte mult utilizatorul în dezvoltarea programelor, realizând o serie de automatisme, dar are dezavantajul că este mai lent. Practic, instrucțiunile MFC sunt fie Win API, fie reuniuni ale acestora, care în momentul compilării sunt translatate în instrucțiuni API. Din acest punct de vedere, o aplicație MFC va rula întotdeauna mai lent decât alta care realizează același lucru, dar scris în Win API. Situația este similară cu un program scris în C, comparativ cu unul scris direct în Assembler.

Însă MFC are și o serie de opțiuni și funcții noi, ce se pot realiza mai greu în Win API, cum este cazul proceselor de serializare, toolbar-uri, ferestre împărțite, arhitectura document/view. MFC se folosește în general în cazul proiectelor de programare la nivel Enterprise.

Dacă se dorește crearea unui program utilitar mic, care să ruleze rapid, atunci se utilizează SDK. Timpul petrecut pentru dezvoltarea unui program în MFC sau SDK este același, însă fișierul executabil va fi mai mic și procesul de instalare va fi mult mai facil. Când se distribuie un program dezvoltat în MFC, acesta trebuie compilat cu librării MFC statice, sau distribuit împreună cu dll-urile MFC necesare. O problemă care apare este incompatibilitatea între versiunile de MFC, deoarece vechile versiuni de MFC-uri nu sunt întotdeauna compatibile cu cele noi. De asemenea, este posibilă blocarea calculatorului în momentul instalării unui proiect MFC nou, deoarece se va face update unor librării ale sistemului. Dacă programul de instalare nu are funcții specializate, atunci se vor suprascrie biblioteci mai noi cu unele mai vechi.

Cunoașterea Win SDK este necesară atât pentru programarea în SDK, cât și pentru programarea MFC, întrucât o mare parte din MFC este SDK. Diferențele sunt în general mici. Deși există o serie de funcții Win SDK care nu se regăsesc în MFC, înțelegerea mecanismelor de programare în SDK vă va ajuta foarte mult în programarea tip MFC.

### 1.6 Mediul Visual C++

Înainte de a începe primul proiect în Visual C++, un utilizator trebuie să se obișnuiască cu mediul de lucru al aplicației, cu barele de meniuri și ferestrele ajutătoare puse la dispoziția sa de către Visual C++.

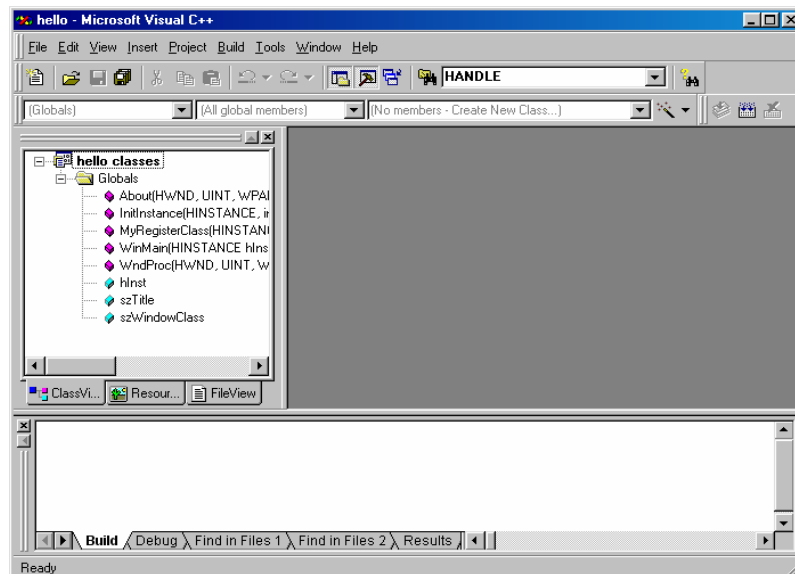


Figura 1.6. Mediul de dezvoltare Visual C++



La deschiderea programului, se poate vedea o fereastră asemănătoare cu cea din figura 1.6. Fiecare zonă are un anumit scop în mediul de dezvoltare. Utilizatorul poate rearanja ferestrele, după nevoile sale particulare.

### 1.6.1 Spațiul de lucru - Workspace

Zona din stânga ecranului poartă numele de Workspace și reprezintă o modalitate facilă de a accesa toate resursele din cadrul proiectului pe care îl dezvoltăți. În momentul când se deschide Visual C++, zona este goală, așa cum se poate observa și în figura următoare:

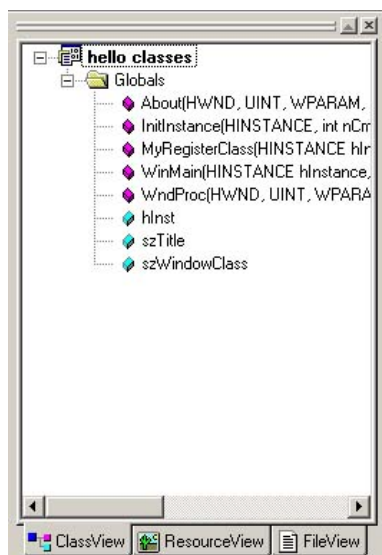


Figura 1.7. Workspace-ul unui proiect în Visual C++

Workspace-ul permite accesarea resurselor unui proiect în trei moduri diferite:

- **Class View** permite navigarea și manipularea codului sursă C++ la nivel de clasă
- **Resource View** permite găsirea și editarea resurselor grafice din aplicația dezvoltată, cum ar fi: ferestre, ferestre de dialog, butoane, icoane, meniuri etc.
- **File View** este un browser la nivel de fișiere de cod sursă. În cadrul acestui tab din workspace găsiți fișierele care sunt incluse în proiectul dezvoltat.

### 1.6.2 The Output Pane

Panoul Output nu apare la prima pornire a Visual C++. Însă după prima compilare, acesta devine vizibil în josul ecranului, și rămâne deschis până când se decide închiderea sa. În fereastra Output sunt afișate informații specifice despre aplicație, generate de compilator, cum ar fi: pașii de compilare, erorile, warning-urile. Tot în cadrul acestui panou sunt afișate valorile specifice ale parametrilor din program, pe care utilizatorul dorește să îi urmărească în cadrul procesului de debug. Practic, panoul Output preia funcția de Watch.

Dacă panoul Output a fost închis, el se va redeschide automat în momentul apariției unui nou mesaj la compilare.

### 1.6.3 The Editor Area – Zona de editare

Partea dreaptă, care ocupă cea mai mare suprafață a mediului de dezvoltare este zona de editare, unde se realizează toate operațiile de editare asupra fișierelor în Visual C++. Aici se poate adăuga, șterge sau modifica codul sursă al unui fișier inclus într-o aplicație. Partea de prelucrarea grafică a unei icoane, ce urmează a fi atribuită aplicației, sau editarea grafică a unei ferestre (butoane noi, meniuri, etc) se realizează tot în aceasta zonă.

Mediul de dezvoltare oferă o facilitate importantă – AutoComplete, ce vine în ajutorul programatorului în procesul de dezvoltare a unui program. Principalul beneficiu al funcției AutoComplete este reducerea timpului de editare.

Astfel, după se poate observa în figura următoare, în momentul în care ați scris primele două litere dintr-o funcție, și apoi apăsați combinația de taste Ctrl+Space, se deschide o fereastră cu sugestii care pot fi completate sub funcția al cărei nume ați început să



îl scrieți. Odată ce ați văzut funcția pe care doriți să o inserați, acționați săgeata jos (sau sus, după caz), până ați selectat-o, și apoi Enter.

În acel moment se inserează în codul sursă funcția aleasă. Deși C++ este un limbaj de programare case-sensitive (face diferența între litere mici și litere mari), dacă folosiți funcția AutoComplete, puteți să scrieți începutul numelui funcției cu litere mici. Fereastra AutoComplete nu face diferența între modul cum ați introdus textul. Odată selectat numele funcției în fereastra AutoComplete, aceasta se introduce corect în codul sursă, respectând literele mici și cele mari.

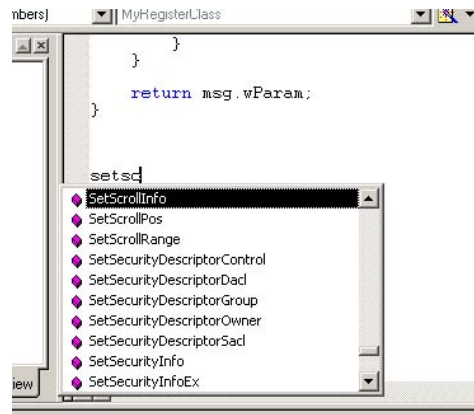


Figura 1.8. Fereastra AutoComplete

Spre exemplu, dacă dorim să inserăm funcția "SetScrollRange", tastăm "set", apoi CTRL+Space, și se va deschide fereastra AutoComplete. Cu fereastra deschisă, continuăm să tastăm "sc" și am derulat automat lista până la funcțiile care încep cu "SetScroll". Apoi acționăm tasta săgeată jos, până ajungem la funcția dorită. Cu ajutorul tastei Enter se inserează în codul sursă funcția "SetScrollRange". Când deschidem o paranteză, pentru a introduce parametrii, se afișează automat sub numele funcției, o fereastră de tip ToolTip, în care ne sunt prezentați parametrii pe care îi necesită funcția respectivă, precum și tipurile acestora. Dacă nu dorim să ne fie afișați parametrii este de ajuns să închidem paranteza și ToolTip-ul dispare.

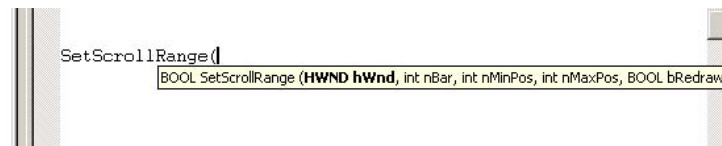


Figura 1.9. ToolTip-ul cu parametrii unei funcții

În acest mod, timpul de editare se reduce foarte mult, precum și posibilitatea de a introduce parametrii greșiți.

O altă facilitare pusă la dispoziția programatorilor de mediul de dezvoltare Visual C++ este aceea de salt rapid la definiția unei funcții. Prin intermediul toolbar-ului WizardBar, se poate sări la orice funcție din program, cu ajutorul unui drop-down box, așa cum se poate vedea în figura următoare:

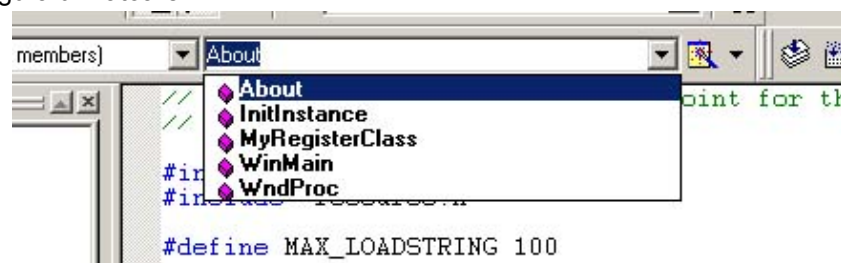


Figura 1.10. Funcțiile definite în cadrul unui fișier sursă

Când un fișier ce conține cod sursă este deschis, funcțiile definite în cadrul acestuia sunt recunoscute automat de către Visual C++, putând fi accesate rapid prin intermediul browser-ului din WizardBar.

### 1.6.4 Meniurile și Toolbar-urile

Prima dată când rulați Visual C++, sunt afișate trei toolbar-uri sub bara principală de meniuri. În mediul Visual C++ sunt disponibile foarte multe toolbar-uri predefinite. De asemenea, se pot crea toolbar-uri personalizate în scopul eficientizării dezvoltării programelor și al economiei de timp. Cele trei toolbar-uri inițiale sunt:

- **Toolbar-ul standard**, ce conține majoritatea uneltelor standard pentru deschiderea și salvarea fișierelor, operații de editare avansată (Cut, Copy, Paste) și o serie de alte comenzi care sunt utile majorității programatorilor.

- **Toolbar-ul WizardBar**, ce permite realizarea diverselor acțiuni asupra claselor și metodelor acestora, fără să fie necesar să deschideți ClassWizard-ul.

- **Minibar-ul Build**, ce oferă acces facil la comenzile de Build, Run și Compile ale unui proiect. Există și un toolbar Build, care are o serie de butoane suplimentare, care nu sunt neapărat necesare în lucrul uzual cu mediul Visual C++.

### 1.6.5 Rearanjarea mediului de dezvoltare Visual C++

Microsoft Developer Studio furnizează două moduri facile pentru rearanjarea mediului de dezvoltare, similare cu cele din alte aplicații Microsoft. Prin acționarea butonului dreapta al mouse-ului în zona unui toolbar se deschide un meniu de tip pop-up, așa cum se poate vedea mai jos. Acesta permite afișarea diverselor toolbar-uri implicite, puse la dispoziția utilizatorului de către Microsoft.



Figura 1.11. Toolbar-urile disponibile

O altă modalitate prin care se poate rearanja mediul de dezvoltare este prin “tragerea” cu mouse-ul a barei de la capătul din stânga al unui toolbar. Acest lucru este valabil în cazul toolbar-urilor ce se află direct sub meniu. În cazul în care se optează să se scoată aceste toolbar-uri de sub meniu, acestea devin ferestre asemănătoare cu cele din Windows, având un Title bar și buton de închidere. Fenomenul este reversibil, în sensul că un toolbar care este fereastră se poate insera lângă toolbar-urile de sub meniu prin “tragerea” acestuia în zona celor din urmă.

### 1.6.6 Help-ul

Așa cum am menționat în capitolul de instalare a MSDN-ului, help-ul oferit de Microsoft Visual Studio este furnizat prin colecția amintită.

Pentru a accesa help-ul, există un meniu în cadrul programului special destinat în acest sens. Se poate căuta o funcție, o constantă predefinită fie apelând la motorul de căutare, fie parcurgând o listă ordonată alfabetic.

Cea mai importantă facilitare a help-ului din Microsoft Visual Studio este help-ul de context. În cazul în care în zona de editare ați scris numele unei funcții, iar cursorul este fie la începutul cuvântului, fie la sfârșitul acestuia, fie în cadrul numelui scris, prin apăsarea tastei F1 se afișează o pagină de help a funcției sau o constantei respective.

## 1.7 Opțiunile globale în Visual C++

După ce am instalat Visual C++, MSDN-ul, am văzut principale ferestre și unelte de care ne vom folosi în dezvoltarea ulterioară a programelor, înainte de a deschide primul proiect, trebuie efectuată setarea opțiunilor globale ale Visual C++. Acesta este un pas semnificativ, deoarece toate proiectele ulterioare se vor baza pe setările efectuate acum.

Evident setările se pot modifica ulterior, dar în general, acestea se schimbă foarte rar și doar în cazul unor necesități speciale.

Fereastra pentru setarea opțiunilor se găsește în meniul Tool/Options. În general opțiunile setate implicit sunt utile pentru majoritatea utilizatorilor. O opțiune esențială se regăsește în tab-ul Editor din fereastra Options. Este vorba de grupa de check-box-uri "Save Option". Dacă "Save before running tools" nu este activată, atunci va trebui să salvați proiectul manual înaintea fiecărei execuții, pentru a fi luate în considerare ultimele modificări.

În caz contrar, dacă operați modificări asupra codului sursă, cu opțiunea mai sus menționată inactivă, și nu salvați manual proiectul, atunci modificările nu se vor regăsi în executabilul generat, acesta fiind de fapt rezultatul compilării ultimelor programe sursă salvate.

În fereastra de opțiuni globale se mai pot seta dimensiunea tab-ului și a alineatului, informații care să fie stocate în executabil pentru Debug, compatibilitate și altele.

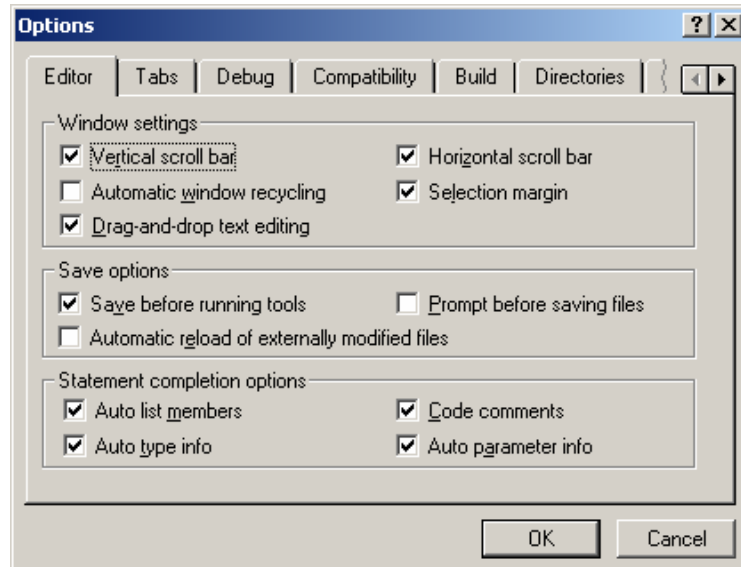


Figura 1.12. Fereastra de opțiuni globale

Trebuie menționat că opțiunile globale se aplică tuturor proiectelor noi generate sub această instalare de Visual C++. Pe lângă aceste opțiuni, fiecare proiect are propriile sale opțiuni, legate de librării incluse, căi către fișiere, etc., pe care le vom vedea în capitolele ulterioare.

## 1.8 Primul proiect în Visual C++

Pentru primul proiect în Visual C++, vom crea o aplicație simplă de tipul HelloWorld. În acest moment nu ne interesează structura programului. Ne vom axa pe partea de creare a aplicației, procedeu ce va fi utilizat mai târziu în dezvoltarea altor programe.

Pentru acesta sunt necesari următorii pași:

1. Crearea unui project workspace (spațiu de lucru al proiectului) nou.
2. Utilizarea Application Wizard pentru crearea mediului de dezvoltare al aplicației.
3. Selectarea opțiunilor care sunt furnizate de Wizard pentru a genera un cod sursă inițial cât mai apropiat de scopul aplicației pe care dorim să o creăm.

### 1.8.1 Crearea workspace-ului proiectului

Fiecare proiect de dezvoltare al unei aplicații necesită un workspace propriu în Visual C++. Workspace-ul include directoarele unde se găsește codul sursă, precum și directoarele unde sunt localizate fișierele de configurație și executabilele generate în urma compilării.

În general, în directoarele unei aplicații găsim următoarele fișiere:

- \*.dsw – fișierul workspace al proiectului. Acest fișier este creat automat când se generează un nou workspace, iar modificările asupra proiectului (opțiuni ale proiectului, modele de compilare, toolbar-uri active când se deschide proiectul, fișiere de cod sursă care se vor deschide când se deschide proiectul) sunt salvate în acest fișier. Trebuie menționat că în acest fișier nu sunt stocate

informații despre modificarea codului sursă în cadrul fișierelor ce alcătuiesc proiectul.

- \*.cpp – fișierele ce conțin cod sursă
- \*.h – fișierele de tip header
- \*.rc – fișierele de tip resursă. În aceste fișiere se salvează partea grafică a programului, care se desenează cu editorul grafic al Visual C++. Fișierul este text, și poate fi citit cu orice editor, cum ar fi Notepad.
- \*.ico – fișierele de tip icoană
- \*.plg – fișierul în care sunt salvate comenzile și mai ales switch-urile utilizate la compilare.
- în directorul Debug găsim executabilul, în cazul în care compilarea s-a finalizat cu succes și o serie de alte fișiere intermediare, rezultate în urma procesului de compilare.

Pentru portabilitate și pentru economie de spațiu, toate fișierele din directorul Debug se pot șterge, ele fiind generate automat în urma compilării. În cazul unui proiect, cele mai mari fișiere se găsesc în directorul Debug.

Pentru a crea un workspace și implicit un proiect nou, trebuie să parcurgeți următorii pași.

- a. Selectați File | New. Aceasta are ca rezultat deschiderea ferestrei următoare:

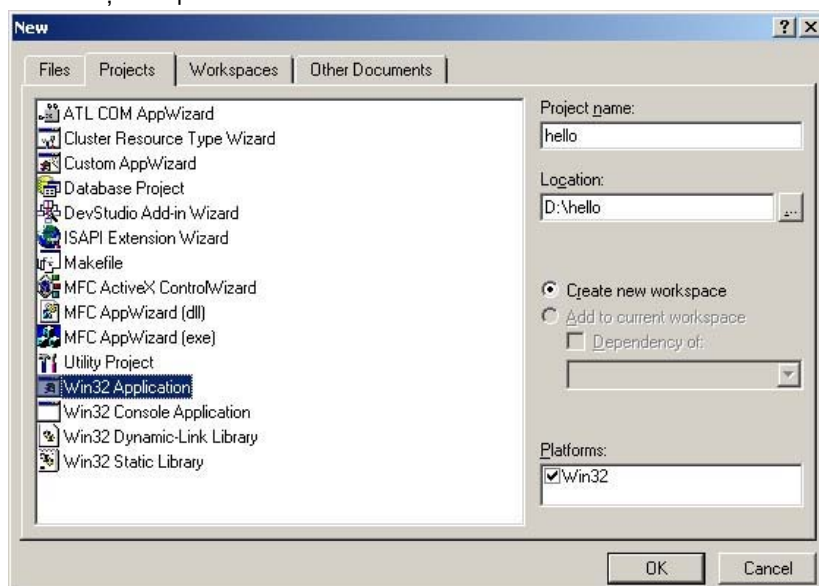


Figura 1.13.

- b. Din tab-ul Projects selectați tipul de aplicație dorit. După cum se poate observa și din figură, sunt disponibile mai multe tipuri: MFC AppWizard – exe sau dll, MFC ActiveX ControlWizard, Win32 Application, Win32 Console Application și altele. Selectați Win32 Application.
- c. Scrieți numele pe care doriți să îl atribuiți proiectului în câmpul Project Name. Am denumit aceasta prima aplicație "hello".
- d. Selectați directorul unde vreți să se creeze structura de directoare specifică proiectului.
- e. Acționați butonul Ok. Aceasta are ca rezultat crearea directoarelor aferente proiectului și pornirea AppWizard.

Trebuie avut în vedere că Visual C++ generează automat un director cu numele proiectului, în interiorul directorului selectat. Dacă dorim să creăm un proiect intitulat "Lab1" în directorul "My Documents" atunci trebuie să selectăm doar "My Documents" în câmpul cale, iar la Project Name să tastăm "Lab1". În general, câmpul de cale fiind mic, iar calea fiind lungă, nu se observă că în momentul când tastăm ceva în câmpul Project Name, se adaugă automat ce am tastat noi la calea directorului. De aceea, se întâmplă frecvent următoarea eroare: dacă avem un director "Lab1" în "My Documents", și se dorește ca proiectul nou creat să se găsească tot în directorul "Lab1", prin scrierea în câmpul Project Name a valorii "Lab1" și selecția căii "My Documents\Lab1", noul proiect se va găsi în directorul "My Documents\Lab1\Lab1". Pentru evitarea acestui caz, trebuie selectat doar "My Documents" și

în numele proiectului "Lab1", ce va avea ca rezultat generarea unui proiect lab1.dsw în directorul deja existent "My Documents\Lab1"

### 1.8.2 Utilizarea Application Wizard pentru a crea mediul de dezvoltare al programului

Wizard-ul care pornește după apăsarea butonului Ok din fereastra New Workspace diferă în funcție de tipul de aplicație ales. În cazul selecției Win32 Application, wizard-ul permite trei tipuri de proiecte:

- **Empty project** – creează un proiect gol, fără nici un fișier
- **Simple Win32 Application** – un proiect cu un fișier ce conține numai funcția WinMain, și care nu realizează nimic
- **"Hello World" Application** – un proiect ce deschide o fereastră și afișează textul Hello World.

În continuare ne vom axa pe acest exemplu, pe care îl selectăm, după care acționăm butonul Finish.

Înainte de a analiza proiectul, îl lansăm în execuție, pentru a vedea rezultatul. Acest lucru se poate realiza fie cu ajutorul comenzii Execute din meniul Build, fie cu combinația de taste Ctrl+F5. În acest pas, programul va întreabă, prin intermediul unei casete de dialog, dacă doriți să compilați sursele, la care se răspunde cu Yes. Dacă nu s-a întâlnit nici o eroare, rezultatul este următorul:

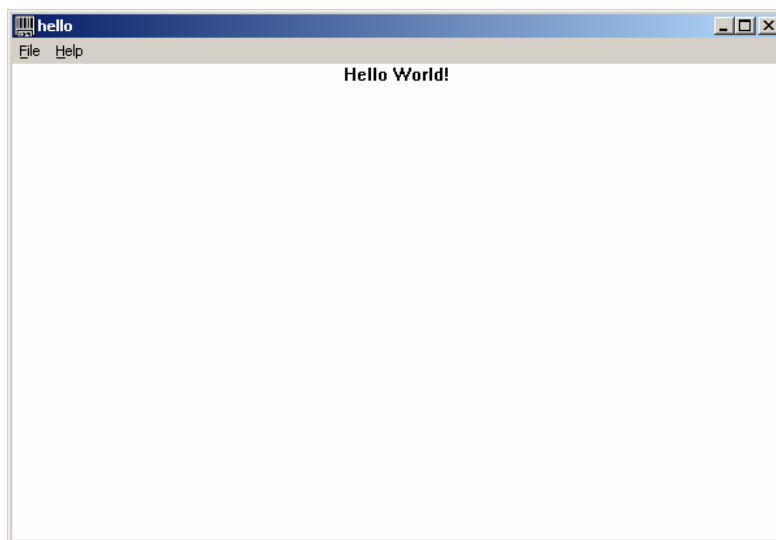


Figura 1.14. Aplicația Hello World

În cazul acestui proiect, nu a fost întâlnită nici o eroare la compilare și nu a fost generat nici un warning. Dacă programul are erori, atunci el nu se poate compila și nu este generat executabilul.

O altă situație este cea în care programul nu are erori și doar sunt generate warning-uri. Un exemplu de warning este cel în care o variabilă a fost declarată și nu a fost folosită în program sau în cazul unui cast implicit, în care o variabilă își pierde din caracteristici (devine short din long). Warning-urile pot cauza blocări ale aplicației sau chiar ale sistemului de operare. De aceea în programare este de preferat ca înainte de a termina o aplicație să vă asigurați că nu se generează nici un warning și nici o eroare la compilare.

### 1.8.3 Structura aplicației

După închiderea ferestrei, care este aplicația rezultată în urma compilării proiectului, se transferă controlul asupra mediului de dezvoltare. În cadrul acestei lucrări de laborator nu vom analiza codul sursă, ci ne vom axa pe structura programului generat automat.

În panoul workspace, avem cele trei tab-uri descrise în prima parte a laboratorului. În secțiunea ClassView, sub numele generic de "hello classes" se găsește un set de funcții, reunite sub numele "Global". Termenul de "classes", este o "moștenire" de la MFC. În cazul lui "Hello World", aplicația nu are nici o clasă, doar o serie de funcții și variabile globale. Funcțiile au un semn roșu în fața lor, și pe lângă numele acestora, observăm și tipurile de date ale parametrilor, în timp ce variabilele globale au un semn bleu și sunt afișate doar numele lor

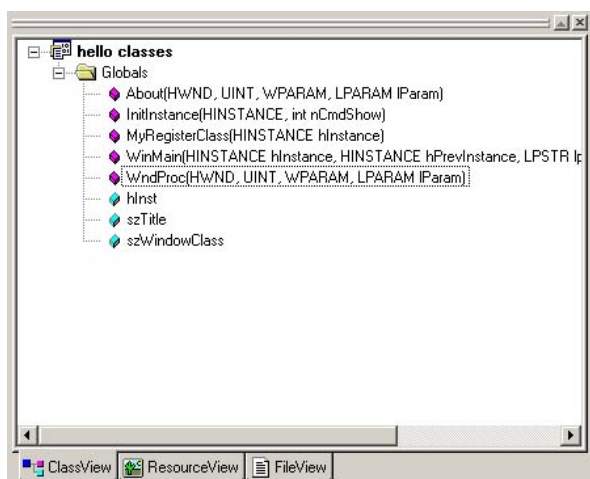


Figura 1.15. Secțiunea ClassView din workspace-ul proiectului “hello”

În secțiunea de resurse – ResourceView sunt grupate elementele grafice ale aplicației. Avem:

- grupul “Accelerator” – unde găsim informații despre combinațiile de taste care realizează diverse acțiuni în cadrul aplicațiilor
- grupul “Dialog” – unde sunt ferestrele de dialog care vor apărea în cadrul aplicației
- grupul “Icon” – icoanele aplicației. Trebuie menționat că o aplicație are două icoane:
  - o icoana mare - 32x32 pixeli, care este afișată în anumite condiții – spre exemplu dacă cream un shortcut pe desktop-ul Windows-ului sau în Windows Explorer când este selectată opțiunea View / Large Icons
  - o icoana mică - 16x16 pixeli, care este afișată în Title bar-ul aplicației sau în taskbar-ul Windows-ului
- grupul “Meniu” - unde se regăsește partea grafică a meniul aplicației
- grupul “String Table” - unde sunt reunite informații despre variabilele aplicației care comunică cu sistemul de operare, variabile care vor fi descrise mai pe larg în laboratoarele următoare.

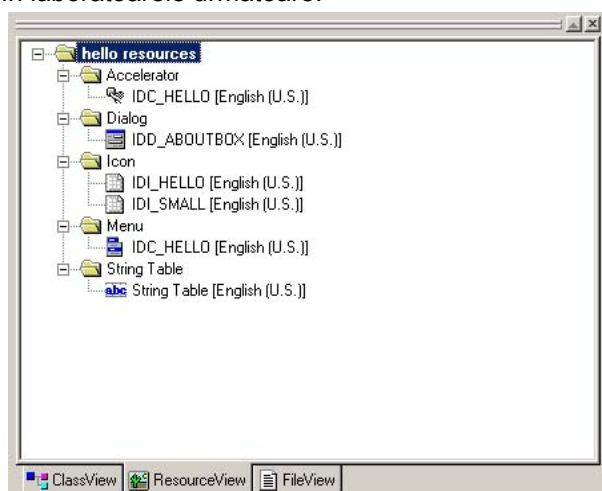


Figura 1.16. Tab-ul ResourceView

Ultima secțiune este cea denumită FileView, unde sunt grupate fișierele aplicației, în funcție de tipurile acestora. În acest sens avem grupurile:

- Source Files. Aici sunt reunite fișierele sursă, inclusiv cel ce conține codul sursă al resurselor grafice
- Header Files, în care sunt incluse fișierele de tip header
- Resource Files, unde găsim cele două icoane ale aplicației
- External Dependencies. În acest grup sunt plasate diverse fișiere externe, dacă este cazul



- Fișierul Readme.txt, care reprezintă un fișier de tip log, rezultat în urma generării programului de către AppWizard, dar care poate fi folosit pentru scrierea diverselor instrucțiuni de folosire a viitoarei aplicații.

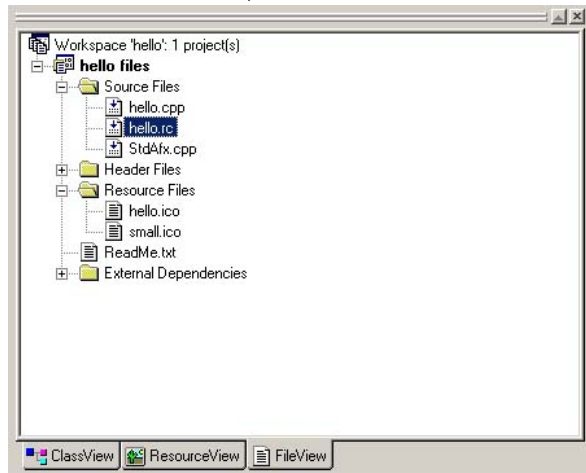


Figura 1.17. Tab-ul FileView

Utilizatorul poate adăuga fișiere manual în diverse grupuri. Este de preferat să se păstreze scopul fiecărui astfel de grup, pentru manipularea facilă a fișierelor în cadrul workspace-ului.

Pentru a adăuga un fișier într-un grup, se efectuează click dreapta pe numele grupului și se selectează opțiunea “Add Files to Folder”, așa cum se poate observa în figura următoare:

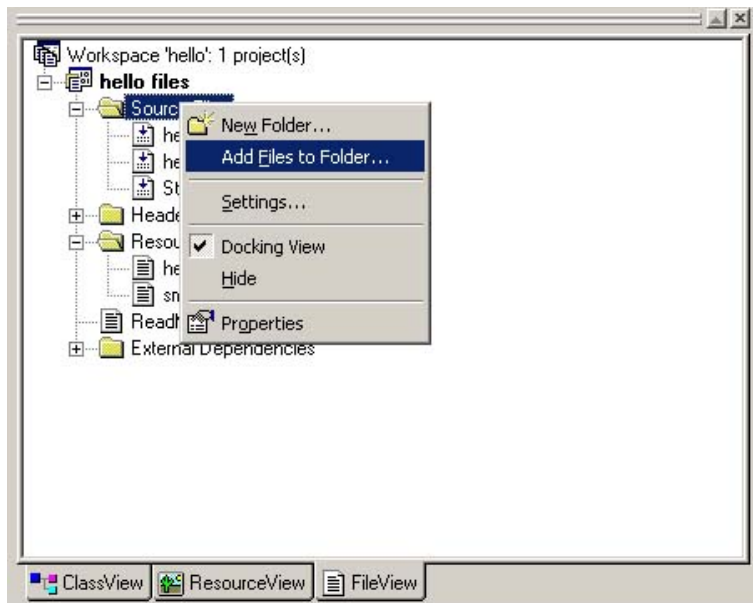


Figura 1.18: Adăugarea unor fișiere noi

După aceea se parcurge calea până la fișierul care se dorește a fi inserat în cadrul proiectului.

Este de preferat ca înainte de a adăuga un fișier sursă în cadrul unui proiect, să îl copiați manual în directorul curent al proiectului.

#### 1.8.4 Opțiunile proiectului

După cum menționam în capitolul “Opțiuni globale în Visual C++”, fiecare proiect are o serie de opțiuni caracteristice. Acestea se pot accesa din meniul Project/Settings sau cu combinația de taste Alt+F7.



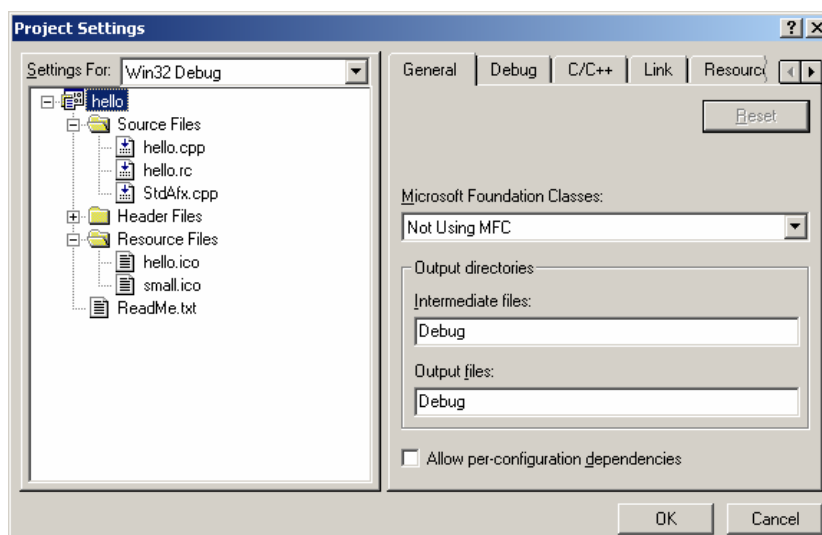


Figura 1.19: Fereastra Project Settings

În primul tab - General, puteți stabili dacă proiectul nu folosește clase MFC, sau dacă folosește clase MFC dintr-o bibliotecă statică sau dinamică. Tot aici se specifică directoarele unde vor fi plasate fișiere rezultate în urma compilării

Din tab-ul Debug se pot stabili argumentele de start ale programului, dacă este cazul. Tot aici se pot specifica căile către fișiere tip .dll folosite de aplicație.

În tab-ul C/C++ se pot stabili nivelele de mesaje de warning, optimizări. Aici se pot selecta limba și opțiunile de compilare. În ceea ce privește nivelul de warning, este recomandabil să rămână pe nivelul 3, care este opțiunea implicită. Nivelul 4 implică o sensibilitate a compilatorului sporită, în multe cazuri nejustificată.

Tab-ul unde se intervine cel mai des din partea utilizatorului, mai ales în cazul folosirii unor biblioteci externe, este tab-ul Link. Aici aveți o listă de fișiere tip .lib, care vor fi incluse la compilarea aplicației.

Ca să adăugați o librărie nouă la un proiect, mai întâi trebuie să o copiați manual în subdirectorul VC98\Lib din directorul unde aveți instalat Microsoft Visual Studio. După aceea, în tab-ul Link din Project Settings, în câmpul Object/library modules, apăsați tasta End pentru a ajunge la sfârșit. La sfârșitul rândului lăsați un spațiu și începeți să scrieți numele librăriei sau librăriilor noi pe care vreți să le adăugați în proiect, despărțite prin spațiu. Orice alt caracter în afara de spațiu între numele librăriilor va rezulta în producerea unei erori.

În cadrul tab-ului Custom Build se pot specifica comenzi, fișiere și parametri utilizați când se efectuează o compilare Custom.

Tab-urile Pre-Link Step și Post-Build Step sunt folosite pentru comenzi ce urmează să fie executate automat după crearea fișierelor .obj, dar înaintea generării executabilului, respectiv după ce procesul de compilare a fost completat cu succes.

### 1.8.5 Debug-ul programului

Unul dintre cele mai importante aspecte ale programării reprezintă partea de debug a unui program. De-a lungul timpului s-a dovedit imposibilitatea realizării unui proiect mediu fără nici o eroare. De aceea, procesul de debug permite rularea programului în mai multe moduri, astfel încât să poată fi urmărite rezultatele unor funcții și valorile unor variabile în decursul execuției unui program. Pentru a putea realiza debug-ul unui program, trebuie să păstrați valorile implicite din setările proiectului, precum și ale mediului de dezvoltare.

Principalele modalități de debug sunt:

- Step Into (F11), debugger-ul va executa linia curentă, și dacă este o funcție, va sări la funcția respectivă și va începe să execute liniile acelei funcții pas cu pas. Procesul acesta poate conduce în scurt timp la intrarea în diverse headere care sunt incluse în pachetul Visual Studio.
- Step Over (F10), la fel ca funcția de mai sus, numai că la întâlnirea unei funcții nu va intra în codul respectivei funcții, ci va întoarce doar rezultatul respectivei funcții. Se recomandă utilizarea celor două moduri împreună, în funcție de necesități.

- Step Out (Shift+F11). În cazul în care debugger-ul a intrat în codul unei funcții și se lansează această comandă, se va parcurge funcția automat până la sfârșit și se va întoarce pe un nivel mai sus.
- Run to cursor (Ctrl+F10), debugger-ul va rula până la linia unde este poziționat cursorul de către utilizator.

În cazul în care porniți compilarea unui program folosind una din opțiunile de mai sus, se va activa în bara de meniuri intrarea Debug, unde veți regăsi același opțiuni care sunt disponibile și în toolbar-ul Debug. În general, este mai facilă utilizarea shortcut-urilor sau cel mult a toolbar-ului Debug.

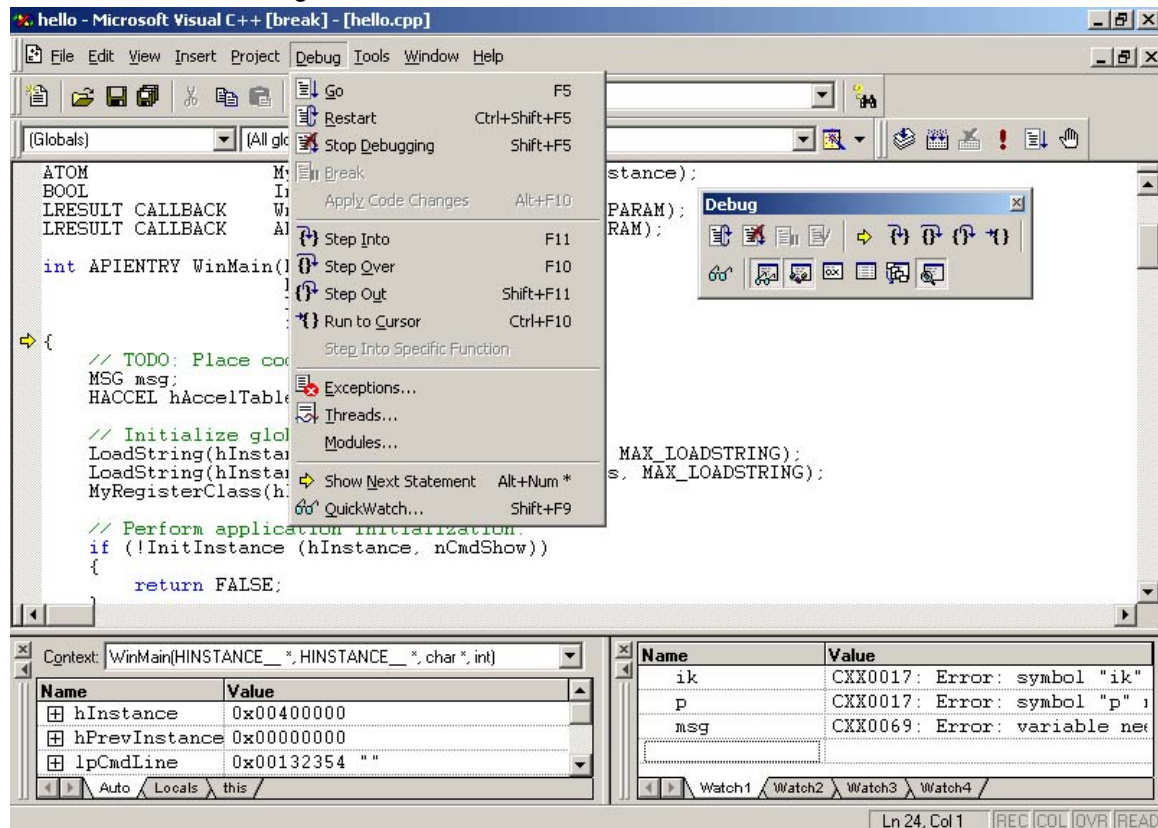


Figura 1.20. Modalități de acces a debugger-ului. În dreapta jos se poate observa fereastra Watch

O modalitate de întrerupere a execuției unui program într-un anumit punct definit de utilizator, cu posibilitatea continuării ulterioare a execuției este folosirea unui breakpoint. Un breakpoint se inserează în codul sursă prin selectarea liniei în care se dorește întreruperea programului și acționarea tastei F9. Pentru a utiliza breakpoint-ul nou creat, se folosește comanda Go din meniul Build/Start Debug sau tasta F5. Programul își va derula execuția normal până va întâlni breakpoint-ul, moment în care se va opri și va aștepta comenzile ulterioare ale programatorului. În acest pas se pot utiliza comenzile Step În, Out, Over.

Dacă ați utilizat mai multe breakpoint-uri, lista lor poate fi obținută fie cu Alt+F9, fie cu ajutorul meniul Edit/Breakpoints. În această listă se pot adăuga noi breakpoint-uri sau se pot șterge cele deja existente, fie pe rând, selectând unul și apoi tasta Del sau butonul Remove fie toate folosind Remove All.

Breakpoint-urile pot fi condiționale. Dacă dorim să urmărim valorile dintr-un ciclu de incrementare, atunci putem seta un breakpoint care să oprească execuția când o variabilă este egală cu o anumită valoare. Pentru aceasta setăm un breakpoint în linia respectivă, și apoi mergem în lista de breakpoint-uri cu Alt+F9. Când selectăm breakpoint-ul creat, observăm că se activează butonul Condition. Acționarea sa are ca efect deschiderea unei ferestre în care se pot stabili valorile la care să se activeze breakpoint-ul.

Procesul de debug este strâns legat de fereastra Watch. În fereastra Watch se adăuga, printr-un click, variabilele a căror valoare dorim să o urmărim pe parcursul rulării programului. Avantajul acestei ferestre este ca în fiecare moment știm valoarea variabilei care ne interesează.

## 1.9 Teme

1. Realizați un toolbar personalizat care să conțină icoanele de salvare a unui fișier, Copy, Paste și execuție a unui program.
2. Căutați în MSDN descrierea funcțiilor WinMain și WndProc
3. Modificați opțiunile Visual C++ pentru ca tab-ul și ident-ul să aibă doar 2 spații normale.
4. Se dau fișierele hello.cpp, StdAfx.cpp, hello.h, StdAfx.h, resource.h, hello.rc, hello.ico, small.ico. Să se creeze în directorul "My Documents" directorul "hello". În directorul hello să se creeze proiectul hello.dsw, care să conțină toate fișiere date și să se compileze proiectul.
5. Proiectul creat la punctul 4 să se parcurgă linie cu linie, folosindu-se Step Over. Să se urmărească în timpul execuției pas cu pas variabilele msg și hdc.

## Laborator 2. Primul program în Visual C++

### 2.1 Interfața grafică cu utilizatorul

Interfața grafică utilizează reprezentări grafice pentru a oferi informațiile utilizatorului. În general simbolurile și reprezentările grafice permit o mai bună utilizare a spațiului de lucru al monitorului precum și o prezentare mai intuitivă a informațiilor. Ecranul monitorului poate să devină o sursă de informații primite de la utilizator. Monitorul poate afișa diferite obiecte grafice sub forma icon-urilor sau obiecte (butoane și scroll bar-uri) care pot primi informații de la utilizator. Prin utilizarea tastaturii (sau, mai direct, prin utilizarea mouse-ului), utilizatorul poate acționa direct asupra acestor obiecte. Obiectele grafice pot fi "agățate", butoanele pot fi "apăsate" și scroll bar-urile pot ajuta la derularea textului sau graficelor. Utilizatorul poate interacționa direct cu obiectele de pe monitor.

#### 2.1.1 Interfața utilizator consistentă

Programele Windows au în general o interfață cu utilizatorul asemănătoare. Programele ocupă o suprafață rectangulară pe ecran numită "fereastră". Aplicația este identificată prin intermediul unei "caption bar", în care este afișată denumirea programului. Majoritatea funcțiilor unui program sunt accesibile prin intermediul unui "menu bar".

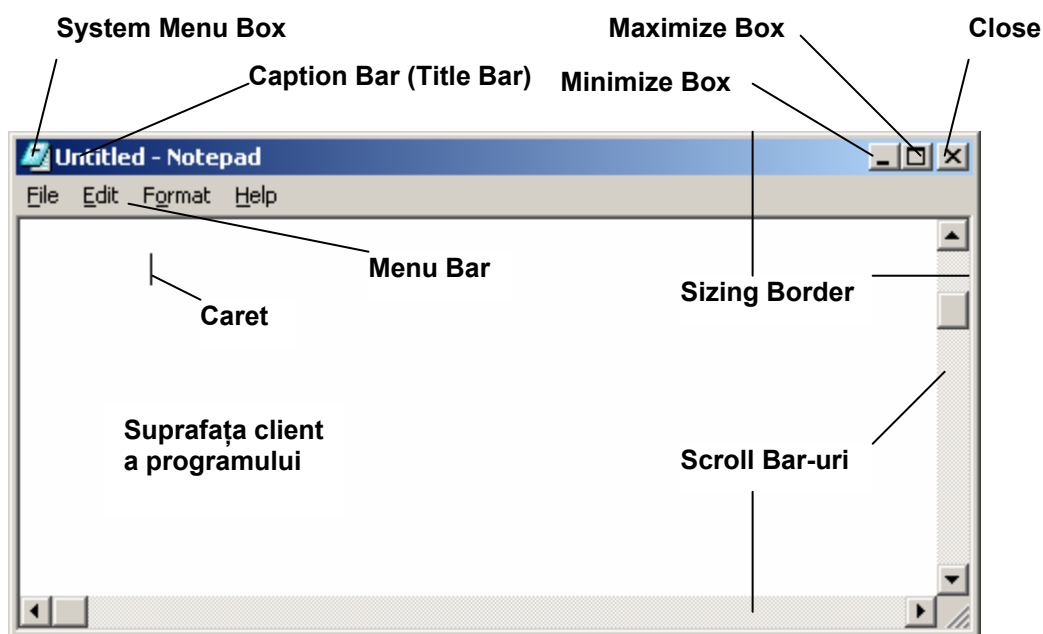


Figura 2.1. O fereastră tipică a unei aplicații Windows.

Din punctul de vedere al programatorului, utilizarea unei interfețe utilizator similare între programe este facilitată de multiplele funcții conținute de Windows pentru utilizarea meniurilor, ferestrelor de dialog și multe alte obiecte grafice predefinite. Toate meniurile și ferestrele de dialog au aceeași interfață de tastatură și de mouse deoarece aceasta este gestionată de Windows și nu de aplicații, și se face transparent pentru program.

### 2.1.2 Avantajul multitasking-ului

Mediul Windows oferă posibilitatea execuției simultane a mai multor aplicații. Fiecare aplicație poate ocupa o fereastră rectangulară pe ecran. Utilizatorul poate deplasa ferestrele pe ecran, să le modifice dimensiunile, să modifice fereastra activă, să comute rapid între diferitele programe, precum și să schimbe informații între aplicațiile care se execută simultan.

### 2.1.3 Gestiunea memoriei

Un sistem de operare nu poate implementa un mediu multitasking fără a realiza o gestiune adecvată a memoriei. Pe măsură ce programele sunt lansate în execuție și altele sunt oprite, zonele de memorie pe care acestea le ocupă pentru memorarea datelor sau a instrucțiunilor devin fragmentate. Sistemul trebuie să fie în măsură să mențină spațiul de memorie liber. Acest lucru necesită ca sistemul de operare să poată muta blocuri întregi de memorie. Programele care se execută sub Windows pot avea o dimensiune a codului executabil chiar mai mare decât ar putea încapa în memorie la un moment dat. Windows poate să renunțe la porțiuni de cod aflate în memorie, urmând ca ulterior, când este necesară execuția sa, acesta să fie reîncărcat din fișierul executabil. Utilizatorul poate executa mai multe ferestre ale aceleiași aplicații (numite "instanțe"). Toate aceste instanțe împart aceeași zonă de memorie pentru instrucțiuni ("cod"), iar zonele de memorie pentru date sunt unice pentru fiecare instanță. Programele executate sub Windows pot să apeleze funcții care se află în alte fișiere executabile denumite "Biblioteci legate dinamic" ("Dynamic Link Libraries" - DLL), Windows include un mecanism de legare a apelurilor din program cu funcțiile aflate în aceste DLL, la momentul execuției aplicației. Chiar însuși Windows nu este altceva decât o colecție de DLL-uri.

### 2.1.4 Interfață grafică independentă de "device" (periferic)

Windows este o interfață grafică și ca urmare toate aplicațiile care se execută sub Windows utilizează integral reprezentările grafice ale informațiilor atât pe monitor, cât și pentru tipărirea pe imprimantă, plotter etc. Interfața grafică este nu numai atractivă pentru utilizator, dar poate să prezinte mult mai multe informații pentru utilizator.

Programele scrise pentru Windows nu au acces direct la nivelul hardware al calculatorului pentru a afișa informațiile pe ecran sau imprimantă. Windows include un limbaj de programare grafică (denumit "Graphics Device Interface" - GDI) care înlesnește afișarea de grafice și texte formate. Un program scris pentru Windows va putea fi executat indiferent de ce tip de placă video conține sistemul sau de tipul de imprimantă instalată. Programul nu are nevoie să determine tipul de periferic instalat în sistem.

### 2.1.5 Apelul funcțiilor Windows

Fiecare funcție Windows are un nume semnificativ scris prin alăturarea de litere mari și mici, ca de exemplu "CreateWindow". Această funcție, după cum sugerează numele său, creează o fereastră pentru program. Un alt exemplu este "IsClipboardFormatAvailable", care determină dacă Clipboard-ul conține un anumit tip de format.

Toate funcțiile Windows sunt declarate în fișiere header, care sunt apelate tot din fișierul Windows.h, din directorul Microsoft Visual Studio\VC98\Include. Spre exemplu, Windows.h include și headerul Winuser.h, unde sunt majoritatea declarațiilor tipurilor de date și funcțiilor folosite în mod uzual în programarea Windows. Aceste funcții pot fi utilizate într-un program C în același mod în care se utilizează funcțiile de bibliotecă clasice ca de exemplu "strlen". Exista totuși câteva deosebiri între funcțiile Windows și funcțiile de bibliotecă standard.

Toate funcțiile Windows sunt declarate utilizând modificatorii tip "far" și "pascal", chiar dacă nu explicit. Acești modificatori sunt incluși în diverse tipuri de date, pe care le găsim înaintea funcțiilor.

Modificatorii tip "far" indică faptul că funcția respectivă se află într-un alt segment de cod decât segmentul de cod al aplicației. Cu puține excepții, orice referință furnizată ca argument al unei funcții Windows trebuie să fie o referință far. În mod normal compilatorul extinde automat referințele near la referințe far în cadrul argumentelor de apel ale funcțiilor, pe baza prototipurilor de funcții declarate în header-ele apelate de Windows.h.

Modificatorii tip "pascal" indică faptul că secvența de apel a funcției este diferită de secvența normală de apel a unei funcții C. În mod normal compilatorul C generează un cod de apel al funcțiilor care împinge în stivă valorile variabilelor de apel începând cu argumentul cel mai din dreapta și sfârșind cu primul argument al funcției. În cazul funcțiilor care au

modificatori tip "pascal", argumentele sunt împinse în stivă de la primul către ultimul argument al funcției, iar funcția apelată are datoria să descarce stiva. Secvența de apel pascal este utilizată în Windows datorită faptului că este mai eficientă, furnizând o viteză sporită de rulare. În mod normal în C, datorită posibilității existenței funcțiilor cu număr variabil de parametri, la fiecare apel de funcție se fac o serie de pași în ceea ce privește parametrii funcției (număr, poziția în lista de parametrii). Acești pași conduc la o viteză scăzută a programului.

### 2.1.6 Legarea dinamică (Dynamic Linking)

O aplicație Windows realizează interfața cu funcțiile Windows printr-un mecanism denumit "legare dinamică", diferit de modul de realizare a interfeței cu funcțiile MS-DOS (prin folosirea întreruperilor software). Putem vorbi în Windows de "Noul format executabil". De fiecare dată când un program cheamă o funcție Windows, compilatorul C generează codul în limbaj de asamblare pentru apelul funcției respective. Un tabel în fișierul executabil identifică funcția apelată prin intermediul unui nume de "Dynamic Link Library" – DLL, numele funcției sau un număr unic asociat cu numele funcției ("numărul ordinal al funcției")

Windows este constituit în mare parte din trei DLL-uri denumite:

- KERNEL - gestionează memoria sistemului, încărcarea și execuția programelor, planificarea task-urilor;
- USER - interfața cu utilizatorul și gestiunea ferestrelor;
- GDI - interfața grafică.

Aceste biblioteci dinamice conțin instrucțiunile și datele necesare funcțiilor Windows. Aceste biblioteci pot fi găsite în subdirectorul Windows\System sau Windows\System32, în funcție de sistemul de operare.

În momentul încărcării unui program în memorie, în apelurile "far" către funcțiile Windows este modificată adresa de salt astfel încât ea să conțină adresele punctelor de intrare ale funcțiilor din bibliotecile utilizate (bibliotecile dinamice sunt încărcate la rândul lor, dacă ele nu existau deja în memorie, utilizate de către o altă aplicație). Acesta este motivul pentru care toate funcțiile Windows sunt declarate "far". Funcțiile din bibliotecile dinamice se află într-un segment de cod diferit de cel al programului care le cheamă. Referințele trimise prin argumentele de apel ale funcțiilor sunt de asemenea "far", pentru a evita referirea lor în segmentul de date al bibliotecii dinamice.

În majoritatea cazurilor, apelul funcțiilor Windows este transparent pentru programator. Compilatorul C generează tot codul necesar apelului dinamic. Din punctul de vedere al programatorului, acesta apelează o funcție clasică C.

În momentul legării (link-ării) unei aplicații Windows, este folosit un modul special, denumit "biblioteca de import" (import library). Aceasta import library conține numele DLL-urilor, numele și numerele ordinale ale funcțiilor pentru toate funcțiile Windows. Link-editor-ul utilizează aceste informații pentru a construi tabelele din fișierul executabil necesare pentru rezolvarea apelurilor în momentul execuției.

### 2.1.7 Programarea orientată pe obiect

Deși Windows nu impune utilizarea unui compilator orientat pe obiect - programele pot fi scrise, în general, în limbaj clasic C, stilul de încapsulare a "obiectelor" Windows și posibilitatea de virtualizare a funcțiilor care interacționează cu aceste obiecte, este mai apropiată de o filosofie de programare orientată pe obiect decât de un stil de programare clasic.

Cel mai frecvent "obiect" cu care lucrează o aplicație este fereastra. Aceasta poate să primească evenimente provocate de către utilizator prin intermediul mouse-ului sau tastaturii și poate să afișeze ieșirea sub forma unor grafice în interiorul său.

O fereastră a unei aplicații conține, în general, titlul aplicației, menu-ul, marginea pentru dimensionare (sizing border) și eventual scroll bar-uri. Ferestrele de dialog sunt ferestre adiționale. În general suprafața unei ferestre de dialog conține mai multe ferestre adiționale, denumite ferestre copil (child windows). Aceste ferestre sunt sub forma butoanelor, check box-uri, radio buttons-uri și câmpuri de introducere de text, list box-uri sau scroll bar-uri.

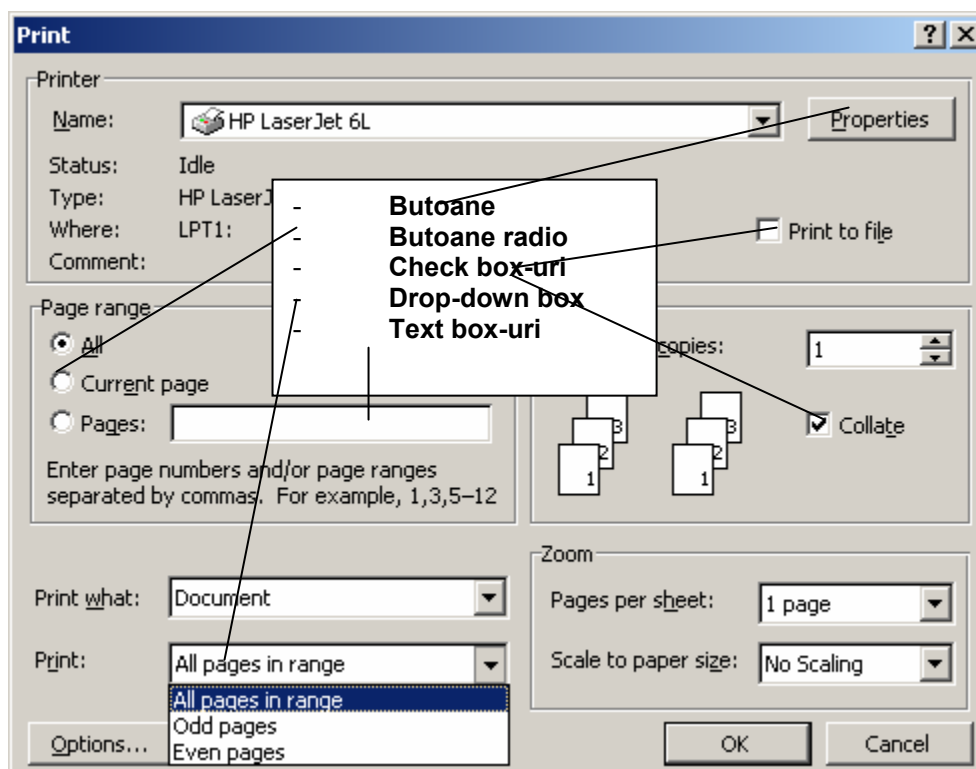


Figura 2.2. Componente fundamentale în Windows

Utilizatorul percepe aceste ferestre ca obiecte pe ecran și interacționează direct cu ele (apăsând un buton de exemplu). Perspectiva programatorului este analoagă cu cea a utilizatorului. Fereastra primește intrările de la utilizator sub forma unor "mesaje" și poate utiliza aceste "mesaje" în scopul de a comunica cu alte ferestre.

Înțelegerea acestor mesaje este una din dificultățile care trebuie întâmpinate pentru a putea proiecta aplicații pentru Windows.

### 2.1.8 Arhitectura aplicațiilor conduse de evenimente (mesaje)

În momentul în care o fereastră a unei aplicații (un editor de texte, de exemplu), este redimensionată, aplicația respectivă poate reacționa astfel încât să se adapteze noilor dimensiuni impuse de utilizator. Amănuntele legate de redimensionarea unei ferestre sunt realizate de către sistemul de operare. Aplicația în schimb trebuie să "știe" faptul că fereastra în care se execută a fost redimensionată și să reacționeze corespunzător.

În Windows, aplicația a cărei fereastră a fost redimensionată primește un mesaj de la sistemul de operare, mesaj care îi indică faptul că dimensiunile ferestrei s-au modificat precum și noile dimensiuni cărora trebuie să li se conformeze. Aplicația poate apoi să modifice conținutului grafic al ferestrei astfel încât acesta să încapă în noile dimensiuni.

Prin afirmația că "aplicația primește un mesaj de la sistemul de operare" se înțelege faptul că acesta (sistemul de operare) apelează o funcție din interiorul programului, trimițând prin intermediul parametrilor de apel informațiile necesare. Această funcție se află în interiorul programului și este denumită WndProc - procedura de fereastră (window procedure).

### 2.1.9 Procedura de fereastră

În aplicațiile scrise pentru sistemul de operare MS-DOS, este normal ca programul să facă apeluri la funcțiile de sistem. În acest mod se poate deschide un fișier, se poate citi tastatura sau se poate afișa informația pe ecran. În Windows (și în general în arhitectura aplicațiilor orientate pe obiecte), frecvent mecanismul este invers și anume, sistemul de operare cheamă proceduri din interiorul programului.

Fiecare fereastră creată de către un program are asociată o procedură de fereastră. De fiecare dată când Windows trimite un mesaj către aplicația care conține fereastra respectivă, este apelată procedura de fereastră. Procedura de fereastră realizează anumite activități în funcție de tipul mesajului recepționat după care se întoarce și cedează controlul înapoi sistemului de operare care a chemat-o.



Mai exact, o fereastră este creată întotdeauna pe baza unei "clase de fereastră" ("window class"). Clasa de fereastră conține adresa procedurii de fereastră care procesează mesajele către acel tip de fereastră. Utilizarea unei "clase de fereastră" permite ca mai multe ferestre (denumite stări sau instanțe ale clasei) să fie create pe baza unor caracteristici comune, memorate în "clasa de fereastră". În cadrul acestor caracteristici este inclusă și adresa procedurii de fereastră, deci este posibil ca prelucrarea evenimentelor trimise către ferestre diferite (provenite din aceeași clasă) să fie realizată de către aceeași funcție. De exemplu toate butoanele din Windows sunt bazate pe aceeași clasă de ferestre, denumită "BUTTON", și a cărei procedură de fereastră se află localizată într-o bibliotecă dinamică de sistem (denumită "User.exe "). Acesta este motivul pentru care toate butoanele din Windows arată și reacționează în același fel.

În programarea orientată pe obiecte, un "obiect" este o combinație de instrucțiuni (reprezentate prin metode) și date (câmpurile obiectului). O fereastră este un obiect (în sensul clasic al cuvântului). Instrucțiunile (metodele) sunt reprezentate de procedura de fereastră, iar câmpurile sunt menținute de către Windows pentru fiecare fereastră și clasă de ferestre existente în sistem. Este posibil să se modifice adresa procedurii de fereastră în structura oricărei clase, în care să se înscrie adresa unei funcții furnizate de către utilizator (de exemplu se poate modifica procedura de fereastră a clasei "BUTTON", și astfel mesajele provenite de la acțiunile asupra tuturor butoanelor din Windows vor trece prin noua funcție de fereastră introdusă). Prin acest mecanism se poate realiza virtualizarea metodelor din programarea orientată pe obiecte (tot într-un sens clasic).

O procedură de fereastră procesează mesajele primite de către o fereastră. De obicei aceste mesaje informează fereastra respectivă asupra acțiunilor utilizatorului de la tastatură sau de la mouse. În acest mod o fereastră buton știe faptul că a fost "apăsă". Alte mesaje pot să informeze fereastra în momentul în care i-au fost modificate dimensiunile sau când suprafața sa trebuie redesenată.

În momentul lansării în execuție a unui program Windows, sistemul de operare creează o "coadă de mesaje" asociată aplicației respective. Aceasta coadă de mesaje conține toate mesajele către toate ferestrele ce sunt create de către acest program.

Fiecare program conține o secvență de cod numită "message loop" - bucla de mesaje, care extrage mesajele din aceasta coadă și le trimite către procedurile de fereastră cărora le-a fost destinat mesajul. Alte mesaje pot fi trimise direct către procedurile de fereastră fără ca acestea să fie plasate în coada de mesaje a aplicației.

## 2.2 Primul program Windows

Clasicul program descris de către Brian Kernighan și Denis Ritchie în cartea "The C Programming Language", constituie tradiționalul punct de plecare în programarea utilizând limbajul C.

Versiunea pentru Windows este mult mai mare ca dimensiuni. Programul HelloWin conține două funcții principale și dimensiunea codului este de peste 80 de linii C. Majoritatea acestor linii de program reprezintă "house-keeping"-ul solicitat de Windows. În general, orice aplicație Windows se axează pe acest program, diferențele fiind în multe cazuri nesemnificative.

Programul HelloWin creează o fereastră cu textul "Hello, Windows!". Diferențele între acest program, și cel prezentat în laboratorul anterior (programul generat implicit de Visual C) constau din:

- simplificarea sursei, prin renunțarea la funcții
- lipsa meniurilor și a castelor de dialog

În directorul aferent laboratorului 2 găsiți fișierul Lab2.dsw, care este workspace-ul acestei aplicații. Programul creează o fereastră, așa cum se poate observa în figura următoare. Fereastra aplicației afișează în centrul suprafeței client mesajul "Hello, Windows!".



Figura 2.3. Rezultatul programului HelloWin

Fereastra afișată conține un grad de funcționalitate foarte mare pentru cele 80 de linii de program necesare pentru a scrie aplicația. Title bar-ul poate fi "prins" (drag) cu mouse-ul pentru a modifica poziția ferestrei pe ecran, este posibilă redimensionarea ferestrei. Când fereastra principală a aplicației își schimbă dimensiunile, programul rezonează automat mesajul astfel încât el se va afla în centrul suprafeței client. Puteți apăsa butonul de maximizare pentru ca fereastra să ocupe întreaga suprafață a ecranului. Prin apăsarea butonului de minimizare, se comprimă toată fereastra într-un icon. Toate aceste opțiuni pot fi comandate din Sysmenu - meniul sistem și în plus puteți alege comanda "Close" pentru a închide aplicația.

### 2.2.1 Fișierul HelloWin.c

Fișierul HelloWin.c conține două funcții: "WinMain" și "WndProc". WinMain este punctul de intrare în program și este echivalentul funcției standard "main" din C-ul clasic. Orice program Windows trebuie să aibă o funcție WinMain, precum orice program C scris pentru MS-DOS trebuie să conțină o funcție main.

WndProc este procedura de fereastră pentru fereastra principală a aplicației HelloWin. Aceasta funcție procesează mesajele trimise de sistemul de operare către fereastră. WndProc nu este chemată direct din programul HelloWin, WndProc este chemată doar de către Windows. Singura referință către WndProc este în definirea clasei aplicației, care se face în WinMain.

### 2.2.2 Identificatorii scriși cu majuscule

Identificatorii scriși cu majuscule în programul sursă sunt constante definite în Winuser.h, majoritatea acestor identificatori sunt prefixați de două sau trei litere majuscule urmate de un semn de subliniere. Acești identificatori sunt: CS\_HREDRAW, CS\_VREDRAW, IDI\_APPLICATION, IDC\_ARROW, WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, WM\_PAINT, WM\_DESTROY, DT\_SINGLELINE, DT\_CENTER, DT\_VCENTER.

Acestea sunt constante numerice. Prefixul indică o categorie generală căreia acești identificatori îi aparțin:

Prefix	Categoria
CS	Class Style (stil de clasă)
IDI	ID (identificator) pentru un Icon
IDC	ID pentru un Cursor (mouse)
WS	Window Style (stil de fereastră)
CW	Create Window (creare de fereastră)
WM	Window Message (mesaj pentru procedura de fereastră)
DT	Draw Text (desenare de text).

### 2.2.3 Tipuri de date

Windows definește în Winuser.h o serie de noi tipuri de date. Acestea sunt:

Tip de date	Semnificație
LRESULT	echivalent cu far. Rezultatul procesării unui mesaj
CALLBACK	echivalent cu Pascal
WINAPI	Convenție folosită pentru Win32API
UINT	Unsigned INT – întreg fără semn
WPARAM	echivalent cu unsigned long int (întreg pe 32 de biți fără semn)
LPARAM	echivalent cu long int (întreg pe 32 de biți cu semn)
LPSTR	echivalent cu char far * (pointer far către un șir de caractere).

Aceste tipuri sunt definite pentru ușurarea eventualei portabilități a aplicațiilor sub Windows. Noile tipuri de date sunt proiectate astfel încât să mențină consistența programelor indiferent de arhitectura procesorului pe care ar fi executate.

HelloWin utilizează de asemenea patru structuri de date definite în Winuser.h. Acestea sunt:

Structura	Semnificație
MSG	Structura de mesaj
WNDCLASS	Structura de clasa de fereastră
PAINTSTRUCT	Structura de informații de desenare
RECT	Structura de dreptunghi

Primele două dintre aceste structuri sunt utilizate în WinMain pentru a defini două variabile: "msg" și "wndclass". Următoarele două sunt utilizate în WndProc pentru a defini variabilele "ps" și "rect", necesare pentru apelul unor funcții Windows.

### 2.2.4 Despre handle-uri

În fișierul sursa HelloWin.c exista trei identificatori pentru diferite tipuri de "handle-uri":

Identificator	Semnificație
HINSTANCE	handle al unei instanțe
HWND	handle al unei ferestre
HDC	handle al unui Device Context (context de periferic)

Handle-urile sunt folosite foarte frecvent în Windows. Un Handle este un număr pe 16 biți care este asociat unic unui obiect. Putem asimila un handle ca un identificator al unui obiect. Handle-urile din Windows sunt asemănătoare handle-urilor folosite de către MS-DOS pentru funcțiile de gestiune a fișierelor. Handle-urile sunt în general obținute de către un program prin apelul unei funcții Windows. Programul poate utiliza handle-ul astfel obținut în apelul altor funcții Windows, pentru a face referința la obiectul respectiv. Valoarea handle-ului este irelevantă pentru program, dar pentru Windows el reprezintă de fapt un pointer near într-un segment de date intern către structura care definește obiectul utilizat.

### 2.2.5 Notăția Ungară

Se observă că anumite variabile din programul HelloWin.c au anumite nume particulare. Un exemplu este lpszCmdParam utilizat ca argument formal al funcției WinMain.

Majoritatea programatorilor în Windows folosesc o convenție de notație a variabilelor cunoscută sub denumirea de "Notăția Ungară". Numele variabilei începe cu un șir de caractere minuscule care indică tipul variabilei respective (prefixul lpsz înseamnă "long pointer to a string zero terminated")

Prefixul "h" din hInstance sau hPrevInstance este utilizat pentru a indica tipul "handle"; n din nCmdShow este utilizat pentru "number" și reprezintă în mod normal un întreg.

Pentru denumirea variabilelor de tip structură se poate folosi denumirea structurii ca prefix sau ca denumire a variabilei sau o prescurtare a acesteia (folosind litere minuscule). De exemplu msg este de tipul MSG, iar wndclass este de tipul WNDCLASS, rect este de tipul RECT, iar ps este de tipul PAINTSTRUCT.

Notăția Ungară este utilă pentru evitarea erorilor de programare, deoarece denumirea unei variabile reprezintă atât utilizarea variabilei respective cit și tipul ei.

Prefixele uzuale ale variabilelor sunt:

Prefix	Tip de date
C	Char
By	BYTE (unsigned char)
N	short sau int
I	Int
x, y	short sau int (coordonate ecran)
cx, cy	short sau int (lungimi pe x sau y c provine de la "count")
B	BOOL (int)
W	WORD (unsigned int)
L	LONG (long int)
Dw	DWORD (unsigned long int)
Fn	Funcție
S	String
Sz	string terminat cu byte-ul 0

### 2.2.6 Punctul de intrare în program

Fișierul sursă începe cu o directivă `#include` necesară pentru a citi fișierul header `Windows.h`, care conține declarațiile simbolurilor publice definite de `Windows`. Acest fișier include la rândul sau alte headere, inclusiv `Winuser.h`, ce conține declarații pentru funcțiile, structurile, tipurile de date și constantele `Windows`. Trebuie menționat că dacă schimbăm `Windows.h` cu `Winuser.h` se vor genera o serie de erori.

Urmează declararea prototipului procedurii de fereastră, necesară datorită faptului că în `WinMain` se face o referință la ea:

**LRESULT CALLBACK WndProc (HINSTANCE, UINT, WPARAM, LPARAM);**

Punctul de intrare pentru un program `Windows` este o funcție denumită `WinMain`. Aici începe execuția programului (de fapt funcția `WinMain` este punctul de intrare pentru porțiunea de program scrisă de către utilizator; compilatorul C inserează un cod de intrare în fișierul executabil. Codul de intrare, după realizarea inițializărilor necesare aplicației cheamă funcția `WinMain` cu parametrii corespunzători). `WinMain` este totdeauna definită astfel:

**int WINAPI WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdParam, int nCmdShow )**

Parametrul `hInstance` este denumit "instance handle". Acesta este un handle (număr) care identifică unic programul în momentul în care se execută sub `Windows`. Este posibil ca utilizatorul să execute simultan mai multe copii ale aceleiași aplicații. De exemplu un utilizator ar putea porni de mai multe ori programul `Clock`. Fiecare copie a aplicației `Clock` este denumită o "instanță", iar fiecare instanță are o valoare diferită a parametrului `hInstance`. Instanța este echivalentul PID-ului (process ID), frecvent în sistemele de operare multitasking.

Parametrul `hPrevInstance` ("previous instance") este handle-ul de instanță a celei mai recente instanțe a aceluiași program, anterior lansată în execuție, dar încă activă. Dacă nici o copie a programului respectiv nu se execută în momentul respectiv, `hInstance` va avea valoarea 0 sau `NULL`.

Parametrul `lpszCmdParam` este un pointer far către un șir de caractere care conține linia de comandă a programului.

Parametrul `nCmdShow` este un număr care indică modul în care fereastra trebuie afișată inițial. În general nu este necesară testarea valorii acestui parametru, el fiind doar o indicație către program asupra modului în care ar trebui să se afișeze inițial. Acest parametru poate lua valori între 1 și 11, adică între `SW_SHOWNORMAL` și `SW_MAX`, constante definite în `Winuser.h` și care indică faptul că fereastra aplicației trebuie afișată normal, respectiv sub formă maximizată. Acest parametru în mod normal este trecut funcției `ShowWindow` (prefixul `SW_` din denumirea constantelor vine de la funcția `ShowWindow`, căreia îi este destinat acest parametru).

### 2.2.7 Înregistrarea clasei de fereastră

O fereastră este întotdeauna creată pe baza unei clase de fereastră. Clasa de fereastră conține adresa procedurii care procesează mesajele de fereastră.

Mai multe ferestre pot fi create pe baza aceleiași clase de fereastră. De exemplu toate butoanele Windows sunt create pe baza unei singure clase. Clasa de fereastră definește, pe lângă procedura de fereastră, și alte caracteristici generale ale tuturor ferestrelor care vor fi create pe baza ei. În momentul creării unei ferestre se definesc caracteristici adiționale ale ferestrei și care sunt specifice ferestrei create.

Înainte de creării unei ferestre trebuie înregistrată clasa de fereastră căreia aceasta îi aparține. Înregistrarea clasei se face cu ajutorul funcției `RegisterClass`. Funcția `RegisterClass` cere un singur argument: un pointer către o structură de tipul `WNDCLASS`. Structura `WNDCLASS` este definită în `Winuser.h` astfel:

```
typedef struct tagWNDCLASS
{
    UINT          style;
    WNDPROC       lpfnWndProc;
    int           cbClsExtra;
    int           cbWndExtra;
    HINSTANCE     hInstance;
    HICON         hIcon;
    HCURSOR       hCursor;
    HBRUSH        hbrBackground;
    LPCSTR        lpstrMenuName;
    LPCSTR        lpstrClassName;
}WNDCLASS;
```

În `WinMain` trebuie definită o variabilă de tipul `WNDCLASS`, în general astfel:

```
WNDCLASS wndclass;
```

După inițializarea celor zece câmpuri conținute de aceasta variabilă, este necesar apelul funcției **`RegisterClass`**, pentru înregistrarea clasei definite:

```
RegisterClass (&wndclass);
```

Este necesară înregistrarea clasei doar de către prima instanță a unui program. Clasa astfel înregistrată devine astfel disponibilă tuturor instanțelor ulterioare ale aceluși program. Din această cauză `HelloWin` inițializează câmpurile structurii **`wndclass`** și cheamă `RegisterClass` doar dacă `hPrevInstance` este egală cu `NULL`.

Structura `WNDCLASS` are zece câmpuri. Cel mai importante sunt `lpfnWndProc` și `lpstrClassName`. Câmpul `lpfnWndProc` conține adresa procedurii de fereastră pentru toate ferestrele create pe baza acestei clase (care în cazul `HelloWin` este funcția `WndProc`). Toate celelalte câmpuri conțin caracteristici comune ale ferestrelor bazate pe această clasă de ferestre.

Instrucțiunea:

```
wndclass.style = CS_HREDRAW | CS_VREDRAW;
```

combină doi identificatori "class style" cu ajutorul operatorului SAU pe biți. În `Winuser.h`, identificatori pentru stilul de clasă (prefixați cu `CS_`) sunt constante pe 16 biți care au un singur bit setat. De exemplu `CS_VREDRAW` este definit ca `0x0001`, iar `CS_HREDRAW` este definit ca `0x0002`. Identificatorii astfel definiți se mai numesc și "flag-uri pe biți". Flag-urile pe biți pot fi combinați cu ajutorul operatorului SAU.

Acești doi identificatori pentru stilul de clasă indică faptul că toate ferestrele create pe baza acestei clase vor fi complet redesenate dacă dimensiunea orizontală (`CS_HREDRAW`) sau cea verticală (`CS_VREDRAW`) suferă modificări. Dacă modificați dimensiunile ferestrei lui `HelloWin`, puteți observa faptul că textul este redesenat în centrul ferestrei. Cei doi identificatori specificați asigură această comportare a ferestrei.

Cel de-al doilea câmp al structurii `WNDCLASS` este inițializat cu instrucțiunea:

```
wndclass.lpfnWndProc = WndProc;
```

Această instrucțiune asociază procedura de fereastră pentru această clasă funcției `WndProc`. Procedura de fereastră va procesa toate mesajele către toate ferestrele create pe baza acestei clase. Prefixul `lpfn` are, în notația Ungară, semnificația: "long pointer to function"

Următoarele două instrucțiuni:

```
wndclass.cbClsExtra = 0;  
wndclass.cbWndExtra = 0;
```

pot alocă un spațiu suplimentar de memorie în structura de clasă, respectiv în structura de fereastră. Aceste structuri sunt menținute intern de către Windows. Un program poate utiliza aceste zone de memorie pentru scopuri proprii, memorând în ele diferite informații suplimentare. HelloWin nu utilizează aceasta facilități și de aceea valoarea celor două câmpuri este inițializată cu zero. Prefixul cb provine de la "count bytes" - număr de octeți.

Următorul câmp reprezintă handle-ul instanței programului (provenit din parametrii funcției WinMain) care realizează înregistrarea clasei.

```
wndclass.hInstance = hInstance;
```

Instrucțiunea:

```
wndclass.hIcon = LoadIcon (NULL, IDI_APPLICATION);
```

asociază un icon pentru toate ferestrele create pornind de la această clasă. Iconul este un desen mic (pentru monitorul de tip VGA, el are întotdeauna dimensiunea 32 x 32 pixeli), și care este afișat în momentul minimizării programului. Iconurile pot fi create de către programator. Pentru exemplul de față s-a utilizat un icon predefinit de către Windows.

Pentru obținerea unui handle către iconul predefinit a fost necesar apelul funcției LoadIcon cu primul parametru având valoarea NULL. Al doilea parametru este un identificator prefixat cu IDI\_ ("ID Icon") definit în Winuser.h. Funcția LoadIcon returnează un handle către acest icon. Acest handle este folosit doar prin atribuirea valorii lui câmpului hIcon din structura wndclass. Câmpul hIcon din structura WNDCLASS este de tipul HICON (adică "handle la ICON").

Instrucțiunea:

```
wndclass.hCursor = LoadCursor (NULL, IDC_ARROW);
```

este asemănătoare cu instrucțiunea anterioară. Rolul ei este de a asocia un cursor de mouse clasei înregistrate. Când cursorul de mouse se află în interiorul suprafeței client a unei ferestre aparținând acestei clase, va căpăta forma specificată de către acest câmp. Funcția LoadCursor încarcă un cursor predefinit (deoarece primul câmp are valoarea NULL) și identificat prin intermediul constantei IDC\_ARROW, apoi returnează un handle către acel cursor. Valoarea acestui handle este atribuită câmpului hCursor. Cursor-urile de mouse pot, de asemenea, să fie create de către programator.

Următorul câmp specifică culoarea fundalului suprafeței client a ferestrelor create pe baza acestei clase. Prefixul hbr al câmpului hbrBackground provine de la "handle la Brush". Un "brush" este un obiect grafic care specifică culorile pixelilor utilizați pentru umplerea unei suprafețe. Windows deține o serie de brush-uri standard (sau de "stock"). Funcția GetObject returnează un handle către un brush alb:

```
wndclass.hbrBackground = GetStockObject (WHITE_BRUSH);
```

Această instrucțiune specifică faptul că fundalul suprafeței client pentru toate ferestrele provenite din wndclass va avea culoarea albă.

Următorul câmp specifică meniul clasei de ferestre. HelloWin nu posedă un meniu al aplicației și deci acestui câmp i se va atribui valoarea NULL.

```
wndclass.lpszMenuName = NULL;
```

În sfârșit trebuie să asociem acestei clase un nume. Numele clasei este cel care identifică unic o anumită clasă înregistrată. Dacă un program înregistrează mai multe clase, atunci ele trebuie să aibă obligatoriu nume diferite. Acest nume va trebui folosit în momentul creării unei ferestre pentru a se specifica faptul că ea aparține clasei respective. Clasei definite în programul HelloWin i s-a atribuit numele aplicației, adică valoarea variabilei szAppName: "HelloWin":

```
wndclass.lpszClassName = szAppName;
```

După inițializarea tuturor celor zece câmpuri ale structurii wndclass, HelloWin înregistrează această clasă prin apelul funcției RegisterClass. Singurul parametru al acestei funcții este o referință către structura WNDCLASS înregistrată:

```
RegisterClass (&wndclass);
```

### 2.2.8 Crearea ferestrei principale a aplicației

Clasa de fereastră definește caracteristicile generale ale unei ferestre. Acest lucru permite ca o singură clasă de ferestre să fie folosită pentru crearea mai multor ferestre diferite. O fereastră se creează efectiv prin apelul funcției `CreateWindow`, prin care se specifică informațiile specifice ferestrei create. În comparație cu funcția `RegisterClass`, funcția `CreateWindow` nu specifică o structură de apel, diferitele informații asociate ferestrei sunt trimise ca valori ale argumentelor de apel ale funcției. Iată secvența prin care `HelloWin` creează fereastra principală a aplicației:

```

hwnd = CreateWindow (
    szAppName,                //numele clasei de fereastră
    "The Hello Program",      //caption-ul (titlul) ferestrei
    WS_OVERLAPPEDWINDOW,     //stilul ferestrei
    CW_USEDEFAULT,           //poziția x inițială
    CW_USEDEFAULT,           //poziția y inițială
    CW_USEDEFAULT,           //dimensiunea orizontală inițială
    CW_USEDEFAULT,           //dimensiunea verticală inițială
    NULL,                     //handle-ul ferestrei părinte
    NULL,                     //handle-ul meniului ferestrei
    hInstance,               //instanța programului
    NULL);                    //parametrii de creare

```

Deși înregistrarea clasei se face doar de către prima instanță a programului, fereastra principală a programului este creată de către fiecare instanță. Fiecare instanță a unui program are propria fereastră asociată, și toate aceste ferestre aparțin clasei de ferestre înregistrate.

Primul parametru al funcției `CreateWindow` are valoarea `szAppName`, adică numele clasei care tocmai am înregistrat-o. În acest mod Windows identifică clasa căreia îi aparține fereastra creată (adică prin compararea dintre primul argument al funcției `CreateWindow` și câmpul `lpClassName` din structura tuturor claselor înregistrate până la momentul respectiv).

Al doilea parametru specifică titlul ferestrei. În cazul nostru acest titlu va fi afișat de către Windows în caption bar-ul asociat aplicației.

Fereastra creată de către aplicație este o fereastră normală conținând un caption-bar, un system menu box în partea stângă, un minimize box și un maximize box în dreapta caption-ului și o margine care permite redimensionarea suprafeței client. Acesta este stilul standard al unei ferestre și este specificat de către identificatorul `WS_OVERLAPPEDWINDOW` (din `Winuser.h`), a cărui valoare este atribuita celui de-al treilea argument de apel.

Următoarele patru argumente reprezintă coordonata inițială (x, y) pe ecran a colțului din dreapta sus al ferestrei precum și dimensiunile (x, y) inițiale ale ferestrei. Toate aceste valori sunt specificate în număr de pixeli. Prin trimiterea valorii identificatorului `CW_USEDEFAULT` (definit în `Winuser.h` și având valoarea `0x8000`), se indică faptul că se dorește utilizarea valorilor implicite pentru poziționarea ferestrei pe ecran.

Parametrul "handle-ul ferestrei părinte" are valoarea `NULL` deoarece aceasta este fereastra principală a aplicației. Când există o relație "părinte" - "copil" între două ferestre, fereastra copil va trebui să trimită valoarea handle-ului ferestrei părinte, ca argument. Ferestrele "copii" apar totdeauna în interiorul suprafeței ferestrei "părinte".

Argumentul "handle-ul meniului ferestrei" are valoarea `NULL` deoarece fereastra nu dispune de un meniu asociat.

Argumentul "instanța programului" are valoarea handle-ului de instanță primit ca valoare a argumentului `hInstance` al funcției `WinMain`.

În sfârșit, pointerul către "parametrii de creare" a ferestrei are valoarea `NULL`. Acest argument poate fi utilizat pentru trimiterea de informații suplimentare în momentul creării ferestrei (pentru scopuri specifice aplicației).

Funcția `CreateWindow` întoarce un handle către fereastra creată. Valoarea acestui handle este memorată în variabila `hwnd` care are tipul `HWND` (handle la window). Fiecare fereastră existentă în Windows are un handle unic. Programul poate utiliza această valoare ca argument al funcțiilor referitoare la o anumită fereastră. Multe funcții Windows necesită un argument de tipul `HWND` astfel încât Windows să identifice fereastra vizată de funcția respectivă. Dacă un program creează mai multe ferestre fiecare din aceste ferestre va avea un handle diferit.



### 2.2.9 Afișarea ferestrei

După revenirea din funcția `CreateWindow`, avem o fereastră care a fost creată intern de către Windows. Pentru a afișa fereastra pe ecran este necesar apelul a două noi funcții. Prima este:

#### **ShowWindow (hwnd, nCmdShow);**

Primul argument este handle-ul de fereastră primit ca rezultat al apelului funcției `CreateWindow`. Parametrul `nCmdShow` determină modul de afișare al ferestrei pe ecran și este trimis ca argument al funcției `WinMain`. Funcția `ShowWindow` afișează fereastra pe ecran, dacă parametrul `nCmdShow` are valoarea `SW_SHOWNORMAL`. Apelul funcției:

#### **UpdateWindow (hwnd);**

determină desenarea suprafeței client a ferestrei. Funcția realizează desenarea suprafeței client a ferestrei prin trimiterea unui mesaj `WM_PAINT` către procedura de fereastră (funcția `WndProc` din `HelloWin.C`).

### 2.2.10 Bucla de mesaje

După apelul funcției `UpdateWindow`, fereastra este vizibilă pe ecranul calculatorului. Programul trebuie acum să poată reacționa la mesajele de la mouse și de la tastatură care pot să apară de la utilizator în timpul execuției. Windows menține o coadă de mesaje pentru fiecare program care se execută. În momentul apariției unui eveniment, Windows "traduce" acest eveniment într-un mesaj pe care apoi îl introduce în coada de mesaje a aplicației.

O aplicație poate extrage aceste mesaje din coadă în interiorul unei secvențe de cod cunoscută sub denumirea de "bucla de mesaje":

```
while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}
return msg.wParam;
```

Variabila `msg` este o structură de tipul `MSG`, declarată în `Winuser.h` după cum urmează:

```
typedef struct tagMSG
{
    HWND          hwnd;
    UINT          message;
    WPARAM        wParam;
    LPARAM        lParam;
    DWORD         time;
    POINT         pt;
} MSG;
```

Apelul funcției `GetMessage` care începe bucla de mesaje extrage un mesaj din coada de mesaje a aplicației:

#### **GetMessage (&msg, NULL, 0, 0)**

Apelul funcției `GetMessage` trimite o referință far către variabila `msg` care are tipul `MSG`. Al doilea, al treilea și al patrulea argument de apel au valoarea `NULL` sau zero pentru a indica faptul că programul dorește să extragă toate mesajele către toate ferestrele create de către program. Windows completează câmpurile structurii `MSG` cu informațiile referitoare la primul mesaj din coada de mesaje. Câmpurile acestei structuri sunt:

- **hwnd**: handle-ul ferestrei către care este îndreptat mesajul. În programul `HelloWin` acest câmp are valoarea `hwnd` deoarece acesta este handle-ul singurei ferestre a aplicației.
- **message**: identificatorul de mesaj (tipul mesajului). Acesta este un număr care identifică mesajul. Pentru fiecare mesaj există un identificator asociat definit în `Winuser.h`. Toți acești identificatori au prefixul `WM_` ("window message"). De exemplu dacă pointerul de mouse se poziționează în suprafața client a ferestrei "HelloWin" și se apasă butonul stânga al mouse-ului, Windows va introduce în coada de mesaje a aplicației un mesaj al cărui

câmp "message" va avea valoarea identificatorului WM\_LBUTTONDOWN (definit în Winuser.h, cu valoarea 0x0201).

- **wParam, lParam** : valori ale căror semnificații depind de tipul de mesaj.
- **time**: momentul de timp la care mesajul a fost introdus în coada de mesaje a aplicației.
- **pt** - coordonatele mouse-ului la momentul în care mesajul a fost introdus în coadă.

Dacă câmpul message al mesajului extras este diferit de valoarea WM\_QUIT (adică 0x0012), funcția GetMessage întoarce o valoare diferită de zero. Un mesaj WM\_QUIT determină întoarcerea unei valori zero ca rezultat al funcției GetMessage provocând terminarea ciclului while și deci a buclei de procesare de evenimente. Programul se termină returnând valoarea câmpului wParam al structurii msg.

Instrucțiunea:

**TranslateMessage (&msg);**

trimite mesajul către Windows pentru a realiza anumite translații de taste apăsate.

Instrucțiunea:

**DispatchMessage (&msg);**

trimite iar mesajul înapoi către Windows. Windows identifică procedura de fereastră asociată ferestrei cu handle-ul hwnd, și apoi cheamă această procedură cu argumentele hwnd, message, wParam, lParam din structura de mesaj. În cazul nostru procedura de fereastră este funcția WndProc din HelloWin.c. După ce WndProc procesează mesajul, revine în funcția DispatchMessage care a chemat-o. Windows revine din funcția DispatchMessage, după care bucla de procesare de mesaje continuă cu o nouă iterație.

### 2.2.11 Procedura de fereastră

Ceea ce s-a discutat până acum a reprezentat mai mult sau mai puțin house-keeping-ul necesar oricărei aplicații. Clasa de fereastră a fost înregistrată; apoi, pe baza ei a fost creată fereastra principală a aplicației, aceasta fereastră apoi a fost afișată, iar programul a intrat într-o buclă de mesaje.

Toate acțiunile importante au loc în procedura de fereastră. Procedura de fereastră este cea care determină ceea ce programul va afișa în suprafața client și cum va reacționa aplicația la acțiunile utilizatorului.

În HelloWin, procedura de fereastră este funcția WndProc. O procedură de fereastră poate avea orice denumire. Un program Windows poate conține oricâte proceduri de fereastră. O procedură de fereastră este totdeauna asociată cu o clasă de fereastră care a fost înregistrată prin apelul funcției RegisterClass.

Definiția procedurii de fereastră trebuie să fie de forma:

**LRESULT CALLBACK WndProc (HWND hwnd, UINT message,  
WPARAM wParam, LPARAM lParam);**

Cele patru argumente ale funcției sunt identice cu primele patru câmpuri ale structurii MSG.

Primul argument este hwnd, și reprezintă handle-ul ferestrei care recepționează mesajul. Pentru un program ca HelloWin, acesta este handle-ul unicii ferestre create de program. Dacă un program creează mai multe ferestre pe baza aceleiași clase de ferestre (și deci având aceeași procedură de fereastră), atunci hwnd identifică fereastra care recepționează mesajul respectiv.

Cel de-al doilea argument este un număr (un întreg fără semn pe 16 biți) care identifică tipul mesajului. Ultimele două argumente (wParam și lParam) furnizează informații suplimentare despre mesaj. Aceste argumente se numesc "parametrii de mesaj". Conținutul acestor argumente este specific fiecărui tip de mesaj.

### 2.2.12 Procesarea mesajelor

Fiecare mesaj primit de către procedura de fereastră este identificat de un număr, valoarea argumentului message al procedurii de fereastră. Winuser.h declară identificatori prefixați cu WM\_ ("window message") pentru toate valorile posibile ale argumentului message.

În cazul general programatorii folosesc o structură switch / case pentru a determina tipul mesajului recepționat de către procedura de fereastră, și pentru a-l putea procesa corespunzător. Când procedura de fereastră primește un mesaj pe care îl procesează, ea

trebuie să întoarcă valoarea zero. Toate mesajele care nu sunt procesate de către procedura de fereastră trebuie trimise către o funcție Windows denumită DefWindowProc. Funcția DefWindowProc realizează procesarea implicită pentru aceste mesaje. Valoarea întoarsă de către DefWindowProc trebuie să fie întoarsă și ca rezultat al procedurii de fereastră.

În HelloWin.c, WndProc alege să proceseze doar două tipuri de mesaje: WM\_PAINT și WM\_DESTROY. Procedura de fereastră este structurată astfel:

```
switch (message)
{
    case WM_PAINT:
        /* procesarea mesajului WM_PAINT */
        return 0;
    case WM_DESTROY:
        /* procesarea mesajului WM_DESTROY */
        return 0;
}
return DefWindowProc (hwnd, message, wParam, lParam);
```

Apelul funcției DefWindowProc în cazul mesajelor neprocesate este esențial pentru execuția corectă a aplicației.

### 2.2.12.1 Mesajul WM\_PAINT

Primul mesaj procesat de către WndProc este mesajul WM\_PAINT. Acest mesaj este foarte important pentru un program Windows. Mesajul WM\_PAINT comunică programului faptul că o parte sau întreaga suprafață client a fost "invalidată" și este necesară redesenarea ei.

O regiune a suprafeței client poate deveni invalidă din mai multe cauze:

- în momentul creării ferestrei, întreaga suprafață client este invalidă deoarece nu s-a desenat încă nimic în fereastră. Primul mesaj WM\_PAINT provocat de apelul funcției UpdateWindow determină procedura de fereastră să deseneze în suprafața client;
- în momentul modificării dimensiunilor ferestrei, suprafața client devine invalidă. Datorită faptului că câmpului style din structura wndclass i-a fost atribuită valoarea CS\_HREDRAW | CS\_VREDRAW, Windows invalidează suprafața client de fiecare dată când dimensiunea orizontală sau verticală a ferestrei suferă modificări.
- când aplicația este minimizată, iar apoi refăcută la dimensiuni normale, Windows nu salvează conținutul zonei grafice a ferestrei. Dimensiunea zonei de memorie necesară pentru această operație ar fi prea mare. Windows invalidează întreaga suprafață client a ferestrei. Procedura de fereastră primește un mesaj WM\_PAINT, și își reface conținutul ferestrei.
- când peste suprafața client a ferestrei se suprapune fereastra altei aplicații, suprafața grafică astfel acoperită nu este salvată. În momentul descoperirii mai târziu a suprafeței acoperite, aceasta regiune devine invalidă. Procedura de fereastră primește ca urmare un mesaj WM\_PAINT pentru a reface porțiunea descoperită.

Windows șterge suprafața regiunii invalide, înainte de a trimite către procedura de fereastră mesajul WM\_PAINT. Ștergerea se realizează prin utilizarea brush-ului specificat în câmpul hbrBackground al structurii de clasă a ferestrei. În cazul lui HelloWin, acesta este un brush alb din "stock", ceea ce înseamnă că Windows șterge fundalul zonei invalide colorând-o cu culoarea albă.

Procesarea mesajului WM\_PAINT începe de obicei cu apelul funcției BeginPaint:

```
hdc = BeginPaint (hwnd, &ps);
```

și se încheie prin apelul funcției EndPaint:

```
EndPaint (hwnd, &ps);
```

Primul argument al celor două funcții este un handle de fereastră iar al doilea este o referință la o variabilă de tipul PAINTSTRUCT. Structura PAINTSTRUCT (declarată în Winuser.h) conține informații pe care procedura de fereastră le poate utiliza pentru a desena în suprafața client.

BeginPaint întoarce un handle către un "device context" (context de periferic). Un device context conține informațiile referitoare la un anumit periferic fizic (cum ar fi display-ul)

și la driverul de periferic asociat. Orice operațiune grafică asupra suprafeței client (desene sau afișare de texte) necesită acest handle al contextului perifericului. Utilizând handle-ul de device context întors de către funcția `BeginPaint`, se evită desenarea (chiar și intenționată) în afara suprafeței client a ferestrei. Funcția `EndPaint` eliberează handle-ul la device context astfel încât acesta devine invalidă. `EndPaint` validează de asemenea întreaga suprafață client a ferestrei.

După apelul funcției `BeginPaint`, `WndProc` cheamă funcția `GetClientRect`:

**`GetClientRect (hwnd, &rect);`**

Primul parametru este handle-ul ferestrei programului. Cel de-al doilea parametru este adresa variabilei `rect` definită în `WndProc` și având tipul `RECT`.

Structura `RECT` este un "rectangle". Aceasta structură (definită în `Winuser.h`) are patru câmpuri de tip întreg denumite `left`, `top`, `right` și `bottom`. Funcția `GetClientRect` atribuie valorile coordonatelor dreptunghiului care delimitează zona client a ferestrei celor patru câmpuri ale acestei structuri. Câmpurile `left` și `top` au totdeauna valoarea zero. Câmpurile `right` și `bottom` au valorile dimensiunilor orizontale, respectiv verticale ale dreptunghiului client, exprimate în pixeli.

`WndProc` utilizează această structură pentru a o trimite ca parametru funcției `DrawText`:

**`DrawText (hdc, "HelloWindows!", -1, &rect,  
DT_SINGLELINE | DT_CENTER | DT_VCENTER);`**

Funcția `DrawText` afișează textul specificat în interiorul dreptunghiului specificat drept al patrulea parametru. Datorită faptului că această funcție desenează ceva pe ecran, ea cere ca primul parametru un handle la device context. Acest handle este valoarea returnată de către apelul funcției `BeginPaint`. Cel de-al doilea argument este șirul de caractere care trebuie desenat, al treilea parametru are valoarea `-1`, specificând faptul că textul este un șir de caractere terminat cu valoarea zero.

Ultimul parametru este o serie de "flag-uri pe biți", care indică ce caracteristici de desenare are textul specificat. În acest caz, flag-urile determină afișarea textului pe o singură linie, centrat orizontal și vertical în interiorul dreptunghiului specificat ca al patrulea parametru. Apelul acestei funcții provoacă afișarea șirului de caractere "Hello, Windows!", centrat în suprafața client a ferestrei.

De fiecare dată când suprafața client devine invalidă (când se modifică dimensiunile orizontale sau verticale ale acesteia, de exemplu), Windows șterge fundalul ferestrei și `WndProc` primește un nou mesaj `WM_PAINT`. `WndProc` obține noile coordonate ale suprafeței client prin apelul funcției `GetClientRect` și afișează din nou textul centrat în noua zonă client.

### 2.2.12.2 Mesajul `WM_DESTROY`

Mesajul `WM_DESTROY` este un alt mesaj important. Acest mesaj indică faptul că Windows este în curs de a distruge fereastra, ca urmare a unei comenzi de la utilizator. Mesajul este rezultatul selectării comenzii `Close` din meniul sistem al programului sau apăsării tastelor `ALT+F4`, sau al unui `double-click` în `System menu box`.

`HelloWin` răspunde acestui mesaj printr-o modalitate standard, apelând:

**`PostQuitMessage (0);`**

Această funcție inserează un mesaj `WM_QUIT` în coada de mesaje a aplicației. Funcția `GetMessage` chemată în bucla de procesare de mesaje a aplicației, extrage un mesaj `WM_QUIT` din coada de mesaje și întoarce valoarea zero. Ca urmare a acestui fapt testul buclei `while` se evaluează la condiția `FALSE`, iar bucla de mesaje încetează, iar programul este părăsit.

## 2.3 Teme

1. Căutați fișierele `Windows.h`, `Winuser.h` în directorul Visual C++
2. Căutați constantele `IDC_ARROW`, `IDI_APPLICATION` în `Winuser.h`
3. Schimbați cursorul în aplicația `HelloWindows`, astfel încât în suprafața client acesta să aibă forma clepsidrei.
4. Schimbați icoana aplicației.
5. Modificați fișierul `Hellowin.c`, astfel încât fereastra inițială să fie afișată la punctele de coordonate `200,200` și să aibă dimensiunea `150x300` pixeli.

## Laborator 3 - Desenarea textelor

### 3.1 Aplicația fundamentală Windows

Concluzionând cele prezentate în laboratorul anterior, o aplicație fundamentală Windows, care realizează funcționalitatea de bază a unui program Windows, este creată din următoarele componente:

1. Includerea fișierului Windows.h, necesar pentru declararea identificatorilor Windows. Acesta se introduce în prima linie a programului prin directiva

```
#include <windows.h>
```

2. Declararea procedurii de fereastră și a funcțiilor de prelucrare de mesaje. În cazul aplicației fundamentale, un singur mesaj trebuie prelucrat și anume mesajul WM\_DESTROY. Prelucrarea acestui mesaj se realizează prin intermediul funcției WMDestroy. În cazul mai general în care se realizează prelucrarea mai multor mesaje, oricărui mesaj de forma WM\_Xxxx i se asociază o funcție de procesare de mesaj denumită WMXxxx, având prototipul identic cu al funcției WMDestroy. Atât procedura de fereastră, cât și funcțiile de procesare de mesaje trebuie declarate în prima parte a programului:

```
LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);  
long WMDestroy (HWND hwnd, WPARAM wParam, LPARAM lParam);
```

pentru a putea fi utilizate ulterior.

3. Funcția InitClass realizează inițializarea câmpurilor structurii de clasă de fereastră:

```
void InitClass (WNDCLASS* wndclass, HINSTANCE hInstance,  
               char* szAppName)  
{  
    wndclass->style           = CS_HREDRAW | CS_VREDRAW;  
    wndclass->lpfnWndProc     = WndProc;  
    wndclass->cbClsExtra      = 0;  
    wndclass->cbWndExtra      = 0;  
    wndclass->hInstance       = hInstance;  
    wndclass->hIcon           = LoadIcon (NULL, IDI_APPLICATION);  
    wndclass->hCursor         = LoadCursor (NULL, IDC_ARROW);  
    wndclass->hbrBackground   = GetStockObject (WHITE_BRUSH);  
    wndclass->lpszMenuName     = NULL;  
    wndclass->lpszClassName    = szAppName;  
}
```

4. Funcția InitApplication realizează inițializarea primei instanțe a unei aplicații Windows prin apelul funcției InitClass ce inițializează corespunzător structura de clasă, după care este apelată funcția Windows RegisterClass, care are ca efect înregistrarea clasei de fereastră. InitApplication întoarce valoarea TRUE dacă a reușit înregistrarea clasei de fereastră respectiv FALSE în caz contrar:

```
BOOL InitApplication (HINSTANCE hInstance, char* szAppName)  
{  
    WNDCLASS wndclass;  
    InitClass (&wndclass, hInstance, szAppName);  
    return RegisterClass (&wndclass);  
}
```

5. Funcția InitInstance realizează inițializarea instanței curente a programului. Funcția InitInstance creează fereastra principală a aplicației și o afișează pe

ecran. În cazul în care fereastra principală a fost creată, funcția `InitInstance` întoarce valoarea `TRUE`. În caz contrar se întoarce valoarea `FALSE`:

**BOOL InitInstance (HINSTANCE hInstance, char\* szAppName, int nCmdShow)**

```
{
    HWND      hwnd;
    hwnd = CreateWindow (    szAppName,
                            "Aplicație Windows",
                            WS_OVERLAPPEDWINDOW,
                            CW_USEDEFAULT,
                            CW_USEDEFAULT,
                            CW_USEDEFAULT,
                            CW_USEDEFAULT,
                            NULL,
                            NULL,
                            hInstance,
                            NULL);

    if (hwnd == NULL)
        return FALSE;
    ShowWindow (hwnd, nCmdShow);
    UpdateWindow (hwnd);
    return TRUE;
}
```

6. Funcția `Applnit` realizează secvența de operațiuni necesare inițializării unui program Windows. `Applnit` testează valoarea argumentului `hPrevInstance` și, în cazul în care acesta are valoarea zero, cheamă funcția `InitApplication` pentru a inițializa prima instanță a unui program. În cazul în care `InitApplication` reușește (valoarea întoarsă este `TRUE`), `Applnit` cheamă funcția `InitInstance` pentru inițializarea instanței curente a programului. Funcția `Applnit` întoarce `TRUE` dacă secvența de inițializare a programului a reușit, `FALSE` în caz contrar.

**BOOL Applnit (HINSTANCE hInstance, HINSTANCE hPrevInstance, int nCmdShow)**

```
{
    static char szAppName [] = "schelet";
    if (!hPrevInstance) {
        if (!InitApplication (hInstance, szAppName))
            return FALSE;
    }
    return InitInstance (hInstance, szAppName, nCmdShow);
}
```

7. Funcția `WinMain` (punctul de intrare (utilizator) al unei aplicații Windows). Funcția `WinMain` cheamă funcția `Applnit` pentru realizarea operațiunilor de inițializare a aplicației, după care intră în bucla de procesare de mesaje. În momentul închiderii buclei de mesaje (momentul în care s-a extras mesajul `WM_QUIT` din coada de mesaje a aplicației), funcția `WinMain` întoarce valoarea din câmpul `wParam` al mesajului `msg`, provocând terminarea aplicației:

**int WINAPI WinMain ( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpszCmdLine, int nCmdShow)**

```
{
    MSG msg;
    if (!Applnit (hInstance, hPrevInstance, nCmdShow))
        return 0;
    while (GetMessage (&msg, NULL, 0, 0)) {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return msg.wParam;
}
```

8. Procedura de fereastră `WndProc` procesează mesajele extrase din coada de mesaje a aplicației. Pentru mesajele pentru care s-a definit o funcție de procesare

de mesaj, funcția WndProc apelează respectiva funcție de procesare de mesaj (De exemplu pentru mesajul WM\_DESTROY este apelată funcția WMDestroy). Mesajele pentru care nu au fost definite funcții de procesare sunt trimise către funcția Windows DefWndProc, care realizează procesarea implicită de mesaj:

```
LRESULT CALLBACK WndProc ( HWND hwnd, UINT message,  
WPARAM wParam, LPARAM lParam)  
{  
    switch (message) {  
        case WM_DESTROY: return WMDestroy (hwnd, wParam,  
                                            lParam);  
    }  
    return DefWindowProc (hwnd, message, wParam, lParam);  
}
```

9. Funcțiile de procesare de mesaje (WMDestroy - pentru mesajul WM\_DESTROY) realizează procesarea utilizator a mesajelor Windows, returnând valoarea zero pentru a indica faptul că mesajul a fost procesat. Funcțiile de mesaje au ca argumente handle-ul de fereastră pentru fereastra care a primit mesajul, precum și argumentele wParam și lParam, a căror semnificație este dependentă de mesajul respectiv. Funcția WMDestroy introduce în coada de mesaje a aplicației mesajul WM\_QUIT, necesar pentru ruperea buclei de mesaje:

```
long WMDestroy (HWND hwnd, WPARAM wParam, LPARAM lParam)  
{  
    PostQuitMessage (0);  
    return 0;  
}
```

### 3.2 Programarea independentă de periferic (device)

Suprafața client a unei ferestre este formată din suprafața ferestrei de aplicație din care se scad suprafețele ocupate de Caption bar-ul aplicației, marginile de dimensionare, meniul aplicației (dacă acesta există), precum și suprafețele ocupate de scroll bar-uri (dacă există). Suprafața client este porțiunea din fereastră în care aplicația are dreptul să deseneze și să afișeze text. Aplicațiile sub Windows nu au dreptul să presupună că fereastra principală are o anumită dimensiune sau că dimensiunea acesteia va rămâne constantă pe tot timpul execuției programului. Aplicația împarte ecranul cu celelalte aplicații Windows. Utilizatorul determină poziția și dimensiunile ferestrelor pe ecran. Aplicația poate să citească dimensiunile ferestrei și să se adapteze la cerințele impuse de utilizator.

Acest capitol prezintă modul în care se realizează un program independent de periferic (monitor). Programele Windows pot presupune relativ puține lucruri despre mediul în care vor fi lansate în execuție. Windows pune acestor programe la dispoziție o serie de funcții prin care pot obține toate informațiile despre mediul pe care se execută (tipul monitorului etc.), necesare în momentul execuției.

#### 3.2.1 Desenarea și redesenarea suprafeței client

Aplicațiile proiectate pentru MS-DOS puteau afișa informații în toate zonele ecranului. Informațiile astfel afișate rămăneau desenate pe ecran fără ca la un moment dat să dispară în chip misterios. Programul putea să renunțe la anumite informații necesare pentru desenarea pe ecran.

În Windows, un program afișează informația în general doar în suprafața client a ferestrei sale. Este posibil ca informația afișată în suprafața client de către un program să fie ștearsă datorită suprapunerii altor ferestre peste zona client a aplicației respective. De exemplu, o fereastră de dialog poate să se suprapună peste fereastra programului considerat. Deși Windows va încerca să salveze și apoi să refacă suprafața acoperită de către fereastra de dialog, în anumite condiții el nu va putea realiza această operație consumatoare de memorie. Când fereastra de dialog va fi îndepărtată de pe ecran, Windows va cere programului să redeseneze porțiunea din suprafața client acoperită de către fereastra de dialog.

Windows este un sistem de operare condus de evenimente. Windows semnalează aplicațiilor apariția diferitelor evenimente introducând mesaje în coada de mesaje a aplicațiilor



sau trimițând mesaje direct către procedurile de fereastră. Windows semnalează unei proceduri de fereastră faptul că o parte din suprafața client trebuie redesenată introducând mesaje WM\_PAINT în coada de mesaje a aplicației.

### 3.2.2 Mesajul WM\_PAINT

Majoritatea programelor Windows apelează funcția UpdateWindow în partea de inițializare a aplicației (în WinMain), imediat înainte de a intra în bucla de procesare a mesajelor. Windows, cu această ocazie, trimite mesajul WM\_PAINT către procedura de fereastră. Acest mesaj specifică procedurii de fereastră faptul că suprafața client a ferestrei trebuie redesenată. Ca urmare, procedura de fereastră trebuie să fie pregătită în permanență să primească mesaje WM\_PAINT și să reafișeze chiar și întreaga suprafața client, dacă este necesar. Procedura de fereastră primește un mesaj WM\_PAINT când apare unul din următoarele evenimente:

- O suprafață din fereastră (anterior acoperită de către o altă fereastră) este descoperită prin deplasarea (sau închiderea) ferestrei acoperitoare.
- Utilizatorul modifică dimensiunea ferestrei (doar dacă stilul de clasă conține flag-urile pe biți CS\_HREDRAW și CS\_VREDRAW setate).
- Programul folosește funcția ScrollWindow pentru a scroll-a o porțiune a suprafeței client.
- Programul apelează funcția InvalidateRect sau InvalidateRgn pentru a genera în mod explicit un mesaj WM\_PAINT.

În anumite cazuri când o porțiune a suprafeței client este acoperită temporar, Windows încearcă să salveze informația grafică din fereastra pentru ca apoi să refacă această suprafață. Nu întotdeauna Windows reușește să realizeze aceasta acțiune. Dacă acțiunea nu reușește, Windows ar putea introduce un mesaj WM\_PAINT în coada de mesaje când:

- Windows închide o fereastră de dialog sau un box de mesaj care acoperea o porțiune din fereastră.
- Un meniu pulldown este afișat apoi el este închis.

În câteva cazuri Windows va salva totdeauna suprafața din afișare suprascrisă, iar apoi va reface informația grafică. Acest lucru se întâmplă de fiecare dată când:

- Cursorul mouse-ului este deplasat peste suprafața client.
- Un icon este tras ("drag") peste suprafața client.

Procesarea mesajelor WM\_PAINT necesită modificarea punctului de vedere al programatorului în legătura cu afișarea informațiilor pe ecran. Aplicația trebuie astfel structurată încât să memoreze toate informațiile necesare să deseneze suprafața client și să deseneze pe baza acestor informații "la cerere" - când Windows trimite un mesaj WM\_PAINT către procedura de fereastră.

### 3.2.3 Dreptunghiuri valide și invalide

O procedură de fereastră trebuie să fie capabilă de a desena întreaga suprafață client în momentul primirii unui mesaj WM\_PAINT. De cele mai multe ori însă, procedura de fereastră trebuie să redeseneze doar o mică zonă rectangulară din suprafața client. Cel mai adesea peste suprafața client se suprapune o fereastră de dialog. În momentul închiderii ferestrei de dialog este necesară redesenarea porțiunii dreptunghiulare care a fost acoperită.

Suprafața dreptunghiulară care necesită redesenare este denumită "dreptunghi invalid". Windows postează un mesaj WM\_PAINT în coada de mesaje a aplicației în momentul apariției unui dreptunghi invalid în suprafața client a ferestrei. Procedura de fereastră care primește un mesaj WM\_PAINT trebuie să reafișeze informația grafică conținută în dreptunghiul invalid.

Windows memorează câte o structură de informații pentru desenare asociată fiecărei ferestre din sistem. Această structură conține (pe lângă alte informații) și coordonatele dreptunghiului invalid. Dacă o altă suprafață rectangulară devine invalidă înainte ca procedura de fereastră să proceseze mesajul WM\_PAINT, Windows calculează un nou dreptunghi invalid care va conține cele două suprafețe și memorează noul dreptunghi invalid în structura de informații pentru desenare. Windows introduce un singur mesaj WM\_PAINT în coada de mesaje a aplicației.

O procedură de fereastră poate "invalida" un dreptunghi în propria suprafață client prin apelul funcției InvalidateRect. Dacă coada de mesaje a aplicației care deține fereastra respectivă conține deja un mesaj WM\_PAINT, Windows calculează un nou dreptunghi invalid.

În caz contrar (nu există mesaje WM\_PAINT în coada de mesaje), Windows plasează un mesaj WM\_PAINT în coada de mesaje. O procedură de fereastră poate obține coordonatele dreptunghiului invalid când procesează un mesaj WM\_PAINT, prin apelul funcției BeginPaint. În orice altă situație, coordonatele dreptunghiului invalid pot fi obținute prin apelul funcției GetUpdateRect.

După ce procedura de fereastră a terminat de desenat în suprafața client, trebuie apelată funcția EndPaint care are ca efect validarea întregii suprafețe client a ferestrei. Un program poate să valideze în orice moment un dreptunghi din suprafața client prin apelul funcției ValidateRect. Dacă apelul funcției ValidateRect are ca efect validarea întregii suprafețe invalide, mesajul WM\_PAINT este scos din coada de evenimente.

### 3.2.4 Introducere în GDI

Pentru a desena în suprafața client a ferestrei trebuie utilizate funcțiile GDI (Graphic Device Interface). Windows oferă cinci funcții GDI pentru afișarea de șiruri de caractere în suprafața client a unei ferestre. În laboratorul anterior am folosit funcția DrawText pentru a afișa "Hello, Windows!". Cea mai frecvent folosită funcție este "TextOut". Această funcție are următorul prototip:

**TextOut (hdc, x, y, lpszString, nLength);**

TextOut afișează un șir de caractere pe monitor. Parametrul lpszString este o referință la un șir de caractere. Parametrul nLength reprezintă numărul de caractere din care este format șirul lpszString. Argumentele x și y precizează poziția de început în cadrul suprafeței client a textului, în "coordoanate logice". Argumentul hdc este un handle către un "device context". Orice funcție GDI necesită ca primul argument acest handle care identifică suprafața client în care se realizează acțiunea grafică.

### 3.2.5 Contextul de periferic - "device context"

Un handle este un număr (de 16 biți) pe care Windows îl utilizează intern pentru a identifica un obiect. Un handle poate fi obținut prin apelul funcțiilor Windows, iar apoi acesta poate fi utilizat ca argument în apelul altor funcții. Handle-ul de device context este necesar apelului funcțiilor GDI. Cu ajutorul lui, aplicația are posibilitatea să deseneze în suprafața client a ferestrei.

Device context-ul (denumit și "DC") este de fapt o structură de date memorată de către GDI ("Graphic Device Interface"). Un device context este asociat cu un anumit tip de periferic care poate fi utilizat pentru desene (de exemplu imprimantă, plotter-ul sau monitorul grafic). În cazul particular al monitorului, device context-ul este asociat cu o anumită fereastră de pe ecran.

Windows utilizează valorile din structura de device context (denumite și attribute ale device context-ului) în legătură cu apelul funcțiilor GDI. De exemplu, în cazul funcției TextOut, attributele device context-ului (specificat prin argumentul hdc al funcției), determină culoarea cu care va fi afișat șirul de caractere pe ecran, tipul de caractere (fontul) pe care Windows îl va utiliza pentru afișarea textului etc.

Când un program trebuie să deseneze în suprafața client a ferestrei proprii, el trebuie să obțină un handle către device context-ul acelei ferestre. După ce programul a terminat acțiunile de desene, programul trebuie să elibereze handle-ul utilizat, deoarece numărul de handle-uri de device context disponibile în sistem este limitat. Când un program eliberează handle-ul către device context, acesta devine invalid și nu mai trebuie utilizat. Programul trebuie să obțină un handle către device context și să-l elibereze în timpul procesării unui singur mesaj. Exceptând cazul în care handle-ul este obținut prin apelul funcției CreateDC, programul nu trebuie să păstreze handle-ul între mesaje.

Există două metode foarte frecvent utilizate pentru a obține un handle către un device context în scopul afișării de informații pe monitor.

### 3.2.6 Prima metodă pentru obținerea unui handle pentru un device context

Această metodă poate fi utilizată doar în timpul procesării mesajului WM\_PAINT. Metoda necesită apelul a două funcții: BeginPaint și EndPaint. Cele două funcții cer ca argumente de apel handle-ul ferestrei care a primit mesajul WM\_PAINT (valoarea acestui argument este trimis ca primul parametru al procedurii de fereastră: argumentul hwnd, și are aceeași valoare cu cea întoarsă de către apelul funcției CreateWindow din InitInstance). Al doilea parametru este o referință către o variabilă de tip PAINTSTRUCT, denumită ps.

În timpul procesării mesajului WM\_PAINT, programul trebuie să apeleze funcția BeginPaint pentru a inițializa câmpurile structurii ps. Valoarea întoarsă de către funcția BeginPaint este un handle către device context-ul asociat ferestrei care a primit mesajul. Această valoare este înscrisă într-o variabilă denumită hdc de tipul HDC.

Tipul de date HDC este definit în Winuser.h drept un HANDLE (întreg pe 16 biți fără semn). Programul poate să folosească după aceea funcții GDI, ca de exemplu TextOut. Apelul funcției EndPaint eliberează handle-ul către device context și validează tot dreptunghiul client al ferestrei.

Procesarea mesajului WM\_PAINT care se realizează în funcția WMPaint are în general următoarea formă:

```
long WMPaint (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    PAINTSTRUCT ps;
    hdc = BeginPaint (hwnd, &ps);
    /* Se pot utiliza funcții GDI cu primul argument hdc */
    EndPaint (hwnd, &ps);
    return 0;
}
```

Dacă o procedură de fereastră nu procesează mesajul WM\_PAINT, atunci acesta trebuie trecut funcției DefWindowProc (procedura implicită de fereastră) care se află în Windows.

Funcția DefWindowProc procesează mesajul WM\_PAINT astfel:

```
BeginPaint (hwnd, &ps);
EndPaint (hwnd, &ps);
return 0;
```

Această secvență care conține un BeginPaint și EndPaint fără nimic între ele realizează validarea dreptunghiului invalid conținut în suprafața client și care a produs mesajul WM\_PAINT. Este greșită procesarea mesajului WM\_PAINT astfel:

```
long WMPaint (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    return 0;
}
```

Windows introduce un mesaj WM\_PAINT în coada de mesaje a aplicației datorită faptului că o porțiune din suprafața client este invalidă. Dacă nu apelați funcțiile BeginPaint și EndPaint (sau ValidateRect), Windows nu va valida zona invalidă și ca urmare va trimite un alt mesaj WM\_PAINT. Acest lucru se va repeta producând o buclă infinită în care se trimit mesaje WM\_PAINT către procedura de fereastră și ca urmare sistemul se va bloca.

### 3.2.7 Structura de informații de desenare

PAINTSTRUCT reprezintă structura de informații de desenare memorată intern de Windows pentru orice fereastră existentă în sistem. În momentul apelului funcției BeginPaint, Windows copiază câmpurile acestei structuri în variabila ps care este trimisă prin referință ca al doilea argument al funcției. PAINTSTRUCT este definită în Winuser.h astfel:

```
typedef tagPAINTSTRUCT
{
    HDC    hdc;
    BOOL   fErase;
    RECT   rcPaint;
    BOOL   fRestore;
    BOOL   fIncUpdate;
    BYTE   rgbReserved [32];
} PAINTSTRUCT;
```

Programul poate utiliza doar primele trei câmpuri ale acestei structuri.

Câmpul hdc reprezintă un handle către device context. Aceasta valoare este întoarsă și ca rezultat al apelului funcției BeginPaint.

Câmpul fErase are în general valoarea TRUE (diferit de zero), semnificând faptul că Windows a șters fundalul conținut în dreptunghiul invalid. Windows șterge fundalul prin umplerea acestuia cu culoarea specificată în câmpul hbrBackground al structurii WNDCLASS

care a fost folosită la înregistrarea clasei de fereastră în funcția `InitApplication`. Majoritatea programelor Windows folosesc culoarea alba:

**`wndclass.hnrBackground = GetStockObject (WHITE_BRUSH);`**

Cu toate acestea, în cazul apelului funcției `InvalidateRect`, utilizată pentru invalidarea unui dreptunghi în suprafața client, este posibil să se specifice prin intermediul unui argument al funcției dacă se dorește ca fundalul dreptunghiului invalid să fie șters sau nu. Dacă acest argument este `FALSE` (adică 0), atunci Windows nu va șterge fundalul anterior mesajului `WM_PAINT`, iar câmpul `fErase` din structura `PAINTSTRUCT` obținută în timpul procesării mesajului `WM_PAINT` va avea valoarea `FALSE`.

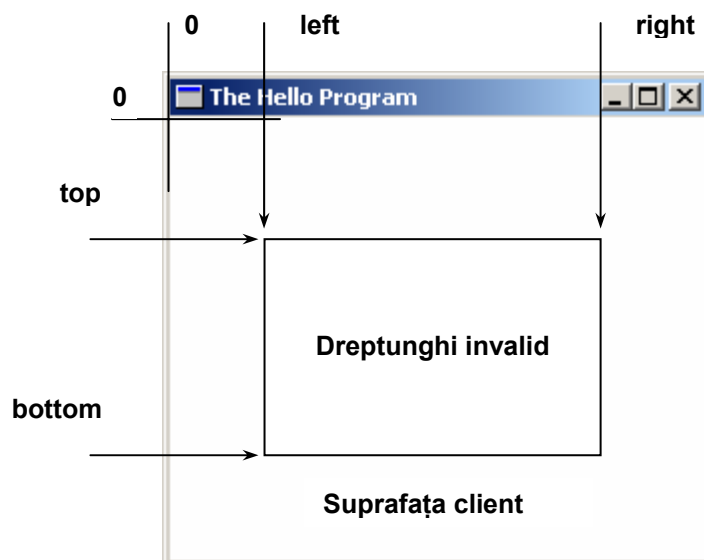


Figura 3.1.

Câmpul `rcPaint` al structurii `PAINTSTRUCT` este o structură de tipul `RECT`. Această structură definește un dreptunghi. Cele patru câmpuri ale structurii sunt `left`, `top`, `right` și `bottom`. Câmpul `rcPaint` definește marginile dreptunghiului invalid relativ la dreptunghiul client al ferestrei, după cum se vede în figura 3.1. Valorile sunt în pixeli relative la colțul stânga sus al suprafeței client. Dreptunghiul invalid este suprafața care ar trebui redesenată. Deși un program Windows poate să redeseneze întreaga suprafață client a ferestrei în momentul primirii unui mesaj `WM_PAINT`, redesenarea doar a suprafeței invalide (definită prin `rcPaint`) realizează o economie de timp.

Câmpul `rcPaint` din structura `PAINTSTRUCT` reprezintă nu numai coordonatele dreptunghiului invalid, el este în același timp un dreptunghi de tăiere ("clipping"). Acest lucru semnifică faptul că Windows nu permite desenul decât în interiorul acestui dreptunghi. Folosirea handle-ului de device context obținut prin apelul funcției `BeginPaint`, restricționează desenarea în interiorul dreptunghiului `rcPaint`.

Pentru a desena în exteriorul acestui dreptunghi, în timpul procesării mesajului `WM_PAINT`, este necesar apelul funcției:

**`InvalidateRect (hwnd, NULL, TRUE);`**

anterior apelului lui `BeginPaint`.

Funcția `InvalidateRect` cu argumentele respective invalidează și șterge întreaga suprafață client. Trimițând valoarea `FALSE` ca ultim argument al funcției nu va produce ștergerea fundalului suprafeței client deci ce este afișat în suprafața client va rămâne nemodificat.

În exemplul din capitolul anterior nu am luat în considerare dreptunghiul invalid și dreptunghiul de clipping în cursul procesării mesajului `WM_PAINT`. Dacă textul afișat se afla în interiorul dreptunghiului invalid, atunci `DrawText` refăcea textul. Dacă textul nu se afla în interiorul dreptunghiului invalid, la un moment dat, în timpul procesării funcției `DrawText`, Windows determină faptul că textul este în exteriorul suprafeței de clipping și deci că nu are nimic de desenat. Acest mecanism este consumator de timp. Un program care urmărește realizarea unor parametri buni de performanță utilizează în general coordonatele

dreptunghiului invalid în timpul procesării mesajului WM\_PAINT pentru a evita apeluri inutile de funcții GDI.

### 3.2.8 A doua metodă pentru obținerea unui handle de device context

Windows oferă posibilitatea de a obține un handle de device context și în alte mesaje decât în mesajul WM\_PAINT. Dacă se dorește desenarea în alt mesaj decât WM\_PAINT sau dacă se dorește obținerea de informații despre device context (cum ar fi de exemplu dimensiunile caracterelor etc.), este posibilă obținerea unui handle prin apelul funcției GetDC, iar eliberarea handle-ului astfel obținut se face prin apelul funcției ReleaseDC, după utilizare:

```
hdc = GetDC (hwnd);  
/* se pot utiliza funcții GDI cu primul argument hdc */  
ReleaseDC (hwnd, hdc);
```

Întocmai ca BeginPaint și EndPaint, GetDC și ReleaseDC trebuie utilizate în pereche. Când se face apelul funcției GetDC în timpul procesării unui mesaj, trebuie apelat ReleaseDC înainte de a se părăsi procedura de fereastră. Nu este recomandat apelul funcției GetDC în timpul procesării unui mesaj și apelul funcției ReleaseDC în timpul procesării altui mesaj.

În comparație cu device context-ul obținut prin apelul funcției BeginPaint, device context-ul întors de funcția GetDC are o suprafață de clipping întotdeauna egală cu dreptunghiul client al ferestrei specificate. EndDC, în comparație cu EndPaint nu realizează validarea dreptunghiurilor invalide din suprafața client.

### 3.2.9 Funcția TextOut

În momentul obținerii unui handle către un device context, Windows înscrie în structura de device context anumite atribute implicite. Aceste valori implicite pot fi modificate prin apelul de funcții GDI. Funcția TextOut folosește următoarea secvență de apel:

```
TextOut (hdc, x, y, lpszString, nLength);
```

Primul parametru al funcției este un handle către un device context: acesta poate fi obținut fie prin apelul funcției BeginPaint (în timpul procesării mesajului WM\_PAINT), fie prin apelul lui GetDC.

Atributele setate în device context realizează controlul caracteristicilor de afișare a textului. De exemplu un atribut al device context-ului specifică culoarea de desenare a textului. Culoarea implicită de desenare a textului este negru. Device context-ul implicit definește, de asemenea, o culoare de fundal albă care se folosește pentru afișarea textelor. Când un program scrie într-un device context, Windows utilizează culoarea de fundal pentru a umple spațiile din jurul caracterelor.

Această culoare de fundal nu este același lucru cu culoarea de fundal utilizată pentru definirea clasei de fereastră. Fundalul ferestrei este definit de către un brush pe care Windows îl utilizează pentru a umple suprafața client a ferestrei, ea nu face parte din structura de device context. Când se definește o structură de clasă de fereastră, majoritatea aplicațiilor utilizează un WHITE\_BRUSH, astfel încât culoarea de fundal implicită a device context-ului să fie identică cu culoarea utilizată pentru ștergerea suprafeței client a ferestrei.

Argumentul lpszString este un pointer far către un șir de caractere, iar argumentul nLength reprezintă lungimea (în număr de caractere) a textului afișat.

Argumentele x și y specifică coordonatele punctului de început de unde va fi afișat șirul de caractere în cadrul suprafeței client. Valoarea x reprezintă coordonata orizontală, iar valoarea y pe cea verticală. Implicit colțul stânga sus al primului caracter al șirului de caractere va fi poziționat la coordonatele x și y. În device context-ul implicit, originea sistemului de coordonate se afla poziționată în colțul stânga sus a suprafeței client a ferestrei.

Coordonatele utilizate de către GDI sunt "coordoanate logice". Windows are mai multe moduri de translație, numite și moduri de mapare care controlează translația de la sistemul de coordonate logice la cel de coordonate fizice asociat dispozitivului periferic utilizat. Modul de mapare este un atribut al device context-ului. Modul implicit de mapare este identificat prin constanta MM\_TEXT (definită în Wingdi.h). Sub modul de mapare MM\_TEXT, unitățile logice sunt identice cu unitățile fizice (adică pixeli). Coordonata x crește către partea dreaptă, iar coordonata y crește către partea inferioară a suprafeței client. Figura 3.2. prezintă sistemul de coordonate logice MM\_TEXT. Sistemul de coordonate MM\_TEXT este identic cu cel pe care Windows îl folosește pentru definirea dreptunghiului invalid în structura de date PAINTSTRUCT.

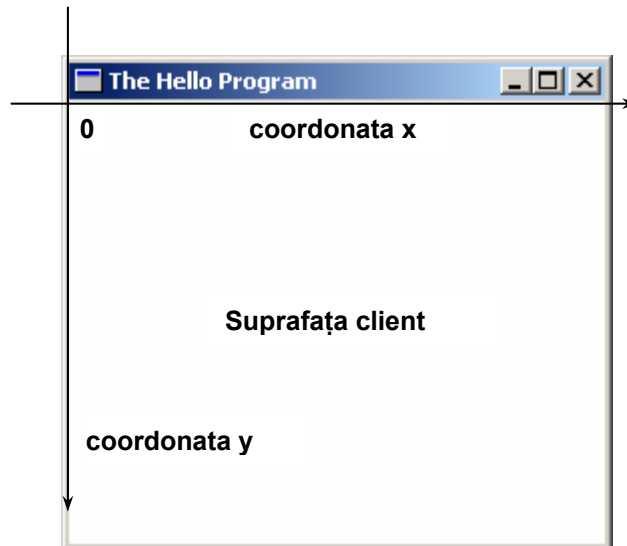


Figura 3.2.

Device context-ul definește de asemenea o suprafață de clipping. După cum am văzut, suprafața implicită de clipping este egală cu suprafața client a ferestrei pentru device context-ul obținut prin apelul funcției `GetDC` și egală cu suprafața invalidă pentru device context-ul obținut din apelul funcției `BeginPaint`. Windows nu afișează porțiunile din șirul de caractere care nu sunt incluse în suprafața de clipping (în general Windows nu afișează decât porțiunile de desen care se află în suprafața de clipping). Dacă un caracter se află numai parțial în suprafața de clipping, Windows nu afișează decât porțiunea din caracter care se află în interiorul acestui dreptunghi.

### 3.2.10 Setul de caractere sistem

Structura de device context conține setul de caractere (font-ul) pe care Windows îl utilizează pentru afișarea textelor în suprafața client. Fontul implicit este denumit "font de sistem - system font" și este descris de către identificatorul `SYSTEM_FONT` (definit în `Wingdi.h`). Fontul sistem este utilizat de către Windows pentru afișarea textelor din interiorul Caption bar-ului, a meniurilor și în cadrul ferestrelor de dialog.

Fontul de sistem are o lățime variabilă a caracterelor, ceea ce înseamnă că două caractere pot să aibă lățimi diferite (de exemplu, caracterul "W" este mai lat decât caracterul "i").

### 3.2.11 Dimensiunea unui caracter

Pentru a afișa mai multe linii de text cu ajutorul funcției `TextOut`, este necesar să se cunoască dimensiunea caracterelor fontului. Liniile succesive de text pot fi spațiate pe baza înălțimii unui caracter, iar coloanele de text pot fi distanțate cunoscând lățimile caracterelor.

Dimensiunile caracterelor pot fi obținute prin apelul funcției `GetTextMetrics`. Funcția `GetTextMetrics` cere ca argument de apel un handle de device context deoarece valoarea pe care această funcție o întoarce este relativă la fonturile selectate într-un anumit device context. Windows copiază diferitele valori care identifică dimensiunile unui caracter într-o structură de date denumită `TEXTMETRIC`. Valorile din câmpurile acestei structuri sunt definite în modul de mapare curent al device context-ului utilizat. În device context-ul implicit, acest mod de mapare este `MM_TEXT`, deci dimensiunile sunt în pixeli.

Pentru a utiliza funcția `GetTextMetrics` este necesară declararea unei variabile de tipul `TEXTMETRIC` denumită `tm`. Apoi, obținerea unui handle către device context-ul implicit asociat ferestrei:

```
hdc = GetDC (hwnd);  
GetTextMetrics (hdc, &tm);
```

După analiza valorilor înscrise în structura `tm` (și probabil salvarea unor valori pentru utilizarea ulterioară), este necesară eliberarea device context-ului:

```
ReleaseDC (hwnd, hdc);
```

### 3.2.12 Funcția GetTextMetric

Structura TEXTMETRICS oferă o serie de informații despre font-ul curent selectat în device context. Dimensiunea verticală a caracterului este definită cu ajutorul a numai puțin de cinci valori, ca în figura 3.3.

Câmpul tmInternalLeading este dimensiunea necesară pentru un semn de accent deasupra caracterului. Câmpul tmExternalLeading reprezintă spațiul minim dintre două rânduri de caractere scrise cu fontul respectiv.

Structura TEXTMETRIC conține două câmpuri care descriu lățimea fontului respectiv: tmAveCharWidth reprezintă lățimea medie a caracterelor din font, iar tmMaxCharWidth reprezintă lățimea celui mai mare caracter al fontului. Pentru un font cu lățime fixă, cele două câmpuri au aceeași valoare. Programele pe care le discutăm în acest capitol necesită cunoașterea lățimii medii a caracterelor majuscule. Aceasta lățime a fost calculată ca 150% din tmAveCharWidth.

Dimensiunile fontului sunt dependente de rezoluția monitorului pentru care este instalat Windows. Din acest motiv este necesar ca dimensiunile caracterelor să fie obținute cu ajutorul funcției GetTextMetrics în momentul execuției programului (pe un tip diferit de monitor, acestea ar putea avea valori diferite).

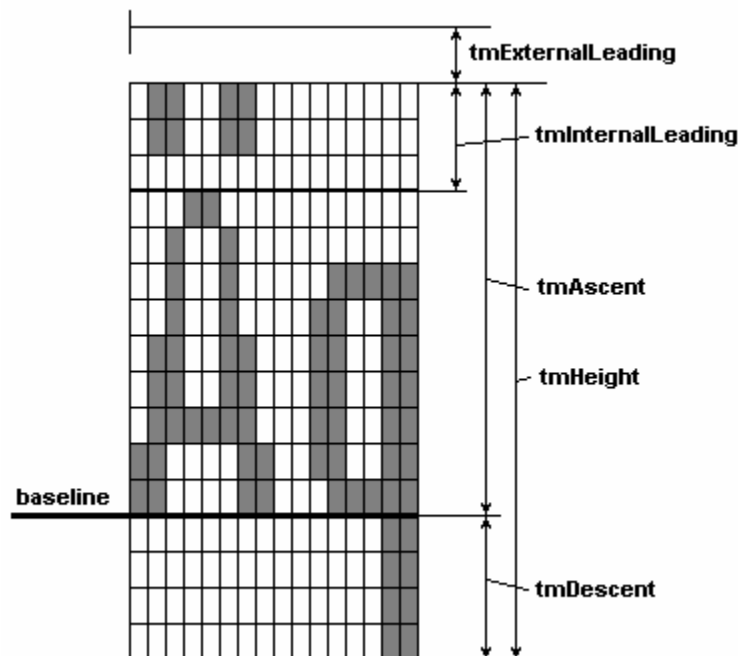


Figura 3.3. Caracteristicile unui caracter

### 3.2.13 Afișarea textelor

Deoarece dimensiunile fontului de sistem nu se modifică în timpul unei sesiuni Windows, este necesar apelul funcției GetTextMetrics o singură dată în timpul execuției programului. Un loc foarte bun pentru a face astfel de apeluri este în timpul funcției de procesare a mesajului WM\_CREATE din procedura de fereastră. Mesajul WM\_CREATE este primul mesaj primit de către procedura de fereastră. Windows apelează procedura de fereastră cu mesajul WM\_CREATE ca urmare a apelului funcției CreateWindow din InitInstance.

Să presupunem că vrem să scriem un program Windows care afișează mai multe linii de text în suprafața client. Valorile lățimii medii și înălțimii unui caracter, de care avem nevoie pentru afișarea textului, le vom memora în două variabile pe care le vom denumi cxChar (lățimea medie) respectiv cyChar (înălțimea caracterului). Funcția de procesare a mesajului WM\_CREATE este următoarea:

```
long WmCreate (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    TEXTMETRIC tm;
    HDC hdc;
```

```

    hdc = GetDC (hwnd);
    GetTextMetrics (hdc, &tm);
    cxChar = tm.tmAveCharWidth;
    cyChar = tm.tmHeight + tm.tmExternalLeading;
    ReleaseDC (hwnd, hdc);
    return 0;
}

```

Utilizarea acestor valori este folosită pentru spațierea textelor pe ecran. O metodă relativ simplă este următoarea: față de marginea ferestrei se lasă un spațiu de `cxChar` (față de stânga) și `cyChar` (față de marginea superioară). Pentru a afișa mai multe linii de text, coordonata `y` va fi incrementată cu câte un `cyChar` la fiecare linie. Deci pentru afișarea liniei de text numărul `i` se vor utiliza următoarele coordonate: `cxChar`, pentru coordonata `x`, respectiv `cyChar * (i + 1)`, pentru coordonata `y`, unde `i` este numărul liniei de text, iar prima linie este numerotată cu 0.

Pentru afișarea de numere într-un anumit format se poate utiliza funcția `sprintf` (similară cu funcția `printf`, dar textul formatat este depus într-un buffer de caractere și nu pe ecran). Afișarea bufferului de text poate fi făcută cu ajutorul funcției `TextOut`. Valoarea întoarsă de către funcția `sprintf` este lungimea șirului de caractere, valoare pe care o vom folosi pentru argumentul `nLength` al funcției `TextOut`. Următoarea secvență de program prezintă o combinație tipică între `TextOut` și `sprintf`:

```

    short  nLength;
    char   szBuffer [40];
    nLength = sprintf (szBuffer, "Suma dintre %d și %d este %d", nA, nB,
                                                                nA + nB);
    TextOut (hdc, x, y, szBuffer, nLength);

```

Putem renunța la utilizarea variabilei `nLength` și să combinăm cele două instrucțiuni într-una singură, bazându-ne pe faptul că `sprintf` returnează numărul de caractere din buffer:

```

    TextOut (hdc, x, y, szBuffer, sprintf (szBuffer,
                                           "Suma dintre %d și %d este %d", nA, nB, nA + nB));

```

Dacă nu este necesară afișarea de valori în virgulă mobilă, se poate utiliza funcția `wsprintf` care este inclusă în Windows. Sintaxa funcției `wsprintf` este identică cu cea a lui `sprintf`.

### 3.3 Programul SysMets

Am discutat toate elementele necesare pentru scrierea unui program care să afișeze mai multe linii de text pe ecran. Știm cum putem să obținem un handle de device context, cum să folosim funcția `TextOut`, și cum să spațiem textul în funcție de dimensiunea unui singur caracter.

Vom utiliza funcția `GetSystemMetrics` care întoarce valorile diferitelor dimensiuni ale obiectelor grafice Windows, cum ar fi: dimensiunile icon-urilor, cursorului, Caption bar-urilor și ale scroll bar-urilor. Aceste dimensiuni variază în funcție de tipul de monitor utilizat. Funcția `GetSystemMetrics` necesită un singur caracter denumit `index`. Acest `index` poate lua valori între cei 37 întregi definiți în `Winuser.h` și care au prefixul `SM_`. `GetSystemMetrics` întoarce un întreg ce reprezintă dimensiunea specificată prin argument.

Programul `Sysmets` este alcătuit din două fișiere:

- `Sysmets.h`
- `Sysmets.c`

În fișierul header `Sysmets.h` se definește un tablou de structuri care conțin atât identificatorul utilizat ca argument al funcției `GetSystemMetrics`, precum și o descriere a indexului respectiv, pe care o vom afișa în dreptul valorii obținute. Programul care afișează toate aceste informații se numește `Sysmets.c`.

În figura 3.4. se observă rezultatul programului ce rulează pe un monitor tip VGA. După cum se poate observa din textul afișat în suprafața client a programului, rezoluția monitorului este de 800 pixeli pe orizontală și 600 pixeli pe verticală. Aceste două valori, precum și multe alte valori afișate de către program vor fi diferite în funcție de tipul de monitor pe care programul este executat.



Metrici sistem No. 1		
SM_CXSCREEN	Dimensiunea orizontala a ecranului in pixeli	800
SM_CYSCREEN	Dimensiunea verticala a ecranului in pixeli	600
SM_CXVSCROLL	Latimea scroll bar-ului vertical	16
SM_CYHSCROLL	Inaltimea scroll bar-ului orizontal	16
SM_CYCAPTION	Inaltimea caption bar-ului	19
SM_CXBORDER	Latimea marginii ferestrei	1
SM_CYBORDER	Inaltimea marginii ferestrei	1
SM_CXDLGFRAME	Latimea marginii ferestrei de dialog	3
SM_CYDLGFRAME	Inaltimea marginii ferestrei de dialog	3
SM_CYVTHUMB	Inaltimea butonului scroll bar-ului vertical	16
SM_CXHTHUMB	Latimea butonului scroll bar-ului orizontal	16
SM_CXICON	Latimea icon-ului	32
SM_CYICON	Inaltimea icon-ului	32
SM_CXCURSOR	Latimea cursorului de mouse	32
SM_CYCURSOR	Inaltimea cursorului de mouse	32
SM_CYMENU	Inaltimea menu bar-ului asociat ferestrei	19
SM_CXFULLSCREEN	Latimea zonei client a ferestrei maximizate	800

Figura 3.4. Programul Systems.c

### 3.3.1 Funcțiile de prelucrare de mesaje

Procedura de fereastră prelucrează trei mesaje: WM\_CREATE, WM\_PAINT și WM\_DESTROY. Pentru procesarea celor trei mesaje sunt apelate funcțiile WMCreate, WMPaint, respectiv WMDestroy. Mesajul WMDestroy este procesat identic cu aplicația schelet prezentată la începutul capitolului.

Mesajul WM\_CREATE este unul din primele mesaje pe care procedura de fereastră îl primește. Mesajul WM\_CREATE este generat de către Windows când funcția CreateWindow creează fereastra principală. Mesajul WMCreate este prelucrat de către funcția WMCreate. Funcția WMCreate obține un handle la device context-ul ferestrei principale a aplicației prin apelul funcției Windows GetDC, după care obține dimensiunile caracterelor necesare pentru afișarea textelor în suprafața client a ferestrei. Dimensiunile sunt obținute prin apelul funcției GetTextMetrics. Programul memorează lățimea medie a caracterului în cxChar și înălțimea totală a caracterului (adică tm.tmHeight + tm.tmExternalLeading) în variabila cyChar.

Programul calculează de asemenea o dimensiune medie a caracterelor majuscule în variabila cxCaps. Pentru un font cu lățime fixă, cxCaps este egal cu cxChar, iar pentru un font cu lățime variabilă, variabila cxCaps este 150% din lățimea medie a caracterului (adică din cxChar). Tipul fontului este dat de către bit-ul mai puțin semnificativ al câmpului tm.tmPitchAndFamily. Dacă acest bit este zero, atunci fontul de sistem este un font fix, iar dacă bitul este setat, fontul de sistem este un font cu dimensiune variabilă. Secvența de calcul al lui cxCaps este următoarea:

```
if ((tm.tmPitchAndFamily & 0x0001) == 0)
    cxCaps = cxChar;
else
    cxCaps = 3 * cxChar / 2;
```

Programul Sysmets realizează desenul suprafeței client în timpul mesajului WM\_PAINT (adică în funcția WMPaint care prelucrează acest mesaj). Funcția obține un handle la device context-ul ferestrei prin apelul funcției Windows BeginPaint. O structură "for" este utilizată pentru iterarea tuturor câmpurilor tabloului sysmetrics definit în Sysmets.h. Fiecare câmp al tabloului sysmets este afișat pe o linie separată. Cele trei coloane de text (specificând denumirea indexului utilizat, descrierea valorii și valoarea respectivă) sunt afișate cu ajutorul a trei funcții TextOut. Parametrul funcției TextOut care identifică coordonata y de afișare a textului în cadrul zonei client are valoarea cyChar \* (i + 1). Acest parametru indică poziția în pixeli a marginii de sus a textului afișat relativ la marginea de sus a zonei client. Deci programul lasă un rând liber, după care prima linie de text (când i = 0) începe la poziția cyChar pe y, a doua linie de text (i = 1) începe la 2 \* cyChar pe coordonata y etc.

Primul apel al funcției TextOut afișează denumirea identificatorului utilizat în apelul funcției GetSystemMetrics pe prima coloană. Coordonata x la care se afișează denumirea

identificatorului are în permanență valoarea `cxChar`. Aceasta lasă un spațiu de dimensiune un caracter între marginea stânga a zonei client și începutul textului. Textul este conținut în câmpul `szLabel` al structurii `sysmetrics` definită în `Sysmets.h`. S-a utilizat funcția `Windows strlen` (similară cu funcția `C strlen`), care obține lungimea unui șir de caractere, necesară pentru ultimul parametru al funcției `TextOut`.

Al doilea apel al funcției `TextOut` afișează descrierea metricilor de sistem. Aceste descrieri au fost memorate în câmpul `szDesc` al structurii `sysmetrics`. Coordonata `x` din apelul funcției `TextOut` are valoarea: `cxChar + 18 * cxCaps`.

Cel mai lung identificator afișat în prima coloană are 16 caractere, deci cea de-a doua coloană trebuie să înceapă la cel puțin `16 * cxCaps` pixeli de la marginea din stânga a suprafeței client.

Cel de-al treilea apel al funcției `TextOut` afișează valorile numerice obținute din apelul funcției `GetSystemMetrics`. Alinierea numerelor afișate se face în raport cu poziția de sfârșit a șirului de caractere. Funcția `SetTextAlign` permite setarea modului de aliniere a șirurilor de caractere. După apelul:

**`SetTextAlign (hdc, TA_RIGHT | TA_TOP);`**

coordonatele `x` și `y` trecute funcției `TextOut` specifică marginea din dreapta sus a șirului de caractere care va fi afișată la coordonatele `(x, y)` în suprafața client.

Funcția `TextOut` afișează coloana de numere aliniată în partea dreaptă la o coordonată `x` cu valoarea `(cxChar + 18 * cxCaps + 50 * cxChar)`, care asigură faptul că coloanele doi și trei nu se suprapun.

După cel de-al treilea apel al funcției `TextOut` din bucla "for" se revine la modul normal de aliniere, prin apelul funcției `SetTextAlign`:

**`SetTextAlign (hdc, TA_LEFT | TA_TOP);`**

Bucloa "for" se repetă pentru toate câmpurile tabloului `sysmets` și pentru toate liniile care trebuie afișate pe ecran.

Ultimele linii afișate de către `Sysmets` nu pot fi vizibile pe ecranele de tip VGA, iar dacă se micșorează dimensiunea orizontală a ferestrei, nu veți putea vedea nici măcar valorile afișate. `Sysmets` nu cunoaște dimensiunea zonei client. Afișarea textului începe cu prima linie și se termină cu ultima linie, bazându-se pe faptul că Windows nu afișează ceea ce nu se încadrează în suprafața client.

## Laborator 4 - Scroll Bar-uri

### 4.1 Dimensiunea suprafeței client

Dimensiunile ferestrelor în Windows pot varia în limite foarte largi. O metodă frecventă de determinare a dimensiunilor ferestrei se realizează prin procesarea mesajului WM\_SIZE. Windows trimite un mesaj WM\_SIZE către procedura de fereastră când dimensiunile ferestrei se modifică. Argumentul LPARAM trimis către procedura de fereastră conține în cuvântul mai puțin semnificativ lățimea suprafeței client, iar înălțimea suprafeței client în cuvântul mai semnificativ. Secvența de program care realizează procesarea acestui mesaj se face astfel:

```
short cxClient, cyClient;
long WMSize (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    cxClient = LOWORD (lParam);
    cyClient = HIWORD (lParam);
    return 0;
}
```

LOWORD și HIWORD sunt macrouri definite în Windef.h. Mesajul WM\_SIZE este urmat de un mesaj WM\_PAINT deoarece stilul clasei de fereastră conținea constantele CS\_HREDRAW și CS\_VREDRAW. Acest stil de clasă de fereastră determină Windows-ului să solicite redesenarea suprafeței client dacă dimensiunea orizontală sau cea verticală se modifică.

Numărul liniilor de text care se pot afișa în noua suprafață client se poate calcula cu expresia  $\text{cyClient} / \text{cyChar}$ . Acest număr poate fi eventual zero, dacă înălțimea suprafeței client este prea mică pentru a afișa integral o linie de text. Numărul mediu de caractere care poate fi afișat într-o linie în cadrul suprafeței client poate fi calculat cu formula  $\text{cxClient} / \text{cxChar}$ .

### 4.2 Scroll bar-uri

Scroll bar-urile pot fi folosite pentru afișarea informațiilor (texte sau grafice) care necesită un spațiu de desenare mai mare decât cel existent în suprafața client.

Scroll bar-urile pot fi de două tipuri: verticale (pentru deplasarea sus / jos a informației afișate) sau orizontale (pentru deplasarea stânga / dreapta). Un buton ("thumb" sau "scroll box") se afla în interiorul scroll bar-ului pentru a indica poziția aproximativă a porțiunii afișate în suprafața client în relație cu întregul document.

Pentru a include un scroll bar vertical în fereastra de aplicație este necesară includerea identicatorului WS\_VSCROLL (scroll bar vertical) sau WS\_HSCROLL (scroll bar orizontal) sau ambele în stilul de fereastră utilizat pentru apelul funcției CreateWindow. Scroll bar-urile astfel create sunt afișate în partea dreaptă și de jos a ferestrei, pe întreaga înălțime, respectiv lățime a ferestrei. Suprafața client nu conține spațiul ocupat de scroll bar-uri. Dimensiunea orizontală a scroll bar-ului vertical și dimensiunea verticală a scroll bar-ului orizontal sunt constante care depind de tipul de monitor folosit. Obținerea acestor valori (în pixeli) se poate face prin apelul funcției GetSystemMetrics.

Windows gestionează toată logica de manevrare cu ajutorul mouse-ului a scroll bar-urilor. Scroll bar-urile nu au, însă și o interfață cu tastatura. Interfață cu tastatura trebuie realizată de către program.

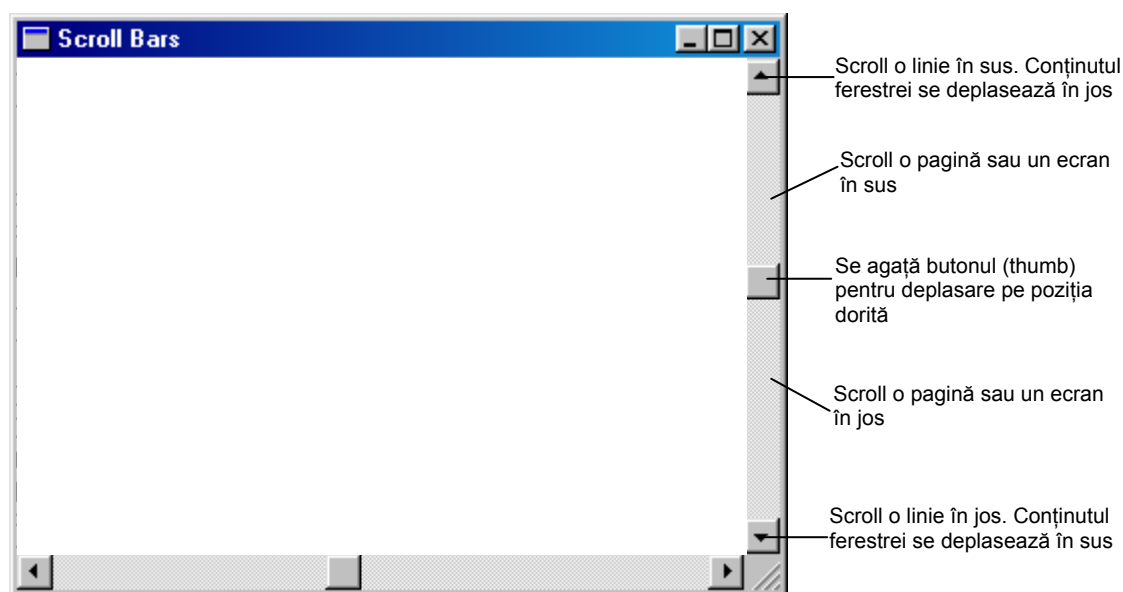


Figura 4.1. Acțiunile posibile cu mouse-ul asupra scroll bar-ului vertical

### 4.3 Intervalul de scroll și poziția scroll bar-urilor

Scroll bar-urile au un "interval de scroll" și o poziție curentă. Intervalul este definit de două valori întregi denumite valoarea minimă și maximă de scroll. Când butonul de scroll se găsește în partea superioară a scroll bar-ului, poziția butonului se află la valoarea minimă a intervalului de scroll. Când butonul de scroll se află în partea inferioară a scroll bar-ului, aceasta corespunde valorii maxime a intervalului de scroll.

Poziția butonului de scroll ia valori discrete (asociate numerelor întregi din interiorul intervalului de scroll). De exemplu, butonul unui scroll bar cu un interval între 0 și 4 are cinci locații posibile. Implicit intervalul asociat unui scroll bar este între 0 (sus sau stânga) și 100 (jos sau dreapta), modificarea intervalului de scrollare se realizează cu ajutorul funcției:

#### **SetScrollRange (hwnd, nBar, nMin, nMax, bRedraw)**

Argumentul nBar poate fi SB\_VERT (pentru scroll bar-ul vertical) sau SB\_HORZ (pentru scroll bar-ul orizontal), argumentele nMin și nMax sunt valorile pozițiilor de minim și de maxim ale scroll bar-ului, iar bRedraw are valoarea TRUE dacă se dorește redesenarea scroll bar-ului pe baza noului interval.

Funcția SetScrollPos poate fi utilizată pentru setarea unei noi poziții a butonului de scroll în interiorul intervalului de scrollare:

#### **SetScrollPos (hwnd, nBar, nPos, bRedraw)**

Argumentul nPos reprezintă noua poziție a butonului de scroll. Această valoare trebuie să se încadreze între nMin și nMax. Citirea domeniului curent asociat unui scroll bar poate fi obținut cu ajutorul funcției GetScrollRange, iar poziția curentă poate fi citită cu funcția GetScrollPos.

Pentru a putea utiliza scroll bar-urile, programul trebuie să împartă cu Windows responsabilitatea gestiunii acestora și actualizarea poziției butonului de scroll. Windows asigură:

- Gestiunea logicii de interacțiune a scroll bar-urilor cu mouse-ul
- Asigurarea unei reacții vizuale în momentul în care utilizatorul acționează cu mouse-ul asupra unui scroll bar.
- Afișarea unui buton "fantomă" când utilizatorul "agață" butonul unui scroll bar.
- Trimiterea de mesaje de scroll către procedura de fereastră a ferestrei care conține scroll bar-urile.

Programul trebuie să asigure:

- Inițializarea domeniului scroll bar-ului.
- Procesarea mesajelor de scroll.
- Actualizarea poziției butonului de scroll.

#### 4.4 Mesajele de scroll

Windows trimite către procedura de fereastră mesajele WM\_VSCROLL și WM\_HSCROLL de fiecare dată când se realizează o acțiune cu mouse-ul asupra scroll bar-urilor. Fiecare acțiune a mouse-ului asupra scroll bar-ului generează cel puțin două mesaje: un mesaj când este apăsat butonul de mouse într-o zonă a scroll bar-ului și un mesaj când butonul de mouse este eliberat.

Valoarea cuvântului cel mai puțin semnificativ din wParam și anume LOWORD(wParam), care însoțește mesajele WM\_VSCROLL și WM\_HSCROLL descrie acțiunea specifică a mouse-ului asupra scroll bar-ului. Valorile posibile din argumentul wParam au asociate constante definite în fișierul Winuser.h și care încep cu prefixul SB\_:

- **SB\_TOP**: scroll bar-ul a ajuns pe poziția de valoare minimă (maxim sus sau maxim stânga).
- **SB\_PAGEUP**: butonul de mouse a fost apăsat în suprafața dintre butonul de scroll și butonul de increment negativ de scroll (identificat printr-o săgeată în sus sau la stânga).
- **SB\_LINEUP**: butonul de mouse a fost apăsat în suprafața butonului de increment negativ de scroll (identificat prin săgeată în sus sau la stânga).
- **SB\_LINEDOWN**: butonul de mouse a fost apăsat în suprafața butonului de increment pozitiv de scroll (identificat prin săgeată în jos sau la dreapta).
- **SB\_PAGEDOWN**: butonul de mouse a fost apăsat în suprafața dintre butonul de scroll și butonul de increment pozitiv de scroll (identificat printr-o săgeată în jos sau la dreapta).
- **SB\_BOTTOM**: scroll bar-ul a ajuns pe poziția de valoare maximă (maxim jos sau maxim dreapta).
- **SB\_THUMBTRACK**: butonul de mouse a fost apăsat în suprafața butonului de scroll, după care poziția mouse-ului s-a modificat. Argumentul HIWORD(wParam) conține în noua valoare curentă a scroll bar-ului.
- **SB\_THUMBPOSITION**: Butonul de mouse (care anterior fusese apăsat în suprafața butonului de scroll) a fost ridicat. Butonul de scroll se află pe o nouă poziție care este trimisă în cuvântul mai puțin semnificativ al argumentul lParam.

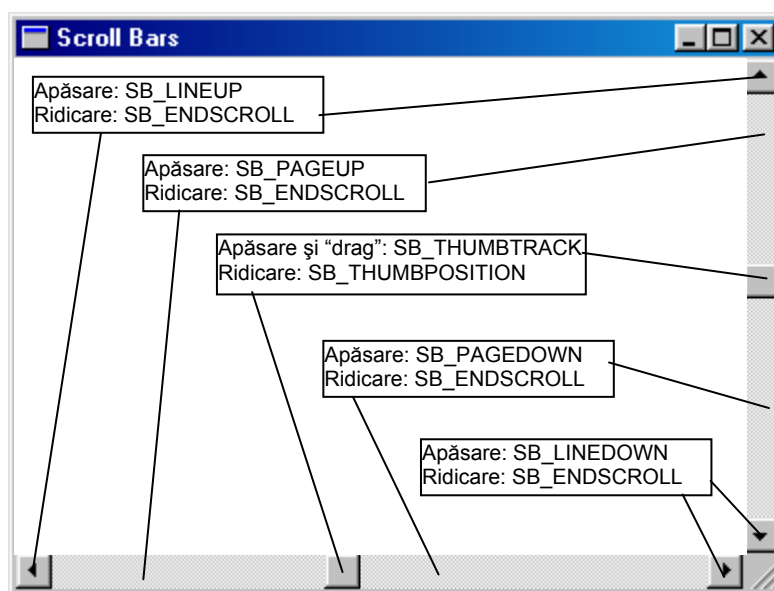


Figura 4.2

Aceste valori pot fi utilizate în conjuncție cu evenimentele provenite atât de la scroll bar-ul vertical cât și de la cel orizontal. Deși acești identificatori utilizează cuvintele "UP" și "DOWN", ei se referă și la evenimentele provenite de la scroll bar-ul orizontal, așa cum se poate observa în figura 4.2. Procedura de fereastră poate să recepționeze mai multe mesaje SB\_LINEUP, SB\_PAGEUP, SB\_LINEDOWN sau SB\_PAGEDOWN dacă butonul mouse-ului este ținut apăsat într-o anumită zonă a scroll bar-ului. Mesajul SB\_ENDSCROLL semnalează

procedurii de fereastră faptul că butonul mouse-ului a fost eliberat. În general mesajul SB\_ENDSCROLL poate fi ignorat.

Când valoarea lui LOWORD(wParam) este SB\_THUMBTRACK sau SB\_THUMBPOSITION, cuvântul cel mai semnificativ al lui wParam indică poziția curentă a butonului de scroll-are care a fost "agățat". Aceasta valoare se găsește întotdeauna în intervalul dintre valoarea minimă și maximă a domeniului de scroll. Valoarea poate fi citită cu ajutorul macro-ului HIWORD și va fi întoarsă valoarea cuvântului cel mai semnificativ al argumentului wParam (pe 32 biți).

Pentru celelalte valori ale lui wParam, valoarea lui HIWORD(wParam) trebuie ignorată. Dacă utilizatorul apasă butonul mouse-ului deasupra butonului de scroll, Windows va trimite mesajul SB\_THUMBTRACK de fiecare dată când mouse-ul (cu butonul stânga apăsat) este deplasat pe o nouă poziție. Dacă programatorul alege să urmărească poziția scroll bar-ului în timpul agățării butonului de scroll, el va trebui să redeseneze suprafața client la fiecare mesaj SB\_THUMBTRACK, utilizând valoarea curentă a scroll bar-ului. Dacă desenarea este lentă, se poate redesena suprafața client doar la primirea mesajului SB\_THUMBPOSITION, adică în momentul în care utilizatorul a deplasat butonul de scroll pe noua poziție și a eliberat butonul mouse-ului.

Documentația de Windows indică faptul că wParam poate avea și valorile SB\_TOP și SB\_BOTTOM, semnificând faptul că butonul scroll bar-ului a fost pus pe poziția de maxim, respectiv minim.

### 4.5 O aplicație Windows care utilizează scroll bar-uri

Exemplul prezintă o aplicație Windows care utilizează scroll bar-uri. Caracteristicile aplicației sunt următoarele:

- Programul afișează informația într-un dreptunghi definit de coordonatele: (xMin, yMin) respectiv (xMax, yMax). Suprafața client a ferestrei este calculată prin procesarea mesajului WM\_SIZE. Suprafața client este cuprinsă între (0, 0) și (cxClient, cyClient). Dacă suprafața client este prea mică pentru afișarea zonei necesare, programul utilizează scroll bar-uri.
- Pozițiile curente ale scroll bar-ului orizontal, respectiv vertical sunt memorate în variabilele nHScrollPos și, respectiv nVScrollPos. Domeniile scroll bar-urilor sunt: pentru scroll bar-ul orizontal 0..nHScrollMax, respectiv pentru scroll bar-ul vertical 0..nVScrollMax. Incrementul de pagină a scroll bar-urilor sunt: nVScrollPage și nHScrollPage și sunt egale cu numărul de pixeli necesari pentru a scrolla un ecran întreg.
- Aplicația specifică la apelul funcției CreateWindow ca flaguri pentru stilul de fereastră identificatorii WS\_HSCROLL și WS\_VSCROLL, necesari pentru includerea scroll bar-urilor în fereastra aplicației.

#### 4.5.1 Mesajele prelucrate de aplicație

Aplicația procesează următoarele mesaje:

1. **WM\_CREATE**: trimis în momentul creării ferestrei. Mesajul este utilizat pentru inițializarea poziției curente de scroll.
2. **WM\_SIZE**: trimis când dimensiunile ferestrei au fost modificate. Argumentul lParam conține în cuvântul mai puțin semnificativ noua dimensiune orizontală a suprafeței client, iar în cuvântul mai semnificativ noua dimensiune verticală a suprafeței client. Mesajul este utilizat pentru recalcularea dimensiunilor suprafeței client:

**cxClient = LOWORD (lParam);**

**cyClient = HIWORD (lParam);**

precum și pentru calculul domeniilor și incrementelor de pagina ale scroll bar-urilor pe baza noilor dimensiuni ale suprafeței client. Scroll bar-urile sunt reafişate (doar dacă domeniul de scrollare este nenul), pentru a reflecta noile valori calculate.

3. **WM\_VSCROLL**: trimis de către scroll bar-ul vertical când s-a produs o acțiune de mouse asupra sa. În funcție de valoarea parametrului LOWORD(wParam) se realizează următoarele acțiuni:

- **SB\_TOP**: Poziția curentă ia valoarea zero.
- **SB\_PAGEUP**: Poziția curentă se decrementează cu valoarea nVScrollPage.
- **SB\_LINEUP**: Poziția curentă se decrementează cu o unitate.
- **SB\_LINEDOWN**: Poziția curentă se incrementează cu o unitate.
- **SB\_PAGEDOWN**: Poziția curentă se incrementează cu valoarea nVScrollPage.
- **SB\_BOTTOM**: Poziția curentă ia valoarea nVScrollMax.
- **SB\_THUMBTRACK**: Poziția curentă ia valoarea cuvântului cel mai semnificativ al argumentului wParam.
- **SB\_THUMBPOSITION**: Poziția curentă ia valoarea cuvântului cel mai semnificativ al argumentului wParam.

Incrementul de poziție este calculat de către funcția GetScrollInc. Funcția GetScrollInc testează valoarea argumentului LOWORD(wParam) și funcție de aceasta valoare calculează o valoare care, prin însumare va actualiza poziția curentă (nVScrollPos) a scroll bar-ului. Dacă valoarea incrementului este nenulă, programul actualizează valoarea curentă a scroll bar-ului și cere redesenarea lui, apoi scrolează conținutul ferestrei cu un număr de pixeli egal cu incrementul de poziție. După scrollarea ferestrei este chemată funcția UpdateWindow care trimite un mesaj WM\_PAINT către procedura de fereastră pentru desenarea suprafeței invalide rămase prin scrollare.

Funcția ScrollWindow permite scrollarea zonei client a aplicației. Informația grafică din suprafața client a ferestrei este copiată pixel cu pixel pe noua poziție, definită ca deplasament față de colțul stânga sus al suprafeței client. Ca urmare a apelului funcției ScrollWindow, zona client rămasă ne-scroll-ată este declarată zonă invalidă.

Funcția UpdateWindow trimite un mesaj WM\_PAINT procedurii de fereastră asociată ferestrei specificată de către primul argument al funcției.

În comparație cu funcția InvalidateRect, funcția UpdateWindow nu introduce mesajul în coada de mesaje a aplicației, ci cheamă direct procedura de fereastră. Mesajele WM\_PAINT sunt mesaje de prioritate redusă, din acest motiv, ele nu sunt trimise către procedura de fereastră decât dacă nu exista alte mesaje în coada de mesaje a aplicației. Funcția UpdateWindow poate fi chemată dacă este necesară o redesenare imediată a suprafeței client. În caz contrar este preferabil apelul funcției InvalidateRect.

4. **WM\_HSCROLL**: trimis de către scroll bar-ul orizontal când s-a produs o acțiune de mouse asupra sa. Prelucrările realizate în acest mesaj sunt identice cu cele din mesajul WM\_VSCROLL, cu singura deosebire că ele interesează poziția curentă a scroll bar-ului orizontal.
5. **WM\_PAINT**: desenează un dreptunghi în suprafața client. Coordonatele dreptunghiului sunt relative la poziția curentă a scroll bar-urilor. Pentru îmbunătățirea vitezei de desenare este utilă folosirea coordonatelor dreptunghiului invalid conținut în câmpul rcPaint al variabilei ps (de tipul PAINTSTRUCT).
6. **WM\_DESTROY**: trimis în momentul închiderii ferestrei. Introduce un mesaj WM\_QUIT în coada de evenimente a aplicației.

#### 4.6 Temă

Să se asocieze o interfață cu scroll bar-uri programului Sysmets discutat în capitolul anterior. Aplicația se va numi Sysmets2.

## Laboratorul 5 - Tastatura

Programele Windows realizează interacțiunea cu utilizatorul folosind ca dispozitive de intrare tastatura și mouse-ul.

### 5.1 *Elementele fundamentale de tratare a evenimentelor provenind de la tastatură*

Apăsarea tastelor provoacă introducerea de mesaje în coada de mesaje a sistemului. Windows extrage mesajele legate de tastatură din coada de mesaje sistem și le introduce în coada de mesaje a aplicației active (ce are "input focus"). Aceste mesaje sunt procesate de către procedurile de fereastră ale programului respectiv.

#### 5.1.1 **Driverul de tastatură**

Driverul de tastatură este un program aflat în fișierul Keyboard.drv. La boot-area Windows-ului se activează driver-ul de tastatură. Acesta instalează o rutină proprie de tratare a întreruperilor hardware de la tastatură.

Apăsarea tastelor generează întreruperea 09H. Aceasta este o întrerupere asincronă, deoarece poate interveni la orice moment de timp. Întreruperea oprește execuția programului curent și trece controlul rutinei care gestionează întreruperea de tastatură. După ce se realizează procesarea întreruperii provocate de apăsarea unei taste, execuția trece înapoi la programul întrerupt. Rutina de tratare a întreruperilor din Keyboard.drv realizează decodificarea tastelor, după care cheamă o funcție din biblioteca de sistem User.exe pentru a introduce mesajul de tastatură în coada de mesaje sistem. Aplicația Windows poate obține mesajul prin intermediul funcției GetMessage.

Windows trimite opt tipuri de mesaje diferite referitoare la evenimentele provenite de la tastatură. Majoritatea acestor mesaje pot fi ignorate și trimise funcției DefWindowProc pentru procesare implicită. Toate funcțiile de sistem care sunt invocate prin apăsarea unei taste sau a unei combinații de taste sunt realizate de către procedura de prelucrare implicită de mesaje DefWindowProc.

#### 5.1.2 **Fereastră selectată**

Tastatura este utilizată de către toate programele care se execută la un moment dat în mediul Windows. Anumite aplicații au ferestre multiple care, la rândul lor trebuie să împartă intrarea de la tastatură. În momentul apăsării unei taste, o singură fereastră primește mesajul de tastă apăsată. Fereastră care primește mesajele de la tastatură de numește fereastră selectată, "focused".

Conceptul de fereastră selectată este strâns legat de cel de fereastră activă. Fereastră care primește intrarea de la tastatură este fie fereastră activă, fie o fereastră copil a acesteia. Fereastră activă poate fi ușor identificată. Dacă fereastră activă are un Caption Bar, Windows evidențiază Caption Bar-ul ferestrei active, desenându-l cu o culoare diferită decât al celorlalte ferestre. Dacă fereastră activă este un icon, Windows desenează fundalul textului din dreptul icon-ului cu o culoare diferită.

Cele mai frecvente ferestre copil sunt "controalele" cum ar fi: butoane, butoane exclusive - radio buttons, butoane non-exclusive - check boxes, scroll bar-uri, ferestrele și liniile de editare de text, listele - list boxes. Toate aceste controale apar de cele mai multe ori în ferestre de dialog. Ferestrele copil nu sunt niciodată ferestre active, ele pot fi însă ferestre selectate și să primească intrarea de la tastatură. Dacă o fereastră copil este selectată, atunci fereastră părinte este activă. Ferestrele "control" indică printr-o anumită modalitate faptul că dețin intrarea de la tastatură.

Procedura de fereastră poate determina dacă fereastră pentru care procesează mesajele este selectată (primește mesaje de la tastatură). Windows trimite mesajul



WM\_SETFOCUS procedurii de fereastră care primește selecția, iar mesajul WM\_KILLFOCUS este trimis către procedura ferestrei care pierde selecția. O fereastră este selectată din momentul în care a primit mesajul WM\_SETFOCUS până în momentul în care primește mesajul WM\_KILLFOCUS. În acest interval, fereastra respectivă va primi mesaje de la tastatură.

### 5.1.3 Taste și caractere

Mesajele primite de către o procedură de fereastră indică fie coduri de taste apăsate, fie caractere introduse. Aceasta corespunde modului în care poate fi privită tastatura: fie ca o colecție de taste (de exemplu: există o unică tastă "A"), fie ca un periferic de intrare care produce caractere afișabile (tasta A poate genera caracterul "a" sau caracterul "A", în funcție de starea tastei Shift, Ctrl sau Caps Lock).

Pentru tastele sau combinațiile de taste care generează caractere afișabile, Windows trimite programului atât mesajele de tastă asociate cât și un mesaj "caracter", conținând codul ASCII al caracterului rezultat. Anumite taste nu produc caractere (tastele Shift, tastele funcționale, tastele de deplasare de cursor și tastele speciale ca Insert și Delete). Pentru aceste taste, Windows generează doar mesaje de tastă.

## 5.2 Mesajele de tastă

Când utilizatorul apasă o tastă, Windows introduce un mesaj WM\_KEYDOWN sau WM\_SYSKEYDOWN în coada de mesaje a aplicației care conține fereastra selectată. Când tasta este eliberată, Windows introduce un mesaj WM\_KEYUP sau WM\_SYSKEYUP în coada de mesaje a aplicației.

	Tasta apăsată	Tasta eliberată
<b>Tasta normală</b>	WM_KEYDOWN	WM_KEYUP
<b>Tasta de sistem</b>	WM_SYSKEYDOWN	WM_SYSKEYUP

În mod obișnuit, mesajele "up" și "down" sosesc în pereche. Dacă utilizatorul ține o tastă apăsată astfel încât se produce autorepeat-ul, Windows trimite o serie de WM\_KEYDOWN (sau WM\_SYSKEYDOWN) către procedura de fereastră și un singur mesaj WM\_KEYUP când tasta este eliberată. Momentul de timp la care o tastă a fost apăsată sau eliberată poate fi obținut cu ajutorul funcției GetMessageTime.

### 5.2.1 Taste sistem și taste normale

Tastele sistem sunt tastele importante pentru sistemul Windows și mai puțin importante pentru aplicație. WM\_SYSKEYDOWN și WM\_SYSKEYUP sunt trimise în conjuncție cu apăsarea tastei Alt. Aceste combinații de taste sunt utilizate de către Windows pentru selectarea opțiunilor din meniul programului sau din meniul sistem. În mod obișnuit, programele nu procesează aceste mesaje, ele sunt trimise către DefWindowProc, care realizează toată logica de procesare a tastelor de sistem.

Dacă se dorește interceptarea acestor mesaje (ca în programul KEYLOOK, discutat mai jos), aceste mesaje trebuie trimise către DefWindowProc pentru că Windows să poată să utilizeze aceste mesaje pentru funcțiile de sistem.

Aproape orice afectează fereastra programului trece în forma de mesaj prin procedura de fereastră. Windows procesează aceste mesaje doar dacă procedura de fereastră le trimite către DefWindowProc. De exemplu, dacă în procedura de fereastră se introduc următoarele linii de program:

```
case WM_SYSKEYDOWN:
case WM_SYSKEYUP:
case WM_SYSCHAR:
return 0;
```

aplicația va dezactiva toate funcțiile de sistem (comenzile de menu, Alt+Tab, Alt+Esc, Ctrl+Esc etc.) în momentul în care programul este selectat.

Mesajele WM\_KEYDOWN și WM\_KEYUP sosesc pentru tastele care nu interesează sistemul. Procedura de fereastră poate să le intercepteze sau să le ignore. Windows nu realizează procesări asupra acestor mesaje.

### 5.3 Argumentul IParam

Pentru toate mesajele de tastă, variabila IParam trimisă ca argument procedurii de fereastră conține șase câmpuri: Contorul de repetiție, Codul de scanare OEM, Indicatorul de tastă extinsă, Codul de context, Starea anterioară a tastei și Starea curentă a tastei.

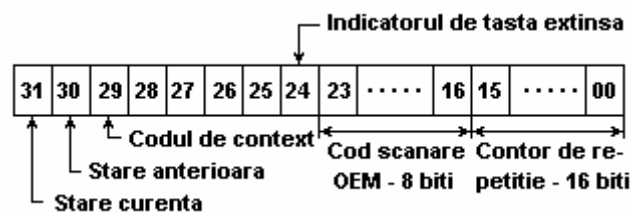


Figura. 5.1.

#### 5.3.1 Contorul de repetiție

Contorul de repetiție conține numărul de taste reprezentat de mesaj. În majoritatea cazurilor, Contorul de repetiție este 1. În cazul în care o tastă este ținută apăsată, iar procedura de fereastră nu procesează destul de rapid, Windows poate să combine mai multe mesaje WM\_KEYDOWN sau WM\_SYSKEYDOWN într-un singur mesaj și incrementează contorul de repetiție. Contorul de repetiție are valoarea 1 pentru mesajele WM\_KEYUP și WM\_SYSKEYUP.

#### 5.3.2 Codul de scanare OEM

Codul de scanare OEM este codul de scanare generat de către portul de tastatură, pentru a identifica tastă apăsată.

#### 5.3.3 Indicatorul de tastă extinsă

Dacă indicatorul de tastă extinsă are valoarea 1, tastă care a provocat mesajul este o tastă extinsă. Acest indicator are valoarea 1 pentru tastele Alt și Ctrl din partea dreaptă a tastaturii, tastele de deplasare de cursor (inclusiv Insert și Delete) care nu fac parte din tastatura numerică, tastele Slash și Enter care fac parte din tastatură numerică și tasta Num Lock. Aplicațiile ignoră în general acest indicator.

#### 5.3.4 Codul de context

Codul de context are valoarea 1 dacă tasta Alt este apăsată. Acest bit are valoare 1 pentru mesajele WM\_SYSKEYDOWN și WM\_SYSKEYUP.

#### 5.3.5 Starea anterioară a tastei

Bitul de stare anterioară are valoarea 0 dacă anterior mesajului, tasta era eliberată și valoarea 1 dacă aceasta era apăsată. Valoarea ei este totdeauna 1 pentru mesajul WM\_KEYUP și WM\_SYSKEYUP. Pentru mesajele WM\_KEYDOWN și WM\_SYSKEYDOWN, valoarea 1 indică faptul că a apărut autorepeat.

#### 5.3.6 Starea curentă a tastei

Bitul de stare curentă a tastei are valoarea 0 dacă tasta este apăsată și 1 dacă tasta este eliberată. Acest câmp are valoarea 0 pentru mesajele WM\_KEYDOWN și WM\_SYSKEYDOWN, respectiv 1 pentru mesajele WM\_KEYUP și WM\_SYSKEYUP.

### 5.4 Codurile virtuale de tastă

Argumentul wParam conține codul virtual al tastei care a generat mesajul. Acest cod identifică unic tasta care prin apăsare sau eliberare a generat mesajul. Aceste coduri virtuale definesc tastele independent de periferic, adică de tipul de tastatură instalat (de aici provine și cuvântul "virtual"). Anumite coduri virtuale de tastă nu pot fi generate de tastaturile utilizate pe calculatoare IBM PC sau compatibile, dar pot fi întâlnite la tastaturile altor producători.

Codurile de taste cele mai des utilizate au identificatori definiți în Winuser.h. Acești identificatori au prefixul VK\_.

#### 5.4.1 Starea tastelor de shift

Informația despre starea oricărei taste, la momentul producerii mesajului, poate fi obținută prin apelul funcției `GetKeyState`. Această funcție este utilizată în mod frecvent pentru obținerea stării tastelor de shift (Shift, Ctrl și Alt) precum și a tastelor de comutație: Caps Lock și Num Lock. De exemplu:

**`GetKeyState (VK_SHIFT)`**

Întoarce o valoare negativă (bitul cel mai semnificativ este setat) dacă tasta Shift este apăsată.

Valoarea întoarsă de către apelul:

**`GetKeyState (VK_CAPITAL)`**

are valoarea 1 (bitul cel mai puțin semnificativ este setat) dacă Caps Lock este comutat (activ).

Funcția `GetKeyState` întoarce starea tastei la momentul generării mesajului care este procesat. Aceasta funcție nu testează starea curentă a tastei, deci următoarea secvență de cod care utilizează `GetKeyState` pentru a aștepta apăsarea tastei F1 este eronată:

**`while (GetKeyState (VK_F1) == 0); // Eronat`**

deoarece valoarea întoarsă de `GetKeyState` este aceeași în timpul procesării unui mesaj, indiferent de starea curentă a tastei. Sincronizarea între `GetKeyState` și mesajul procesat este utilă programului, deoarece în general acesta are nevoie de starea tastelor de Shift pentru mesajul de tastă procesat. `GetKeyState` întoarce valoarea corectă chiar dacă momentul de procesare a mesajului de tastă se petrece după ce tasta de shift a fost eliberată. Pentru obținerea stării curente a tastelor se poate utiliza funcția `GetAsyncKeyState`.

#### 5.4.2 Procesarea mesajelor de tastă

În general o aplicație procesează relativ puține mesaje generate de tastatură: mesajele `WM_SYSKEYDOWN` și `WM_SYSKEYUP` sunt necesare în special funcțiilor de sistem Windows și sunt trimise funcției `DefWindowProc`. Mesajul `WM_KEYUP` poate fi de asemenea ignorat.

Singurul mesaj procesat este `WM_KEYDOWN`, utilizat pentru detectarea apăsării de taste care nu generează caractere (taste de deplasare a cursorului etc.). Deși aparent este posibilă utilizarea mesajelor de tastă împreună cu informația asupra tastelor shift oferită de funcția `GetKeyState` pentru a transla mesajele de tastă în caractere afișabile, mesajul `WM_CHAR` este mai sigur și produce rezultate corecte indiferent de tipul de tastatură instalată. Deci mesajele `WM_KEYDOWN` sunt utile pentru tastele de deplasare a cursorului, tastele funcționale și tastele speciale ca Insert și Delete.

#### 5.4.3 Interfață de tastatură pentru programul Sysmets

Pentru adăugarea unei interfețe de tastatură programului Sysmets descris în laboratorul 4, vom utiliza tastele de deplasare a cursorului care vor dubla evenimentele produse de mouse asupra scroll bar-ului. Pentru deplasarea verticală vom utiliza tastele săgeată sus și jos, `PageUp` și `PageDown`, respectiv Home - deplasarea la prima linie a textului și End - deplasarea la ultima linie a textului. Pentru scroll-ul orizontal vom utiliza tastele săgeată stânga și dreapta.

#### 5.4.4 Funcția de procesare a mesajelor de tastă

Pentru procesarea mesajelor de tastă s-ar putea scrie o secvență de program foarte asemănătoare celei utilizate în funcțiile `WMVScroll` și `WMHScroll` și care, în funcție de tasta apăsată, să calculeze noua poziție a scroll bar-ului, să reafișeze scroll bar-ul, dacă este necesar, să scroleze fereastra client, precum și să trimită mesaj de redesenare a suprafeței client către procedura de fereastră. Rezultatul ar fi copierea unor porțiuni mari de cod.

O cale mai simplă este aceea de a transla mesajele de la tastatură în mesaje de scroll bar, pe care apoi să le trimitem procedurii de fereastră.

#### 5.4.5 Trimiterea de mesaje către procedura de fereastră

Funcția Windows care permite trimiterea de mesaje către o procedură de fereastră este:

**`SendMessage (hwnd, message, wParam, lParam);`**

Apelul funcției `SendMessage` provoacă apelul procedurii de fereastră asociate ferestrei identificate prin handle-ul de fereastră `hwnd` cu tipul de mesaj specificat de către argumentul `message` și argumentele specificate în `wParam` și `lParam`. În momentul în care procedura de fereastră termină procesarea mesajului respectiv, funcția `SendMessage` returnează valoarea întoarsă de către procedura de fereastră. Procedura de fereastră chemată de către `SendMessage` poate fi aceeași procedură de fereastră care apelează funcția `SendMessage` sau o procedură de fereastră asociată altei ferestre.

Apelul procedurii de fereastră se face cu ajutorul funcției `SendMessage`. Chiar dacă cunoaștem adresa procedurii de fereastră, este interzis apelul direct al acestei funcții. O instrucțiune de forma:

```
WndProc (hwnd, message, wParam, lParam); /* Greșit ! */
```

ce chemă direct procedura de fereastră va provoca blocarea sistemului!

Procesarea mesajelor de tastă se va face în următorul mod::

```
long WMKeyDown (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    switch (wParam) {
        case VK_HOME: /* Home */
            SendMessage (hwnd, WM_VSCROLL, SB_TOP, 0L);
            break;
        case VK_END: /* End */
            SendMessage (hwnd, WM_VSCROLL, SB_BOTTOM, 0L);
            break;
        case VK_PRIOR: /* PageUp */
            SendMessage (hwnd, WM_VSCROLL, SB_PAGEUP, 0L);
            break;
        case VK_NEXT: /* PageDown */
            SendMessage (hwnd, WM_VSCROLL, SB_PAGEDOWN, 0L);
            break;
        case VK_UP: /* Săgeată sus */
            SendMessage (hwnd, WM_VSCROLL, SB_LINEUP, 0L);
            break;
        case VK_DOWN: /* Săgeată jos */
            SendMessage (hwnd, WM_VSCROLL, SB_LINEDOWN, 0L);
            break;
        case VK_LEFT: /* Săgeată stânga */
            SendMessage (hwnd, WM_HSCROLL, SB_PAGEUP, 0L);
            break;
        case VK_RIGHT: /* Săgeată dreapta */
            SendMessage (hwnd, WM_HSCROLL, SB_PAGEDOWN, 0L);
            break;
    }
    return 0;
}
```

Deci tastele de deplasare a cursorului generează mesaje de scroll care sunt retransmise procedurii de fereastră prin intermediul funcției `SendMessage`. Când procedura de fereastră primește aceste mesaje, ea va reacționa exact în același mod în care reacționează la mesajele de scroll provenite de la "adevăratele" scroll bar-uri.

Urmează prelucrarea mesajului `WM_KEYDOWN` în interiorul funcției `WndProc` sub forma:

```
case WM_KEYDOWN: return WMKeyDown (hwnd, wParam, lParam);
```

Acestea sunt singurele diferențe între programul `Sysmets` fără interfață pentru tastatură și cel care oferă o interfață și pentru tastatură.

## 5.5 Mesaje de caracter

Windows realizează traducerea mesajelor de tastă în mesaje de caracter (dacă tasta sau combinațiile de taste definesc un caracter afișabil). Traducerea între taste și caractere afișabile se realizează în funcție de tipul de tastatură instalat.

Traducerea mesajelor de tastă în mesaje caracter se realizează chiar în bucla de procesare de mesaje. Bucla de mesaje a unei aplicații se realizează printr-un ciclu while de forma:

```
while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}
```

Funcția GetMessage copiază în variabila msg informațiile referitoare la mesajul găsit în coada de mesaje a aplicației. DispatchMessage apelează procedura de fereastră asociată ferestrei care a primit mesajul extras din coada de mesaje. Între aceste două funcții, TranslateMessage transformă mesajele de tastă în mesaje caracter. Dacă mesajul de tastă este WM\_KEYDOWN sau WM\_SYSKEYDOWN, iar combinațiile tastelor Ctrl, Shift și Alt cu tasta apăsată generează (în funcție de tipul de tastatură) un caracter afișabil, funcția TranslateMessage introduce un mesaj WM\_CHAR în coada de mesaje a aplicației. Mesajul WM\_CHAR va fi următorul mesaj pe care funcția GetMessage îl va extrage (la următoarea iterație) din coada de mesaje a aplicației.

Există patru tipuri de mesaje caracter:

	Caracter	Caracter "mort"
Caracter normal	WM_CHAR	WM_DEADCHAR
Caracter de sistem	WM_SYSCHAR	WM_SYSDEADCHAR

Mesajele WM\_CHAR și WM\_DEADCHAR sunt obținute din transformarea mesajului WM\_KEYDOWN. Mesajele WM\_SYSCHAR și WM\_SYSDEADCHAR sunt obținute ca urmare a traducerii unui mesaj WM\_SYSKEYDOWN. Un program Windows procesează în general doar mesajul WM\_CHAR. Argumentul lParam trimis procedurii de fereastră este identic cu argumentul lParam descris la mesajele de tastă. Argumentul wParam reprezintă codul ASCII al caracterului introdus.

Mesajele caracter sosesc la procedura de fereastră între mesaje de tastă. Secvența de mesaje generată de apăsarea tastei "A" este următoarea:

Mesajul	wParam
WM_KEYDOWN	VK_A (0x41)
WM_CHAR	Cod ASCII pentru caracterul "a" (0x41)
WM_KEYUP	VK_A (0x41)

Pentru un caracter A majuscul, generat prin apăsarea tastei Shift, urmată de tasta A, secvența de mesaje va fi următoarea:

Mesajul	wParam
WM_KEYDOWN	VK_SHIFT (0x10)
WM_KEYDOWN	VK_A (0x41)
WM_CHAR	Cod ASCII pentru caracterul "A" (0x61)
WM_KEYUP	VK_A (0x41)
WM_KEYUP	VK_SHIFT (0x10)

Apăsarea tastei Shift nu generează un mesaj caracter.

Dacă tasta A este ținută apăsată, astfel încât să se producă autorepeat-ul, secvența de mesaje generată va fi următoarea:

Mesajul	wParam
WM_KEYDOWN	VK_A (0x41)
WM_CHAR	Cod ASCII pentru caracterul "a" (0x41)
WM_KEYDOWN	VK_A (0x41)
WM_CHAR	Cod ASCII pentru caracterul "a" (0x41)
WM_KEYDOWN	VK_A (0x41)
WM_CHAR	Cod ASCII pentru caracterul "a" (0x41)
WM_KEYDOWN	VK_A (0x41)
WM_CHAR	Cod ASCII pentru caracterul "a" (0x41)
WM_KEYUP	VK_A (0x41)

Apariția unor mesaje WM\_KEYDOWN cu contorul de repetiție mai mare decât 1 va genera apariția de mesaje WM\_CHAR cu același contor de repetiție.

Tasta Ctrl împreună cu anumite taste poate genera coduri ASCII cuprinse între 0x01 (Ctrl+A) și 0x1A (Ctrl+Z). Unele dintre aceste coduri pot fi generate și de către taste independente:

Tasta	Codul ASCII	Ctrl+Tastă
Backspace	0x08	Ctrl+H
Tab	0x09	Ctrl+I
Ctrl+Enter	0x0A	Ctrl+J
Enter	0x0D	Ctrl+M
Esc	0x1B	Ctrl+[

### 5.5.1 Mesajele WM\_CHAR

Când programul trebuie să proceseze caracterele introduse de la tastatură (de exemplu un program care realizează editarea de texte), acesta va procesa mesajul WM\_CHAR. Anumite coduri ASCII (Backspace, Tab, Enter etc) vor avea asociate procesări speciale. Restul caracterelor alfanumerice vor fi procesate în același mod. Funcția de procesare a mesajului WM\_CHAR va avea următoarea formă:

```

long WMChar (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    int i;
    for (i = 0; i < LOWORD (lParam); i++)    // LOWORD (lParam) = contor de
                                                // repetare
    {
        switch (wParam) {
            case 'b':    // backspace
                ....
                break;
            case '\n':    // linefeed
                ....
                break;
            case '\r':    // carriage return
                ....
                break;
            default:    // caractere afișabile normale
                ....
                break;
        }
    }
    return 0;
}

```

### 5.5.2 Mesaje caracter "moarte"

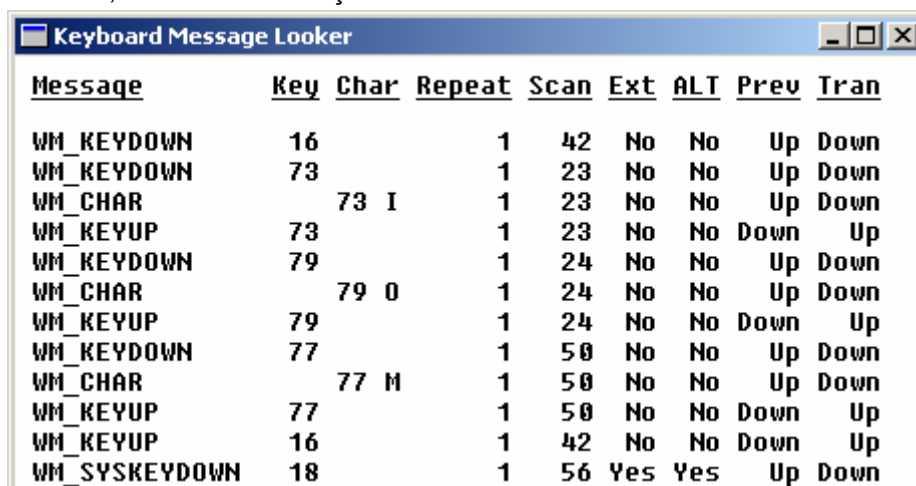
Programele Windows ignoră în general mesajele caracter "moarte": WM\_DEADCHAR și WM\_SYSDEADCHAR. Aceste mesaje sunt destinate tastaturilor diferite de cele U.S. Anumite tastaturi au o tastă specială pentru a adăuga diacritice deasupra literelor. Tastele care adaugă diacritice sunt denumite "taste moarte" deoarece ele nu generează caractere.

Procedura de fereastră primește un mesaj de caracter "mort" în momentul în care utilizatorul apasă o tastă "moartă". Parametrul wParam conține codul ASCII pentru semnul diacritic. Când utilizatorul apasă o tastă (tasta A de exemplu), procedura de fereastră primește un mesaj WM\_CHAR care conține în câmpul wParam codul ASCII al diacritice respective.

### 5.5.3 Afișarea mesajelor de tastatură

Aplicația KeyLook afișează în suprafața client informații referitoare la toate mesajele de la tastatură pe care le primește.

Afișarea liniilor de text referitoare la mesajele de la tastatură se realizează prin scrollarea în sus a suprafeței client a ferestrei (cu un offset egal cu înălțimea unei linii de text). După scroll-are, pe ultima linie este afișată informația referitoare la mesajul recepționat, prin apelul funcției TextOut. În figura 5.2. se observă rezultatele aplicației KeyLook după introducerea cuvântului "IOM". Prima coloană afișată indică tipul mesajului primit. A doua coloană indică codul virtual de tastă pentru mesajele de tastă. Cea de-a treia coloană indică codul ASCII și caracterul pentru mesajele de caracter.. Următoarea coloană conține contorul de repetiție. Următoarele coloane conțin codul OEM de scanare, indicatorul de tastă extinsă, codul de context, starea anterioară și starea curentă a tastei.



Message	Key	Char	Repeat	Scan	Ext	ALT	Prev	Tran
WM_KEYDOWN	16		1	42	No	No	Up	Down
WM_KEYDOWN	73		1	23	No	No	Up	Down
WM_CHAR		73 I	1	23	No	No	Up	Down
WM_KEYUP	73		1	23	No	No	Down	Up
WM_KEYDOWN	79		1	24	No	No	Up	Down
WM_CHAR		79 O	1	24	No	No	Up	Down
WM_KEYUP	79		1	24	No	No	Down	Up
WM_KEYDOWN	77		1	50	No	No	Up	Down
WM_CHAR		77 M	1	50	No	No	Up	Down
WM_KEYUP	77		1	50	No	No	Down	Up
WM_KEYUP	16		1	42	No	No	Down	Up
WM_SYSKEYDOWN	18		1	56	Yes	Yes	Up	Down

Figura. 5.2.

Pentru a se ușura aranjarea textului pe coloane a fost utilizat un set de caractere de dimensiune fixă. Selectarea fontului fix de sistem în device context-ul ferestrei s-a realizat prin apelul a două funcții Windows într-o singură linie:

**SelectObject (hdc, GetStockObject (SYSTEM\_FIXED\_FONT));**

KeyLook selectează fontul fix de sistem de fiecare dată când obține un device context. Funcția GetStockObject obține un handle către fontul descris de către indicatorul SYSTEM\_FIXED\_FONT. Funcția SelectObject introduce acest font în device context, astfel încât funcțiile de scriere de texte vor utiliza acest font. Este posibil să se revină la fontul de dimensiune variabilă (implicit) prin apelul funcțiilor:

**SelectObject (hdc, GetStockObject (SYSTEM\_FONT));**

Funcția ShowKey apelează ScrollWindow pentru a scroll-a în sus liniile anterioare de text afișate în suprafața client a ferestrei, înaintea afișării unei noi linii de text. În mod normal, scrolarea unei zone din fereastră ar genera un mesaj de WM\_PAINT (datorită apariției unui dreptunghi invalid în suprafața client a ferestrei). Pentru a preveni apariția mesajului de WM\_PAINT, ShowKey apelează funcția ValidateRect pentru a valida întreaga suprafață client.

KeyLook nu memorează mesajele de tastatură și informația asociată acestora. Din acest motiv, este imposibilă redesenarea suprafeței client în momentul primirii unui mesaj WM\_PAINT. De aceea, KeyLook afișează doar capul de tabel, fără a reface informația din interiorul zonei client. Device context-ul obținut prin apelul funcției BeginPaint are suprafața de clipping egală cu dreptunghiul invalid. Pentru a desena în întreaga suprafață client la primirea mesajului WM\_PAINT, este necesar ca dreptunghiul invalid să fie egal cu suprafața client. Programul cheamă funcția Windows InvalidateRect pentru a invalida întregul dreptunghi client al ferestrei, după care este chemată funcția BeginPaint care întoarce un device context a cărui suprafață de clipping este acum egală cu întreaga suprafață client a ferestrei.

Faptul că programul KeyLook nu salvează informația de desenare și este deci incapabil să reafișeze aceasta informație la primirea mesajului WM\_PAINT este desigur o lipsă a programului.

Programul KeyLook desenează un cap de tabel în partea de sus a suprafeței client pentru a indica cele 9 coloane de informație afișate. Deși este posibilă crearea unui font subliniat, metoda utilizată este aceea de a afișa textul asociat capului de tabel și, suprapus peste acesta o linie de sublinieri. În mod normal, Windows afișează textul într-un mod "opac", ceea ce înseamnă că Windows șterge suprafața de fundal a caracterelor pentru fiecare caracter din text. Dacă am afișa în acest mod șirul de caractere de subliniere, textul asociat capului de tabel ar fi șters prin afișarea sublinierilor. Pentru a se evita acest lucru s-a trecut în modul de afișare "transparent" prin apelul funcției:

**SetBkMode (hdc, TRANSPARENT);**

Trecerea înapoi la modul "opac" se poate face cu:

**SetBkMode (hdc, OPAQUE);**

## 5.6 Caret-ul (cursorul text)

Când este necesară introducerea de texte într-un program, un mic dreptunghi este afișat pe poziția unde va fi introdus următorul caracter. Acest dreptunghi este cunoscut sub denumirea de "cursor". În Windows el se numește "caret", pentru a se deosebi de cursorul care indică poziția curentă a mouse-ului.

### 5.6.1 Funcțiile de caret

Cele mai utilizate funcții pentru manevrarea caret-ului sunt:

Funcția	Acțiune
CreateCaret	creează un caret asociat unei ferestre.
SetCaretPos	setează poziția caret-ului în cadrul ferestrei.
ShowCaret	afișează caret-ul.
HideCaret	ascunde caret-ul.
DestroyCaret	distruge caret-ul.

Windows pune la dispoziție și alte funcții asociate caret-ului (GetCaretPos - citește poziția caret-ului; GetCaretBlinkTime și SetCaretBlinkTime - pentru citirea și setarea intervalului de "blink" a caret-ului).

Caret-ul poate fi o linie orizontală, verticală sau un dreptunghi de dimensiunea unui caracter, în funcție de aplicația care îl deține.

Caret-ul este o resursă sistem, asta înseamnă că există un singur caret la un moment de timp în întregul sistem. Apariția unui caret într-o fereastră nu are sens decât atunci când fereastra respectivă este selectată (are intrare de la tastatură). Apariția caret-ului indică utilizatorului faptul că poate să introducă texte în fereastra respectivă. Deoarece o singură fereastră poate avea intrarea de la tastatură în întregul sistem, un singur caret este necesar întregului sistem.

O aplicație Windows poate determina dacă este selectată sau nu cu ajutorul mesajelor WM\_SETFOCUS și WM\_KILLFOCUS. Procedura de fereastră care primește WM\_SETFOCUS devine selectată (deci va primi mesaje de la tastatură) până în momentul în care se primește mesajul WM\_KILLFOCUS. Un program Windows va primi în timpul execuției



un număr de mesaje WM\_SETFOCUS egal cu numărul de mesaje WM\_KILLFOCUS. Mesajul WM\_KILLFOCUS este precedat întotdeauna de mesajul WM\_SETFOCUS.

Regulile de utilizare a caret-ului sunt următoarele:

- Procedura de fereastră cheamă funcția CreateCaret în timpul procesării mesajului WM\_SETFOCUS.
- Procedura de fereastră cheamă funcția DestroyCaret în timpul procesării mesajului WM\_KILLFOCUS.
- Caret-ul este creat ascuns.
- După apelul funcției CreateCaret, programul trebuie să cheme funcția ShowCaret pentru afișarea caret-ului.
- Procedura de fereastră trebuie să cheme funcția HideCaret pentru a ascunde caret-ul înainte de a desena ceva în suprafața client în afara de cazul în care procesează mesajul WM\_PAINT.
- După terminarea instrucțiunilor de desenare, procedura de fereastră trebuie să cheme funcția ShowCaret pentru a reafișa caret-ul.

## 5.7 Temă

Să se realizeze un editor de texte cu numele Type. Editorul de texte va utiliza fontul de sistem cu dimensiune fixă. Poziția de introducere a caracterelor va fi marcată cu ajutorul unui caret. La apăsarea tastelor de poziționare a cursorului - săgeată sus, jos, stânga, dreapta, Home, End, PageUp, PageDown, caret-ul va fi deplasat corespunzător în cadrul zonei client a aplicației. Editarea se realizează doar în cadrul zonei client, nu există noțiunea de scroll. Deplasarea cursorului este limitată la marginile zonei client.

Editorul va permite utilizarea următoarelor taste de editare: Delete - pentru ștergerea caracterului din dreptul cursorului; Backspace pentru ștergerea caracterului de pe poziția anterioară cursorului; Enter pentru trecerea la începutul următoarei linii (fără a modifica textul de pe ecran).

Pentru realizarea editorului se vor defini următoarele funcții de procesare de mesaje:

- WM\_CREATE: WMCreate (hwnd, wParam, lParam);
- WM\_SIZE: WMSize (hwnd, wParam, lParam);
- WM\_SETFOCUS: WMSetFocus (hwnd, wParam, lParam);
- WM\_KILLFOCUS: WMKillFocus (hwnd, wParam, lParam);
- WM\_KEYDOWN: WMKeyDown (hwnd, wParam, lParam);
- WM\_CHAR: WMChar (hwnd, wParam, lParam);
- WM\_PAINT: WMPaint (hwnd, wParam, lParam);
- WM\_DESTROY: WMDestroy (hwnd, wParam, lParam);

Suprafața de editare va fi egală cu suprafața client a ferestrei.

Variabilele cxClient, cyClient vor conține dimensiunile (orizontală, respectiv verticală) a suprafeței client și vor fi calculate în cadrul mesajului WMSize:

```
int cxClient, cyClient;
long WMSize (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    cxClient = LOWORD (lParam);
    cyClient = HIWORD (lParam);
    ....
    return 0;
}
```

Variabilele cxChar, cyChar vor conține dimensiunile în pixeli ale unui caracter din fontul de sistem de dimensiune fixă. Calculul valorilor celor două variabile se va realiza în mesajul WMCreate:

```
short cxChar, cyChar;
long WMCreate (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    HDC hdc;
    TEXTMETRIC tm;
    hdc = GetDC (hwnd);
    SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT));
    GetTextMetrics (hdc, &tm);
    cxChar = tm.tmAveCharWidth;
```

```
cyChar = tm.tmHeight;  
ReleaseDC (hwnd, hdc);  
.....  
return 0;  
}
```

Textul introdus se va memora într-o matrice alocată dinamic de dimensiuni `cxBuffer` și `cyBuffer`, unde valorile variabilelor `cxBuffer` și `cyBuffer` conțin respectiv numărul maxim de caractere dintr-o linie și numărul maxim de linii din suprafața client. Aceste valori sunt calculate în funcție de dimensiunea suprafeței client și de dimensiunile caracterelor, deci în cadrul mesajului `WM_SIZE`:

```
char* pBuffer;  
int cxBuffer, cyBuffer;  
#define BUFFER(x, y) (*(pBuffer + y*cxBuffer + x))  
long WMSize (HWND hwnd, WPARAM wParam, LPARAM lParam)  
{  
    int x,y;  
    .....  
    cxBuffer = cxClient / cxChar;  
    if (cxBuffer == 0) cxBuffer = 1;  
    cyBuffer = cyClient / cyChar;  
    if (cyBuffer == 0) cyBuffer = 1;  
    if (pBuffer != NULL)  
        free (pBuffer);  
    pBuffer = NULL;  
    if ((LONG) cxBuffer * cyBuffer <= 65535L)  
        pBuffer = (char *) malloc (cxBuffer * cyBuffer);  
    if (pBuffer == NULL)  
        MessageBox (    hwnd,  
                        "Window too large. Cannot allocate enough  
                        memory",  
                        "Type",  
                        MB_ICONEXCLAMATION | MB_OK);  
    else  
        for (y = 0; y < cyBuffer; y++)  
            for (x = 0; x < cxBuffer; x++)  
                BUFFER (x,y) = ' '  
    .....  
    return 0;  
}
```

Poziția curentă a caret-ului este memorată în variabilele `xCaret` și `yCaret`. Aceste variabile conțin indicii (linie / coloană) din matricea de caractere unde se află cursorul. Acesta este caracterul care va fi modificat dacă se introduce un caracter în fereastra de editare:

```
int xCaret, yCaret;
```

Poziția `x`, `y` în pixeli a caret-ului în suprafața client se poate calcula după următoarele formule:

- `xCaret*cxChar` - pentru coordonata `x`;
- `yCaret*cyChar` - pentru coordonata `y`;

Poziția inițială a caret-ului este (0, 0). Aceasta poziție se setează în mesajul `WMCreate`:

```
long WMCreate (HWND hwnd, WPARAM wParam, LPARAM lParam)  
{  
    .....  
    xCaret = 0;  
    yCaret = 0;  
    return 0;  
}
```

Prelucrarea mesajului WM\_SETFOCUS se face astfel: se creează caret-ul pentru fereastra asociată aplicației; se poziționează caret-ul pe poziția curentă (xCaret\*cxChar, yCaret\*cyChar); se afișează caret-ul:

```
long WMSetFocus (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    CreateCaret (hwnd, NULL, cxChar, cyChar);
    SetCaretPos (xCaret*cxChar, yCaret*cyChar);
    ShowCaret (hwnd);
    return 0;
}
```

Prelucrarea mesajului WM\_KILLFOCUS se realizează astfel: se ascunde caret-ul și apoi se distruge caret-ul:

```
long WMKillFocus (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    HideCaret (hwnd);
    DestroyCaret ();
    return 0;
}
```

Mesajul WM\_KEYDOWN se realizează în funcția WMKeyDown. Tastele de interes sunt tastele de deplasare a cursorului și tasta Delete.

Deplasarea cursorului se realizează utilizând următoarea structură de procesare:

```
long WMKeyDown (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    .....
    switch (wParam) {
        case VK_HOME:
            xCaret = 0;
            break;
        case VK_END:
            xCaret = cxBuffer - 1;
            break;
        case VK_PRIOR:
            yCaret = 0;
            break;
        case VK_NEXT:
            yCaret = cyBuffer - 1;
            break;
        case VK_LEFT:
            if (xCaret > 0)
                xCaret--;
            break;
        case VK_RIGHT:
            if (xCaret < cxBuffer - 1)
                xCaret++;
            break;
        case VK_UP:
            if (yCaret > 0)
                yCaret--;
            break;
        case VK_DOWN:
            if (yCaret < cyBuffer - 1)
                yCaret++;
            break;
        .....
    }
    SetCaretPos (xCaret*cxChar, yCaret*cyChar);
    return 0;
}
```

Introducerea caracterelor se realizează în mesajul WM\_CHAR. Caracterul introdus se memorează în matricea de caractere (utilizând macro-ul BUFFER) pentru a putea fi afișat la un mesaj WM\_PAINT, după care se afișează pe poziția xCaret\*cxChar, yCaret\*cyChar caracterul introdus de la tastatură. Anterior afișării caracterului se obține un handle la device context-ul ferestrei cu ajutorul funcției GetDC și se selectează fontul sistem de dimensiune fixă. După afișarea caracterului, se eliberează handle-ul de device context obținut și se actualizează poziția caret-ului.

**long WMChar (HWND hwnd, WPARAM wParam, LPARAM lParam)**

```
{
    HDC hdc;
    int i, x, y;
    for (i = 0; i < LOWORD (lParam); i++)
        switch (wParam) {
            ....
            default:
                BUFFER (xCaret, yCaret) = (char) wParam;
                HideCaret (hwnd);
                hdc = GetDC (hwnd);
                SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT));
                TextOut (hdc, xCaret*cxChar, yCaret*cyChar, &BUFFER (xCaret,
                                                                    yCaret), 1);
                ReleaseDC (hwnd, hdc);
                ShowCaret (hwnd);
                if (++xCaret == cxBuffer) {          /* Actualizare poziție caret. */
                    xCaret = 0;
                    if (++yCaret == cyBuffer)
                        yCaret = 0;
                }
                break;
        }
    SetCaretPos (xCaret*cxChar, yCaret*cyChar);
    return 0;
}
```

În cadrul mesajului WM\_CHAR vor fi procesate și codurile speciale pentru trecerea la linie nouă, Backspace etc.

Mesajul WM\_PAINT este utilizat pentru afișarea matricei de caractere în suprafața client a aplicației. Matricea de caractere este alocată dinamic în cadrul mesajului WM\_SIZE și inițializată cu spații. Introducerea caracterelor în această matrice se realizează în cadrul mesajului WM\_CHAR. Afișarea caracterelor din matricea de caractere se realizează astfel:

**long WMPaint (HWND hwnd, WPARAM wParam, LPARAM lParam)**

```
{
    HDC      hdc;
    PAINTSTRUCT ps;
    int y;
    hdc = BeginPaint (hwnd, &ps);
    SelectObject (hdc, GetStockObject (SYSTEM_FIXED_FONT));
    for (y = 0; y < cyBuffer; y++)
        TextOut (hdc, 0, y*cyChar, & BUFFER (0, y), cxBuffer);
    EndPaint (hwnd, &ps);
    return 0;
}
```

## Laboratorul 6 – Mouse-ul

### 6.1 Elementele fundamentale de tratare a mouse-ului

Prezența mouse-ului poate fi detectată prin apelul funcției `GetSystemMetrics`:

**`fMouse = GetSystemMetrics (SM_MOUSEPRESENT);`**

valoarea variabilei `fMouse` este `TRUE` (diferită de zero) dacă mouse-ul este instalat, respectiv `FALSE` (egală cu zero), dacă mouse-ul nu este instalat. Nu există o metodă standard pentru determinarea numărului de butoane ale mouse-ului.

#### 6.1.1 Definiții

Poziția mouse-ului este reprezentată pe ecran cu ajutorul unei imagini denumită cursor de mouse. Cursorul are un pixel unic (în interiorul imaginii) denumit "hot spot" care selectează un punct la o anumită coordonată pe ecran. Driver-ul de display (VGA.DRV) conține mai multe cursoare predefinite care pot fi utilizate de către un program Windows. Cursorul de mouse cel mai frecvent utilizat este identificat de constanta `IDC_ARROW` din `Winuser.h`. Hot spot-ul acestui cursor este în vârful săgeții. Cursorul `IDC_CROSS` (format din două linii încrucișate) are hot spot-ul la intersecția celor două linii. Cursorul `IDC_WAIT` este o clepsidră și este utilizat de către programe pentru a indica faptul că realizează procesări care necesită intervale mari de timp.

Cursorul implicit asociat unei clase de fereastră este specificat la inițializarea structurii de clasă de fereastră, de exemplu:

**`wndclass.hCursor = LoadCursor (NULL, IDC_ARROW);`**

asociază un cursor `IDC_ARROW` clasei de ferestre specificată prin structura `wndclass`.

Utilizatorul poate realiza următoarele acțiuni cu ajutorul butoanelor de mouse:

- Click - apăsarea și eliberarea unui buton de mouse.
- Double-click - apăsarea și eliberarea de două ori a unui buton de mouse într-o succesiune rapidă.
- Drag - deplasarea mouse-ului în timp ce un buton este ținut apăsat.

Pentru un mouse cu trei butoane, acestea sunt denumite butoanele `left`, `right` și `middle`. Un mouse cu două butoane nu deține decât butoanele `"left"` și `"right"`, iar un mouse cu un singur buton nu are decât butonul `"left"`.

### 6.2 Mesajele mouse în zona client

În comparație cu mesajele de la tastatură pe care Windows le trimite doar către fereastra focusată, mesajele de mouse sunt trimise către fereastra în dreptul căreia se află mouse-ul, indiferent dacă aceasta este focusată sau nu.

Windows declară 21 de mesaje de mouse dintre care 10 mesaje sunt trimise când mouse-ul se află în suprafața client. Restul de 11 mesaje se referă la suprafața non-client și, în general, un program poate să le ignore. Din cele 10 mesaje "client", 6 mesaje se referă la butoanele dreapta și mijloc ale mouse-ului și, în general, sunt ignorate.

Când mouse-ul este deplasat deasupra zonei client a unei ferestre, procedura de fereastră primește mesajul `WM_MOUSEMOVE`. Când un buton de mouse este apăsat sau eliberat în interiorul suprafeței client, procedura de fereastră primește unul din următoarele mesaje:

Buton	Apăsat	Eliberat	Double-click
stânga	<code>WM_LBUTTONDOWN</code>	<code>WM_LBUTTONUP</code>	<code>WM_LBUTTONDBLCLK</code>
mijloc	<code>WM_MBUTTONDOWN</code>	<code>WM_MBUTTONUP</code>	<code>WM_MBUTTONDBLCLK</code>
dreapta	<code>WM_RBUTTONDOWN</code>	<code>WM_RBUTTONUP</code>	<code>WM_RBUTTONDBLCLK</code>

Procedura de fereastră va primi mesaje "WM\_MBUTTONDOWNxxx" doar pentru un mouse cu trei butoane, iar mesajele "WM\_RBUTTONDOWNxxx" sunt generate doar pentru un mouse cu două sau trei butoane. Procedura de fereastră primește mesaje "WM\_xBUTTONDOWNBLCLK" (double-click) doar dacă în stilul clasei de fereastră a fost introdus identificatorul CS\_DBLCLKS. De exemplu:

**wndclass.style = CS\_HREDRAW | CS\_VREDRAW | CS\_DBLCLKS;**

Pentru toate aceste mesaje valoarea argumentului lParam conține coordonatele mouse-ului relativ la suprafața client a ferestrei. Cuvântul mai puțin semnificativ conține coordonata x a cursorului, iar cuvântul mai semnificativ conține coordonata y a cursorului. Coordonatele sunt exprimate în pixeli relativ la sistemul de coordonate al suprafeței client (adică originea în colțul stânga sus al suprafeței client, coordonata x reprezintă coordonata orizontală, iar y pe cea verticală). Cele două coordonate pot fi obținute cu ajutorul macro-urilor LOWORD și HIWORD definite în Windef.h astfel:

**xMouse = LOWORD (lParam);**

**yMouse = HIWORD (lParam);**

Valoarea argumentului wParam indică starea butoanelor de mouse și a tastelor Shift și Ctrl. Starea acestor taste se poate obține cu ajutorul operatorului "și" pe biți dintre argumentul wParam și flagurile cu prefixul MK\_ definite în Winuser.h:

- wParam & MK\_LBUTTONDOWN != 0 dacă butonul stânga al mouse-ului este apăsat.
- wParam & MK\_RBUTTONDOWN != 0 dacă butonul dreapta al mouse-ului este apăsat.
- wParam & MK\_MBUTTONDOWN != 0 dacă butonul mijloc al mouse-ului este apăsat.
- wParam & MK\_SHIFT != 0 dacă tasta Shift este apăsată.
- wParam & MK\_CONTROL != 0 dacă tasta Ctrl este apăsată.

Dacă se apasă butonul stânga al mouse-ului în interiorul unei ferestre inactive, Windows activează fereastra respectivă după care trimite procedurii de fereastră mesajul WM\_LBUTTONDOWN. Când o procedură de fereastră primește mesajul WM\_LBUTTONDOWN fereastra respectivă este deja activată. Mesajul WM\_LBUTTONUP poate să sosească la o procedură de fereastră fără ca anterior să fi venit mesajul WM\_LBUTTONDOWN. Acest lucru se întâmplă în cazul în care utilizatorul apasă butonul mouse-ului în dreptul unei ferestre, deplasat în interiorul ferestrei respective, după care butonul este eliberat. În mod similar, o procedură de fereastră poate primi un mesaj WM\_LBUTTONDOWN fără a primi și mesajul WM\_LBUTTONUP asociat acestuia, dacă eliberarea butonului se produce în dreptul altei ferestre

O procedură de fereastră poate să "captureze" mouse-ul pentru a evita acest fenomen. Mesajele de mouse vor fi trimise către procedura de fereastră care a capturat mouse-ul indiferent dacă acesta nu se mai afla în interiorul ferestrei respective.

### 6.3 Aplicație pentru procesarea mesajelor de mouse

Program Connect este un exemplu simplu de procesare a mesajelor WM\_LBUTTONDOWN, WM\_LBUTTONUP și WM\_MOUSEMOVE provenite de la mouse:

Acest program procesează următoarele mesaje:

- **WM\_LBUTTONDOWN**: șterge suprafața client a ferestrei.
- **WM\_MOUSEMOVE**: dacă butonul stânga este apăsat, programul afișează un punct negru în suprafața client pe poziția mouse-ului.
- **WM\_LBUTTONUP**: programul unește fiecare punct desenat în suprafața ferestrei cu fiecare alt punct.

Connect utilizează câteva apeluri simple de funcții GDI. Funcția SetPixel desenează un pixel în suprafața client. Desenarea unei linii se realizează prin apelul a două funcții: MoveToEx și LineTo. Funcția MoveToEx marchează coordonatele x și y ale unei extremități a liniei, iar funcția LineTo desenează linia.

Dacă mouse-ul este deplasat în exteriorul suprafeței client, înainte de eliberarea butonului de mouse, Connect nu unește punctele din suprafața client deoarece procedura de fereastră nu primește mesajul WM\_LBUTTONUP. Intervalul de timp necesar pentru desenarea liniilor poate fi mare (în funcție de numărul de puncte din suprafața client).

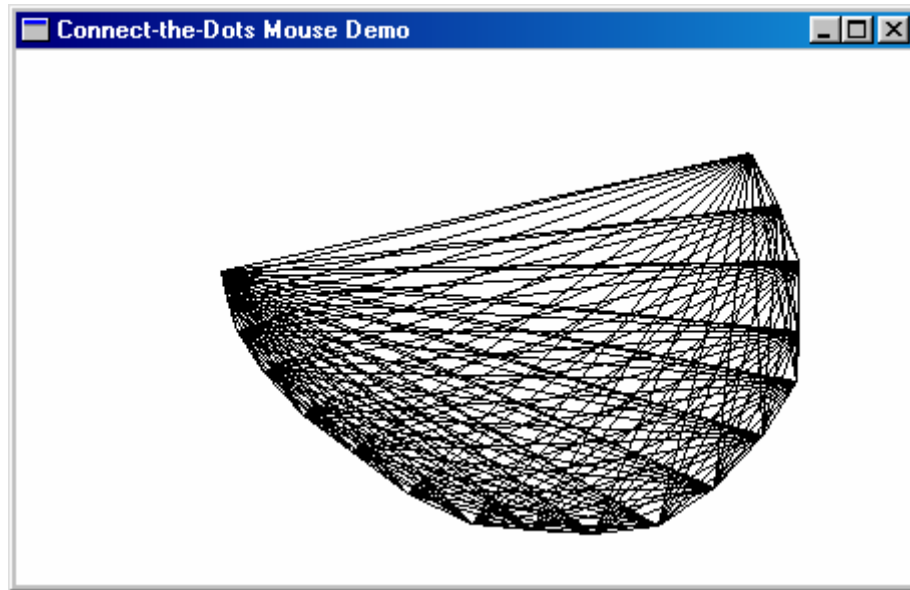


Figura 6.1. Aplicația Connect

În timp ce programul este ocupat cu desenarea de linii, utilizatorul poate să apese un buton de mouse în suprafața client, apoi să deplaseze mouse-ul și să elibereze butonul de mouse. Datorită faptului că programul nu apelează funcția `GetMessage` în timpul desenării, nu se produce nici un efect ca urmare a acestor acțiuni. Apariția unui mesaj `WM_LBUTTONDOWN` în timp ce mesajul `WM_LBUTTONDOWN` asociat nu a fost încă procesat (el se afla încă în coada de mesaje a aplicației) provoacă eliminarea din coada de mesaje a aplicației a mesajului `WM_LBUTTONDOWN` precum și a mesajelor `WM_MOUSEMOVE` care l-au succedat, iar mesajul `WM_LBUTTONDOWN` nu mai este introdus în coadă. Din acest punct de vedere, mesajele de mouse nu prezintă aceeași importanță ca mesajele de la tastatură. Dacă un buton de mouse a fost apăsat și eliberat în suprafața client a unei ferestre, iar programul nu cheamă funcția `GetMessage` în acest răstimp, evenimentele de mouse sunt ignorate.

#### 6.4 Evenimentele double-click

Un double-click de mouse este realizat prin două apăsări în succesiune rapidă. Pentru a deveni double-click, cele două apăsări de buton trebuie să apară într-un interval de timp specificat. Pentru ca procedura de fereastră să primească mesaje de tip double-click, trebuie inclus în stilul de clasă de fereastră identificatorul `CS_DBLCLKS`, anterior înregistrării clasei de fereastră:

```
wndclass.style = CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS;
```

```
.....
```

```
RegisterClass (&wndclass);
```

Dacă în stilul de clasă nu se include identificatorul `CS_DBLCLKS`, apăsarea de două ori rapidă se traduce într-o succesiune de mesaje:

- **WM\_LBUTTONDOWN,**
- **WM\_LBUTTONUP,**
- **WM\_LBUTTONDOWN**
- **WM\_LBUTTONUP.**

Procedura de fereastră trebuie să implementeze logica de double-click utilizând funcția `GetMessageTime` pentru calculul intervalului între cele două click-uri.

Inserarea stilului `CS_DBLCLKS` în stilul de clasă de fereastră determină ca succesiunea de mesaje provocată de două apăsări rapide asupra unui buton de mouse să fie:

- **WM\_LBUTTONDOWN,**
- **WM\_LBUTTONUP,**
- **WM\_LBUTTONDBLCLK**
- **WM\_LBUTTONUP.**

Mesajul WM\_LBUTTONDOWNBLCLK ia locul celui de-al doilea mesaj WM\_LBUTTONDOWN.

## 6.5 Mesaje mouse non-client

Primele 10 mesaje de mouse discutate pot apărea în momentul în care mouse-ul este mișcat sau apăsat în suprafața client a ferestrei. Dacă mouse-ul este în interiorul suprafeței non-client a ferestrei, Windows trimite procedurii de fereastră un mesaj "non-client". Suprafața non-client conține Caption bar-ul, Meniu bar-ul și Scroll bar-urile (dacă există).

Mesajele non-client în general nu sunt procesate de către procedura de fereastră și sunt trimise către DefWindowProc pentru realizarea funcțiilor de sistem. Din acest punct de vedere, mesajele non-client sunt similare cu mesajele de tasta sistem: WM\_SYSKEYDOWN, WM\_SYSKEYUP și WM\_SYSCHAR.

Mesajele de mouse non-client sunt similare mesajelor client. Identificatorii asociați acestor mesaje includ prefixul NC, provenit din prescurtarea lui "non-client". Când mouse-ul este deplasat în interiorul suprafeței non-client, procedura de fereastră primește mesajele WM\_NCMOUSEMOVE. Butoanele de mouse generează următoarele mesaje:

Buton	Apăsat	Eliberat	Double-click
stânga	WM_NCLBUTTONDOWN	WM_NCLBUTTONUP	WM_NCLBUTTONDOWNBLCLK
mijloc	WM_NCMBBUTTONDOWN	WM_NCMBUTTONUP	WM_NCMBUTTONDBLCLK
dreapta	WM_NCRBUTTONDOWN	WM_NCRBUTTONUP	WM_NCRBUTTONDOWNBLCLK

Valorile argumentelor wParam și lParam sunt diferite de cele ale mesajelor de mouse din zona client. Argumentul wParam indică zona suprafeței non-client în interiorul căreia mouse-ul a fost deplasat sau butonul a fost apăsat. Valoarea acestui argument este unul din identificatorii prefixați cu HT\_ și definiți în Winuser.h (de exemplu HTCAPTION, HTSYSTEMMENU etc.).

Valoarea argumentului lParam conține coordonatele x (în cuvântul mai puțin semnificativ) și y (în cuvântul mai semnificativ) ale cursorului. Coordonatele sunt exprimate în pixeli relativ la sistemul de coordonate al întregului ecran. Sistemul de coordonate al ecranului se definește astfel: colțul stânga sus al ecranului reprezintă originea sistemului de coordonate. Valorile coordonatei x cresc pe orizontală de la stânga spre dreapta, iar coordonata y crește pe verticală de sus în jos. (Figura. 6.2.)

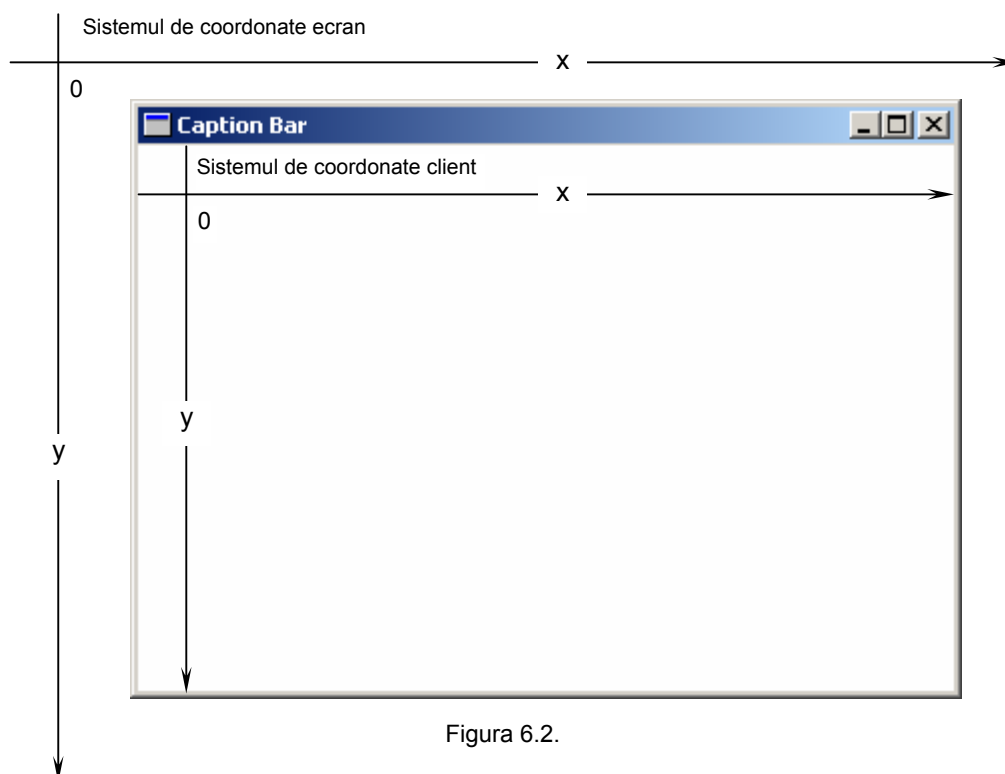


Figura 6.2.



Trecerea de la sistemul de coordonate ecran la sistemul de coordonate client se poate realiza cu ajutorul funcțiilor:

**ScreenToClient (hwnd, lpPoint);**  
**ClientToScreen (hwnd, lpPoint);**

Argumentul lpPoint este un pointer far către o structură de tip POINT. Cele două funcții convertesc valorile coordonatelor din structura lpPoint.

## 6.6 Mesajul Hit-Test

Ultimul mesaj generat de mouse este WM\_NCHITTEST, adică "non-client hit test". Mesajul Hit-Test este utilizat de către Windows pentru a determina zona din fereastră în interiorul căreia se găsește cursorul de mouse. Acest mesaj precedă orice alt mesaj client sau non-client de mouse. Argumentul lParam conține poziția x și y a cursorului de mouse în sistemul de coordonate ecran. Argumentul wParam nu este utilizat.

Aplicațiile Windows trimit acest mesaj către DefWindowProc. DefWindowProc utilizează mesajul WM\_NCHITTEST pentru a genera toate mesajele de mouse în funcție de poziția acestuia. Valoarea întoarsă de către DefWindowProc identifică regiunea în care se află mouse-ul. Aceasta valoare devine valoarea câmpului wParam pentru mesajele non-client. Pe lângă valorile întâlnite în mesajul în câmpul wParam al mesajelor non-client, valoarea întoarsă de către DefWindowProc poate să fie:

- **HTCLIENT** - suprafața client.
- **HTNOWHERE** - în exteriorul oricărei ferestre.
- **HTTRANSPARENT** - în interiorul unei zone acoperite de o altă fereastră.
- **HTERROR** - provoacă emiterea unui sunet de către DefWindowProc.

Dacă valoarea întoarsă de către DefWindowProc este HTCLIENT, Windows convertește coordonatele ecran ale cursorului de mouse în coordonate client și apoi generează un mesaj de mouse "client".

Funcțiile sistem accesibile prin intermediul mouse-ului pot fi dezactivate în totalitate dacă se introduce următoarea secvență în procedura de fereastră:

```
case WM_NCHITTEST:
    return (long) HTNOWHERE;
```

## 6.7 Implementarea Hit Test-ului în zona client

Foarte frecvent o procedură de fereastră trebuie să determine zona din suprafața client în care a apărut un eveniment de mouse. Aceasta procedură se numește "hit-test". Întocmai ca DefWindowProc, procedura de fereastră trebuie să determine regiunea "lovită", pe baza coordonatelor client de mouse din argumentul lParam.

### 6.7.1 Exemplu de aplicație ce realizează hit-test în zona client

Programul Checker1 prezintă o procedură simplă de hit-test. Suprafața client a programului este împărțită într-un tablou de dimensiune 5 x 5 celule de dimensiune egală. Prin apăsarea butonului stânga de mouse în dreptul unei celule, în dreptunghiul asociat celei respective este desenat un "X". Prin apăsarea a doua oară a butonului mouse-ului, "X"-ul din celula respectiva este șters.

Matricea fState definită în cadrul programului memorează starea celulelor afișate în zona client a programului. Inițial valoarea elementelor acestei matrice este zero, adică toate dreptunghiurile sunt libere. În momentul în care valoarea unui element din acest tablou devine diferită de zero (adică TRUE), în dreptunghiul asociat elementului respectiv se desenează un "X".

Funcția WMLButtonDown testează, pe baza coordonatelor de mouse, dreptunghiul în interiorul căruia a fost apăsă butonul stânga al mouse-ului. În cazul obținerii unor coordonate (linie / coloana) valide, starea celulei respective, memorată în matricea fState, este modificată. Determinarea numărului celulei selectate se face prin:

```
x = LOWORD (lParam) / cxBlock;
y = HIWORD (lParam) / cyBlock;
```

Unde x reprezintă coloana, iar y reprezintă linia dreptunghiului selectat. Pentru a reflecta modificarea de stare a zonei selectate, funcția WMLButtonDown apelează funcția InvalidateRect căreia îi transmite coordonatele în pixeli ale zonei selectate:

```

rect.left = x*cxBlock;
rect.top = y*cyBlock;
rect.right = (x + 1)*cxBlock;
rect.bottom = (y + 1)*cyBlock;
InvalidateRect (hwnd, &rect, FALSE);

```

Variabila `rect` conține coordonatele în pixeli în sistemul de coordonate client pentru celula a cărei stare a fost modificată. Ca urmare a invalidării unui dreptunghi în suprafața client a ferestrei, procedura de fereastră primește un mesaj `WM_PAINT` și redesenează zona modificată.

Funcția `WMPaint` răspunde mesajului `WM_PAINT` prin afișarea, în concordanță cu starea curentă, a dreptunghiurilor din zona client.

Aplicția `Checker2` adaugă programului `Checker1` interfața cu tastatura.

## 6.8 Utilizarea ferestrelor copil pentru hit-testing

În cazul unor situații mai complexe, procedura de determinare a zonei selectată cu ajutorul mouse-ului poate fi simplificată utilizând ferestre "copil". Ferestrele copil împart suprafața client a aplicației în zone dreptunghiulare mai mici. Procedura de fereastră pentru ferestrele copil primește mesaj de eveniment de mouse doar atunci când cursorul mouse-ului se afla în interiorul acestor ferestre. Argumentul `LPARAM` conține coordonatele cursorului relative la colțul stânga sus a ferestrei copil. Pe baza acestor coordonate procedura de hit-testing poate fi mult simplificată.

### 6.8.1 Aplicație ce utilizează ferestre copil pentru hit-testing

Noua versiune a programului `Checker` utilizează 25 de ferestre copil pentru a procesa mesajele de la mouse. `Checker3` conține două proceduri de fereastră denumite `WndProc`, respectiv `ChildWndProc`. `WndProc` este procedura de fereastră asociată ferestrei principale a aplicației (fereastră părinte). `ChildWndProc` este procedura de fereastră pentru toți cei 25 de copii.

Datorită faptului că adresa procedurii de fereastră este conținută în structura de clasă de fereastră, cele două proceduri de fereastră trebuie asociate cu două clase de fereastră. Prima clasă se numește "`Checker3`" și este clasa ferestrei părinte, iar cea de-a doua clasă se numește "`Checker3_Child`" și este asociată tuturor celor 25 de ferestre copil.

Diferențele între valorile cu care sunt inițializate câmpurile celor două structuri de clasă de fereastră (între clasa părinte și clasa copil) sunt următoarele:

- câmpul `lpszClassName` este inițializat cu "`Checker3_Child`" pentru clasa de ferestre copil;
- câmpul `lpfnWndProc` primește adresa funcției `ChildWndProc`, care este funcția de procesare a mesajelor clasei copil;
- câmpul `hIcon` primește valoarea `NULL`, deoarece aceste ferestre copil nu sunt minimizabile;
- câmpul `cbWndExtra` este inițializat cu valoarea 2. Ca urmare, Windows alocă 2 octeți suplimentari în fiecare structură de fereastră copil. În spațiul alocat suplimentar, programul memorează starea ferestrei respective (cu "X" sau fără "X").

Apelul funcției `CreateWindow` din `InitInstance` creează fereastra principală a aplicației, pe baza clasei "`Checker3`". Ca urmare a apelului funcției `CreateWindow`, `WndProc` primește mesajul `WM_CREATE`. În funcția `WMCreat`, care procesează mesajul `WM_CREATE`, programul apelează de 25 de ori funcția `CreateWindow` pentru a crea cele 25 ferestre copil ale ferestrei principale a aplicației. Ferestrele copil sunt create pe baza "`Checker3_Child`". Următorul tabel compară argumentele de apel ale funcției `CreateWindow` pentru fereastra principală și ale funcției `CreateWindow` pentru cei 25 de copii ai ferestrei principale.

Argumentul	Fereastra principală	Fereastra copil
clasa de fereastră	"Checker3"	"Checker3_Child"
titlul ferestrei	"Checker3..."	NULL
stilul ferestrei	WS_OVERLAPPED WINDOW	WS_CHILDWINDOW   WS_VISIBLE
poziția orizontală	CW_USEDEFAULT	0
poziția verticală	CW_USEDEFAULT	0

Lățimea	CW_USEDEFAULT	0
Înălțimea	CW_USEDEFAULT	0
handle-ul părintelui	NULL	hwnd
handle-ul de meniu / ID-ul copilului	NULL	y*DIVISIONS + x
handle-ul de instanță	hInstance	GetWindowLong (hwnd, GWL_HINSTANCE);
parametrii suplimentari	NULL	NULL

În mod normal, coordonatele ferestrei, lățimea și înălțimea sunt nenule în cazul ferestrelor copil, dar Checker3 re poziționează copii mai târziu, în cadrul mesajului WM\_SIZE, din acest motiv, valorile acestor argumente sunt irelevante la momentul apelului funcției CreateWindow.

Handle-ul ferestrei părinte este NULL în cazul ferestrei principale (aceasta este fereastra părinte a aplicației). În cazul ferestrelor copil, acest handle are valoarea handle-ului ferestrei părinte (fereastra principală), pentru a realiza relația de subordonare dintre părinte și ferestrele copil. Acest argument trebuie să fie diferit de NULL la apelul funcției CreateWindow pentru o fereastra copil.

Fereastra principală nu deține un meniu, de aceea acest argument este NULL. Pentru ferestrele de tip copil acest argument are semnificația de "identificator de copil". ID-ul de copil este un număr care identifică în mod unic fereastra copil dintre toate ferestrele copil ale aceluiași părinte. ID-ul de copil este foarte important în cazul ferestrelor control copii, deoarece sursa mesajelor pe care acestea le trimit către fereastra părinte poate fi stabilită prin acest ID.

Handle-ul de instanță este hInstance în ambele cazuri (adică valoarea argumentului hInstance din WinMain). În momentul creării ferestrei copil, acest handle este extras din structura de fereastră (menținută de către Windows) a ferestrei principale cu ajutorul funcției GetWindowLong. Același efect l-ar fi avut dacă valoarea lui hInstance ar fi putut fi memorată într-o variabilă globală și apoi folosită ca argument al funcției CreateWindow pentru ferestrele copil.

Fiecare fereastră copil deține un handle de fereastră diferit. Acest handle este valoarea întoarsă de către funcția CreateWindow. Handle-ul de fereastră al copiilor este memorat în tabloul hwndChild de tipul HWND, dimensionat pentru a memora handle-urile tuturor ferestrelor copil. Când fereastra principală primește mesajul WM\_SIZE, este apelată funcția MoveWindow pentru toți cei 25 de copii. Argumentele funcției MoveWindow sunt handle-ul de fereastră al copilului, coordonatele colțului stânga sus ale ferestrei, lățimea și înălțimea ferestrei și dacă este necesară redesenarea ferestrei re poziționate.

Funcția ChildWndProc realizează procesarea mesajelor pentru toate cele 25 de ferestre copil ale aplicației. Funcția este chemată de Windows de fiecare dată când apare un mesaj pentru unul din cei 25 de copii.

Argumentul hwnd al funcției ChildWndProc este handle-ul de fereastră pentru fereastra copil care primește mesajul.

ChildWndProc procesează mesajul WM\_CREATE (care va sosi de 25 de ori, pentru cele 25 de ferestre copil) în funcția ChildWmCreate. Funcția ChildWmCreate utilizează funcția SetWindowLong pentru a inițializa cu zero zona suplimentară de doi octeți alocată în structura de fereastră (Acest spațiu a fost alocat punând valoarea 4 adică sizeof (LONG) în câmpul cbWndExtra al structurii de clasă de fereastra copil). ChildWndProc utilizează acest spațiu de memorie pentru a memora starea ferestrei (cu "X" sau fără "X").

Când butonul stânga al mouse-ului este apăsat în interiorul uneia din ferestrele copil, procedura de fereastră copil este chemată cu mesajul WM\_LBUTTONDOWN. Mesajul WM\_LBUTTONDOWN provoacă inversarea stării ferestrei copil care a fost click-ată și invalidarea suprafeței client a ferestrei copil. Ca urmare este generat un mesaj WM\_PAINT care reafixează fereastra copil în conformitate cu noua stare a acesteia.

## 6.9 Captura mouse-ului

O procedură de fereastră primește în mod normal mesaje de mouse când cursorul acestuia se află în interiorul ferestrei respective. Un program care dorește să primească

mesaje de mouse și atunci când mouse-ul se află în exteriorul ferestrei, trebuie să "captureze" mouse-ul.

Capturarea mouse-ului se realizează prin apelul funcției:

**SetCapture (hwnd);**

După întoarcerea funcției SetCapture, Windows va trimite toate mesajele de mouse ferestrei identificate prin handle-ul hwnd. Mesajele de mouse capturate vor fi întotdeauna mesaje client, chiar dacă mouse-ul se află în zona non-client a ferestrei. Argumentul lParam indică poziția cursorului relativ la sistemul de coordonate client. Coordonatele x și y ale mouse-ului vor avea însă valori pozitive sau negative (dacă mouse-ul se află în stânga sau deasupra colțului stânga sus al zonei client).

Pe timpul cât mouse-ul este capturat, funcțiile sistem sunt dezactivate. Mouse-ul și tastatura nu produc evenimente de sistem. Eliberarea mouse-ului se realizează prin apelul funcției:

**ReleaseCapture (hwnd);**

### 6.9.1 Aplicație ce utilizează captura mouse-ului

Programul CapScr utilizează funcțiile SetCapture și ReleaseCapture. Programul permite selectarea oricărei zone dreptunghiulare din ecran. CapScr copiază apoi informația grafică din zona selectată în suprafața client, lărgind sau micșorând imaginea astfel încât să se potrivească dimensiunii suprafeței client.



Figura. 6.3. Aplicația CapScr

Ajustarea dimensiunilor imaginii este realizată de către funcția GDI StretchBlt (prescurtarea Blt provine de la "block transfer").

CapScr nu memorează imaginea capturată și deci nu poate procesa mesajul WM\_PAINT. Modificarea dimensiunilor ferestrei va șterge conținutul suprafeței client.

### 6.9.2 Modificarea formei cursorului de mouse

CapScr utilizează funcția SetCursor pentru a modifica forma cursorului dintr-o săgeată într-un "cross hair" și apoi într-o clepsidră. Toate aceste forme de cursor sunt forme predefinite și care sunt puse la dispoziție de Windows. Handle-ul unui cursor predefinit poate fi obținut utilizând funcția LoadCursor cu primul argument NULL.

Funcția SetCursor permite atribuirea unei noi imagini cursorului de mouse. Valoarea întoarsă de către aceasta funcție este handle-ul anterior al cursorului.

Atribuirea unei forme de clepsidră cursorului de mouse pentru a semnaliza o procesare îndelungată se face astfel:

Se declară o variabilă necesară pentru memorarea handle-ului anterior de cursor:

**HCURSOR hcursor;**

Înainte de secvența de program care necesită un timp îndelungat de procesare se introduc următoarele linii:

**hCursor = SetCursor (LoadCursor (NULL, IDC\_WAIT));**

**ShowCursor (TRUE);**

După încheierea procesării se apelează:

**ShowCursor (FALSE);**

**SetCursor (hCursor);**

Apelul funcției ShowCursor are ca efect afișarea și ștergerea cursorului dacă nu este instalat un mouse în sistem. În cazul în care este instalat un mouse, efectul celor două funcții este de incrementare, respectiv decrementare a unui contor de "vizibilitate" a mouse-ului, deci nu provoacă nici un efect asupra cursorului.

În mod normal, Windows modifică forma cursorului cu cea specificată în clasa de fereastră de fiecare dată când procedura de fereastră primește un mesaj WM\_MOUSEMOVE. Excepție face cazul în care mouse-ul este capturat (ca în programul CapScr). Dacă se dorește afișarea unui cursor clepsidră pe timpul unei procesări îndelungate, cursorul va rămâne nemodificat datorită faptului că în timpul procesării nu se cheamă funcția GetMessage, și deci procedura de fereastră nu va procesa nici un mesaj în respectivel interval de timp.

Dacă în structura de clasă de fereastră se înscrie câmpul hCursor cu valoarea NULL:

**wndclass.hCursor = NULL;**

Windows nu va modifica forma de cursor atribuită de funcția SetCursor la primirea mesajelor WM\_MOUSEMOVE.

### 6.9.3 Apelul funcției StretchBlt

CapScr utilizează funcția StretchBlt în timpul procesării mesajului WM\_LBUTTONDOWN, pentru a transfera imaginea din zona selectată în zona client:

**StretchBlt (     hdc, 0, 0, rect.right, rect.bottom,  
                  hdc, ptBeg.x, ptBeg.y, ptEnd.x - ptBeg.x,  
  ptEnd.y - ptBeg.y, SRCCOPY);**

Această funcție realizează transferul unei imagini "bitmap" de la o sursă către o destinație. Sursa, respectiv destinația, trebuie să fie device context-e. Primele cinci argumente reprezintă device context-ul destinație și coordonatele în acest device context unde va fi introdusă imaginea, precum și noile dimensiuni ale imaginii. Următoarele cinci argumente reprezintă sursa de imagine, adică device context-ul sursă și coordonatele, respectiv dimensiunile imaginii sursă. StretchBlt poate să ajusteze corespunzător imaginea astfel încât să se potrivească cu dimensiunile destinației (care pot fi diferite de dimensiunile imaginii sursă).

În cazul programului CapScr, coordonatele sursei sunt memorate în punctele ptBeg (începutul zonei sursă) și ptEnd (sfârșitul zonei sursă); ele pot fi chiar și în exteriorul suprafeței client. Funcția StretchBlt poate citi o imagine din exteriorul suprafeței de clipping a device context-ului, dar ca orice funcție GDI, nu poate scrie informația decât în interiorul suprafeței de clipping a device context-ului.

### 6.9.4 Desenarea zonei de captură a imaginii

Programul CapScr desenează suprafața de imagine capturată (în funcția InvertBlock) în exteriorul suprafeței client. Acest lucru este posibil prin obținerea unui device context utilizând funcția CreateDC:

**hdc = CreateDC ("DISPLAY", NULL, NULL, NULL);**

obține un device context pentru întregul ecran. Utilizând handle-ul de device context astfel obținut se poate desena în exteriorul suprafeței client a ferestrei.

Funcția InvertBlock utilizează funcția GDI PatBlt ("pattern bit-block transfer") pentru a inversa culorile în zona selectată.

## 6.10 Temă

Se cere realizarea unei interfețe cu mouse-ul a programului Type, din laboratorul anterior. Interfață cu mouse-ul va permite poziționarea cursorului în fereastra client a aplicației, în momentul apăsării butonului stânga a mouse-ului.

Apăsarea butonului de mouse și mișcarea mouse-ului cu butonul stânga apăsător permite selectarea textului între poziția inițială și poziția curentă a mouse-ului.

Forma cursorului de mouse va fi cursorul predefinit prin identificatorul IDC\_IBEAM.

## Laboratorul 7 - Timer-ul

### 7.1 *Timerul în programarea Windows*

Timer-ul Windows este un periferic de intrare care generează evenimente periodice în timp. Aplicația poate să specifice intervalul de timp dorit la care să se producă evenimentele. Windows va trimite un mesaj WM\_TIMER la scurgerea fiecărui interval de timp.

Timer-ul este util pentru următoarele scopuri:

- Evidența timpului. Timerul indică momentul de timp la care este necesară reîmprospătarea informației de timp afișată. Programul DigClock, descris în acest capitol, utilizează timer-ul pentru a afișa un ceas digital.
- Afișarea unor informații de stare. Programul Freemem, prezentat în acest capitol, utilizează timer-ul pentru a afișa memoria disponibilă în sistem. Afișarea este reîmprospătată la fiecare secundă.
- Alarmare. Programul Windows Calendar utilizează timer-ul pentru a genera un semnal de atenționare la un anumit moment de timp.
- Multitasking. Datorită faptului că Windows este un sistem multitasking nonpreemptiv, este foarte important ca toate aplicațiile să revină în Windows cât mai rapid. Deci intervalele de procesare de mesaje trebuie să fie cât mai scurte. Dacă un program trebuie să realizeze o secvență de procesare consumatoare de timp, este posibil să divizeze secvența în intervale mici și să proceseze fiecare interval într-un mesaj WM\_TIMER. Ca urmare, multitaskingul Windows va continua să funcționeze fără ca programul respectiv să acapareze totalitatea resurselor de procesare.
- Implementarea unui mecanism de "autosave". Timer-ul poate să indice programului Windows să salveze informația când un anumit interval de timp s-a scurs.
- Animație. Obiectele grafice într-un program de animație trebuie deplasate la un anumit interval de timp. Utilizarea timer-ului poate elimina diferențele de viteză de procesare între diferite calculatoare.
- Programe demonstrative. Anumite programe cu scop demonstrativ pot fi proiectate astfel încât să se încheie după scurgerea unui anumit interval de timp. Timer-ul poate să fie utilizat pentru închiderea acestor aplicații.
- Comunicațiile seriale sau paralele. Driverile Windows de comunicație serială și paralelă nu generează mesaje către procedura de fereastră care realizează comunicația. Pentru realizarea transferului prin interfață serială sau paralelă, un program trebuie să utilizeze mesajele de timp WM\_TIMER, pentru citirea intrării.

În acest laborator este prezentată tehnica de folosire a funcțiilor tip CALLBACK, utilizate pentru mesajele de timer, și pentru alte funcții de sistem Windows. Acest capitol prezintă și modalitățile de rezolvare a situațiilor când programul nu poate obține un timer. Problema poate apare datorită faptului că Windows gestionează un număr limitat de timere. Rezolvarea acestor situații nu este specifică timer-ului, ea poate fi folosită și în cazul altor resurse finite ale sistemului. Programele prezintă de asemenea modalități de utilizare a culorilor, utilizarea ferestrelor tip "pop-up", modalitatea de a executa o aplicație în task-bar-ul sistemului Windows, obținerea spațiului de memorie disponibilă, citirea informațiilor referitoare la formatele de afișare a datei și a timpului din fișierul Win.ini.

### 7.2 *Elementele fundamentale de tratare a timer-ului*

Alocarea unui timer de către un program se face prin apelul funcției SetTimer. Funcția SetTimer folosește un argument pentru specificarea intervalului între mesajele WM\_TIMER.

Acest interval poate lua valori între 1 milisecunda (teoretic) și 65535 msec, aproximativ 65,5 secunde. Un interval de 1000 va provoca trimiterea unui mesaj WM\_TIMER la fiecare 1 sec.

Programul trebuie să cheme funcția KillTimer pentru a elibera timer-ul utilizat atunci când nu-i mai este necesar. Funcția KillTimer elimina toate mesajele WM\_TIMER din coada de mesaje a aplicației, deci procedura de fereastră nu va mai primi nici un mesaj WM\_TIMER după apelul funcției KillTimer.

### 7.2.1 System.drv și Timer-ul Windows

Windows inițializează ceasul de timp real pentru a genera întreruperea hardware 08H. Ca urmare, întreruperea 08H va apare la un interval de 54.925 msec, sau de aproximativ 18.2 ori într-o secundă. Acest fapt are următoarele implicații:

- Un program Windows nu poate primi mesaje de timp cu o frecvență mai mare de 18.2 ori pe secundă.
- Intervalul de timp trimis funcției SetTimer este rotunjit la un multiplu întreg al perioadei de generare a întreruperilor de ceas. De exemplu, dintr-un interval de 1000 msec împărțit la perioada de 54.925 msec se obțin 18.207 intervale de timp. Windows rotunjește acest număr la 18 intervale de timp, rezultând o perioadă efectivă de primire a mesajelor de timp de 0.989 secunde. Pentru durate sub 55 msec, fiecare întrerupere de timp generează un mesaj WM\_TIMER.

Fișierul System.drv conține secvența de program care tratează întreruperea 08H. Când System.drv primește o întrerupere 08H, apelează o funcție în bibliotecă User.exe care decrementează contoarele asociate tuturor timerelor alocate. Când contorul atinge valoarea 0, User.exe introduce un mesaj WM\_TIMER în coada de mesaje a aplicației și reînscrie contorul de timer cu valoarea inițială.

Deoarece programele Windows extrag mesajele WM\_TIMER din coada de mesaje a aplicației, programele nu sunt "întrerupte" de mesaje WM\_TIMER. Din acest punct de vedere, timer-ul este similar tastaturii și mouse-ului: driver-ul tratează întreruperea asincronă generată de periferic, iar Windows transformă evenimentul asincron într-un mesaj pe care îl introduce în coada de mesaje a aplicației.

Ca și întreruperea hardware 09H de tastatură, întreruperea 08H de timer este o întrerupere asincronă, ea poate interveni la orice moment de timp. Apariția întreruperii 08H provoacă oprirea programului care se execută și trece controlul rutinei care gestionează întreruperea de timer. După ce se realizează procesarea întreruperii provocate de scurgerea intervalului de timp, execuția trece înapoi la programul întrerupt.

Deși întreruperea 08H este asincronă, mesajele WM\_TIMER nu sunt asincrone. Mesajele WM\_TIMER sunt introduse ca oricare alt mesaj în coada de mesaje a aplicației, și sunt extrase de către funcția GetMessage și trimise către procedura de fereastră prin intermediul funcției DispatchMessage. Deci dacă argumentul funcției SetTimer are, de exemplu valoarea 1000 msec, sosirea mesajelor WM\_TIMER la procedura de fereastră nu se va face în mod sigur la o perioadă de 1 sec. Dacă o altă aplicație realizează o procesare îndelungată în interiorul unui singur mesaj, programul nu va primi nici un mesaj WM\_TIMER în aceasta perioadă de timp. Mesajul WM\_TIMER este de prioritate scăzută (ca și mesajul WM\_PAINT). Dacă în coada de mesaje a unei aplicații nu se găsesc decât mesaje WM\_PAINT și WM\_TIMER, iar coada de mesaj a unei alte aplicații conține alte mesaje, Windows va activa cea de-a doua aplicație.

Mesajele WM\_TIMER sunt asemănătoare cu mesajele WM\_PAINT și din alt punct de vedere: Windows nu introduce mai multe mesaje WM\_TIMER în coada de mesaje a unei aplicații. Windows combină mai multe mesaje WM\_TIMER într-unul singur. O aplicație nu poate determina numărul de mesaje WM\_TIMER care nu au fost astfel introduse în coada de mesaje, deci o aplicație Windows nu poate să calculeze durata intervalelor de timp cu ajutorul mesajelor WM\_TIMER.

## 7.3 Utilizarea timer-ului

Dacă aplicația necesită utilizarea unui timer pe întreaga durată de execuție, se va apela funcția SetTimer în InitInstance sau în timpul procesării mesajului WM\_CREATE și KillTimer în timpul procesării mesajului WM\_DESTROY. În funcție de argumentele apelului funcției SetTimer, exista trei modalități de utilizare a timer-ului.

### 7.3.1 Prima metodă de utilizare a timer-ului

Aceasta metoda utilizează procesarea mesajului WM\_TIMER în procedura de fereastră. Apelul funcției SetTimer se face cu următoarele argumente:

**SetTimer (hwnd, 1, wMsec, NULL);**

Primul argument este handle-ul de fereastră a cărei procedură de fereastră va primi mesajele WM\_TIMER. Următorul argument este identificatorul de timer, care trebuie să fie un număr nenul și a cărui valoare va fi trimisă în argumentul wParam al procedurii de fereastră. Cel de-al treilea argument este un cuvânt pe 16 biți care specifică intervalul în milisecunde între două mesaje WM\_TIMER. Valoarea maximă (65535) va provoca câte un mesaj de timer la aproximativ 1 minut.

Mesajele de timer pot fi oprite în orice moment și eliminate, dacă există, din coada de mesaje a aplicației prin apelul funcției KillTimer:

**KillTimer (hwnd, 1);**

Cel de-al doilea argument este același identificator de timer utilizat în apelul funcției SetTimer. Aplicația are datoria ca înainte de terminare să dezactiveze toate timerele.

Când procedura de fereastră primește un mesaj WM\_TIMER, argumentul wParam are valoarea identificatorului timer-ului care a produs mesajul (în cazul de mai sus, valoarea 1), iar argumentul lParam are valoarea 0. Dacă o fereastră utilizează mai multe timere, este necesar să atribuie identificatori de timer diferiți, pentru a diferenția timer-ul care a generat mesajul WM\_TIMER:

```
#define TIMER_SEC    1
#define TIMER_MIN    2
```

Obținerea celor două timere se realizează cu apelurile:

```
SetTimer (hwnd, TIMER_SEC, 1000, NULL);
SetTimer (hwnd, TIMER_MIN, 60000, NULL);
```

Procesarea mesajului WM\_TIMER se va realiza în următorul mod:

```
long WMTimer (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    switch (wParam) {
        case TIMER_SEC:
            /* procesarea se face la interval de o secundă */
            break;

        case TIMER_MIN:
            /* procesarea se face la interval de un minut */
            break;
    }
    return 0;
}
```

Dacă se dorește reprogramarea intervalului de timp pentru un timer, este necesar apelul funcției KillTimer, după care se poate apela funcția SetTimer cu noul interval de timp:

```
KillTimer (hwnd, 1);
SetTimer (hwnd, 1, wMsec, NULL);
```

Argumentul wMsec specifică noul interval de timp în milisecunde. Aplicația Windows Clock utilizează această metodă pentru a modifica intervalul de timp de la 1000 msec la 60000 msec când este iconizată.

### 7.3.2 Tratarea cazului în care nu există timere disponibile

Windows permite ca numai un număr de 16 timere să fie active simultan. Dacă nici un timer nu este disponibil, funcția SetTimer întoarce valoarea NULL. Programul poate fi terminat în cazul în care nu există timere disponibile. Dacă apelul funcției SetTimer se realizează în InitInstance, întoarcerea unei valori FALSE ca rezultat al acestei funcții va provoca încheierea funcției WinMain, și deci încheierea programului.

Să presupunem că dorim un timer cu un interval de 1000 msec. După apelul funcției CreateWindow, se pot introduce următoarele instrucțiuni:

```
if (!SetTimer (hwnd, 1, 1000, NULL))
    return FALSE;
```



Efectul acestor instrucțiuni este terminarea programului în cazul în care acesta nu poate alocă timer-ul necesar. Pentru a oferi utilizatorului o interfață mai prietenoasă, este necesar să se utilizeze o fereastră de mesaj - Message Box care să explice cauza din care s-a hotărât terminarea aplicației.

Un message box este o fereastră "pop-up" care este afișată în centrul ecranului. Message box-urile au caption bar, dar nu au margini de dimensionare. Caption bar-ul conține de obicei numele aplicației. Message box-ul conține un mesaj și unul, două sau trei butoane (o combinație între: "OK", "Retry", "Cancel", "Yes", "No" sau altele). Message box-urile pot conține de asemenea, câte un icon, putând simboliza: o litera "i" (provenind de la "information"), un semn de exclamare, un semn de întrebare, sau un semn de stop.

Următoarea secvență de program creează un message box de informare, care poate fi utilizată când funcția SetTimer eșuează să aloce un timer:

```
if (!SetTimer (hwnd, 1, 1000, NULL)) {
    MessageBox (hwnd,
        "Too many clocks or timers!", "Program Name",
        MB_ICONEXCLAMATION | MB_OK);
    return FALSE;
}
```

Message box-ul creat este prezentat în figura 7.1. Când utilizatorul apasă tasta ENTER sau apasă butonul "OK", funcția WinMain se încheie, întorcând valoarea FALSE.

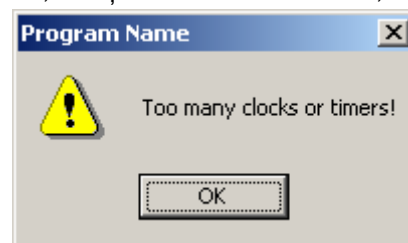


Figura. 7.1.

În mod implicit message box-urile sunt ferestre modale în raport cu aplicația. Acest lucru semnifică faptul că utilizatorul trebuie să închidă fereastra de mesaj înainte ca aplicația să poată continua. Cu toate acestea, este posibil să se comute într-o altă aplicație (prin apăsarea butonului stânga al mouse-ului în suprafața ferestrei altei aplicații).

Este posibil ca utilizatorul să închidă astfel una din aplicațiile care țin ocupate toate timerele sistemului. Programul poate să reîncece să aloce un timer. Secvența următoare de program afișează o fereastră de dialog cu două butoane: "Retry" și "Cancel". Dacă utilizatorul apasă butonul "Cancel", funcția MessageBox întoarce valoarea IDCANCEL și aplicația se încheie. Dacă utilizatorul apasă butonul "Retry", programul reîncearcă să aloce un timer, în caz de insucces, se reafișează fereastra de mesaj:

```
while (!SetTimer (hwnd, 1, 1000, NULL)) {
    if (IDCANCEL = MessageBox (hwnd,
        "Too many clocks or timers!", "Program Name",
        MB_ICONEXCLAMATION | MB_RETRYCANCEL))
        return FALSE;
}
```

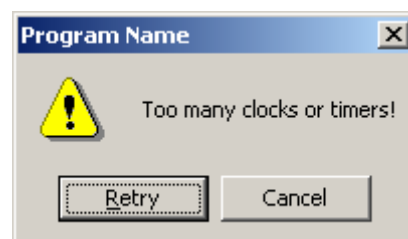


Figura. 7.2.

### 7.3.3 Aplicație ce utilizează timere

Programul Beeper1 utilizează un timer cu un interval de o secundă. La primirea mesajului WM\_TIMER, el își alternează culoarea cu care este umplută suprafața client din roșu în albastru, și emite un beep cu ajutorul funcției MessageBeep. Beeper1 inițializează

timer-ul în funcția `InitInstance` și procesează mesajul `WM_TIMER` în procedura de fereastră `WndProc`. În funcția de procesare a mesajului de timer, `Beeper1` cheamă funcția `MessageBeep` și inversează valoarea variabilei `bFlipFlop`, după care își invalidează suprafața client astfel încât să se genereze un mesaj `WM_PAINT`. În funcția de procesare a mesajului `WM_PAINT`, `Beeper1` colorează dreptunghiul client cu ajutorul funcției `FillRect`.

## 7.4 Utilizarea culorilor în Windows

Windows utilizează o valoare de tipul `unsigned long` pentru a reprezenta culorile. Cei trei octeți mai puțini semnificativi specifică valoarea pentru nuanțele roșu, verde și albastru printr-un număr între 0 și 255 (un octet). În total sunt admise un număr de  $2^{24}$  nuanțe de culori (aproximativ 16 milioane de culori).

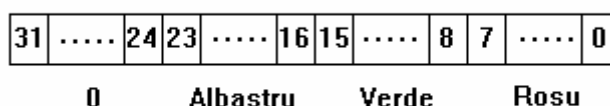


Figura. 7.3.

Această valoare `unsigned long` este denumită "RGB color". Fișierul `Wingdi.h` oferă diferite macro-uri utile lucrului cu valori pentru culori RGB. Macro-ul `RGB` definit în `Wingdi.h` cere trei argumente reprezentând componentele roșu, verde și albastru pe care le combină într-un `unsigned long` pentru a obține o culoare RGB. Valoarea obținută prin chemarea macro-ului `RGB`:

**RGB (255, 0, 255);**

este `0x00FF00FF`, adică valoarea pentru culoarea magenta. Când toate cele trei argumente au valoarea 0, culoarea este neagră, iar când au valoarea 255, culoarea obținută este albă.

Macro-urile `GetRValue`, `GetGValue` și `GetBValue` extrag valoarea componentelor roșu, verde și albastru dintr-o culoare RGB.

Monitoarele de tip VGA permit afișarea simultană a unui număr de 16 culori distincte. Restul de culori sunt afișate de către Windows printr-un procedeu denumit "dithering". Prin acest procedeu o zonă de o anumită culoare este creată prin afișarea unui șablon de pixeli din culori pure.

Programul `Beeper1` utilizează funcția `FillRect` pentru a umple suprafața client. Primul argument al funcției `FillRect` este un handle de device context, al doilea este o referință către o variabilă de tip structură `RECT`, iar al treilea argument este un handle la un "brush". Un brush este un obiect grafic utilizat de către Windows pentru umplerea suprafețelor. Brush-urile pot avea textura solidă, hașurată sau compusă dintr-un șablon care se repetă pe orizontală și pe verticală.

`Beeper1` creează un brush cu textura solidă, prin apelul funcției `CreateSolidBrush`. Unicul parametru al funcției este o valoare de culoare RGB, obținută cu ajutorul macro-ului `RGB`. În funcție de valoarea variabilei `bFlipFlop`, `Beeper1` utilizează culoarea RGB (255, 0, 0) adică roșu, sau RGB (0, 0, 255), adică albastru.

Un brush este un obiect grafic. Numărul de obiecte grafice de care dispune biblioteca `Gdi.exe` este finit. Crearea unui astfel de obiect grafic trebuie să fie însoțită (după ce nu mai este necesară utilizarea sa) de ștergerea sau eliberarea sa. Programul `Beeper1`, după apelul funcției `FillRect` în care utilizează brush-ul creat, apelează funcția `DeleteObject` pentru a elibera obiectul grafic.

## 7.5 A doua metodă de utilizare a timer-ului

Prima metodă de utilizare a timer-ului constă în procesarea mesajului `WM_TIMER` care sosește la procedura de fereastră a aplicației. Cea de-a doua metodă permite direcționarea mesajelor de timer către o funcție specială utilizată doar pentru procesarea evenimentelor de timp.

Funcția care recepționează mesajele de timer este o funcție de tip "call-back". Acest tip este constituit din acele funcții care nu sunt apelate direct din program, ci

ele sunt apelate de către Windows. Programul trimite sistemului de operare adresa funcției respective, iar Windows apelează funcția "call-back" de câte ori este necesar. Aceste funcții sunt foarte asemănătoare cu procedurile de fereastră. Adresa procedurii de fereastră este făcută cunoscută sistemului de operare atunci când se înregistrează clasa de fereastră, iar Windows cheamă această procedură când trimite mesaje către aplicație. Cu toate acestea, funcțiile "call-back" care nu sunt proceduri de fereastră, trebuie gestionate într-un mod mai special.

Funcția `SetTimer` nu este singura funcție Windows care utilizează funcții "call-back".

Ca și o procedură de fereastră, o funcție "call-back" trebuie declarată cu modificatorul `CALLBACK`, deoarece este chemată de către Windows dintr-un segment de cod diferit de cel al programului care o definește. Tipul argumentelor și valoarea întoarsă de către funcțiile "call-back" depind de scopul în care este utilizată funcția respectivă. În cazul unei funcții call-back asociată timer-ului, aceste argumente sunt identice cu cei ai unei proceduri de fereastră, chiar dacă sunt definiți diferit. Funcția definită nu întoarce nici o valoare.

Dacă denumim funcția call-back utilizată pentru timer `TimerProc`, prototipul acestei funcții va fi:

```
VOID CALLBACK TimerProc ( HWND hwnd,  
                        UINT message,  
                        UINT iTimerID,  
                        DWORD dwTime)  
  
{  
    /* procesarea mesajelor WM_TIMER */  
    return 0;  
}
```

Argumentul `hwnd` este handle-ul de fereastră specificat în apelul funcției `SetTimer`. Windows trimite doar mesaje `WM_TIMER` procedurii `TimerProc`, deci argumentul `message` va fi totdeauna egal cu `WM_TIMER`. Argumentul `iTimerID` conține valoarea identificatorului de timer, iar argumentul `dwTime` poate fi ignorat.

Prima metodă de utilizare a timer-ului realizează un apel la funcția `SetTimer` de forma:

```
SetTimer (hwnd, 1, 1000, NULL);
```

Când se utilizează o funcție call-back pentru procesarea mesajelor `WM_TIMER`, cel de-al patrulea argument este adresa far către o scurtă secvență de cod denumită "reload thunk" și care are ca efect apelul funcției call-back. Reload thunk-ul are ca scop încărcarea registrului `DS` cu valoarea adresei de segment a datelor programului. În acest caz, funcția `SetTimer` se apelează cu cel de-al patrulea argument `TimerProc`, care este chiar adresa reload thunk-ului pentru funcția `TimerProc`:

```
SetTimer (hwnd, 1, 1000, TimerProc);
```

Instrucțiunea de mai sus are ca efect alocarea unui timer pentru program și trimiterea tuturor mesajelor `WM_TIMER` către funcția call-back `TimerProc`.

Programul `Beeper2` implementează cea de-a doua metodă de utilizare a timer-ului pentru a realiza un program cu aceeași funcționalitate ca cel prezentat în primul exemplu.

## 7.6 Utilizarea funcțiilor "call-back"

Cerințele pentru utilizarea unei funcții call-back sunt următoarele:

1. Funcția trebuie să fie definită cu modificatorul `CALLBACK`.
2. Adresa funcției care este dată sistemului de operare este chiar numele funcției call-back.

Cele două cerințe rezultă din faptul că Windows utilizează același segment de cod pentru toate instanțele ale aceleiași aplicații, iar fiecare instanță deține un segment de date diferit. Windows are nevoie de acest house keeping pentru a putea deplasa în memorie segmentele de cod și de date ale aplicației. Implicațiile acestui mecanism sunt următoarele:

1. Definirea unui funcții `CALLBACK`, ce implică automat modificatorul `FAR`, determină compilatorul de C să insereze o secvență de intrare specială în funcția respectivă. Aceasta secvență, în limbaj de asamblare este următoarea:

```
push    ds      ;  
pop     ax      ; registrului AX i se atribuie valoarea registrului DS  
                ; de la intrarea în funcție  
nop
```

```
inc    bp
push   bp
mov     bp,sp
push   ds    ; se salvează valoarea registrului ds
mov     ds,ax ; în registrul DS se încarcă valoarea registrului AX, adică
              ; valoarea din registrul DS rămâne nemodificată.
```

- Windows înlocuiește primele două instrucțiuni din secvența de intrare a funcției cu nop-uri în momentul încărcării segmentului de cod respectiv în memorie. Prin această modificare, prologul funcției va arata astfel:

```
nop
nop
nop
inc     bp
push    bp
mov     bp,sp
push    ds    ; se salvează valoarea registrului ds
mov     ds,ax ; în registrul DS se încarcă valoarea registrului AX.
```

Deci în registrul DS se încarcă valoarea care se găsea în registrul AX în momentul apelului funcției.

- La apelul funcției de tip call-back se creează o scurtă secvență de instrucțiuni numită reload thunk. Secvența de instrucțiuni este înscrisă într-un segment special al Windows. Această secvență de instrucțiuni încarcă registrul AX cu valoarea segmentului de date al programului după care face un salt la secvența de intrare în funcția definită. Secvența de intrare în funcție încarcă această valoare din registrul AX în registrul DS, pe care o utilizează ca adresă a segmentului de date.

## 7.7 A treia metodă de utilizare a timer-ului

Cea de-a treia metodă de utilizare a timer-ului este similară cu metoda anterioară. Argumentul hwnd din apelul funcției SetTimer are însă valoarea NULL, iar cel de-al doilea argument (identificatorul de timer) este ignorat. Funcția SetTimer întoarce un identificator de timer:

**nTimerID = SetTimer (NULL, 0, wMSecInterval, TimerProc);**

Valoarea variabilei nTimerID întoarce de către SetTimer va fi NULL dacă nu există nici un timer disponibil.

În acest caz, apelul funcției KillTimer trebuie să se realizeze cu primul argument (handle-ul de fereastră) egal cu NULL, iar cel de-al doilea argument va fi valoarea întoarsă de către funcția SetTimer, adică nTimerID:

**KillTimer (NULL, nTimerID);**

Argumentul hwnd cu care este chemată funcția TimerProc are valoarea NULL. Argumentul iTimerID conține valoarea identificatorului de timer, adică nTimerID, iar argumentul lParam este lpfnTimerProc.

## 7.8 Utilizarea timer-ului pentru afișarea unor informații de stare

O utilizare foarte frecventă a timer-ului este pentru a se reîmprospăta periodic anumite informații de stare afișate pe ecran. Programul FreeMem afișează spațiul de memorie liberă din Windows în megaocteți. Valoarea spațiului de memorie disponibilă este reîmprospătată la fiecare secundă.

Deoarece FreeMem nu are nevoie de un spațiu foarte mare de afișare, el poate afișa toată informația în interiorul suprafeței aferente din taskbar-ul Windows-ului - figura. 7.4.



Figura. 7.4.

### 7.8.1 Pornirea unei aplicații sub forma minimizată

În mod normal, un program afișează fereastra principală utilizând apelul la funcția ShowWindow cu argumentele:

**ShowWindow (hwnd, nCmdShow);**

Variabila nCmdShow este unul din argumentele funcției WinMain. Nu este însă obligatoriu utilizarea variabilei nCmdShow ca argument de apel al funcției ShowWindow.

Pentru a forța o aplicație să pornească în forma minimizată (ca în cazul programului FreeMem), se pot utiliza în apelul funcției ShowWindow următoarele argumente:

**ShowWindow (hwnd, SW\_SHOWMINNOACTIVE);**

Dacă se dorește ca aplicația să pornească maximizată, apelul funcției ShowWindow se va face cu argumentele:

**ShowWindow (hwnd, SW\_SHOWMAXIMIZED);**

### 7.8.2 Afișarea informațiilor în taskbar

FreeMem afișează informațiile utile în taskbar cu ajutorul unui mecanism foarte simplu: schimbarea Caption bar-ului aplicației. În acest sens, în cadrul funcției care prelucrează mesajul WM\_TIMER se actualizează la fiecare secundă Caption bar-ul. Această schimbare se reflectă și în taskbar.

Actualizarea Caption bar-ului se realizează cu ajutorul funcției SetWindowText astfel:

**SetWindowText(hwnd, cBuffer);**

unde cBuffer reprezintă un șirul de caractere ce urmează a fi afișat în Caption bar.

### 7.8.3 Spațiul de memorie disponibil

Când procedura de fereastră primește un mesaj WM\_TIMER, FreeMem determină numărul de octeți disponibili în memoria sistem, folosind o variabilă de tip MEMORYSTATUS și funcția GlobalMemoryStatus.

## 7.9 Utilizarea timer-ului pentru realizarea unui ceas

Programul DigClock creează o fereastră popup care se poziționează automat în colțul dreapta jos al ecranului. Programul afișează ora exactă și data (luna, zi, an) curentă.

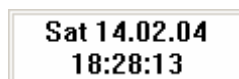


Figura. 7.5.

### 7.10 Temă

Se cere să se realizeze un ceas analog pe baza programului anterior.

## Laborator 8 - Ferestre control

### 8.1 Ferestrele de control în Windows

Ferestrele de control realizează procesarea mesajelor de la tastatură și de la mouse și trimit mesaje de notificare către fereastra părinte dacă starea lor s-a modificat. În acest mod, fereastra control copil devine o sursă de mesaje de intrare pentru fereastra părinte.

Windows pune la dispoziția aplicațiilor o serie de clase (și de proceduri de fereastră) pentru ferestre de control predefinite. Pe baza acestor clase de fereastră, orice aplicație Windows își poate crea propriile ferestre control copil. Aceste controale se prezintă sub forma butoanelor, check box-urilor (butoane non-exclusive), radio buttons (butoane exclusive), ferestre de editare (edit box), list box-uri (liste de selecție), combo box-uri (communication box - controale compuse dintr-o fereastră de editare asociată cu un list box), ferestre pentru afișare de texte, scroll bar-uri etc. Introducerea unui buton în interiorul unei ferestre de aplicație se realizează cu ajutorul funcției `CreateWindow`. Acțiunile mouse-ului și funcțiile de desenare a butonului în funcție de starea curentă sunt realizate automat de către procedura de fereastră a butonului conținută într-o bibliotecă Windows. Singura responsabilitate a aplicației este să gestioneze mesajul `WM_COMMAND`, care notifică fereastra părinte de faptul că butonul a fost apăsat.

Ferestrele de control sunt utilizate foarte frecvent în interiorul ferestrelor de dialog. Este însă posibilă utilizarea acestor ferestre control în interiorul suprafeței client a ferestrei principale a unei aplicații. Fiecare fereastră de control copil trebuie creată cu ajutorul funcției `CreateWindow`, iar poziția sa poate fi modificată prin apelul funcției `MoveWindow`. Procedura de fereastră a ferestrei părinte poate să trimită mesaje către procedura de fereastră a ferestrei de control copil, iar aceasta, în cele mai multe cazuri, trimite mesaje către procedura de fereastră a ferestrei părinte.

O aplicație Windows pornește prin a-și defini clasa de fereastră a ferestrei principale pe care apoi o înregistrează prin apelul funcției `RegisterClass`. Apoi fereastra principală este creată cu ajutorul funcției `CreateWindow`. Când o aplicație dorește utilizarea unei ferestre control copil nu mai este necesară inițializarea și înregistrarea structurii de clasa de fereastră pentru fereastra de control, deoarece Windows a făcut aceasta în momentul pornirii sistemului. Clasa de fereastră este deja înregistrată în Windows și are una din denumirile: "button", "static", "scrollbar", "edit", "listbox" sau "combobox". Aplicația Windows trebuie doar să utilizeze una din aceste denumiri ca nume de clasă în apelul funcției `CreateWindow` și să specifice poziția dorită a ferestrei în interiorul suprafeței client. Argumentul "style" al funcției `CreateWindow` permite specificarea mai precisă a felului în care va fi desenată fereastra pe ecran și a funcționalității acesteia. De exemplu butoanele, check box-urile și radio buttons fac toate parte din aceeași clasă de ferestre denumită "button", specificul fiecărei tip de buton este stabilit de către argumentul style al funcției `CreateWindow`.

### 8.2 Clasa "button"

Programul `BtnLook` prezentat în continuare arată stilurile de butoane predefinite de către Windows precum și modul de funcționare a lor. `BtnLook` creează 10 ferestre copil, fiecare fiind un control button, câte una pentru cele 10 stiluri de butoane posibile.

Apăsarea fiecărui buton provoacă trimiterea unui mesaj `WM_COMMAND` către procedura de fereastră părinte, denumită `WndProc`. Procedura de fereastră a ferestrei principale a programului `BtnLook` afișează în partea dreaptă a ecranului conținutul argumentelor `wParam` și `lParam` ale acestui mesaj, după cum se poate observa în figura 8.1.

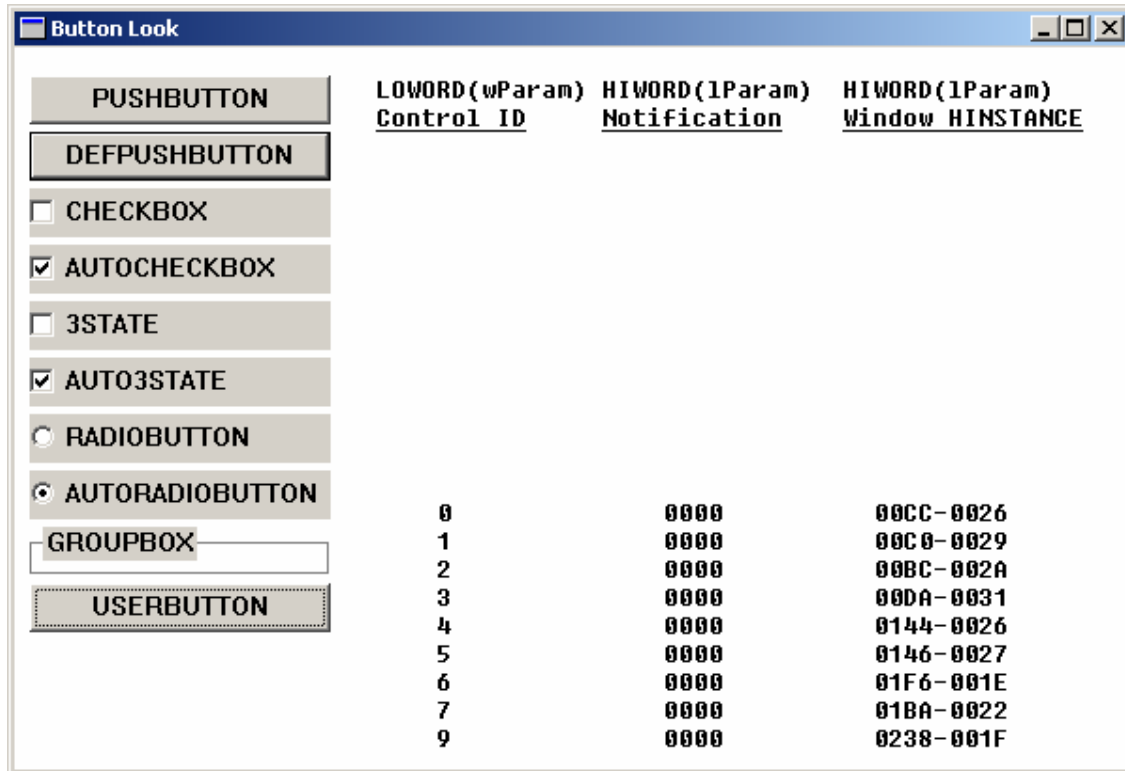


Figura 8.1. Programul BtnLook

### 8.2.1 Crearea ferestrelor copil

BtnLook definește o structură denumită `buttonstruct` care conține stilurile de fereastră buton precum și un text descriptiv al celor 10 tipuri de butoane. Stilurile celor 10 tipuri de butoane încep cu prefixul `BS_`, de la "button style".

Cele 10 ferestre copil sunt create într-o buclă `for` în interiorul mesajului `WM_CREATE` procesat în funcția de procesare de mesaj `WmCreate`. Funcția `CreateWindow` utilizează următoarele valori ale argumentelor de apel:

Argument	Valoare
Numele clasei de fereastră	"button"
Textul ferestrei	button [i]. text
Stilul ferestrei	WS_CHILD   WS_VISIBLE   button [i]. style
Poziția orizontală	cxChar
Poziția verticală	cyChar*(2*i + 1)
Lățimea ferestrei	20*cxChar
Înălțimea ferestrei	7*cyChar / 4
Handle-ul ferestrei părinte	hwnd
Identificatorul de copil	i
Handle-ul de instanță	((LPCREATESTRUCT) lParam)->hInstance
Parametrii suplimentari	NULL

Numele clasei de fereastră este un nume predefinit. Stilul ferestrei combină stilurile `WS_CHILD` (obligatoriu pentru toate ferestrele copil) și `WS_VISIBLE` și unul dintre cele 10 stiluri de butoane (`BS_PUSHBUTTON`, `BS_DEFPUSHBUTTON` etc.) memorate în câmpul `style` al structurii `buttonstruct`. Textul ferestrei (care pentru o fereastră normală cu stilul `WS_OVERLAPPEDWINDOW` apare în interiorul caption bar-ului) este textul care va fi afișat în interiorul fiecărui buton. S-a utilizat descrierea stilului de buton din câmpul `text` al structurii `buttonstruct`.

Poziția orizontală și verticală reprezintă coordonatele colțului stânga sus al butonului în sistemul de coordonate al ferestrei client. Argumentul identificator copil conține o valoare întreagă și trebuie să identifice unic ferestrele control copil. Acest argument este utilizat pentru identificarea ferestrei control copil care trimite mesajul `WM_COMMAND` către părinte.

Argumentul handle de instanță utilizează pointer-ul către o structură de tipul CREATESTRUCT ("creation structure") pe care Windows îl trimite în argumentul lParam al mesajului WM\_CREATE. Un membru al acestei structuri este valoarea hInstance, handle-ul de instanță al aplicației trimis ca argument al funcției WinMain. Pentru a obține valoarea acestui câmp este necesară realizarea unui type cast cu tipul LPCREATESTRUCT ("long pointer to CREATESTRUCT") asupra argumentului. Programele Windows pot utiliza o variabilă globală denumită hInst, care este inițializată în WinMain cu valoarea argumentului hInstance:

**hInst = hInstance;**

și pe care apoi o utilizează pentru a obține valoarea handle-ului de instanță a programului, sau utilizează funcția GetWindowLong pentru a obține valoarea handle-ului de instanță din structura de fereastră a ferestrei principale:

**GetWindowLong (hwnd, GWL\_HINSTANCE);**

Oricare dintre aceste metode este la fel de bună.

După apelul funcției CreateWindow nu mai este necesară gestionarea funcționării ferestrelor copil de tip buton. Ferestrele copil primesc mesajele de mouse sau tastatură, se redesenează dacă este necesar. Singura excepție este butonul cu stilul BS\_USERBUTTON, care cere ca fereastra părinte să deseneze și să redeseneze suprafața sa client. La terminarea aplicației, Windows distruge automat ferestrele copil când fereastra părinte (fereastra principală a aplicației) este distrusă.

## 8.2.2 Conversația dintre fereastra copil și fereastra părinte

Conversația dintre ferestrele copil și părinte se realizează prin intermediul mesajelor pe care aceste ferestre le schimbă între ele. În momentul lansării în execuție a aplicației, în partea stângă a suprafeței client se afișează o serie de tipuri de butoane (butonul cu stilul BS\_USERBUTTON nu este vizibil). Apăsarea unui buton cu mouse-ul sau acționarea lui de la tastatură cu ajutorul tastei spațiu provoacă trimiterea unui mesaj de tipul WM\_COMMAND către procedura de fereastră a ferestrei părinte. Procedura de fereastră procesează acest mesaj în funcția WMCommand și afișează în interiorul suprafeței client valorile argumentelor wParam și lParam, care au următoarea semnificație:

<b>LOWORD(wParam)</b>	<b>ID-ul de copil</b>
<b>HIWORD (wParam)</b>	<b>Codul de notificare</b>
<b>lParam</b>	<b>Handle-ul ferestrei copil</b>

Valoarea ID-ului de copil este valoarea care a fost specificată ca argument al funcției CreateWindow. ID-urile utilizate în programul BtnLook au valorile cuprinse între 0 și 9 pentru cele 10 butoane afișate în suprafața client a aplicației. Valoarea handle-ului de fereastră copil este valoarea întoarsă ca rezultat al funcției CreateWindow pentru butoanele respective.

Codul de notificare este un subcod de mesaj utilizat de către ferestrele copil pentru a trimite ferestrei părinte tipul de acțiune realizat asupra lor. Valorile acestui argument pot fi:

Identificatorul codului de notificare	Semnificație
BN_CLICKED	Butonul a fost apăsător
BN_PAINT	Butonul trebuie redesenat (doar pentru butoane cu stilul BS_USERBUTTON)
BN_HILITE	Butonul trebuie redesenat focusat și apăsător (doar pentru butoane cu stilul BS_USERBUTTON)
BN_UNHILITE	Butonul trebuie redesenat focusat și ridicat (doar pentru butoane cu stilul BS_USERBUTTON)
BN_DISABLE	Butonul trebuie desenat dezactivat (doar pentru butoane cu stilul BS_USERBUTTON)
BN_DOUBLECLICKED	Butonul a fost double-click (doar pentru butoane cu stilul BS_RADIOBUTTON și BS_USERBUTTON)

Pentru toate stilurile de butoane (cu excepția stilului BS\_USERBUTTON) valoarea codului de notificare este BN\_CLICKED. Celelalte coduri de notificare sunt trimise de către butoanele cu stilul BS\_USERBUTTON, pentru care funcția de desenare a suprafeței client este realizată de către părinte.



Se poate observa că atunci când se apasă un buton cu mouse-ul, o linie întreruptă apare în jurul textului din butonul respectiv. Aceasta linie întreruptă semnalizează faptul că focus-ul de intrare de la tastatură este îndreptat către fereastra butonului respectiv. Butonul ignoră apăsarea oricărei taste cu excepția tastei spațiu care are același efect ca apăsarea butonului de mouse în interiorul sau.

Procedura de fereastră a ferestrei părinte poate trimite la rândul sau mesaje către ferestrele copil. Cinci tipuri de mesaje sunt asociate ferestrelor de clasa "button". Toate aceste mesaje au prefixul BM\_ ("button message"). Aceste mesaje sunt definite în Windows.h:

```
#define BM_GETCHECK    (WM_USER + 0)
#define BM_SETCHECK    (WM_USER + 1)
#define BM_GETSTATE    (WM_USER + 2)
#define BM_SETSTATE    (WM_USER + 3)
#define BM_SETSTYLE    (WM_USER + 4)
```

Identificatorul WM\_USER reprezintă cea mai mică valoare de mesaj disponibilă programelor utilizator (această valoare și valorile mai mari nu sunt utilizate ca mesaje predefinite de către WINDOWS). Orice mesaj utilizator trebuie să aibă valoarea tipului de mesaj (argumentul message cu care este chemată funcția de fereastră) mai mare decât aceasta valoare.

Mesajele BM\_SETCHECK și BM\_GETCHECK pot fi trimise de către procedură de fereastră a ferestrei părinte pentru a citi sau scrie starea butoanelor de tip check box și radio button. Mesajele BM\_GETSTATE și BM\_SETSTATE sunt utilizate în legătură cu starea apăsată sau ridicată a butoanelor când sunt apăstate cu mouse-ul sau cu ajutorul tastei spațiu. Mesajul BM\_SETSTYLE permite modificarea stilului butonului după crearea ferestrei asociate.

### 8.2.3 Butoanele cu revenire

Primele două butoane din aplicația BtnLook sunt două butoane cu revenire (sau "push buttons"). Butoanele cu revenire sunt utilizate pentru lansarea de acțiuni. Cele două butoane au stilurile BS\_PUSHBUTTON și BS\_DEFPUSHBUTTON. "DEF" provine de la "default". Când aceste butoane sunt utilizate în ferestre de dialog, ele indică acțiuni implicite, care se realizează prin apăsarea tastei Enter.

Apăsarea butonului de mouse în interiorul ferestrei unui buton provoacă redesenarea acestuia precum și capturarea mouse-ului. Mesajul WM\_COMMAND este trimis de către butoane la ridicarea butonului de mouse dacă cursorul acestuia se află în interiorul ferestrei asociate butonului.

Procedura de fereastră părinte poate să simuleze apăsarea și eliberarea butonului de mouse utilizând mesajul BM\_SETSTATE:

```
SendMessage (hwndButton, BM_SETSTATE, 1, 0L);
```

provoacă apăsarea butonului, iar:

```
SendMessage (hwndButton, BM_SETSTATE, 0, 0L);
```

provoacă revenirea butonului la poziția normală.

Mesajul BM\_GETSTATE permite interogarea stării butonului. Întoarcerea valorii TRUE ca rezultat al funcției SendMessage specifică faptul că butonul respectiv este apăsat; valoarea FALSE indică faptul că butonul se află în stare normală.

### 8.2.4 Butoane non-exclusive (check box-uri)

Check box-urile sunt utilizate pentru a selecta sau deselecta opțiuni ale unei aplicații. Check box-urile funcționează ca un comutator. Când în interiorul unui check box este afișat un "v" (o bifă), opțiunea respectivă este selectată; când check box-ul nu conține un "v" opțiunea este deselectată. Apăsarea butonului stânga al mouse-ului provoacă selectarea check box-ului, iar apăsarea din nou a butonului de mouse în interiorul check box-ului provoacă deselectarea acestuia.

Cele mai des utilizate stiluri de check box sunt BS\_CHECKBOX și BS\_AUTOCHECKBOX. Utilizarea stilului BS\_CHECKBOX necesită ca procedura de fereastră părinte să pună sau să îndepărteze semnul de selecție ca rezultat al mesajelor WM\_COMMAND trimise de către check box-uri cu codul de notificare BM\_CLICKED. Selectarea, respectiv deselectarea, check box-urilor se face prin trimiterea mesajului BM\_SETCHECK. Argumentul wParam are valoarea 1 dacă se dorește selectarea respectiv 0

În caz contrar. Starea curentă a unui check box se poate obține cu ajutorul mesajului BM\_GETSTATE care întoarce 0 dacă check box-ul nu este selectat. Secvența de program utilizată pentru a comuta semnul "v" într-un check box, în cadrul mesajului WM\_COMMAND din procedura de fereastră părinte poate fi realizată în următoarea manieră:

```
SendMessage ( (HWND) lParam, BM_SETCHECK,  
              (LPARAM) !SendMessage ((HWND) lParam, BM_GETCHECK, 0, 0), 0);
```

Operatorul "!" din fața celui de-al doilea SendMessage este utilizat pentru inversarea stării curente a check box-ului. Argumentului lParam are valoarea handle-ului de fereastră a ferestrei check box care a trimis mesajul WM\_COMMAND. Starea unui check box poate fi inițializată cu ajutorul unui mesaj BM\_SETCHECK:

```
SendMessage (hwndButton, BM_SETCHECK, 1, 0);
```

inițializează starea 1 a check box-ului (selectat).

Stilul BS\_AUTOCHECKBOX realizează comutarea automată a semnelui de check din interiorul butonului, procedura de fereastră poate astfel ignora mesajele WM\_COMMAND. Starea curentă a unui check box poate fi citită cu ajutorul mesajului BM\_GETCHECK:

```
nCheck = SendMessage (hwndButton, BM_GETCHECK, 0, 0);
```

valoarea variabilei nCheck va fi, la revenirea din funcție, TRUE, dacă check box-ul este selectat, respectiv FALSE dacă nu.

Stilurile BS\_3STATE și BS\_AUTO3STATE conțin o stare suplimentară nedeterminată în care se afișează un dreptunghi de culoare gri în interiorul check box-ului. Această stare are valoarea 2.

### 8.2.5 Butoane exclusive (radio)

Butoanele radio sunt foarte asemănătoare check box-urilor, cu excepția faptului că au o formă circulară. Un punct negru în interiorul butonului indică faptul că butonul respectiv este selectat. Radio butoanele pot avea stilul BS\_RADIOBUTTON sau BS\_AUTORADIOBUTTON.

În ferestre de dialog, butoanele radio sunt utilizate pentru a indica opțiuni exclusive. Radio butoanele nu realizează comutarea de stare (ca check box-urile), dacă se apasă pe un buton radio selectat, starea sa va rămâne selectată.

Primirea unui mesaj WM\_COMMAND de către procedura de fereastră determină selectarea butonului radio respectiv (wParam egal 1):

```
SendMessage (hwndButton, BM_SETCHECK, 1, 0);
```

Toate celelalte radio butoane din același grup trebuie deselectionate prin trimiterea mesajului BM\_SETCHECK cu argumentul wParam egal cu 0:

```
SendMessage (hwndButton, BM_SETCHECK, 0, 0);
```

### 8.2.6 Grupurile

Stilul grup, BS\_GROUPBOX, reprezintă o zonă rectangulară utilizată în general pentru a înconjura un grup de butoane radio sau de check box-uri. Grupurile nu procesează mesaje de la mouse și de la tastatură și nu trimit mesaje WM\_COMMAND către fereastra părinte.

### 8.2.7 Butoanele definite de utilizator

Clasa BS\_USERBUTTON permite definirea butoanelor desenate de către utilizator. Această clasă de butoane trimite următoarele mesaje de notificare către procedura de fereastră părinte:

BN_PAINT	Butonul trebuie desenat normal
BN_HILITE	Butonul trebuie desenat apăsat și focusat
BN_UNHILITE	Butonul trebuie desenat eliberat și focusat
BN_DISABLE	Butonul trebuie desenat dezactivat

Toate aceste coduri de notificare indică faptul că butonul trebuie desenat. Fereastra părinte are responsabilitatea de a desena butonul. Procedura de fereastră poate utiliza argumentului lParam, care conține handle-ul de fereastră al butonului ca argument al funcțiilor GetClientRect pentru a obține dreptunghiul butonului și GetDC pentru a obține un handle de context la fereastra butonului care poate fi utilizat în funcțiile de desenare.

### 8.2.8 Modificarea textului din interiorul butoanelor

Textul din interiorul unui buton (sau din interiorul oricărei ferestre) poate fi modificat cu ajutorul funcției `SetWindowText`:

**`SetWindowText (hwnd, lpszText);`**

unde `hwnd` reprezintă handle-ul de fereastră al ferestrei al cărui text se modifică, iar `lpszText` reprezintă un pointer către un șir de caractere.

Textul curent dintr-o fereastră se poate obține cu ajutorul funcției `GetWindowText`:

**`nLength = GetWindowText (hwnd, lpszBuffer, nMaxLength);`**

unde `nMaxLength` specifică numărul maxim de caractere care poate fi copiat în `lpszBuffer`. Funcția întoarce lungimea textului copiat în bufferul `lpszBuffer`. Dimensionarea bufferului se poate face cu ajutorul funcției `GetWindowTextLength` care întoarce lungimea șirului de caractere din care este format textul ferestrei:

**`nLength = GetWindowTextLength (hwnd);`**

### 8.2.9 Butoane vizibile, invizibile, active și inactive

Pentru a primi mesaje de la mouse și de la tastatură, o fereastră copil trebuie să fie atât vizibilă cât și activă. Când o fereastră este vizibilă, dar nu este activă, Windows afișează fereastra respectivă utilizând o culoare gri.

Dacă la apelul funcției, în argumentul `style` nu se include atributul `WS_VISIBLE`, fereastra astfel creată nu va fi afișată până nu se apelează funcția `ShowWindow`:

**`ShowWindow (hwndChild, SW_SHOWNORMAL);`**

Dacă stilul `WS_VISIBLE` este inclus în stilul ferestrei, nu este necesar apelul funcției `ShowWindow`. Fereastra copil poate fi însă ascunsă prin apelul funcției `ShowWindow` cu argumentul `SW_HIDE`:

**`ShowWindow (hwndChild, SW_HIDE);`**

ascunde fereastra `hwndChild`. Determinarea stării de vizibilitate a unei ferestre se poate face prin apelul funcției `IsWindowVisible`:

**`bVisible = IsWindowVisible (hwndChild);`**

întoarce `TRUE` dacă fereastra este vizibilă, respectiv `FALSE` dacă nu.

O fereastră poate fi activată sau dezactivată prin apelul funcției `EnableWindow`. Implicit o fereastră este activată (poate primi mesaje de la mouse și tastatură). Dezactivarea unei ferestre se realizează prin apelul:

**`EnableWindow (hwndChild, FALSE);`**

Pentru ferestrele de tip "button", efectul apelului acestei funcții este desenarea textului din interiorul butonului cu o nuanță gri. De asemenea, butonul respectiv nu va mai răspunde la mesaje de la mouse și tastatură. Acest mod este modul ideal de a indica faptul că acțiunea asociată unui buton nu este admisă într-un anumit moment.

Fereastra poate fi reactivată prin apelul funcției:

**`EnableWindow (hwndChild, TRUE);`**

care reactivează fereastra `hwndChild`.

Starea activă sau inactivă a unei ferestre poate fi determinată prin intermediul funcției `IsWindowEnabled`:

**`bEnabled = IsWindowEnabled (hwndChild);`**

întoarce `TRUE` dacă fereastra este activată și `FALSE` în caz contrar.

### 8.3 Culorile ferestrelor control

Windows gestionează un număr de 19 culori denumite culori de sistem. Cele 19 culori definesc culorile cu care sunt afișate anumite porțiuni ale ferestrelor sau ale ecranului. Toate cele 19 culori au asociate identificatori definiți în `Winuser.h` care au prefixul `COLOR_`. Valorile acestor culori sub forma de valori RGB pot fi obținute cu ajutorul funcției `GetSysColor` și pot fi modificate prin apelul funcției `SetSysColor`.

### 8.3.1 Culorile butoanelor

Identificatorii `COLOR_WINDOW` și `COLOR_WINDOWTEXT` sunt utilizați de către majoritatea ferestrelor pentru desenarea suprafeței client. Ferestrele de control "button" utilizează `COLOR_WINDOW` pentru desenarea suprafeței și `COLOR_TEXT` pentru afișarea textului butonului. Există două metode prin care culoarea ferestrei principale și a ferestrelor de control copil să coincidă atât ca fundal cât și culoare de afișare a textului. Prima metodă utilizează culorile de sistem în definirea brush-ului de fundal al ferestrei. Pentru ca fereastra principală să conțină un brush de fundal cu culoarea sistem de fereastră este suficientă specificarea identificatorului `COLOR_WINDOW` în inițializarea câmpului `hbrBackground` al structurii `WNDCLASS`:

```
wndclass.hbrBackground = COLOR_WINDOW + 1;
```

Este necesară incrementarea valorii identificatorului asociat culorii de sistem dorite, pentru a se preveni utilizarea valorii 0. Afișarea textelor în suprafața client a ferestrei trebuie însoțită de setarea corespunzătoare a fundalului precum și a culorii de desenare a caracterelor. Imediat după obținerea handle-ului la device contextul ferestrei se pot utiliza funcțiile:

```
SetBkColor (hdc, GetSysColor (COLOR_WINDOW));
```

```
SetTextColor (hdc, GetSysColor (COLOR_WINDOWTEXT));
```

care setează culorile de desenare ale fundalului textului, respectiv al caracterelor.

Cea de-a doua metodă este de a forța ferestrele de control să fie desenate cu ajutorul culorilor dorite. Aceasta metodă se bazează pe procesarea mesajului `WM_CTLCOLOR`.

### 8.3.2 Mesajul WM\_CTLCOLOR

Mesajul `WM_CTLCOLOR` este un mesaj trimis de către ferestrele control copil către fereastra părinte pentru a cere culorile de desenare a suprafeței client. Procedura de fereastră părinte poate procesa acest mesaj în scopul de a specifica culori de desenare ale ferestrelor de control diferite de cele implicite.

Valorile argumentelor `wParam` și `lParam` ale mesajului `WM_CTLCOLOR` au următoarea semnificație:

<code>wParam</code>	Handle-ul device contextului ferestrei copil
<code>LOWORD (lParam)</code>	Handle-ul ferestrei copil
<code>HIWORD (lParam)</code>	Tipul ferestrei copil

Argumentul `wParam` conține handle-ul de device context a ferestrei copil care va fi utilizat pentru desenarea suprafeței client a ferestrei copil. În acest device context pot fi modificate culorile de fundal și de desenare ale textelor. Cuvântul mai puțin semnificativ al argumentului `lParam` conține handle-ul de fereastră a ferestrei copil. Cuvântul mai semnificativ al argumentului `lParam` conține o valoare care specifică tipul ferestrei control care trimite mesajul. Ferestrele de control predefinite au asociate identificatori cu prefixul `CTLCOLOR_` definiți în `Winuser.h`. Pentru clasa "button", acest identificator este `CTLCOLORBTN`.

Valoarea întoarsă de către procedura de fereastră a ferestrei părinte ca răspuns la mesajul `WM_CTLCOLOR` reprezintă un handle la un brush pe care fereastra control copil îl va utiliza pentru ștergerea fundalului suprafeței client.

Un brush este un obiect GDI utilizat pentru umplerea suprafețelor și care poate fi obținut prin apelul uneia din funcțiile: `GetStockObject`, `CreateSolidBrush`, `CreateHatchBrush` sau `CreatePatternBrush`. Orice brush creat de către utilizator (nu și cele obținute prin apelul funcției `GetStockObject`) trebuie eliberate înainte de terminarea programului prin apelul funcției `DeleteObject` cu argument handle-ul brush-ului respectiv. De obicei acest lucru se realizează în timpul procesării mesajului `WM_DESTROY`.

De exemplu, procesarea unui mesaj `WM_CTLCOLOR` poate arăta în felul următor:

```
HBRUSH hBrush;  
long WmCreate (HWND hwnd, WPARAM wParam, LPARAM lParam)  
{  
.....  
hBrush = CreateSolidBrush (GetSysColor (COLOR_WINDOW));
```

```

    return 0;
}
long WMCtlColor (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    POINT point;
    if (HIWORD (lParam) == COLOR_BTN) {
        SetBkColor ((HDC) wParam, GetSysColor (COLOR_WINDOW));
        SetTextColor ((HDC) wParam, GetSysColor (COLOR_WINDOWTEXT));
        UnrealizeObject (hBrush);
        point.x = point.y = 0;
        ClientToScreen (hwnd, &point);
        SetBrushOrgEx ((HDC) wParam, point.x, point.y, NULL);
        return (LONG) hBrush;
    }
    return DefWindowProc (hwnd, WM_CTLCOLOR, wParam, lParam);
}

```

Un brush definește un șablon de pixeli care se repetă pe orizontală și pe verticală pentru a umple o suprafață închisă. Originea brush-ului este coordonată pe ecran de unde Windows începe repetarea șablonului pe orizontală și pe verticală. În mod implicit, originea brush-ului este egală cu originea device context-ului, adică colțul stânga sus al suprafeței client.

Dacă se dorește umplerea atât a suprafeței client a părintelui cât și a suprafeței client a copiilor cu același brush, șablonul nu se va potrivi corect doar dacă originile brush-ului tuturor copiilor coincid cu originea utilizată pentru brush-ul părintelui. Pentru a realiza acest lucru este necesar apelul funcției `UnrealizeObject` și atribuirea unei noi coordonate de origine a brush-ului prin intermediul funcției `SetBrushOrgEx`. Apelul funcției `UnrealizeObject` are ca efect reinițializarea originii brush-ului specificat ca argument de apel cu prima ocazie când este selectat într-un device context (adică imediat după întoarcerea din funcția de procesare a mesajului `WM_CTLCOLOR`). Originea utilizată pentru reinițializarea brush-ului este cea specificată în argumentele funcției `SetBrushOrgEx`. În exemplul precedent, originea brush-ului este poziționată în colțul stânga sus a suprafeței client a ferestrei părinte.

Brush-ul creat în exemplul anterior se bazează pe o culoare de sistem: `COLOR_WINDOW`. Dacă această culoare se modifică în timpul execuției programului, procedura de fereastră va primi un mesaj `WM_SYSCOLORCHANGE`. Programul trebuie să șteargă brush-ul utilizat și să creeze un nou brush pe baza noii culori de sistem:

```

long WMSysColorChange (HWND hwnd, WPARAM wParam,
                        LPARAM lParam)
{
    DeleteObject (hBrush);
    hBrush = CreateSolidBrush (GetSysColors (COLOR_WINDOW));

    return 0;
}

```

În momentul închiderii aplicației, mesajul `WM_DESTROY` va arăta astfel:

```

long WMDestroy (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    DeleteObject (hBrush);
    PostQuitMessage (0);
    return 0;
}

```

#### 8.4 Clasa "static"

O fereastră de control copil statică poate fi creată utilizând "static" ca denumire a clasei de fereastră în argumentele funcției `CreateWindow`. Ferestrele statice nu primesc intrare de mouse sau de tastatură și nu trimit mesaje `WM_COMMAND` către ferestrele părinte. Mesajul `WM_NCHITTEST` trimis către procedura de fereastră a unei ferestre statice întoarce întotdeauna valoarea `HTTRANSPARENT`, ceea ce face ca mesajul de mouse să fie trimis de Windows ferestrei de sub fereastra statică, adică de obicei ferestrei părinte.

Primele șase stiluri de fereastră statică desenează dreptunghiuri sau margini rectangulare în interiorul suprafeței client părinte. Cele trei stiluri statice "RECT" sunt dreptunghiuri umplute, iar cele trei stiluri statice "FRAME" sunt dreptunghiuri transparente:

Stilul RECT	Stilul FRAME
SS_BLACKRECT	SS_BLACKFRAME
SS_GRAYRECT	SS_GRAYFRAME
SS_WHITERECT	SS_WHITEFRAME

BLACK, GRAY și WHITE sunt bazate pe culorile sistem astfel:

BLACK	COLOR_WINDOWFRAME
GRAY	COLOR_BACKGROUND
WHITE	COLOR_WINDOW

Clasa de ferestre "static" include și trei stiluri de ferestre text: SS\_LEFT, SS\_RIGHT și SS\_CENTER. Aceste ferestre afișează texte aliniate la stânga, dreapta sau central. Textul afișat este cel trimis ca argument al textului de fereastră din apelul funcției CreateWindow și poate fi modificat cu ajutorul funcției SetWindowText. Procedura de fereastră a ferestrelor statice text utilizează funcția DrawText cu argumentele DT\_WORDBREAK, DT\_NOCLIP și DT\_EXPANDTABS. Textul este introdus în interiorul dreptunghiului ferestrei control. Fundalul textului este COLOR\_WINDOW, iar textul este afișat cu culoarea COLOR\_WINDOWTEXT. Aceste atribute pot fi modificate prin procesarea mesajului WM\_CTLCOLOR.

Ferestrele statice includ și stilul de fereastră SS\_ICON și SS\_USERITEM.

## 8.5 Clasa "scrollbar"

Ferestrele de control copil scroll bar sunt create pe baza clasei predefinite "scrollbar", cu unul din stilurile de fereastră SBS\_HORZ sau SBS\_VERT.

Scroll bar-urile nu trimit mesaje WM\_COMMAND către fereastră părinte. Ferestrele de control scroll bar trimit mesajele WM\_VSCROLL și WM\_HSCROLL cu argumentul lParam egal cu ID-ul de copil. Pentru scroll bar-urile de fereastră (create cu ajutorul stilului WS\_VSCROLL și WS\_HSCROLL combinat cu stilul WS\_OVERLAPPEDWINDOW în argumentul style al funcției CreateWindow) lParam are valoarea 0. Argumentul wParam, ca și la scroll bar-urile de fereastră indică tipul de acțiune realizat cu mouse-ul asupra scroll bar-ului.

Intervalul și poziția ferestrelor scroll bar de control se pot seta cu ajutorul funcțiilor:

**SetScrollRange (hwndScroll, SB\_CTL, nMin, nMax, bRedraw);**

**SetScrollPos (hwndScroll, SB\_CTL, nPos, bRedraw);**

Interiorul scroll bar-urilor este umplut cu culoarea de sistem COLOR\_SCROLLBAR. Butoanele de poziție și de direcție sunt create cu ajutorul culorilor de butoane. Dacă se procesează mesajul WM\_CTLCOLOR se poate întoarce un handle la un brush care să umple interiorul scroll bar-ului.

### 8.5.1 Programul Colors

Programul Colors utilizează trei scroll bar-uri în partea stângă a suprafeței client etichetate "Red", "Green" și "Blue". Domeniul celor trei scroll bar-uri este 0..255. Partea dreaptă a suprafeței client este desenată cu un brush obținut prin compunerea pe componentele R, G și B a valorilor celor trei scroll bar-uri. Valorile numerice conținute de cele trei scroll bar-uri sunt afișate dedesubtul lor.

Programul Colors utilizează 10 ferestre control copil: 3 scroll bar-uri, 6 ferestre cu text static și 1 dreptunghi static. Colors procesează mesajul WM\_CTLCOLORSCROLLBAR pentru a colora interiorul scroll bar-urilor.

Programul Colors nu procesează mesajul WM\_PAINT, desenarea culorii selectate de către scroll bar-uri se face cu ajutorul brush-ului de fundal al ferestrei.

Coordonatele x, y, lățimea și înălțimea ferestrelor copil sunt zero la apelul funcției CreateWindow, deoarece acestea se poziționează în cadrul mesajului WM\_SIZE al ferestrei părinte cu ajutorul funcției MoveWindow.

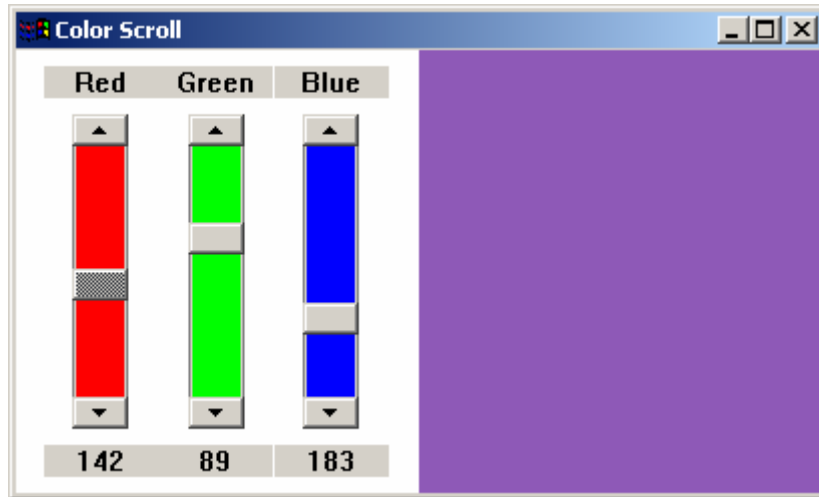


Figura 8.1. Aplicația Colors

Mesajul WM\_VSCROLL conține în argumentul lParam handle-ul de fereastră a scroll bar-ului a cărei poziție a fost modificată. ID-ul de copil al ferestrei respective poate fi citit din structura de fereastră cu ajutorul funcției GetWindowLong:

**n = GetWindowLong ((HWND) lParam, GWL\_ID);**

ID-ul celor trei scroll bar-uri sunt 0, 1, 2 pentru "Red", "Green", "Blue", astfel încât procedura de fereastră poate cunoaște culoarea modificată.

### 8.5.2 Interfață de tastatură a ferestrelor scroll bar de control

Ferestrele scroll bar de control pot procesa și taste în momentul în care au intrare de la tastatură. Mesajele asociate tastelor procesate sunt următoarele:

Tasta	Valoarea argumentului wParam din mesajul de scroll bar
Home	SS_TOP
End	SS_BOTTOM
PageUp	SS_PAGEUP
PageDown	SS_PAGEDOWN
Left sau Up	SS_LINEUP
Right sau Down	SS_LINEDOWN

Pentru ca un scroll bar să poată primi intrarea de la tastatură este necesară includerea stilului WS\_TABSTOP în argumentul style al funcției CreateWindow. Procedura de fereastră părinte trimite focusul de la tastatură când aceasta îl primește către scroll bar-ul curent:

**SetFocus (hwndScroll [nFocus]);**

Trecerea de la un scroll bar la altul utilizând tasta Tab sau Shift+Tab se poate realiza doar printr-o tehnică numită "window subclassing" deoarece scroll bar-urile consumă absolut toate tastele primite, fără a acorda o importanță deosebită decât tastelor de deplasare a cursorului.

### 8.5.3 "Window subclassing"

Procedura de fereastră a scroll bar-urilor este conținută de către Windows. Adresa procedurii de fereastră poate fi obținută cu ajutorul funcției GetWindowLong cu argumentul GWL\_WNDPROC. Este posibil să se introducă adresa altei funcții pe poziția procedurii de fereastră prin apelul funcției SetWindowLong cu argumentul GWL\_WNDPROC. Ca urmare toate mesajele provenite de la fereastra de tipul scroll bar vor fi trimise către funcția introdusă în locul procedurii de fereastră. Aceasta tehnică se numește "Window subclassing" și permite procesarea anumitor mesaje de interes trimise către o fereastră predefinită, iar restul pot fi trimise către vechea procedură de fereastră.

Procedura de fereastră care "preprocesează" mesajele către scroll bar-uri se numește ScrollProc. Funcția ScrollProc este chemată de către Windows, deci este o funcție

CALLBACK.

Pentru fiecare din cele trei scroll bar-uri se citesc apoi adresele procedurilor originale de fereastră și se memorează în variabila `lpfnOldScroll`:

```
lpfnOldScroll = (WNDPROC) GetWindowLong ( hwndScroll [n],  
GWL_WNDPROC);
```

Programul scrie apoi valoarea adresei thunk-ului de reîncărcare a procedurii `ScrollProc` ca adresă a procedurii de fereastră ale celor trei scroll bar-uri:

```
SetWindowLong (hwndScroll [n], GWL_WNDPROC,  
(LONG) lpfnScrollProc);
```

Funcția `ScrollProc` este chemată acum pentru orice mesaj Windows către ferestrele scroll bar. Funcția `ScrollProc` modifică intrarea de tastatură ("input focus") către următorul sau către anteriorul scroll bar dacă au fost apăsat tastele Tab sau Shift+Tab. După care apelează procedura de fereastră originală, prin intermediul funcției `CallWindowProc`:

```
return CallWindowProc (lpfnOldScroll, hwnd, message, wParam, lParam);
```

#### 8.5.4 Desenarea fundalului

`Colors` inițializează brush-ul de fundal cu un brush solid negru. Acesta corespunde poziției inițiale a scroll bar-urilor care au valoarea 0:

```
wndclass->hbrBackground = CreateSolidBrush (RGB (0, 0, 0));
```

Modificarea poziției unuia din scroll bar-uri provoacă trimiterea mesajului `WM_VSCROLL`. Procesarea acestui mesaj se face în funcția `WMVScroll` care creează un nou brush cu culorile curente ale scroll bar-urilor și îl scrie în structura de clasă a ferestrei principale, după care cheamă funcția `DeleteObject` pentru a elibera vechiul brush utilizat pentru desenarea fundalului ferestrei principale:

```
hbrBackgroundOld = (HBRUSH) GetClassLong (hwnd,  
GCL_HBRBACKGROUND);  
hbrBackground = CreateSolidBrush (RGB (color [0], color [1], color [2]));  
SetClassLong (hwnd, GCL_HBRBACKGROUND, (LONG) hbrBackground);  
DeleteObject (hbrBackgroundOld);
```

Redesenarea suprafeței client a ferestrei se va face cu noul brush selectat în câmpul `hbrBackground` al structurii de clasă. Programul forțează redesenarea ferestrei prin apelul funcției `InvalidateRect`:

```
GetClientRect (hwnd, &rcPaint);  
rcPaint.left = rcPaint.right / 2;  
InvalidateRect (hwnd, &rcPaint, TRUE);
```

Valoarea `TRUE` al celui de-al treilea argument al funcției `InvalidateRect` specifică faptul că fundalul trebuie șters cu brush-ul de fundal.

Procedura de fereastră nu procesează mesajul `WM_PAINT`, dar îl trimite către `DefWindowProc`, care realizează procesarea implicită a acestui mesaj. Aceasta se realizează printr-un `BeginPaint` urmat de un `EndPaint`:

```
BeginPaint (hwnd, &ps);  
EndPaint (hwnd, &ps);
```

Apelul funcției `BeginPaint` provoacă trimiterea unui mesaj `WM_ERASEBKGD` către aceeași procedură de fereastră. Mesajul `WM_ERASEBKGD` este, la rândul său, trimis către funcția `DefWindowProc`, care utilizează brush-ul memorat în structura de clasă în câmpul `hbrBackground` pentru a umple întreaga suprafață invalidă (adică jumătatea din dreapta a suprafeței client).

Ca orice obiect GDI, brush-urile create prin apelul funcției `CreateSolidBrush` nu sunt șterse automat la terminarea programului. De aceea, este de datoria programului să șteargă aceste brush-uri (și în general toate obiectele GDI create) în timpul procesării mesajului `WM_DESTROY`. Funcția `DeleteObject` este apelată în cursul acestui mesaj pentru a dealoca brush-ul de fundal al ferestrei principale:

```
hbrBackground = (HBRUSH) GetClassLong (hwnd, GCL_HBRBACKGROUND);  
DeleteObject (hbrBackground);
```



### 8.5.5 Colorarea interiorului scroll bar-urilor

Pe un monitor color, interiorul scroll bar-urilor este roșu, verde respectiv albastru. Aceasta colorare se produce prin intermediul procesării mesajului WM\_CTLCOLORSCROLLBAR.

Funcția WMCreate inițializează trei variabile globale de tipul HBRUSH (handle la brush) cu brush-uri solide conținând culorile fundamentale:

```
HBRUSH hBrush [3];
long WMCreate (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    ....
    hBrush [0] = CreateSolidBrush (RGB (255, 0, 0));
    hBrush [1] = CreateSolidBrush (RGB (0, 255, 0));
    hBrush [2] = CreateSolidBrush (RGB (0, 0, 255));
    return 0;
}
```

Procesarea mesajului WM\_CTLCOLORSCROLLBAR se face prin setarea culorilor normale ale fundalului și caracterelor. Brush-ul întors de către funcția de procesare a mesajului este unul din brush-urile inițializate mai sus:

```
long WMCtlColorScrollBar (HWND hwnd, WPARAM wParam,
                          LPARAM lParam)
{
    HWND hwndChild;
    HDC hdc;
    POINT point;
    short n;

    hdc = (HDC) wParam;
    hwndChild = (HWND) lParam;
    n = GetWindowLong (hwndChild, GWL_ID);
    SetBkColor (hdc, GetSysColor (COLOR_CAPTIONTEXT));
    SetTextColor (hdc, GetSysColor (COLOR_WINDOWFRAME));
    point.x = point.y = 0;
    ClientToScreen (hwnd, &point);
    UnrealizeObject (hBrush [n]);
    SetBrushOrgEx (hdc, point.x, point.y, NULL);
    return ((long) hBrush [n]);
}
return DefWindowProc (hwnd, WM_CTLCOLORSCROLLBAR, wParam,
                      lParam);
}
```

Cele trei brush-uri sunt distruse în timpul procesării mesajului WM\_DESTROY:

```
long WMDestroy (HWND hwnd, WPARAM wParam, LPARAM lParam)
{
    short n;
    ....
    for (n = 0; n < 3; n++)
        DeleteObject (hBrush [n]);
    ....
    return 0;
}
```

## 8.6 Clasa "edit"

Clasa "edit" definește o zonă rectangulară ce permite editarea textelor. Când fereastra copil are input focus, utilizatorul poate să introducă text de la tastatură, să deplaseze caret-ul cu tastele de deplasare a cursorului, să selecteze porțiuni din text (ținând tasta Shift apăsată și acționând tastele de deplasare a cursorului), să le copieze în clipboard

(apăsând tastele Ctrl+Ins sau Ctrl+C), să citească din clipboard informații reprezentate în format text (apăsând tastele Shift+Ins sau Ctrl+V).

Controalele de editare pot fi folosite fie ca ferestre cu o singură linie de text de editare, fie ca ferestre care admit editarea mai multor linii de text.

Programul PopPad folosește o fereastră de control copil din clasa "edit" cu dimensiunea egală cu dimensiunea suprafeței client pentru a realiza un editor rudimentar.

Programul PopPad este un editor pe mai multe linii, în aproximativ 100 de linii de program. Programul nu realizează foarte multe procesări. Controlul predefinit "edit" procesează intrarea de la utilizator și reacționează în mod corespunzător.

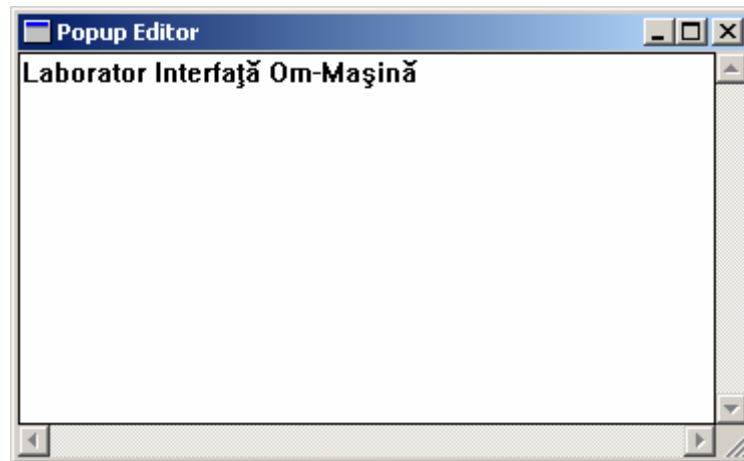


Figura 8.3.

### 8.6.1 Stilurile ferestrelor din clasa "edit"

Ferestrele de control de editare pot fi create cu ajutorul funcției `CreateWindow`, utilizând ca argument al funcției denumirea "edit" a clasei de ferestre. Fereastra utilizează stilul `WS_CHILD` combinat cu mai multe opțiuni. Textul afișat în ferestrele "edit" poate fi aliniat la stânga, dreapta sau centrat, prin utilizarea stilurilor statice `ES_LEFT`, `ES_RIGHT` sau `ES_CENTER`.

Implicit fereastra control de editare are o singură linie. Prin precizarea stilului `ES_MULTILINE` se realizează o fereastră edit pe mai multe linii. Prin utilizarea stilului `ES_AUTOHSCROLL`, poate fi introdus text care să depășească marginea dreaptă a suprafeței client a ferestrei, întreg textul fiind scroll-at automat spre stânga. Începutul unei noi linii se realizează prin apăsarea tastei `Enter`. Stilul `ES_AUTOVSCROLL` permite scrolarea automată pe verticală când, după apăsarea tastei `Enter` se depășește marginea de jos a ferestrei de editare.

Introducerea stilurilor de scolare `WS_HSCROLL` și `WS_VSCROLL` permite utilizarea scroll bar-urilor pentru scrolarea textului din fereastra de editare.

Implicit, o fereastră de editare nu afișează margini. Este posibilă afișarea marginilor ferestrei de editare prin introducerea stilului `WS_BORDER`.

Dacă se selectează o porțiune din textul dintr-o fereastră de editare, acesta este afișat cu culorile de fundal și de text inversate. În momentul pierderii focusului și revenirii focusului către fereastra de editare, fereastra "edit" ascunde în mod implicit selecția de text. Pentru a preîntâmpina acest fenomen, este necesară includerea stilului `ES_NOHIDESEL`.

Valoarea argumentului `style` al funcției `CreateWindow` utilizat în programul PopPad este:

```
WS_CHILD | WS_VISIBLE | WS_HSCROLL | WS_VSCROLL |  
WS_BORDER | ES_LEFT | ES_MULTILINE |  
ES_AUTOHSCROLL | ES_AUTOVSCROLL,
```

Programul PopPad dimensionează fereastra de editare la primirea mesajului `WM_SIZE` de către fereastra principală a aplicației, cu ajutorul funcției `MoveWindow`. Argumentele funcției `MoveWindow` specifică dimensiunile ferestrei de editare egale cu dimensiunea suprafeței client a aplicației:

```
long WMSize (HWND hwnd, WPARAM wParam, LPARAM lParam)  
{
```

```

MoveWindow (hwndEdit, 0, 0, LOWORD (iParam),
            HIWORD (iParam), TRUE);
return 0;
}

```

### 8.6.2 Mesajele de notificare a ferestrelor control "edit"

Ferestrele de control "edit" trimit mesajul WM\_COMMAND către fereastra părinte. Semnificația argumentelor wParam și lParam este similară cu cea din cazul ferestrelor de tip "button":

Argument	Semnificație
LOWORD (wParam)	ID-ul de copil
HIWORD (wParam)	Cod de notificare
lParam	Handle-ul de fereastra a ferestrei copil

Codurile de notificare sunt descrise de către următorii identificatori definiți în Winuser.h:

EN_SETFOCUS	Controlul edit a primit focus-ul.
EN_KILLFOCUS	Controlul edit a pierdut focus-ul.
EN_CHANGE	Conținutul textului din controlul edit va fi modificat.
EN_UPDATE	Conținutul textului din controlul edit a fost modificat.
EN_ERRSPACE	Controlul edit nu mai poate aloca spațiu de memorie.
EN_MAXTEXT	Controlul edit nu mai poate aloca spațiu pentru introducere.
EN_HSCROLL	Controlul edit va fi scroll-at pe orizontală.
EN_VSCROLL	Controlul edit va fi scroll-at pe verticală.

Ferestrele de control edit memorează textul introdus într-un spațiu local de alocare dinamică (în segmentul de date al programului care îl utilizează). Spațiu de memorare este limitat la 32KB.

### 8.6.3 Mesaje către un control edit

Mesajele care pot fi recepționate de către un control edit permit selectarea de text, interogarea coordonatelor textului selectat, copiere în și din clipboard a textului selectat etc.

Mesajele către controlul edit se trimit prin apelul funcției SendMessage:

```

SendMessage (hwndEdit, WM_CUT, 0, 0);
SendMessage (hwndEdit, WM_COPY, 0, 0);
SendMessage (hwndEdit, WM_CLEAR, 0, 0);

```

Mesajul WM\_CUT șterge textul selectat și îl introduce în clipboard. Mesajul WM\_COPY copiază textul selectat în clipboard, iar mesajul WM\_CLEAR șterge textul selectat.

Inserarea conținutului clipboard-ului (în format text) se realizează pe poziția curentă a cursorului cu ajutorul mesajului:

```

SendMessage (hwndEdit, WM_PASTE, 0, 0);

```

Începutul și sfârșitul zonei selectate pot fi obținute cu ajutorul mesajului EM\_GETSEL:

```

SendMessage (hwndEdit, EM_GETSEL, (LPARAM) &iStart,
            (LPARAM) &iEnd);

```

unde iStart reprezintă poziția de început a selecției, iar iEnd poziția de sfârșit plus 1.

Textul într-un edit control poate fi selectat cu ajutorul mesajului EM\_SETSEL:

```

SendMessage (hwndEdit, EM_SETSEL, iStart, iEnd);

```

selectează textul între poziția iBegin și iEnd - 1.

Textul selectat poate fi înlocuit cu un alt șir de caractere utilizând mesajul EM\_REPLACESEL:

```

SendMessage (hwndEdit, EM_REPLACESEL, 0, (LPARAM) lpszString);

```

Mesajul EM\_GETLINECOUNT obține numărul de linii dintr-o fereastra edit multi-linie:

```

nCount = SendMessage (hwndEdit, EM_GETLINECOUNT, 0, 0);

```

Pentru fiecare linie de text, offsetul față de începutul bufferului de editare de text poate fi obținut cu ajutorul mesajului EM\_LINEINDEX:

**nOffset = SendMessage (hwndEdit, EM\_LINEINDEX, iLine, 0);**

Liniile într-o fereastră control edit sunt numerotate începând de la 0. Valoarea -1 a argumentului wLine întoarce offsetul liniei care conține caret-ul. Lungimea unei anumite linii se poate obține folosind mesajul EM\_LINELENGTH:

**nLength = SendMessage (hwndEdit, EM\_LINELENGTH, wLine, 0);**

Orice linie poate fi copiată într-un buffer separat cu ajutorul mesajului EM\_GETLINE:

**nLength = SendMessage (hwndEdit, EM\_GETLINE, wLine,  
(LPARAM) lpszBuffer);**

## Laborator 9 – GDI - Graphic Device Interface

### 9.1 Funcții GDI de desenare

Un sistem grafic necesită cel puțin două funcții: o funcție de desenare de pixel și o funcție de citire a pixel-ului. Teoretic, plecând de la cele două funcții se poate realiza orice operație grafică, oricât de complexă, asupra ecranului. În realitate, este mult mai eficient pentru un sistem grafic să realizeze funcțiile mai complexe (desenare de linii etc.) la nivelul driver-ului de display. Acest lucru realizează o îmbunătățire a performanțelor de timp.

Desigur, nici un sistem grafic nu ar fi însă complet fără cele două funcții de desenare și citire de pixel, de aceea discuția despre funcțiile de desenare GDI va începe de la aceste două funcții.

#### 9.1.1 Desenarea de puncte (pixeli)

Un pixel de o anumită culoare poate fi desenat prin apelul funcției SetPixel:

**rgbActualColor = SetPixel (hdc, x, y, rgbColor);**

Argumentul rgbColor este un întreg lung fără semn (pe 4 octeți), din care cei trei octeți mai puțini semnificativi reprezintă valoarea componentelor roșu, verde și albastru în culoarea dorită. Valoarea rgbColor poate fi construită cu ajutorul macro-ului RGB:

**rgbColor = RGB (nRed, nGreen, nBlue);**

Deși coordonatele x și y sunt exprimate în unități logice, funcția SetPixel desenează un singur pixel fizic indiferent de modul de mapare. Datorită faptului că SetPixel desenează un singur pixel, această funcție nu poate utiliza procedeul de dithering (combinarea mai multor pixeli din culori pure pentru a genera o nuanță intermediară) pentru a afișa culoarea specificată în argumentul de apel. SetPixel aproximează valoarea din argumentul rgbColor la cea mai apropiată culoare pură, pe care o afișează și o întoarce ca rezultat.

Funcția SetPixel este folosită foarte rar în programele Windows. Culoarea pixel-ului aflat la o anumită coordonată pe ecran poate fi obținută cu ajutorul funcției GetPixel:

**rgbColor = GetPixel (hdc, x, y);**

#### 9.1.2 Desenarea de linii

Windows oferă posibilitatea desenării liniilor drepte sau eliptice (segmente din circumferința unei elipse). Cele trei funcții utilizabile pentru desenarea liniilor sunt LineTo și PolyLine (linii drepte) și Arc (linii eliptice). Cinci atribute ale device context-ului afectează aspectul liniilor desenate: poziția curentă a pen-ului (pentru LineTo), pen-ul, modul de desenare a fundalului (pentru pen-uri non-solide), culoarea fundalului (pentru modul de desenare a fundalului OPAQUE) și modul de desenare.

Funcția LineTo desenează o linie între poziția curentă a pen-ului și poziția finală (fără însă a o include) care este specificată în argumentele de apel. În device contextul implicit, poziția inițială a pen-ului este ( 0, 0).

Pentru a desena o linie între punctele de coordonate logice (xStart, yStart) și (xEnd, yEnd) sunt necesare două apeluri de funcții. Primul apel este necesar pentru stabilirea poziției curente a pen-ului:

**MoveToEx (hdc, xStart, yStart, NULL);**

Al doilea apel pentru desenarea efectivă a liniei:

**LineTo (hdc, xEnd, yEnd);**

Linia este desenată până la punctul (fără însă a-l include) de coordonate (xEnd, yEnd). După apelul funcției LineTo, poziția curentă a pen-ului este setată la punctul de coordonate (xEnd, yEnd). LineTo este singura funcție Windows care utilizează poziția curentă

a pen-ului. Funcțiile MoveToEx și LineTo sunt singurele care modifică poziția curentă. Poziția curentă a pen-ului poate fi obținută prin apelul funcției:

```
dwPoint = GetCurrentPositionEx (hdc, &pt);
```

unde pt este o variabilă de tip POINT. Funcția GetCurrentPosition plasează în variabila ps coordonatele poziției curente, care pot fi ulterior aflate cu ajutorul macro-urilor LOWORD și HIWORD.

Următoarea secvență de instrucțiuni desenează un caroiaj spațiat la o distanță de 1 cm pe orizontală și verticală:

```
SetMapMode (hdc, MM_LOMETRIC);  
GetClientRect (hwnd, &rect);  
DPTOLP (hdc, (LPPOINT) &rect, 2);  
for (x = 0; x < rect.right; x += 100)  
{  
    MoveToEx (hdc, x, 0, NULL);  
    LineTo (hdc, x, rect.bottom);  
}  
for (y = 0; y > rect.bottom; y -= 100)  
{  
    MoveToEx (hdc, 0, y, NULL);  
    LineTo (hdc, rect.right, y);  
}
```

Dimensiunile suprafeței client sunt citite în variabila rect și sunt convertite în unități logice prin intermediul funcției DPTOLP. După conversie rect.right conține lățimea suprafeței client în 0.1mm, iar -rect.bottom (semnul negativ se datorează orientării sistemului de axe de coordonate) reprezintă înălțimea în 0.1mm a suprafeței client.

Pentru desenarea unui tablou de puncte se poate utiliza funcția PolyLine:

```
Polyline (hdc, lpPoints, nPoints);
```

Apelul funcției Polyline unește punctele din tabloul de structuri POINT a cărei adresă este trimisă ca argument de apel, și a cărei dimensiune este nPoints. Funcția Polyline trasează nPoints-1 segmente de dreaptă. Dacă primul și ultimul argument al tabloului coincid se obține o figură închisă. Funcția Polyline nu utilizează și nu modifică poziția curentă a pen-ului.

Apelul funcției Arc este următorul:

```
Arc (hdc, xLeft, yTop, xRight, yBottom, xStart, yStart, xEnd, yEnd);
```

Funcția Arc desenează un arc de elipsă. Elipsa este încadrată de dreptunghiul de coordonate (xLeft, yTop), (xRight, yBottom). Arcul este desenat în sens trigonometric începând de la punctul de intersecție al elipsei cu dreapta care unește punctul (xStart, yStart) cu centrul elipsei și până la punctul de intersecție al elipsei cu dreapta care unește punctul (xEnd, yEnd) cu centrul elipsei. Modul de desenare al arcului este prezentat în figura 9.1.

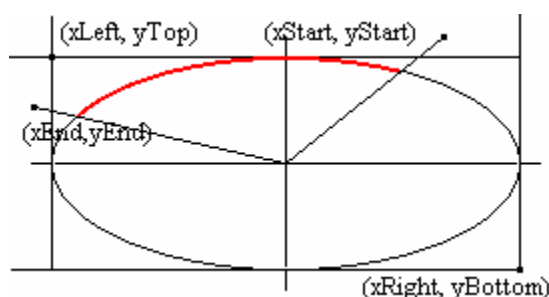


Figura. 9.1.

### 9.1.3 Utilizarea pen-urilor predefinite (Stock Pens)

Funcțiile LineTo, Polyline și Arc utilizează pen-ul curent selectat în device context-ul utilizat. Caracteristicile pen-ului determină culoarea, grosimea și textura liniei (solidă, punctată, întreruptă). Pen-ul utilizat în device context-ul implicit se numește BLACK\_PEN și este un pen predefinit furnizat de Windows. Celelalte două pen-uri predefinite sunt WHITE\_PEN și NULL\_PEN.

În aplicațiile Windows, pen-urile sunt utilizate prin handle-uri. Windef.h definește tipul HPEN care este un handle la pen. O variabilă de tipul handle la pen poate fi definită prin:

**HPEN hPen;**

Un handle către un pen predefinit poate fi obținut prin apelul funcției GetStockObject. Apelul:

**hPen = GetStockObject (WHITE\_PEN);**

obține un pen predefinit de culoare albă, grosime un pixel și textura solidă. Pentru că pen-ul astfel obținut să devină pen-ul curent selectat în device context-ul de desenare (deci utilizat de către funcțiile LineTo etc.) este necesar apelul funcției SelectObject:

**SelectObject (hdc, hPen);**

După acest apel, liniile desenate de către funcțiile LineTo, PolyLine și Arc vor utiliza WHITE\_PEN până în momentul în care un alt pen este selectat în device context-ul de desenare.

Funcția SelectObject întoarce handle-ul către pen-ul care era selectat în device context, înainte de apelul funcției:

**hPenOld = SelectObject (hdc, GetStockObject (WHITE\_PEN));**

#### 9.1.4 Crearea, selectarea și eliberarea pen-urilor

Pentru a utiliza un pen cu caracteristicile definite de către utilizator sunt necesare următoarele acțiuni: Se obține un handle de pen utilizând funcțiile CreatePen sau CreatePenIndirect. Se selectează handle-ul de pen în device context prin apelul funcției SelectObject. În acest moment liniile desenate vor utiliza noul pen. După eliberarea device context-ului (sau după selectarea altui pen în device context), aplicația trebuie să elibereze pen-ul prin apelul funcției DeleteObject.

Un pen este un obiect GDI, de aceea aplicația poate să utilizeze pen-ul, dar pen-ul nu aparține programului. Pen-ul aparține efectiv modulului GDI, din acest motiv pen-ul (ca oricare obiect GDI) nu este eliberat automat în momentul închiderii programului, el trebuie să fie eliberat explicit de către aplicația care l-a creat.

Pentru a utiliza obiecte GDI (pen-uri, brush-uri, bitmap-uri, regiuni, font-uri, și palete) programul trebuie să respecte trei reguli:

- Programul trebuie să elibereze explicit toate obiectele GDI create.
- Nu este permisă ștergerea obiectelor GDI selectate într-un device context valid.
- Nu este permisă ștergerea obiectelor GDI predefinite.

Funcția CreatePen are următoarea formă de apel:

**hPen = CreatePen (nPenStyle, nWidth, rgbColor);**

Argumentul nPenStyle specifică textura liniei desenate. Argumentul poate lua una din valorile definite prin identificatorii PS\_SOLID, PS\_DASH, PS\_DOT, PS\_DASHDOT, PS\_DASHDOTDOT, PS\_NULL sau PS\_INSIDEFRAME. Stilurile de textură rezultate sunt prezentate în figura 9.2.

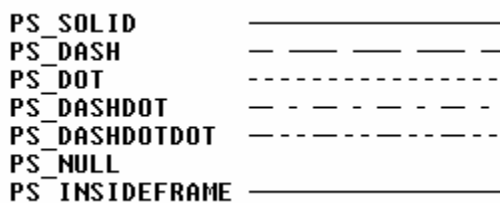


Figura. 9.2.

Pentru stilurile de pen PS\_DASH, PS\_DOT, PS\_DASHDOT și PS\_DASHDOTDOT spațiul între segmentele liniilor este colorat cu culoarea fundalului (dacă modul de desenare al fundalului este OPAQUE). Dacă modul de desenare al fundalului este TRANSPARENT, spațiile dintre segmentele de linie rămân nemodificate.

Pentru stilurile PS\_SOLID, PS\_NULL și PS\_INSIDEFRAME argumentul nWidth specifică grosimea liniei în unități logice. Dacă valoarea acestui argument este 0, linia este desenată cu o grosime de un pixel, indiferent de modul de mapare. Pentru toate celelalte stiluri de pen, acest argument nu poate lua decât valoarea 0.

Argumentul rgbColor specifică culoarea pen-ului. Pentru toate stilurile în afară de PS\_INSIDEFRAME, Windows convertește această valoare la cea mai apropiată culoare pură

din device context-ul în care este selectat pen-ul. PS\_INSIDEFRAME poate utiliza o culoare non-pură (obținută prin efectul de dithering) și aceasta doar dacă dimensiunea sa este mai mare decât 0.

Pen-urile pot fi create și prin inițializarea unei structuri de tipul LOGPEN – logical pen, care este apoi trecută funcției CreatePenIndirect. O variabilă de tipul LOGPEN se declară prin:

**LOGPEN      logpen;**

Structura LOGPEN conține trei câmpuri: lopnStyle (UINT) reprezintă stilul pen-ului, lopnWidth (POINT) este dimensiunea pen-ului în unități logice, iar câmpul lopnColor (COLORREF) este culoarea pen-ului. Argumentul lopnWidth este o structură de tipul POINT, Windows folosește însă doar câmpul lopnWidth.x, valoarea lopnWidth.y este ignorată. Pen-ul poate fi creat trecând ca argument al funcției CreatePenIndirect adresa variabilei structură LOGPEN:

**hPen = CreatePenIndirect (&logpen);**

Pen-urile create prin intermediul funcțiilor CreatePen sau CreatePenIndirect trebuie selectate în device context pentru ca funcțiile de desenare de linii să utilizeze caracteristicile pen-ului respectiv. Selectarea în device context-ul de desenare se face prin apelul funcției SelectObject:

**hPenOld = SelectObject (hPen);**

După utilizarea pen-ului programul trebuie să elibereze pen-ul creat. Eliberarea pen-ului se realizează prin funcția DeleteObject:

**DeleteObject (hPen);**

Funcția DeleteObject nu poate fi apelată decât doar dacă pen-ul nu este selectat în device context (adică fie după selectarea unui alt pen în device context sau după eliberarea device context-ului).

## 9.2 Modurile de desenare

Aspectul liniilor pe ecran este afectat de modul de desenare al device context-ului. În general, culoarea unei linii afișate pe ecran depinde nu numai de culoarea pen-ului selectat ci și de culoarea zonei peste care se desenează.

Culoarea fiecărui pixel din interiorul unei linii afișate pe ecran este obținută printr-o operație logică între valoarea originală a culorii pixel-ului peste care se desenează și culoarea pen-ului. Operația logică se numește "raster operation" sau "ROP". Deoarece desenarea unei linii nu implică decât doi operanzi (pen-ul și fundalul), operația logică este denumită și "raster operation" binara sau "ROP2". Windows definește în total 16 operații logice (moduri de combinare a culorii fundalului cu culoarea pen-ului). Valoarea implicită a modului de desenare este R2\_COPYPEN, ceea ce semnifică faptul că Windows copiază pixelii pen-ului pe ecran. În total există 16 coduri de ROP2. Numărul de coduri ROP2 este egal cu numărul operatorilor logici care aplicați asupra a două variabile logice dau rezultate diferite.

Pen (P):	1	1	0	0	Operația	Modul de desenare
Destinație (D):	1	0	1	0	Logica	
Rezultat (R):	0	0	0	0	$R = 0$	R2_BLACK
	0	0	0	1	$R = \sim(P \mid D)$	R2_NOTMERGEPEN
	0	0	1	0	$R = \sim P \& D$	R2_MASKNOTPEN
	0	0	1	1	$R = \sim P$	R2_NOTCOPYPEN
	0	1	0	0	$R = P \& \sim D$	R2_MASKPENNOT
	0	1	0	1	$R = \sim D$	R2_NOT
	0	1	1	0	$R = P \wedge D$	R2_XORPEN
	0	1	1	1	$R = \sim(P \& D)$	R2_NOTMASKPEN
	0	0	0	0	$R = P \& D$	R2_MASKPEN
	0	0	0	1	$R = \sim(P \wedge D)$	R2_NOTXORPEN
	1	0	1	0	$R = D$	R2_NOP
	1	0	1	1	$R = \sim P \mid D$	R2_MERGEENOTPEN
	1	1	0	0	$R = P$	R2_COPYPEN
	1	1	0	1	$R = P \mid \sim D$	R2_MERGEENNOT
	1	1	1	0	$R = P \mid D$	R2_MERGEEN
	1	1	1	1	$R = 1$	R2_WHITE



Tabelul anterior prezintă codurile ROP2 împreună cu rezultatul aplicării lor asupra combinației între culoarea pen-ului (P) și culoarea suprafeței de desenare (sau destinația, D). S-au luat doar cazurile de Pen alb (valoare 1) sau negru (valoare 0) și suprafața de desenare albă (valoare 1) sau neagră (valoare 0).

Modul de desenare este selectat în device context prin apelul funcției SetROP2:

**SetROP2 (hdc, nDrawMode);**

Argumentul nDrawMode are una din valorile listate în coloana "Modul de desenare" a tabelului. Valoarea modului curent de desenare poate fi obținută prin apelul funcției:

**nDrawMode = GetROP2 (hdc);**

Valoarea din device context-ul implicit a modului de desenare este R2\_COPYPEN, ceea ce semnifică faptul că pen-ul va fi copiat direct în suprafața de desenare. Modul R2\_NOT inversează culoarea suprafeței de desenare. Acest mod este util când nu se cunoaște culoarea fundalului pe care se desenează, deoarece el garantează că desenul va fi întotdeauna vizibil. Aplicarea consecutivă a două linii delimitate de aceleași coordonate lasă culoarea fundalului nemodificată.

### 9.2.1 Programul Rop2Look

Programul Rop2Look afișează efectul aplicării celor 16 moduri de desenare. Programul desenează pe un fundal împărțit pe verticală în cinci suprafețe colorate cu alb, gri deschis, gri, gri închis și negru cu ajutorul unor brush-uri predefinite. Fiecare secțiune orizontală folosește un mod de desenare diferit, care este specificat în partea stângă a ferestrei. În partea superioară a fiecărei secțiuni este desenată o linie cu un pen alb, iar în partea inferioară linia este desenată cu un pen black. Rezultatul programului este prezentat în figura 9.3.

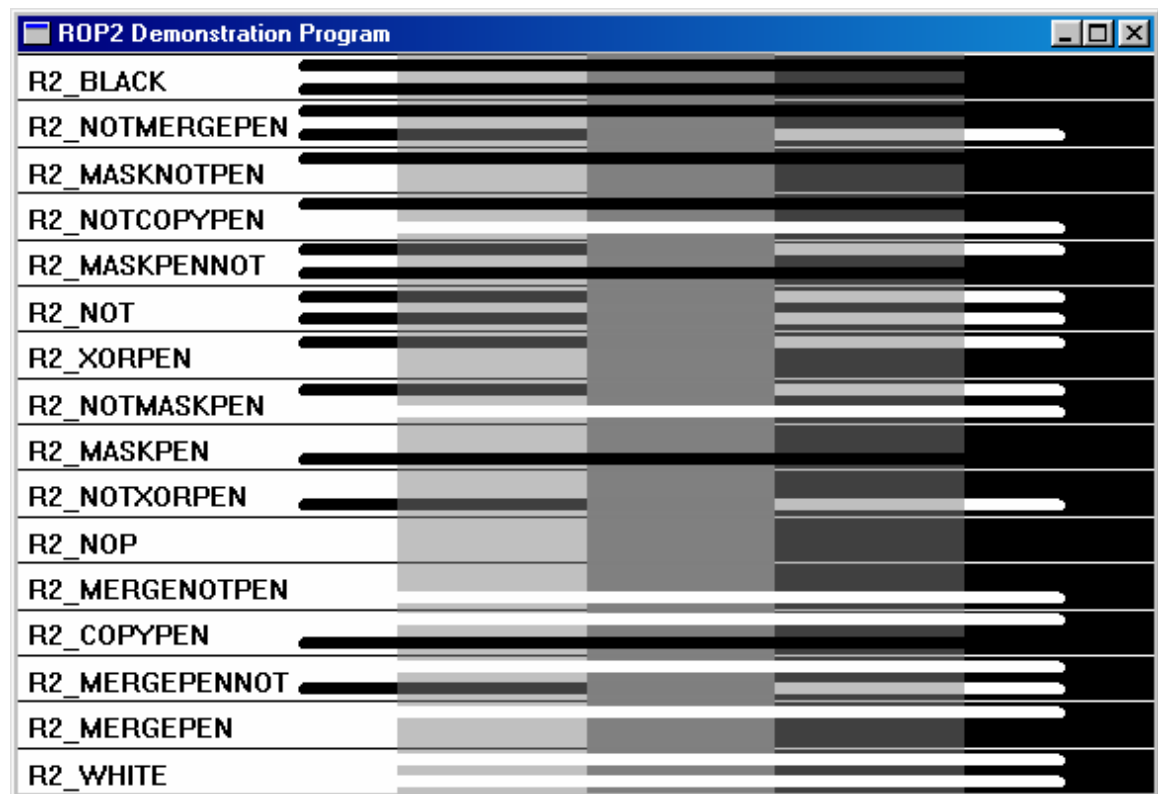


Figura. 9.3. Programul Rop2Look

### 9.2.2 Modul de desenare și culorile

În cazul monitoarelor color, operațiile ROP2 afectează separat fiecare plan de culoare. De exemplu, dacă culoarea fundalului este galben (roșu = 255, verde = 255, albastru = 0), culoarea pen-ului este verde (roșu = 0, verde = 255, albastru = 0), iar modul de desenare este R2\_XORPEN, linia desenată va avea culoarea roșie (roșu = 255, verde = 0, albastru = 0).

### 9.3 Desenarea figurilor închise

Windows oferă cinci funcții de desenare ale curbelor închise:

Funcția	Figura
Rectangle	Dreptunghi cu colțurile drepte.
Ellipse	Elipsă.
RoundRect	Dreptunghi cu colțurile rotunde.
Chord	Arc pe perimetrul unei elipse cu extremitățile conectate printr-o secantă.
Pie	Arc pe perimetrul unei elipse cu extremitățile conectate prin două raze.
Polygon	Poligon.
PolyPolygon	Poligoane multiple.

Windows desenează conturul figurii cu ajutorul pen-ului curent din device context-ul utilizat și utilizând modul de desenare curent, modul de desenare al fundalului și culoarea curentă a fundalului, similar cu desenarea unei linii obișnuite.

Corpul figurii este colorat cu brush-ul curent selectat în device context. Brush-ul implicit al device context-ului este WHITE\_BRUSH, adică interiorul figurii va fi umplut cu alb. Windows oferă șase brush-uri predefinite: WHITE\_BRUSH, LTGRAY\_BRUSH, GRAY\_BRUSH, DKGRAY\_BRUSH, BLACK\_BRUSH și NULL\_BRUSH (sau HOLLOW\_BRUSH). Primele cinci brush-uri predefinite au fost utilizate în programul Rop2Look.

Brush-urile pot fi selectate în device context în același mod în care se selectează pen-urile. Windows definește tipul de date HBRUSH pentru un handle la un brush. Declararea unei variabile de tipul handle la brush se face prin:

```
HBRUSH hBrush;
```

Un handle la GRAY\_BRUSH (de exemplu) poate fi obținut prin:

```
hBrush = GetStockObject (GRAY_BRUSH);
```

Handle-ul astfel obținut poate fi acum selectat în device context:

```
hOldBrush = SelectObject (hdc, hBrush);
```

Funcția SelectObject întoarce handle-ul brush-ului care fusese anterior selectat în device context-ul respectiv. Ulterior apelului funcției SelectObject toate figurile închise vor fi colorate cu ajutorul brush-ului selectat (adică GRAY\_BRUSH în exemplul nostru).

Dacă se dorește desenarea figurilor fără un contur este necesară selectarea pen-ului predefinit NULL\_PEN în device context:

```
SelectObject (hdc, GetStockObject (NULL_PEN));
```

sau se utilizează modul de desenare R2\_NOP:

```
SetROP2 (hdc, R2_NOP);
```

Dacă se dorește doar desenarea conturului figurii, fără să se coloreze interiorul său, se poate selecta brush-ul NULL\_PEN în device context-ul de desenare:

```
SelectObject (hdc, GetStockObject (NULL_BRUSH));
```

#### 9.3.1 Dreptunghiul de încadrare

Funcțiile Rectangle, Ellipse, RoundRect, Chord și Pie (ca și funcția Arc) utilizează un "dreptunghi de încadrare". Argumentele funcțiilor definesc coordonatele unui dreptunghi, în interiorul căruia vor fi desenate figurile.

Cea mai simplă figură închisă este dreptunghiul, desenul unui dreptunghi se face prin apelul:

```
Rectangle (hdc, xLeft, yTop, xRight, yBottom);
```

Punctul de coordonate (xLeft, yTop) reprezintă colțul stânga sus al dreptunghiului, iar (xRight, yBottom) este colțul dreapta jos. Coordonatele sunt exprimate în unități logice.

Funcția Ellipse utilizează aceleași argumente ca și funcția Rectangle:

```
Ellipse (hdc, xLeft, yTop, xRight, yBottom);
```

Rezultatul acestei funcții este o elipsă încadrată de dreptunghiul de coordonate (xLeft, yTop), (xRight, yBottom).

Windows nu conține funcții de desenare de pătrate sau cercuri, aceste figuri pot fi desenate ușor utilizând funcțiile Rectangle sau Ellipse în modurile de mapare izotropice utilizând dimensiuni orizontale și verticale egale pentru dreptunghiul de încadrare. Pentru modul de mapare MM\_TEXT este necesară utilizarea funcției GetDeviceCaps cu argumentele ASPECTX și ASPECTY pentru a obține factorii de scalare pe orizontală și verticală, care apoi vor fi aplicați coordonatelor orizontale și verticale ale dreptunghiului de încadrare.

### 9.3.2 Brush-uri

Interiorul figurilor închise sunt umplute cu brush-ul curent selectat în device context-ul de desenare. Un brush este o imagine șablon de 8 pixeli pe 8 pixeli care se repetă atât pe orizontală cât și pe verticală pentru a umple interiorul figurii.

Windows pune la dispoziție patru funcții pentru crearea brush-uri logice. Brush-ul este selectat în device context prin intermediul funcției SelectObject. Ca și pen-urile, brush-urile sunt obiecte GDI, deci regulile enunțate pentru crearea, selectarea și eliberarea pen-urilor se aplică și brush-urilor. Orice brush creat trebuie eliberat prin intermediul funcției DeleteObject, dar nu atât timp cât acesta este selectat într-un device context valid.

Prima funcție utilizată pentru crearea unui brush logic este CreateSolidBrush:

**hBrush = CreateSolidBrush (rgbColor);**

Brush-ul rezultat prin apelul funcției CreateSolidBrush este un brush cu textură solidă și culoarea specificată prin argumentul rgbColor.

Este posibilă realizarea de brush-uri cu textura hașurată. Funcția CreateHatchBrush are următoarea formă de apel:

**hBrush = CreateHatchBrush (nHatchStyle, rgbColor);**

unde argumentul nHatchStyle definește stilul de hașură. Argumentul poate lua una din valorile definite prin identificatorii HS\_HORIZONTAL, HS\_VERTICAL, HS\_FDIAGONAL, HS\_BDIAGONAL, HS\_CROSS sau HS\_DIAGCROSS definiți în Windows.h. Rezultatul utilizării diferiților identificatori, ca valoarea a argumentului nHatchStyle, este prezentat în figura. 9.4. Argumentul rgbColor specifică culoarea de desenare a liniilor de hașură. Spațiul dintre hașuri este desenat în funcție de modul de desenare al fundalului și de culoarea fundalului.

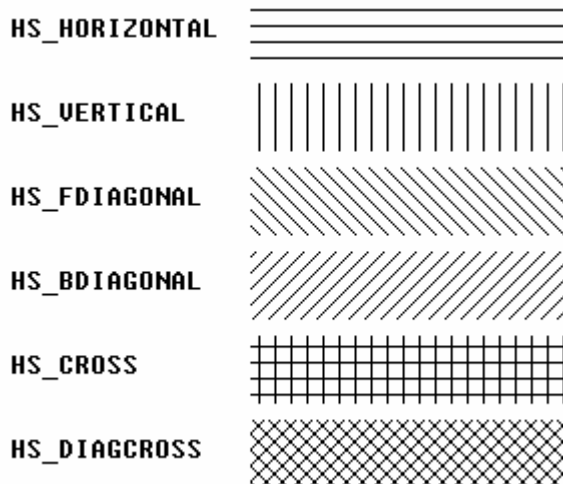


Figura. 9.4.

Programele Windows pot utiliza funcția CreatePatternBrush pentru a utiliza propriul șablon pentru brush-ul rezultat. Apelul funcției se face prin:

**hBrush = CreatePatternBrush (hBitmap);**

unde argumentul hBitmap este un handle la un bitmap de dimensiune 8 pe 8 pixeli.

Windows conține o funcție care include toate cele trei funcții listate anterior. Funcția se numește CreateBrushIndirect și utilizează ca argument o referință la o structură de date denumită LOGBRUSH - logical brush. Cele trei câmpuri ale structurii LOGBRUSH sunt prezentate în tabelul următor:

<b>lbStyle (WORD)</b>	<b>lbColor (DWORD)</b>	<b>lbHatch (short)</b>
BS_SOLID	Culoarea brush-ului	fără semnificație
BS_HOLLOW	fără semnificație	fără semnificație
BS_HATCHED	Culoarea hașurilor	Stilul de hașură (HS_)
BS_PATTERN	fără semnificație	Handle-ul către bitmap

Funcțiile `SelectObject` (pentru selectarea în device context) și `DeleteObject` (pentru eliberarea obiectului GDI) pot fi utilizate la obiectele de tip brush în mod similar cum au fost utilizate la pen-uri. Odată ce programul deține un handle la un brush, acest brush poate fi selectat în device context pentru a fi utilizat la desenare prin:

**`hOldBrush = SelectObject (hdc, hBrush);`**

După eliberarea device context-ului sau selectarea altui brush în device context, handle-ul brush-ului utilizat trebuie să fie eliberat prin `DeleteObject`:

**`DeleteObject (hBrush);`**

### 9.3.3 Alinierea brush-urilor

Când Windows umple o suprafață cu un brush, șablonul de 8 x 8 pixeli care definește brush-ul este repetat atât pe orizontală cât și pe verticală. Aspectul suprafeței de umplere poate să difere în funcție de poziția relativă a originii bitmap-ului șablon și a suprafeței de umplere. Atributul din device context care determină această aliniere se numește "brush origin" și este exprimat întotdeauna în coordonate ecran (pixeli față de colțul stânga sus al monitorului). Pentru device context-ul obținut prin `GetDC` sau `BeginPaint`, originea brush-ului se identifică cu originea sistemului de coordonate client relativ la sistemul de coordonate ecran.

Punctul din stânga sus al bitmap-ului de 8 x 8 pixeli utilizat pentru definirea unui brush coincide implicit cu coordonate multiplu de 8 în sistemul fizic de coordonate client. Realinierea bitmap-ului utilizat pentru definirea unui brush se realizează prin următoarea secvență de instrucțiuni:

1. Se apelează funcția `UnrealizeObject` pentru brush-ul respectiv (`UnrealizeObject` nu poate fi apelat pentru brush-uri predefinite).
2. Se setează originea brush-ului prin apelul funcției `SetBrushOrg`, în coordonate ecran.
3. Se selectează brush-ul realiniat în device context.

## 9.4 Dreptunghiuri, regiuni și clipping

O regiune este o suprafață din ecran care este o combinație de dreptunghiuri, poligoane și elipse. Windows oferă o serie de funcții care utilizează pentru desenare aceste obiecte GDI.

### 9.4.1 Funcții de desenare care utilizează dreptunghiuri

Următoarele trei funcții de desenare utilizează argumente de tip referință către o structură de tip `RECT`:

**`FillRect (hdc, &rect, hBrush);`**  
**`FrameRect (hdc, &rect, hBrush);`**  
**`InvertRect (hdc, &rect);`**

Câmpurile structurii `rect` sunt `left`, `top`, `right` și `bottom` și reprezintă coordonatele în valori logice ale dreptunghiului care va fi desenat.

Funcția `FillRect` colorează dreptunghiul cu brush-ul specificat. Aceasta funcție nu necesită selectarea unui brush în device context-ul de desenare, handle-ul la brush fiind specificat ca argument de apel.

Funcția `FrameRect` permite desenarea conturului unui dreptunghi cu ajutorul brush-ului specificat ca argument de apel. Grosimea conturului desenat va fi de o unitate logică a device context-ului de desenare. Funcția `FrameRect` permite desenarea conturului cu ajutorul unui brush, spre deosebire de funcția `Rectangle` care utilizează pentru desenarea conturului dreptunghiului pen-ul curent al device context-ului de desenare.

Funcția `InvertRect` inversează culorile pixelilor din interiorul dreptunghiului specificat.

Windows conține de asemenea un număr de nouă funcții pentru manipularea câmpurilor unei structuri de tipul `RECT`. De exemplu, inițializarea câmpurilor unei structuri

RECT poate fi realizata fie prin:

```
rect.left = xLeft;
rect.top = yTop;
rect.right = xRight;
rect.bottom = yBottom;
```

sau direct prin apelul funcției SetRect, care are același efect:

```
SetRect (&rect, xLeft, yTop, xRight, yBottom);
```

Celelalte opt funcții sunt listate în tabelul următor.

Funcția	Efect
OffsetRect (&rect, x, y);	Deplasează dreptunghiul cu x, y unități.
InflateRect (&rect, x, y);	Modifică dimensiunea dreptunghiului cu x, y unități
SetRectEmpty (&rect);	Șterge câmpurile variabilei rect cu valoarea 0.
CopyRect (&DestRect, &SrcRect);	Copiază valorile câmpurilor SrcRect în DestRect.
IntersectRect (&DestRect, &SrcRect1, &SrcRect2);	Obține dreptunghiul de intersecție al SrcRect1 cu SrcRect2.
UnionRect (&DestRect, &SrcRect1, &SrcRect2);	Obține dreptunghiul minim care cuprinde SrcRect1 și SrcRect2.
bEmpty = IsRectEmpty (&rect);	Determină dacă dreptunghiul este vid.
BInRect = PtInRect (&rect, point);	Determină dacă punctul de coordonate point se află în interiorul dreptunghiului rect.

Codul echivalent al acestor funcții este relativ simplu, el realizând doar manipulări ale coordonatelor structurilor rect.

#### 9.4.2 Crearea și desenarea regiunilor

Regiunea este descrierea unei suprafețe ca o combinație de dreptunghiuri, poligoane și elipse. Regiunile pot fi utilizate pentru desenare și clipping (constrângerea desenării doar în interiorul unei anumite regiuni). Utilizarea unei regiuni pentru clipping se realizează prin selectarea sa în device context-ul de desenare.

Ca și pen-urile și brush-urile, regiunile sunt obiecte GDI. Regiunile trebuie deci utilizate în același mod în care se utilizează orice obiect GDI. Programul va șterge orice regiune pe care o creează, dar nu atât timp cât ea este selectată într-un device context valid.

Când o aplicație Windows creează o regiune, Windows întoarce o valoare de tipul HRGN (un handle către regiune). Declararea unei variabile de tip HRGN se face prin:

```
HRGN hRgn;
```

Crearea unei regiuni rectangulare se face prin apelul funcției CreateRectRgn:

```
hRgnRect = CreateRectRgn (xLeft, yTop, xRight, yBottom);
```

sau prin apelul funcției CreateRectRgnIndirect:

```
hRgnRect = CreateRectRgnIndirect (&rect);
```

Regiunile sunt exprimate întotdeauna în coordonate fizice (pixeli). Regiunile eliptice pot fi create prin apelul funcției CreateEllipticRgn:

```
hRgnElliptic = CreateEllipticRgn (xLeft, yTop, xRight, yBottom);
```

sau prin CreateEllipticRgnIndirect:

```
hRgnElliptic = CreateEllipticRgnIndirect (&rect);
```

Funcția CombineRgn permite combinarea a două regiuni și depunerea rezultatului într-o regiune destinație. Regiunea destinație trebuie să fi fost anterior creată printr-una din funcțiile anterioare. Apelul funcției CombineRgn se face prin:

```
hRgnType = CombineRgn (hDestRgn, hSrcRgn1,
                        hSrcRgn2, nCombine);
```

Argumentul nCombine specifică modul de combinare al celor două regiuni. El poate lua una din valorile listate în tabelul de mai jos:

nCombine	Modul de combinare
RGN_AND	Suprafața comună a celor două regiuni.
RGN_OR	Suma suprafețelor celor două regiuni.
RGN_XOR	Suma suprafețelor celor două regiuni excluzând suprafața comună.

RGN_DIFF	Suprafața din hSrcRgn1 din care se extrage suprafața comună cu hSrcRgn2.
RGN_COPY	Suprafața primei regiuni.

Rezultatul funcției CombineRgn poate fi: NULLREGION, dacă regiunea rezultată este vidă; SIMPLEREGION, dacă regiunea rezultată este dreptunghiulară, COMPLEXREGION dacă regiunea rezultată este o combinație de dreptunghiuri, elipse sau poligoane și ERROR, dacă GDI nu a avut spațiu suficient de memorie pentru a memora regiunea.

Handle-ul de regiune poate fi utilizat împreună cu 4 funcții de desenare:

Funcția	Efect
FillRgn (hdc, hRgn, hBrush);	Colorează regiunea cu brush-ul specificat.
FrameRgn (hdc, hRgn, hBrush, xFrame, yFrame);	Desenează un contur de dimensiune xFrame, yFrame exprimați în unități logice și cu brush-ul specificat.
InvertRgn (hdc, hRgn);	Inversează culoarea pixelilor din interiorul regiunii.
PaintRgn (hdc, hRgn);	Colorează regiunea cu brush-ul selectat în device context.

O regiune creată de aplicația Windows trebuie eliberată după ce a fost utilizată (ca orice obiect GDI). Eliberarea regiunilor se face prin apelul funcției DeleteObject:

**DeleteObject (hRgn);**

#### 9.4.3 Dreptunghiuri și regiuni de clipping

Device context-ul conține un atribut care definește regiunea de clipping utilizator. Regiunea de clipping a unui device context este intersecția dintre regiunea de clipping utilizator și regiunea vizibilă a ferestrei. Un program care folosește device context-ul pentru desenare nu poate să afecteze zonele din suprafața client din exteriorul regiunii de clipping. Regiunea de clipping utilizator a device context-ului obținut prin funcția BeginPaint este egală cu regiunea invalidă a suprafeței client. Regiunea de clipping utilizator a device context-ului obținut prin funcția GetDC este egală cu întreaga suprafață client. Windows oferă funcții de modificare a regiunii de clipping utilizator cât și a regiunii invalide.

Funcțiile care acționează asupra regiunii invalide a unei ferestre (regiunea care necesită redesenare la primirea mesajului WM\_PAINT) sunt:

**InvalidateRgn (hwnd, hRgn, bErase);**

adaugă regiunea specificată prin hRgn la regiunea invalidă. Argumentul bErase specifică dacă se dorește ștergerea fundalului (adică trimiterea mesajului WM\_ERASEBKGD) ca rezultat al apelării funcției BeginPaint. Funcția:

**ValidateRgn (hwnd, hRgn);**

scade din regiunea invalidă regiunea specificată de argumentul hRgn. Dacă funcțiile sunt utilizate înainte de apelul funcției BeginPaint, regiunea de clipping a device context-ului întors de BeginPaint va fi egală cu regiunea invalidă rezultată, intersectată cu zona din suprafața client vizibilă.

Funcțiile SelectObject și SelectClipRgn permit modificarea regiunii de clipping utilizator a device context-ului:

**hRgnClipOrg = SelectObject (hdc, hRgn);**

**nRgnType = SelectClipRgn (hdc, hRgn);**

Regiunea de clipping utilizator rezultată este regiunea specificată prin hRgn. Funcția SelectObject întoarce un handle către regiunea de clipping utilizator anterioară a device context-ului, iar funcția SelectClipRgn întoarce tipul regiunii hRgn. Rezultatul poate fi NULLREGION, SIMPLEREGION, COMPLEXREGION sau ERROR. Cele două funcții lucrează pe o copie a regiunii trimise ca argument de apel, deci această regiune poate fi refolosită.

#### 9.4.4 Programul Clover

Programul Clover desenează o figură în formă de trifoi prin combinarea a patru regiuni eliptice și selectarea regiunii rezultate ca regiune de clipping utilizator în timpul procesării mesajului WM\_PAINT. După selectarea regiunii de clipping, în mesajul WM\_PAINT se desenează o serie de raze din grad în grad pe un cerc de diametru egal cu diagonala

suprafeței client. Datorită regiunilor de clipping, figura desenată va arăta ca în figura. 9.5.

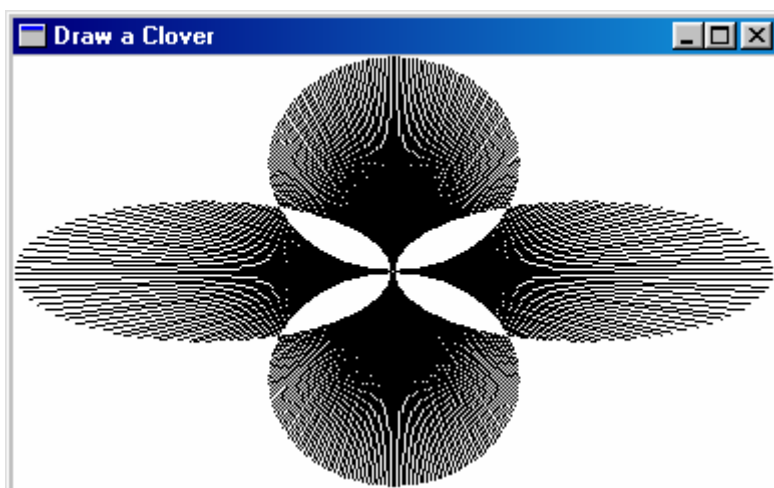


Figura. 9.5. Programul Clover

Regiunile eliptice și poligonale sunt obiecte GDI care ocupă un spațiu semnificativ de memorie. Din acest motiv, programul Clover testează dacă funcțiile de combinare a regiunilor întorc valoarea ERROR, caz în care se afișează un mesaj de eroare.

Datorită faptului că coordonatele regiunilor sunt exprimate în unități fizice (pixeli), programul creează regiunile și le combina în regiunea de clip pe parcursul procesării mesajului WM\_SIZE.

În timpul procesării mesajului WM\_DESTROY, regiunea de clipping creată în mesajul WM\_SIZE este eliberată.

## 9.5 Programe care desenează continuu

O metodă de a structura diferit bucla de mesaje a unei aplicații Windows se bazează pe utilizarea funcției PeekMessage în locul funcției GetMessage. Funcția PeekMessage (ca și GetMessage) realizează comutarea de taskuri dacă aceasta este necesar, dar față de GetMessage, funcția PeekMessage se întoarce dacă una din următoarele două condiții este adevărată:

- Când există un mesaj în coada de mesaje a aplicației, caz în care PeekMessage întoarce valoarea TRUE. Această condiție este similară cu condiția de întoarcere a funcției GetMessage. În plus, PeekMessage se întoarce și dacă:
- Nu există mesaje atât în coada de mesaje a aplicației cât și în nici o altă coadă de mesaje a oricărei aplicații care se execută în momentul respectiv. Aceasta condiție este echivalentă cu momentul când Windows este "în așteptare". Funcția PeekMessage întoarce în acest caz valoarea FALSE.

Semnificația unei bucle de mesaje care utilizează PeekMessage este următoarea: cât timp oricare aplicație din Windows are mesaje de procesat se vor procesa mesajele respective, dar în momentul în care aplicațiile sunt în așteptare (cozile de mesaje sunt vide), este permisă realizarea de acțiuni ale programului respectiv.

Modul de apel al funcției PeekMessage este următorul:

**bMessage = PeekMessage (&msg, NULL, 0, 0, PM\_REMOVE);**

Primele patru argumente ale funcției PeekMessage sunt identice cu argumentele corespunzătoare ale funcției GetMessage (adică adresa unei variabile de tip MSG, în care se depune informația referitoare la mesajul recepționat, dacă acesta există; handle-ul de fereastră pentru care se dorește extragerea mesajului sau NULL dacă se dorește extragerea mesajului pentru toate ferestrele din aplicația respectivă; intervalul între care se dorește extragerea mesajelor sau 0, 0 pentru extragerea tuturor mesajelor din coada de mesaje a aplicației).

Cel de-al cincilea argument al funcției PeekMessage indică dacă aplicația dorește extragerea mesajului din coada de mesaje a aplicației. Dacă valoarea acestui argument este

PM\_REMOVE, mesajul este scos din coada de mesaje. Dacă valoarea argumentului este PM\_NOREMOVE, mesajul nu va fi scos din coada de mesaje a aplicației.

Valoarea întoarsă de funcția PeekMessage este o valoare logică. PeekMessage întoarce TRUE dacă există cel puțin un mesaj în coada de mesaje a aplicației, iar în caz contrar întoarce FALSE.

O buclă de mesaje structurată în jurul funcției GetMessage arată astfel:

```
while (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}
return msg.wParam;
```

O buclă de mesaje structurată în jurul funcției PeekMessage va arăta astfel:

```
while (TRUE)
{
    if (PeekMessage (&msg, NULL, 0, 0))
    {
        if (msg.message == WM_QUIT) break;
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    else
    {
        /* Prelucrări periodice pe timpul cât sistemul este "în așteptare" */
    }
}
return msg.wParam;
```

Se observă faptul că valoarea WM\_QUIT din câmpul message este testată explicit pentru a întrerupe bucla de mesaje a aplicației la apariția acestui mesaj.

Dacă valoarea întoarsă de PeekMessage este TRUE, mesajul se procesează în mod normal, dacă valoarea întoarsă este însă FALSE, atunci programul poate să realizeze o secvență de acțiuni înainte de a întoarce controlul înapoi la Windows (prin apelul funcției PeekMessage).

### 9.5.1 Programul RandRect

Programul RandRect utilizează o buclă de mesaje structurată în jurul funcției PeekMessage pentru a desena în mod continuu în suprafața client dreptunghiuri aleatoare.

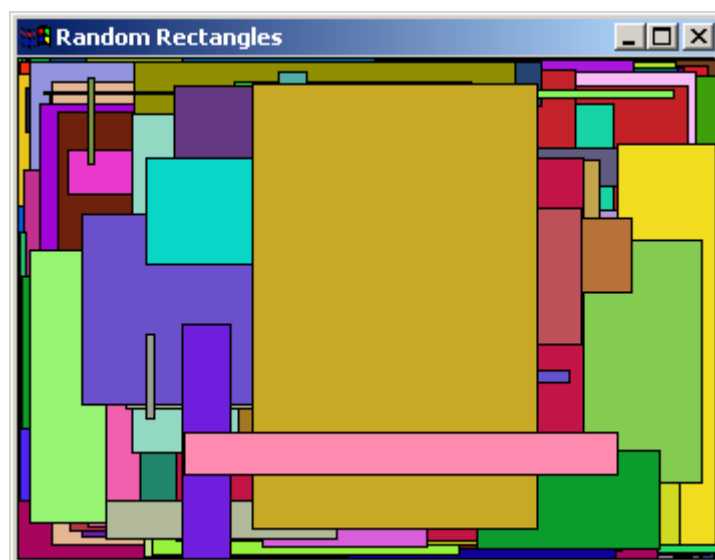


Figura 9.6 Programul RandRect



## Laborator 10 - Introducere în Visual Basic

Microsoft Visual Basic este un mediu de programare vizual, orientat pe obiecte. Obiectele au anumite *proprietăți*, *metode* și *evenimente*.

Visual Basic este alcătuit din două componente: componenta vizuală și componenta de cod. Componenta vizuală sau interfața utilizator este creată utilizând forme și controale care permit afișare/introducere de informații, comenzi, etc

### 10.1 Introducere în mediul de dezvoltare

Principalele facilități oferite de mediul de lucru Visual Basic 6 sunt următoarele:

- Interfață de tip MDI (*Multiple-Document Interface* - posibilitatea de a vizualiza la un anumit moment mai multe ferestre), asemănătoare altor medii MDI cum ar fi Word sau Excel.
- Se oferă posibilitatea de a se edita mai multe proiecte în aceeași sesiune Visual Basic (nu este necesar să se închidă un proiect pentru a se deschide și efectua modificări în altul).
- Facilități de înlesnire a scrierii codului prin "Auto List Member" și "Auto Quick Info" care furnizează constante și proprietăți ale controalelor pe măsură ce se editează codul.

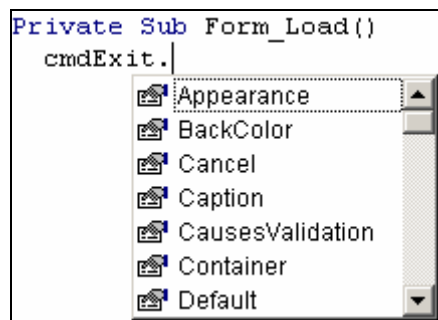


Figura 10.1 Auto List Member

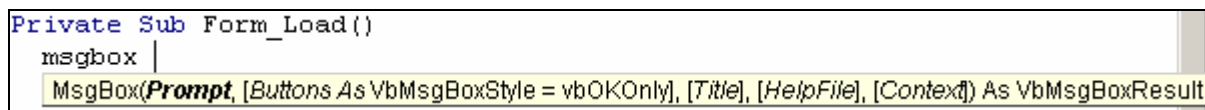


Figura 10.2 Auto Quick Info

La pornirea Visual Basic apare caseta de dialog *New Project* din figura 10.3 care cuprinde trei Tab-uri:

- *New* – permite alegerea tipului de proiect
- *Existing* – permite selectarea unui proiect creat și salvat anterior
- *Recent* – prezintă o listă a proiectelor la care s-a lucrat anterior (cel mai recent proiect este listat primul)

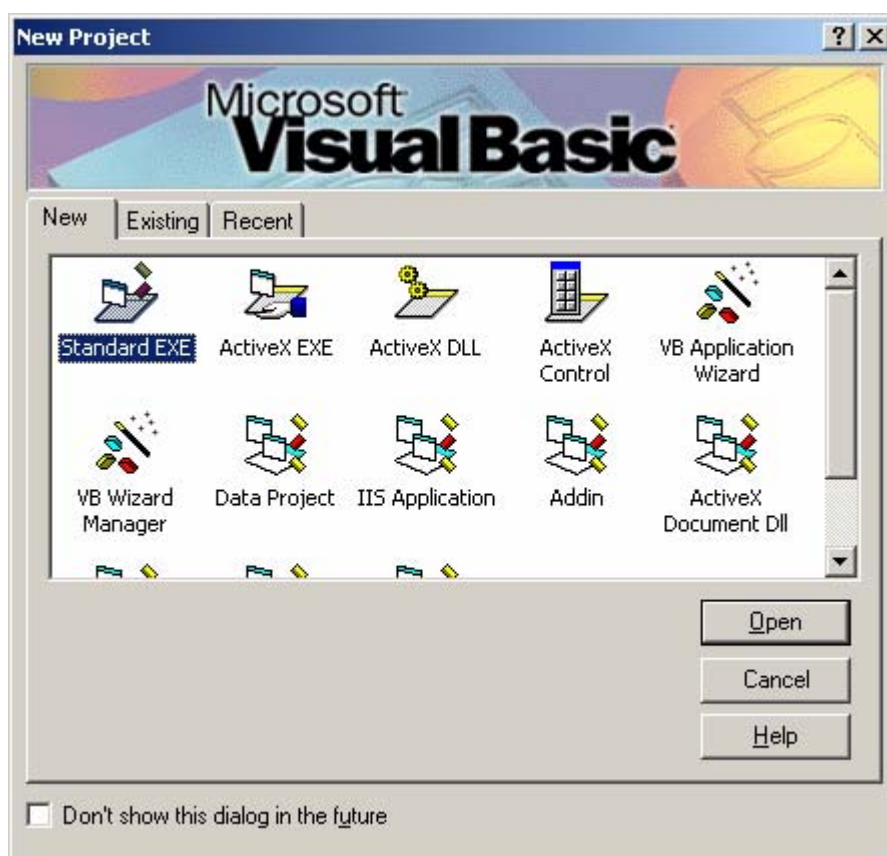


Figura 10.3 Fereastra New Project

Pentru a crea o aplicație EXE Windows alegeți opțiunea *New / Standard EXE*. După selectarea tipului de aplicație se deschide mediul de lucru Visual Basic, după cum se poate observa în figura 10.4.

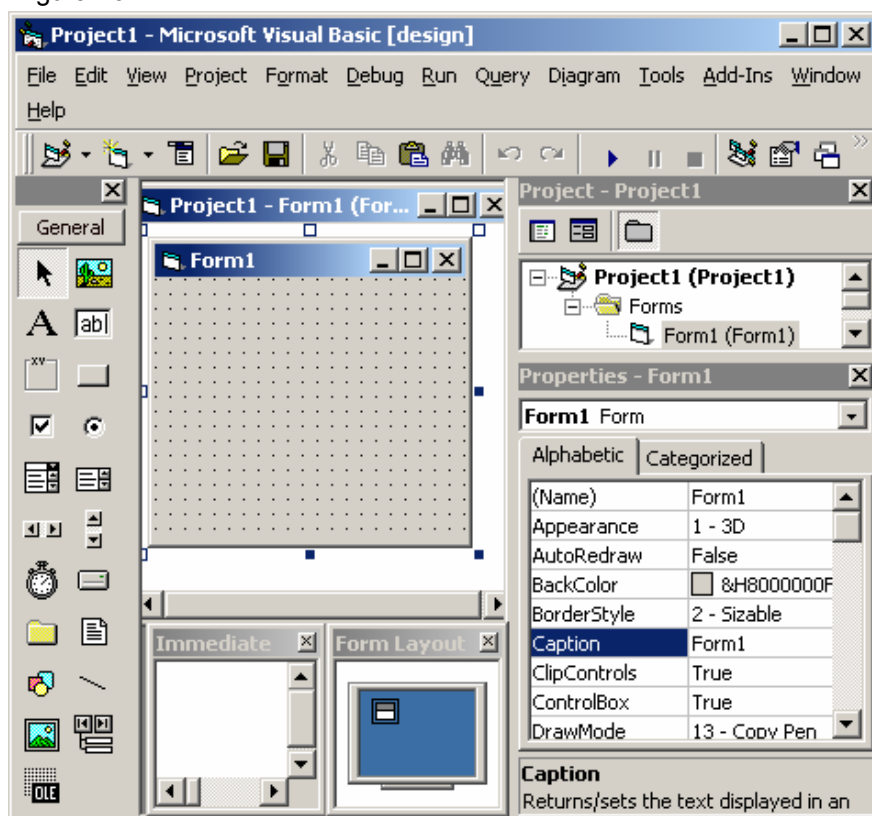


Figura 10.4 Mediul de lucru Visual Basic

În partea de sus se află **meniul** și **Toolbar**-ul care conțin comenzile mediului de programare Visual Basic: *Open Project* (deschide un proiect), *Save Project* (salvează un proiect), etc.

Mai jos, în partea stânga, se află **Toolbox**-ul care conține *controalele standard* Visual Basic ce permit: afișare/editare de text, conectare la o bază de date, etc. Acestui **Toolbox** i se pot adăuga controale adiționale (OCX) prin selectarea meniului *Project/Components*. Controlul selectat apare în **Toolbox** după apăsarea butoanelor *OK* sau *Apply*.

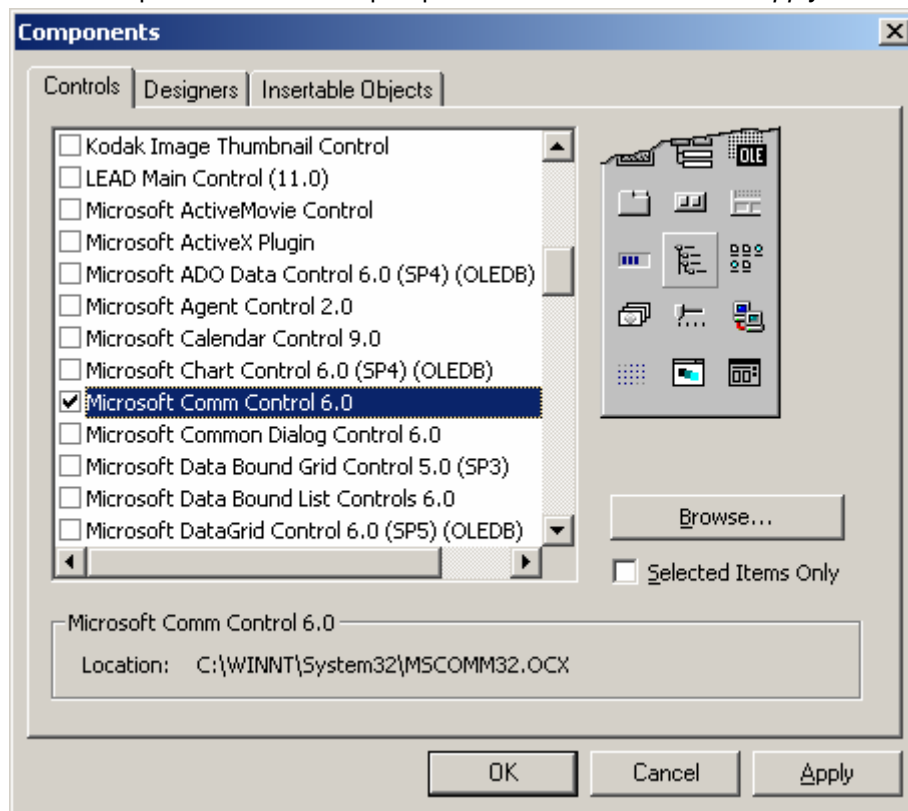


Figura 10.5

În partea centrală se găsește fereastra în care veți realiza aplicația dorită. Când începeți un nou proiect ea va purta numele de *Form1 (Project1 – Form1)*. Prin dublu click pe această fereastră, sau utilizând comenzile *View/Code*, se accesează fereastra **Code**.

În partea dreaptă veți observa două ferestre, una sub alta: **Project** și **Properties**. Fereastra **Project** listează toate formele, modulele de cod și fișierele de control care alcătuiesc proiectul curent. Dacă aceasta fereastră nu este vizibilă, utilizați comenzile *View/Project Explorer* din meniu.

Fereastra **Properties** permite selectarea proprietăților pentru forma sau controlul selectat. Dacă această fereastră nu este vizibilă, selectați obiectul ale cărui proprietăți doriți să le vizualizați și/sau modificați și apăsați tasta *F4*, sau utilizați comenzile *View/Properties Window* din meniu. Proprietățile se pot modifica la "*design time*" (de exemplu proprietatea *Caption* a unui control de tip *Label* se poate modifica prin scrierea textului "*Hello Word*" în câmpul corespunzător proprietății *Caption* din fereastra **Properties**) sau în timpul execuției programului - "*runtime*" (modificare în cod: *Form1.Label1.Caption="Hello Word"*).

La salvarea proiectului, se salvează de fapt, pe lângă proiectul respectiv (*.VBP – Visual Basic Project*), o listă de diferite fișiere care alcătuiesc proiectul (*.FRM – Visual Basic Form*; *.BAS – Code Module*; *.cls – Class Module*; etc).

Pentru accesarea ferestrei de ajutor – **Help** se utilizează fie comanda *Help* din meniu, fie tasta *F1*. Fereastra de ajutor conține informații referitoare la: elementele și controalele din **Toolbox** și **Toolbar**, obiectele și proprietățile lor, procedurile și comenzile din fereastra **Code**, mesajele de eroare. Prin selectarea unui control particular sau a unei anumite funcții și apăsarea tastei *F1* se obține *Help* pentru acel control/funcție.

## 10.2 Primul program în Visual Basic

O aplicație creată cu Visual Basic trebuie să respecte în totalitate caracteristicile unei aplicații pentru Windows. În acest scop, aceasta va conține o interfață formată din una sau mai multe ferestre conținând diverse controale și va reacționa la diverse evenimente declanșate de utilizator.

### 10.2.1 Proiectarea interfeței

Proiectarea interfeței se va realiza în fereastra **Form**, unde veți adăuga obiectele (controalele) din **Toolbar** (cu butonul din stânga al mouse-ului apăsați pe un obiect dorit din **Toolbar**, după care plasați obiectul tot cu butonul din stânga al mouse-ului în fereastra **Form**).

Setarea proprietăților se va face în fereastra **Properties** (selecțați un obiect deja plasat în fereastra **Form**, cu butonul din dreapta al mouse-ului selecțați **Properties** sau utilizați comenzile *View/Properties Window* din meniu – fereastra **Properties** a obiectului respectiv devine activă). Fereastra **Properties** afișează: numele și tipul obiectului; proprietățile obiectului; valorile proprietăților obiectului; o descriere a proprietăților obiectului.

#### 10.2.1.1 Exemplul 1

Modificați proprietatea *Caption* a formei *Form1* din Figura 10.4.

Pentru aceasta se parcurg următoarele etape:

- deschideți Visual Basic-ul și selecțați din fereastra *New Project – New/Standard EXE*
- În fereastra *Properties*, corespunzător proprietății *Caption* scrieți textul “*Hello World*” - textul care apare în bara titlu.

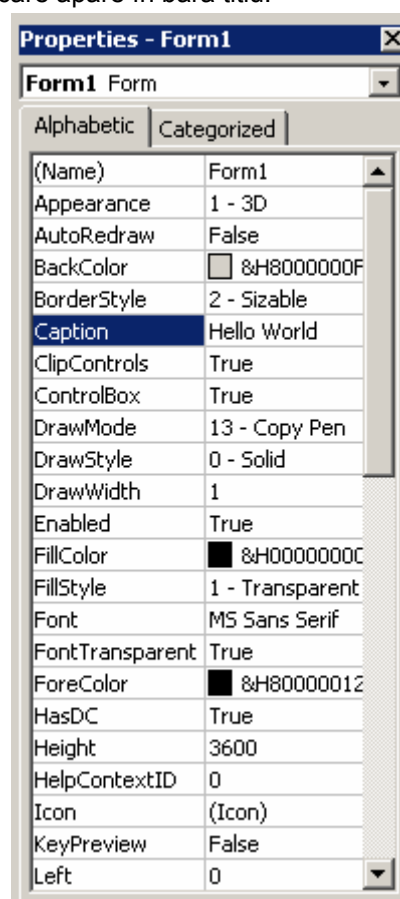


Figura 10.6

O proprietate a unui obiect se formează din variabila și valoarea sa. Variabila este reprezentată printr-un nume sugestiv, iar valoarea este reprezentată fie printr-o cifră fie printr-un șir de caractere. Setarea proprietăților înseamnă modificarea valorilor variabilelor.

### 10.2.1.2 Exemplul 2

În cazul formei *Form1* – setați următoarele proprietăți:

- *Name*...frmMain
- *Caption*...Hello World
- *Height*...2400
- *Width*...3600
- *StartPosition*...2 – Center Screen

Compilați programul apăsând tasta *F5* sau utilizând comenzile *Run/Start* din meniu. Veți obține următorul rezultat:

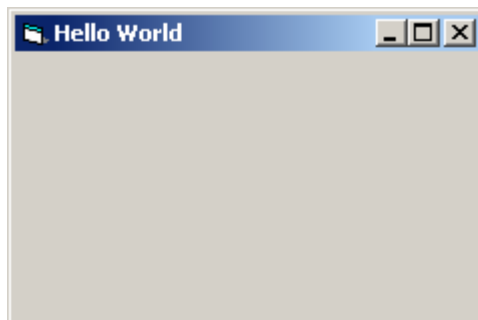


Figura 10.7

Majoritatea programelor necesită instrumente care să permită utilizatorilor să introducă date. În Visual Basic, cel mai utilizat control în acest scop este *TextBox*. După adăugarea controlului trebuie să-i setați proprietățile:

- Proprietatea *Name* este utilizată în cod la identificarea controlului (inițial este *Text1*)
- Proprietatea *Text* conține textul care apare în caseta de text, etc.

Cea mai simplă modalitate de identificare a datelor introduse de utilizator este prin introducerea obiectelor de tip *Label*. Utilizând proprietatea *Caption* se poate realiza o descriere a datelor introduse, precizând astfel utilizatorului unde să introducă datele de intrare.

### 10.2.1.3 Exemplul 3

Utilizați trei controale de tip *TextBox* (cu numele *txtDay*, *txtMonth*, *txtYear*) și trei controale de tip *Label* (*lblDay*, *lblMonth*, *lblYear*) și încercați să obțineți următorul rezultat:

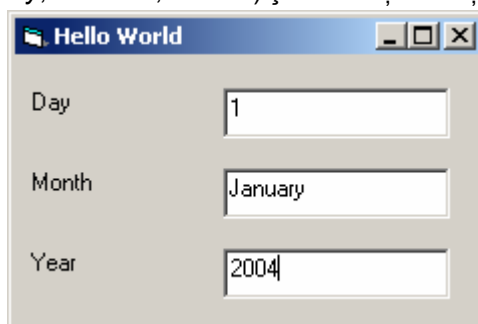


Figura 10.8

Primele controale despre care am discutat: *Label* și *TextButton* permit introducerea sau afișarea de informații. Controlul de tip *CommandButton* se utilizează pentru a iniția o comandă/secvență de instrucțiuni.

### 10.2.1.4 Exemplul 4

Adăugați un control de tip *CommandButton* formei din figura 10.7. Setați proprietățile controlului astfel:

- *Name*...cmdExit
- *Caption*...Exit

Dublu click pe obiectul recent creat. În procedura *cmdExit\_Click* introduceți următoarea secvență de cod, care se va executa la fiecare click pe buton:

```
Private Sub Command1_Click()
    End
End Sub
```

### 10.3 Elemente de bază în utilizarea controalelor

Modul în care arată și se comportă controalele este controlat de trei elemente de bază: *proprietăți*, *metode* și *evenimente*. Acest subcapitol explică aceste trei elemente și modul în care acestea afectează o formă. Procedul poate fi extins ulterior la toate tipurile de controale.

#### 10.3.1 Proprietățile unei forme

O formă este definită de o serie de *proprietăți*. Poziția formei pe ecran este controlată de proprietățile *Left* și *Top*, iar dimensiunea acesteia este indicată prin *Height* și *Width*. Titlul formei (din bara titlu) afișează conținutul proprietății *Caption*.

Proprietatea *StartPosition* controlează locația formei pe ecran (atunci când este afișată prima dată). Proprietatea *BorderStyle* are 6 stări posibile care controlează tipul de cadru. Setarea – 2 *Sizable*; permite utilizatorului să modifice dimensiunea formei atunci când programul rulează.

Proprietatea *Font* permite setarea tipului de font pentru orice text afișat pe formă utilizând metoda *Print* a formei.

O formă în Visual Basic conține toate cele 3 elemente care ne așteptăm să apară într-un program Windows: bara de titlu, un meniu de control, un set de butoane *Minimize*, *Maximize*, *Restore* și *Close*.

Proprietățile *ControlBox*, *MaxButton* și *MinButton* pot fi setate cu valorile *True* sau *False* în funcție de elementele pe care utilizatorul le dorește ca părți componente ale formei respective.

Chiar dacă aplicația include mai multe forme și controale, probabil că nu doriți ca utilizatorul să aibă acces la toate în același timp. În acest scop pot fi utilizate proprietățile: *Visible* și *Enabled*. Prin setarea proprietății *Visible* a unui anumit obiect cu *True*, acel obiect devine vizibil pe ecran. Prin setarea proprietății *Enabled* a unui anumit obiect cu *True*, utilizatorul poate interacționa cu acel obiect.

Utilizați un editor de text (de exemplu Notepad) pentru a deschide fișierul \*.FRM. Acest fișier stochează proprietățile formei.

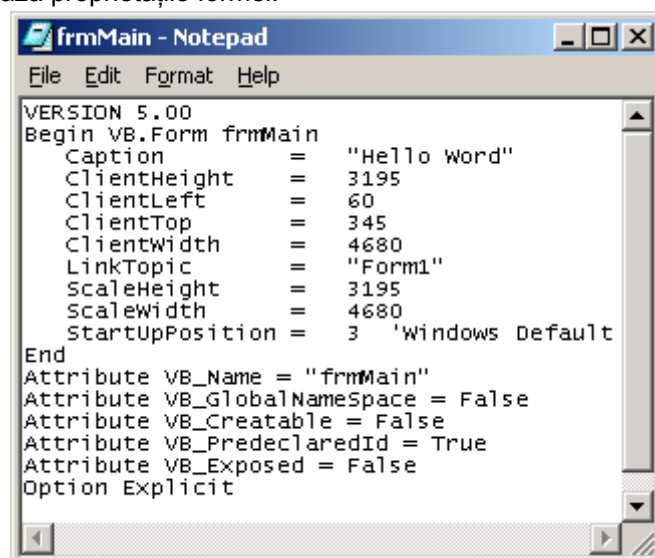


Figura 10.9

Pentru majoritatea formelor, Visual Basic creează un fișier cu extensia \*.FRX. Acesta stochează grafice sau alte elemente care nu pot fi definite prin text. Este important ca atunci când mutați formele în alt subdirector să copiați și fișierele \*.FRX.

### 10.3.1.1 Exemplul 5

Puteți controla dimensiunea formei prin modificarea valorii proprietăților *Height* și *Width* fie la design time, fie la runtime.

Creați un proiect Standard EXE alcătuit dintr-o formă. În fereastra *Properties Window* setați proprietatea *Name* a formei: *Name...frmMain* și apăsați tasta *F5* să-l rulați. Apăsați *Ctrl+Break* pentru a intra în mod *Break*. Apăsați *Ctrl+G* pentru a activa *Immediate Window*. În *Immediate Window* introduceți: *Print frmMain.Height*. Valoarea curentă a proprietății *Height* este tipărită (figura 1.10)

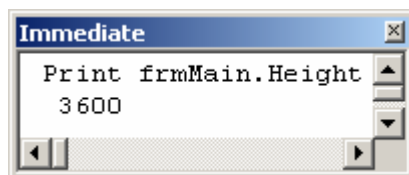


Figura 10.10

În *Immediate Window* tipăriți *frmMain.Height = frmMain.Height\*2* și apăsați tasta *F5*. Ce observați?

### 10.3.1.2 Exemplul 6

Creați un proiect *Standard EXE* alcătuit dintr-o formă. În fereastra *Properties Window* setați proprietatea *Name* a formei: *Name...frmMain*. Adăugați un control de tipul *CommandButton* formei respective și setați următoarele proprietăți: *Name...cmdResize*; *Caption...Resize*.

Dublu-click pe butonul respectiv. Adăugați în procedura *cmdResize\_Click* următoarea secvență de cod:

```
Private Sub Command1_Click()
    If frmMain.Height >= Screen.Height Then
        frmMain.Top = 0
    Else
        frmMain.Top = (Screen.Height - frmMain.Height) / 2
    End If
    If frmMain.Width >= Screen.Width Then
        frmMain.Left = 0
    Else
        frmMain.Left = (Screen.Width - frmMain.Width) / 2
    End If
End Sub
```

Apăsați tasta *F5*, modificați dimensiunile formei și apăsați butonul *Resize*. Observați că forma rămâne centrată după modificarea dimensiunilor

### 10.3.2 Metode și Evenimente

Pe lângă *proprietăți* un obiect poate avea *metode*, care definesc acțiuni pe care acesta le poate executa. Acțiunile pot fi simple (de exemplu mutarea unui obiect către altă locație, sau pot fi complexe, de exemplu actualizarea informației dintr-o bază de date).

O metodă este ca o funcție program care aparține obiectului respectiv. Exemple de metode:

- *Move* – modifică poziția obiectului;
- *SetFocus* – focusează obiectul respectiv
- *Zorder* – determină când un obiect apare în fața sau în spatele obiectului care îl conține

În Visual Basic un program acționează ca răspuns la un eveniment. Fiecare control pe care-l utilizăm în program este construit să recunoască câteva tipuri de evenimente. Dacă doriți ca un obiect să răspundă unui eveniment, trebuie să scrieți cod în procedura eveniment respectivă, altfel, programul ignora acel eveniment. În partea de sus a ferestrei **Code** veți observa două casete de tip drop-down. La stânga este lista obiectelor care au fost plasate în forma respectivă, la dreapta este lista evenimentelor disponibile pentru obiectul care este selectat.

Exemple de evenimente:

- apăsarea unui buton cauzează apariția evenimentului *Click*
- modificarea textului unui *TextBox* determină apariția evenimentului *Change*, etc

Atunci când se invocă o metodă a unui obiect, se pot modifica de asemenea proprietățile obiectului respectiv.

#### 10.3.2.1 Exemplul 7

Creați un proiect Standard EXE alcătuit dintr-o formă. În fereastra *Properties Window* setați proprietatea *Name* a formei: *Name...frmMain*. Adăugați un *TextBox* formei și setați proprietatea *Name...Text*.

Dublu click pe formă. Adăugați următoarea secvență de cod:

```
Private Sub Form_Load()  
    Text.Top = 1000 'mutarea controlului prin setarea proprietății  
End Sub
```

Utilizați metoda *Move* pentru a obține un efect similar:

```
Private Sub Form_Load()  
    Text.Move 100, 1100 'mutarea controlului folosind metoda Move  
End Sub
```

Analog, metodele *Show* și *Hide* au același efect cu proprietatea *Visible*. Când se invocă metoda *Show*, efectul este asemănător cu setarea proprietății *Visible* cu *True*.

De cele mai multe ori când se utilizează metode sau se modifică proprietăți, acest lucru se întâmplă ca răspuns la un eveniment. De exemplu, în cazul unei forme, modificarea proprietăților *Height* sau *Width* generează evenimentul *Resize*. Starea unei forme este controlată de proprietățile *Load* și *Unload*, precum și de metodele *Show* și *Hide*.

Există câteva evenimente speciale care pot să apară în cazul unei forme *Form1*:

- *Load* – atunci când forma *Form1* este încărcată în memorie
- *Activate* – atunci când forma este afișată inițial sau când utilizatorul se întoarce la forma *Form1* din altă formă
- *Deactivate* – atunci când utilizatorul se mută pe o altă formă sau când forma *Form1* este *Hidden*
- *Unload* – când forma *Form1* este scoasă din memorie
- *Initialize* – când este creată o instanță a obiectului formă

Reluați exemplul 1 utilizând controale de tipul: *Label*, *TextBox* și *CommandButton*.

#### 10.3.3 Adăugarea elementelor de control

Deschideți Visual Basic-ul și selectați din fereastra *New Project – New/Standard EXE*. În fereastra *Properties*, corespunzător proprietății *Name* scrieți *frmMain*

##### 10.3.3.1 Exemplul 8

În forma *frmMain* adăugați un control *TextBox* și setați proprietatea *Name...txtName*. Dublu-click pe *frmMain* și adăugați următoarea secvență de cod:

```
Private Sub frmMain_Load()  
    txtName.Text="Hello World"  
End Sub
```

Apăsați tasta F5

##### 10.3.3.2 Exemplul 9

În forma *frmMain* adăugați un control *CommandButton* și setați proprietatea *Name...cmdPrint*. Dublu click pe butonul *cmdPrint*. Adăugați următoarea secvență de cod:

```
Private Sub cmdPrint_Click()  
    Print "Hello World"  
End Sub
```

Apăsați tasta F5

##### 10.3.3.3 Exemplul 10

În forma *frmMain* adăugați un control *CommandButton* și setați proprietatea *Name...cmdUcase*



În forma frmMain adăugați un control TextBox și setați proprietatea Name...txtName. Adăugați următoarea secvența de cod:

```
Sub cmdUcase_Click()  
    Dim sName As String  
    sName = txtName.Text  
    sName = Ucase$(sName)  
    txtName.Text = sName  
End Sub
```

Apăsați tasta F5, scrieți în caseta de text *Hello World* și apăsați butonul *cmdUcase*.

#### **10.3.3.4 Exemplul 11**

În forma frmMain adăugați un control TextBox și setați proprietatea Name...Text1 și Text..."Adăugați text". Adăugați următoarea secvența de cod pentru selectare text:

```
Private Sub Text1_GotFocus()  
    Text1.Text = "Text"  
    SendKeys "{Home}+{End}"  
End Sub
```

Apăsați tasta F5 și scrieți în caseta de text *Hello World*

#### **10.3.3.5 Exemplul 12**

În forma frmMain adăugați un control Label și setați proprietatea Name...lblName , Caption ...Hello World. În forma frmMain adăugați un control CommandButton și setați proprietatea Name cmdName. Adăugați următoarea secvența de cod:

```
Private Sub cmdName_Click()  
    lblName.Caption = "Hello" & vbCrLf & "World"  
End Sub
```

Apăsați tasta F5

## Bibliografie:

1. **Chandler D., Fotsch M.** - *Windows 2000 Graphics API Black Book*, Paraglyph Publishing, 2002.
2. **Eckel B.** - *Using C++*, Osborne McGraw – Hill, 1989.
3. **Gouker M., Simon R., Barnes B.** - *Windows 95 Win 32 Programming Api Bible*, Waite Group Press, 1996.
4. **Hart J.** - *Win32 System Programming: A Windows® 2000 Application Developer's Guide*, Addison-Wesley Pub Co., 2<sup>nd</sup> Edition, 2000
5. **Ladd S. R.** - *Win 32 Api: A Programmer's Reference*, Hungry Minds. Inc, 1995
6. **McCord J.** - *Win 32 Api Desktop Reference*, SAMS, 1993
7. **McKay E., Woodring M.** - *Debugging Windows Programs: Strategies, Tools, and Techniques for Visual C++ Programmers*, Addison-Wesley Pub Co., 2000
8. **Meyers S.** - *Effective C++: 50 Specific Ways to Improve Your Programs and Design*, 2<sup>nd</sup> edition, 1997
9. **Petzold C.** - *Programming Windows*, Microsoft Press, 5<sup>th</sup> edition, 1998.
10. **Rechtin E.** - *The Synthesis of Complex Systems*, IEEE Spectrum, p. 51-55, iulie 1997.
11. **Rector B., Newcomer J.** - *Win32 Programming*, Addison-Wesley Pub Co., 1997)
12. **Simon R.** - *Windows NT Win32 API SuperBible*, Waite Group Press, 1997
13. **Simon R.** - *Windows 2000 API SuperBible*, SAMS, 2000
14. **Stroustrup B.** - *The C++ Programming Language (3<sup>rd</sup> Edition)*, Addison – Wesley, 2000.
15. **Villani P.** - *Programming Win32 Under the API*, CMP Books, 2001
16. **Woodring M., Cohen A.** - *Win32 Multithreaded Programming*, O'Reilly & Associates, 1997

## CUPRINS

<b>Prefață</b>	<b>3</b>
<b>Laborator 1. Mediul Visual C++</b>	<b>5</b>
1.1 Instalarea mediului de dezvoltare Microsoft Visual Studio 6.0	5
1.2 Instalarea MSDN Library pentru Microsoft Visual Studio 6.	7
1.3 Visual C++ ca mediu de dezvoltare	8
1.4 Programarea în Visual C++	8
1.5 Diferențe între programarea Win API (SDK) și MFC	9
1.6 Mediul Visual C++	9
1.6.1 Spațiul de lucru - Workspace	10
1.6.2 The Output Pane	10
1.6.3 The Editor Area – Zona de editare	10
1.6.4 Meniurile și Toolbar-urile	12
1.6.5 Rearanjarea mediului de dezvoltare Visual C++	12
1.6.6 Help-ul	12
1.7 Opțiunile globale în Visual C++	12
1.8 Primul proiect în Visual C++	13
1.8.1 Crearea workspace-ului proiectului	13
1.8.2 Utilizarea Application Wizard pentru a crea mediul de dezvoltare al programului	15
1.8.3 Structura aplicației	15
1.8.4 Opțiunile proiectului	17
1.8.5 Debug-ul programului	18
1.9 Teme	20
<b>Laborator 2. Primul program în Visual C++</b>	<b>21</b>
2.1 Interfața grafică cu utilizatorul	21
2.1.1 Interfața utilizator consistentă	21
2.1.2 Avantajul multitasking-ului	22
2.1.3 Gestiunea memoriei	22
2.1.4 Interfață grafică independentă de "device" (periferic)	22
2.1.5 Apelul funcțiilor Windows	22
2.1.6 Legarea dinamică (Dynamic Linking)	23
2.1.7 Programarea orientată pe obiect	23
2.1.8 Arhitectura aplicațiilor conduse de evenimente (mesaje)	24
2.1.9 Procedura de fereastră	24
2.2 Primul program Windows	25
2.2.1 Fișierul HelloWin.c	26
2.2.2 Identificatorii scriși cu majuscule	26
2.2.3 Tipuri de date	27
2.2.4 Despre handle-uri	27
2.2.5 Notăția Ungară	27
2.2.6 Punctul de intrare în program	28
2.2.7 Înregistrarea clasei de fereastră	29
2.2.8 Crearea ferestrei principale a aplicației	31
2.2.9 Afișarea ferestrei	32
2.2.10 Bucla de mesaje	32
2.2.11 Procedura de fereastră	33
2.2.12 Procesarea mesajelor	33
2.2.12.1 Mesajul WM_PAINT	34
2.2.12.2 Mesajul WM_DESTROY	35
2.3 Teme	35
<b>Laborator 3 - Desenarea textelor</b>	<b>36</b>
3.1 Aplicația fundamentală Windows	36
3.2 Programarea independentă de periferic (device)	38
3.2.1 Desenarea și redesenarea suprafeței client	38
3.2.2 Mesajul WM_PAINT	39
3.2.3 Dreptunghiuri valide și invalide	39
3.2.4 Introducere în GDI	40
3.2.5 Contextul de periferic - "device context"	40
3.2.6 Prima metodă pentru obținerea unui handle pentru un device context	40

3.2.7	Structura de informații de desenare .....	41
3.2.8	A doua metodă pentru obținerea unui handle de device context .....	43
3.2.9	Funcția TextOut.....	43
3.2.10	Setul de caractere sistem.....	44
3.2.11	Dimensiunea unui caracter.....	44
3.2.12	Funcția GetTextMetric.....	45
3.2.13	Afișarea textelor .....	45
3.3	Programul SysMets.....	46
3.3.1	Funcțiile de prelucrare de mesaje .....	47
<b>Laborator 4 - Scroll Bar-uri</b>		<b>49</b>
4.1	Dimensiunea suprafeței client.....	49
4.2	Scroll bar-uri.....	49
4.3	Intervalul de scroll și poziția scroll bar-urilor .....	50
4.4	Mesajele de scroll .....	51
4.5	O aplicație Windows care utilizează scroll bar-uri.....	52
4.5.1	Mesajele prelucrate de aplicație .....	52
4.6	Temă .....	53
<b>Laboratorul 5 - Tastatura</b>		<b>54</b>
5.1	Elementele fundamentale de tratare a evenimentelor provenind de la tastatură .....	54
5.1.1	Driverul de tastatură .....	54
5.1.2	Fereastra selectată .....	54
5.1.3	Taste și caractere .....	55
5.2	Mesajele de tastă .....	55
5.2.1	Taste sistem și taste normale.....	55
5.3	Argumentul IParam .....	56
5.3.1	Contorul de repetiție .....	56
5.3.2	Codul de scanare OEM .....	56
5.3.3	Indicatorul de tastă extinsă .....	56
5.3.4	Codul de context .....	56
5.3.5	Starea anterioară a tastei.....	56
5.3.6	Starea curentă a tastei .....	56
5.4	Codurile virtuale de tastă .....	56
5.4.1	Starea tastelor de shift .....	57
5.4.2	Procesarea mesajelor de tastă .....	57
5.4.3	Interfață de tastatură pentru programul SysMets.....	57
5.4.4	Funcția de procesare a mesajelor de tastă .....	57
5.4.5	Trimiterea de mesaje către procedura de fereastră.....	57
5.5	Mesajele de caracter.....	58
5.5.1	Mesajele WM_CHAR.....	60
5.5.2	Mesaje caracter "moarte".....	61
5.5.3	Afișarea mesajelor de tastatură .....	61
5.6	Caret-ul (cursorul text) .....	62
5.6.1	Funcțiile de caret .....	62
5.7	Temă .....	63
<b>Laboratorul 6 – Mouse-ul</b>		<b>67</b>
6.1	Elementele fundamentale de tratare a mouse-ului .....	67
6.1.1	Definiții.....	67
6.2	Mesajele mouse în zona client.....	67
6.3	Aplicație pentru procesarea mesajelor de mouse.....	68
6.4	Evenimentele double-click .....	69
6.5	Mesaje mouse non-client .....	70
6.6	Mesajul Hit-Test .....	71
6.7	Implementarea Hit Test-ului în zona client.....	71
6.7.1	Exemplu de aplicație ce realizează hit-test în zona client .....	71
6.8	Utilizarea ferestrelor copil pentru hit-testing.....	72
6.8.1	Aplicație ce utilizează ferestre copil pentru hit-testing.....	72
6.9	Captura mouse-ului.....	73
6.9.1	Aplicație ce utilizează captura mouse-ului .....	74
6.9.2	Modificarea formei cursorului de mouse .....	74
6.9.3	Apelul funcției StretchBlt.....	75
6.9.4	Desenarea zonei de captură a imaginii.....	75

6.10	Temă .....	75
<b>Laboratorul 7 - Timer-ul .....</b>		<b>76</b>
7.1	Timerul în programarea Windows .....	76
7.2	Elementele fundamentale de tratare a timer-ului .....	76
7.2.1	System.drv și Timer-ul Windows .....	77
7.3	Utilizarea timer-ului .....	77
7.3.1	Prima metodă de utilizare a timer-ului .....	78
7.3.2	Tratarea cazului în care nu există timere disponibile .....	78
7.3.3	Aplicație ce utilizează timere .....	79
7.4	Utilizarea culorilor în Windows .....	80
7.5	A doua metodă de utilizare a timer-ului .....	80
7.6	Utilizarea funcțiilor "call-back" .....	81
7.7	A treia metodă de utilizare a timer-ului .....	82
7.8	Utilizarea timer-ului pentru afișarea unor informații de stare .....	82
7.8.1	Pornirea unei aplicații sub forma minimizată .....	83
7.8.2	Afișarea informațiilor în taskbar .....	83
7.8.3	Spațiul de memorie disponibil .....	83
7.9	Utilizarea timer-ului pentru realizarea unui ceas .....	83
7.10	Temă .....	83
<b>Laborator 8 - Ferestre control .....</b>		<b>84</b>
8.1	Ferestrele de control în Windows .....	84
8.2	Clasa "button" .....	84
8.2.1	Crearea ferestrelor copil .....	85
8.2.2	Conversația dintre fereastra copil și fereastra părinte .....	86
8.2.3	Butoanele cu revenire .....	87
8.2.4	Butoane non-exclusive (check box-uri) .....	87
8.2.5	Butoane exclusive (radio) .....	88
8.2.6	Grupurile .....	88
8.2.7	Butoanele definite de utilizator .....	88
8.2.8	Modificarea textului din interiorul butoanelor .....	89
8.2.9	Butoane vizibile, invizibile, active și inactive .....	89
8.3	Culorile ferestrelor control .....	89
8.3.1	Culorile butoanelor .....	90
8.3.2	Mesajul WM_CTLCOLOR .....	90
8.4	Clasa "static" .....	91
8.5	Clasa "scrollbar" .....	92
8.5.1	Programul Colors .....	92
8.5.2	Interfață de tastatură a ferestrelor scroll bar de control .....	93
8.5.3	"Window subclassing" .....	93
8.5.4	Desenarea fundalului .....	94
8.5.5	Colorarea interiorului scroll bar-urilor .....	95
8.6	Clasa "edit" .....	95
8.6.1	Stilurile ferestrelor din clasa "edit" .....	96
8.6.2	Mesajele de notificare a ferestrelor control "edit" .....	97
8.6.3	Mesaje către un control edit .....	97
<b>Laborator 9 – GDI - Graphic Device Interface .....</b>		<b>99</b>
9.1	Funcții GDI de desenare .....	99
9.1.1	Desenarea de puncte (pixeli) .....	99
9.1.2	Desenarea de linii .....	99
9.1.3	Utilizarea pen-urilor predefinite (Stock Pens) .....	100
9.1.4	Crearea, selectarea și eliberarea pen-urilor .....	101
9.2	Modurile de desenare .....	102
9.2.1	Programul Rop2Look .....	103
9.2.2	Modul de desenare și culorile .....	103
9.3	Desenarea figurilor închise .....	104
9.3.1	Dreptunghiul de încadrare .....	104
9.3.2	Brush-uri .....	105
9.3.3	Alinierea brush-urilor .....	106
9.4	Dreptunghiuri, regiuni și clipping .....	106
9.4.1	Funcții de desenare care utilizează dreptunghiuri .....	106
9.4.2	Crearea și desenarea regiunilor .....	107
9.4.3	Dreptunghiuri și regiuni de clipping .....	108
9.4.4	Programul Clover .....	108

9.5	Programe care desenează continuu .....	109
9.5.1	Programul RandRect .....	110
<b>Laborator 10 - Introducere în Visual Basic .....</b>		<b>111</b>
10.1	Introducere în mediul de dezvoltare .....	111
10.2	Primul program în Visual Basic .....	114
10.2.1	Proiectarea interfeței .....	114
10.2.1.1	Exemplul 1 .....	114
10.2.1.2	Exemplul 2 .....	115
10.2.1.3	Exemplul 3 .....	115
10.2.1.4	Exemplul 4 .....	115
10.3	Elemente de bază în utilizarea controalelor .....	116
10.3.1	Proprietățile unei forme .....	116
10.3.1.1	Exemplul 5 .....	117
10.3.1.2	Exemplul 6 .....	117
10.3.2	Metode și Evenimente .....	117
10.3.2.1	Exemplul 7 .....	118
10.3.3	Adăugarea elementelor de control .....	118
10.3.3.1	Exemplul 8 .....	118
10.3.3.2	Exemplul 9 .....	118
10.3.3.3	Exemplul 10 .....	118
10.3.3.4	Exemplul 11 .....	119
10.3.3.5	Exemplul 12 .....	119
<b>Bibliografie .....</b>		<b>120</b>