1. Introducere în framework-ul DIBLook

1.1. Introducere

Obiectivul acestui prim laborator este familiarizarea studenţilor cu aplicaţia framework care va fi utilizată pentru realizarea activităţilor practice din cadrul disciplinei Procesarea Imaginilor.

Cunoştinţele fundamentale necesare pentru a realiza cu succes laboratorul de Procesarea Imaginilor sunt:

- **Obligatoriu**: *C, Programarea Calculatoarelor, Structuri de Date şi Algoritmi.*
- **Opţional (recomandat)**: *C++, Visual C++ 12.0 (Visual Studio 2013), Metode Orientate Obiect, Algoritmi Fundamentali, Tehnici de Programare, Algebra Liniară şi Geometrie, Matematici Discrete, Calcul Numeric, Matematici Speciale.*

1.2. Descrierea generală a framework-ului DIBLook

Framework-ul care va fi utilizat pentru implementarea şi testarea algoritmilor de procesarea imaginilor învăţaţi este bazat pe aplicaţia demonstrativă *DIBLook* disponibilă în MSDN. O versiune modificată a acestei aplicaţii (pentru utilizarea mai uşoară) este disponibilă pe pagina de web a laboratorului de Procesarea Imaginilor.

DIBLook este o aplicaţie *MDI (Multiple Document Interface)* [1] respectând arhitectura *Document-View* [2], [3] (Fig. 1.1) din cadrul *MFC (Microsoft Foundation Class) Library* [4].

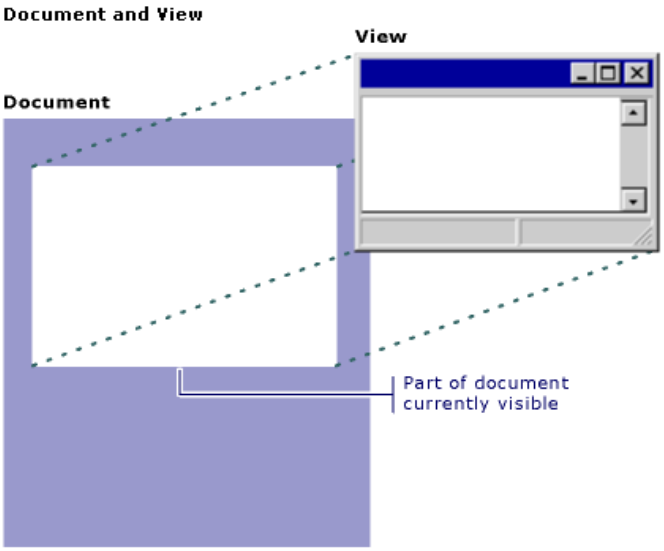

Fig. 1.1. Arhitectura Document View [3]

Versiunea originală de *DIBLook* permite utilizatorului să deschidă, să vizualizeze şi să salveze imagini bitmap (*.bmp, *.dib) (Fig. 1.2). Fiecare imagine este deschisă într-o fereastră diferită şi are asociat câte un obiect propriu *View* (instanţiat din clasa *CDibView*) şi un obiect *Document* (instanţiat din clasa *CDibDoc*) (Fig. 1.6). Obiectul *View* este utilizat pentru interacţiunea cu datele asociate bitmap-ului care sunt memorate în obiectul *Document.*

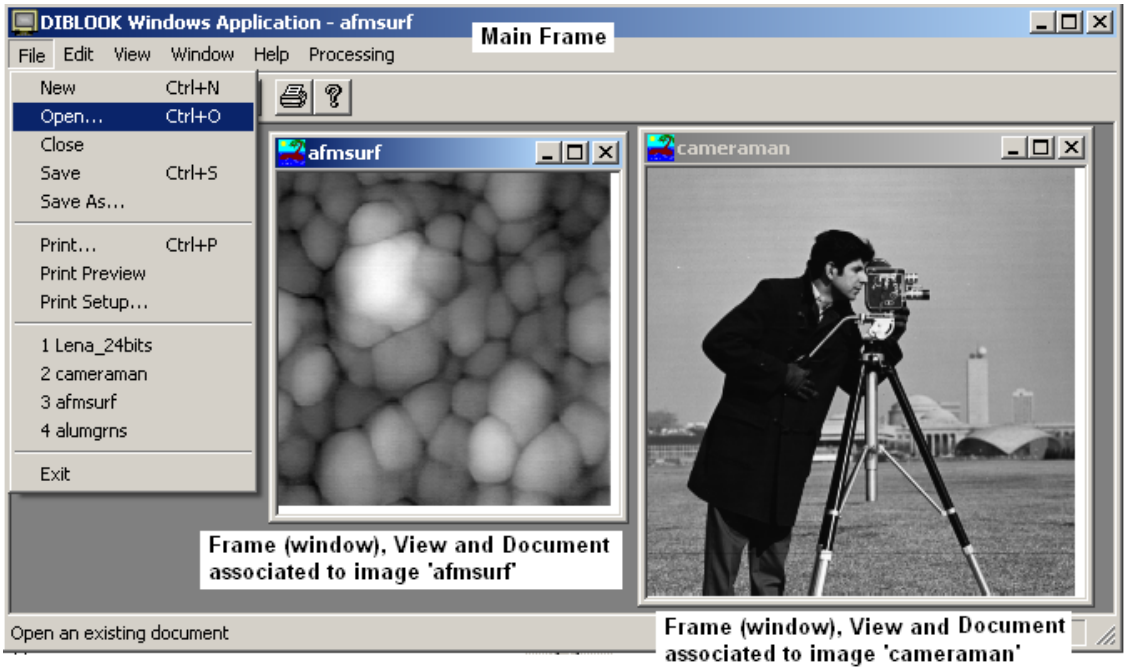

Fig. 1.2. Fiecare imagine deschisă este afişată într-o fereastră diferită şi are propriile obiecte *View* şi *Document*

1.3. Adăugarea unei funcţii de procesare la aplicaţia DIBLook

Pentru a putea efectua orice tip de procesare asupra unei imagini deschise sunt necesari următorii paşi:

1. Comutaţi pe tabulatorul *Resource View* (daca nu există îl veţi deschide din meniul *View -> [Other Windows->] Resource View*) şi deschideţi meniul *IDR_DIBTYPE* (Fig. 1.3):

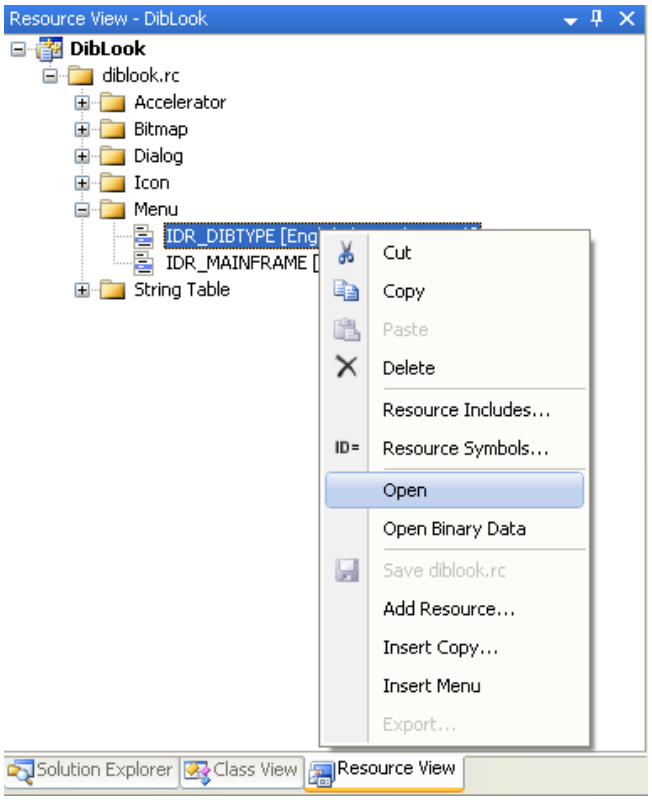

Fig. 1.3. Fereastra de resurse ale aplicaţiei

2. Adăugaţi o intrare nouă în meniu prin scrierea numelui sub intrarea deja existentă (Fig.1.4):

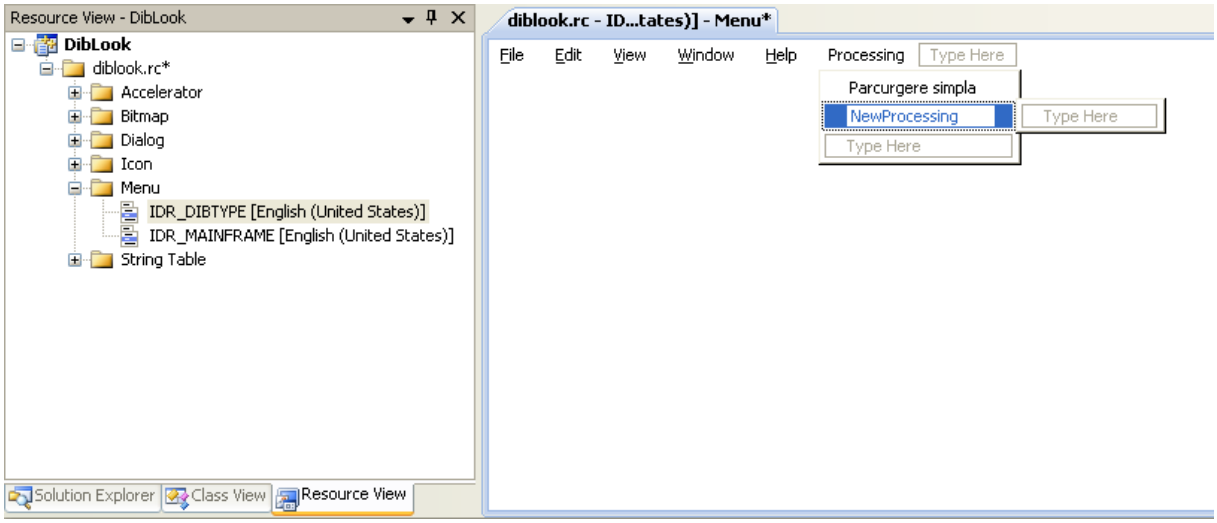

Fig. 1.4. Adăugarea în meniu a intrării *NewProcessing*

3. Asociaţi o funcţie care să fie executată când meniul este accesat (click) prin *Add Event Handler*...(click dreapta pe meniu)

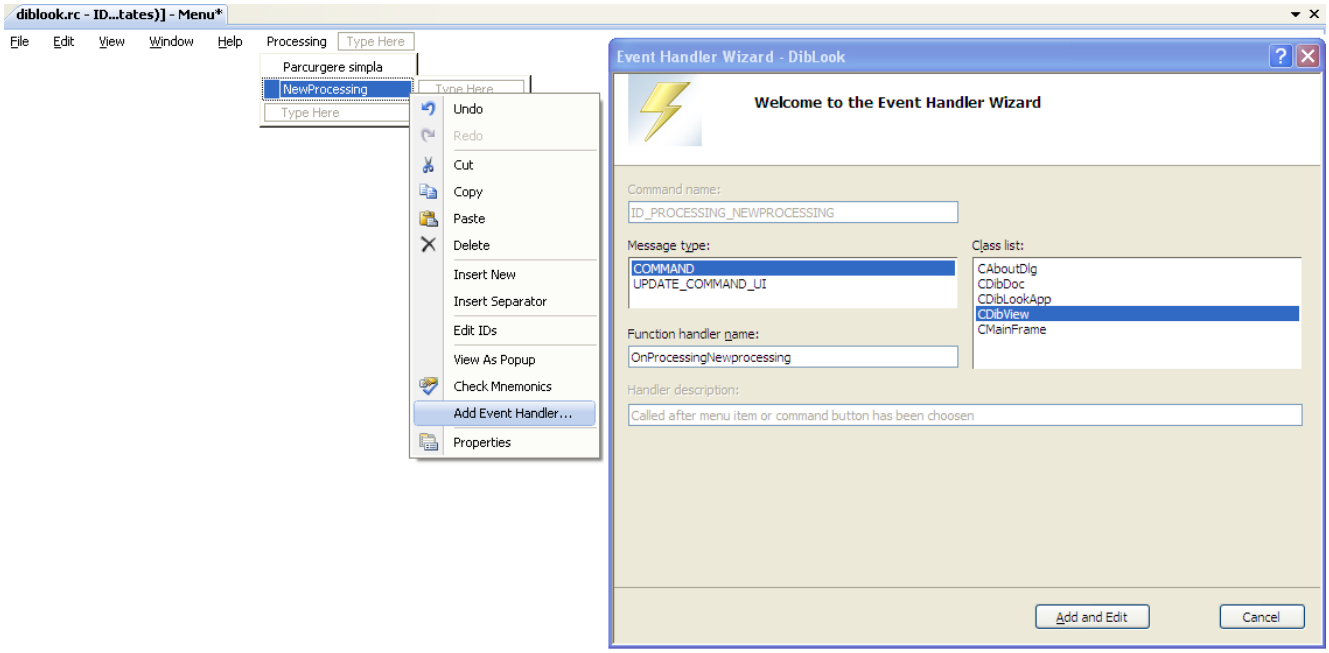

Fig. 1.5. Adăugarea funcţiei asociată evenimentului de click pe intrarea *NewProcessing*

Observaţii importante:

a. Noua funcţie trebuie să fie membră a clasei *CDibView* !

b. Funcţia trebuie să fie apelată de mesajul *COMMAND* (generat prin apăsarea 'click' pe meniu)

4. Adăugarea şi accesarea codului noii funcţii se face prin intermediul butonului *Add and Edit*

```
void CDibView::OnProcessingNewprocessing()
{
	// TODO: Add your command handler code here
}
```

1.4. Un exemplu de funcţie de procesare de imagini

Un exemplu simplu de funcţie de procesare de imagini este oferită în codul sursă al aplicaţiei *DIBLook*. Funcţia *OnProcessingParcurgereSimpla* este membru al clasei *CDibView* (obligatoriu) şi a fost creată urmărind paşii descrişi în secţiunea anterioară (1.3). Aceasta arată cum se accesează pixelii unei imagini bitmap sursă cu 8 biţi/pixel, efectuează câteva operaţii simple (egalează intrările din LUT (grayscale) şi negativează fiecare pixel din imagine) şi afişează rezultatele într-o fereastră nouă/fereastra destinaţie (asociată cu obiectele noi corespunzătoare destinaţiei *Document* şi *View*).

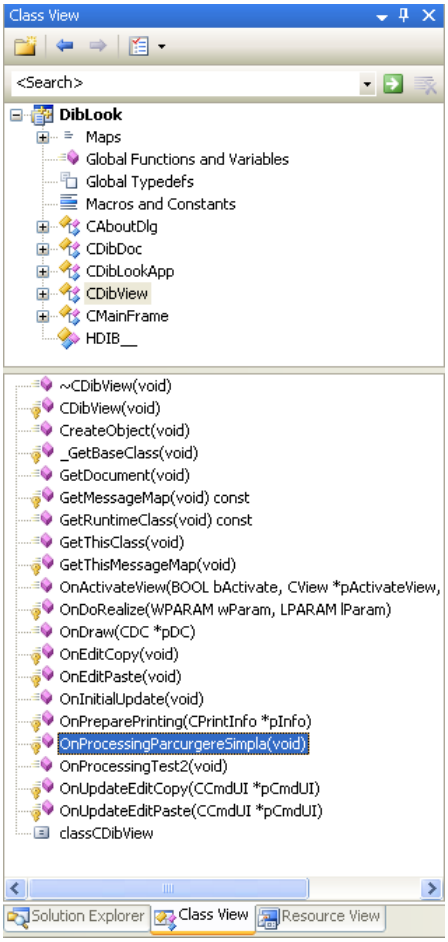

Fig. 1.6. Fereastra *Class View* şi metodele clasei *CDibView*

```
void CDibView::OnProcessingParcurgereSimpla()
{
      INCEPUT_PRELUCRARI();

      // Makes a grayscale image by equalizing the R, G, B components from the LUT
      for (int k=0;  k < iColors ; k++)
            bmiColorsDst[k].rgbRed=bmiColorsDst[k].rgbGreen=
                  bmiColorsDst[k].rgbBlue=k;

      // Goes through the bitmap pixels and performs their negative
      for (int i=0;i<dwHeight;i++)
            for (int j=0;j<dwWidth;j++)
                  lpDst[i*w+j]= 255 - lpSrc[i*w+j]; //makes image negative

      SFARSIT_PRELUCRARI("Operation name");
}
```

1.4.1. Definiţia macroului: `INCEPUT_PRELUCRARI()`

Oferă toate iniţializările, definirile şi alocările necesare. Este definit la începutul fişierului *dibview.cpp* (nu este oferit cu versiunea originală demonstrativă de *DIBLook* din MSDN). Aveţi grijă dacă doriţi să-l editaţi (fiecare linie trebuie să se termine printr-un '\' + <ENTER>, nu sunt permise comentariile etc.)

```
#define INCEPUT_PRELUCRARI()                                                    \
      CDibDoc* pDocSrc=GetDocument();                                           \
      CDocTemplate* pDocTemplate=pDocSrc->GetDocTemplate();                     \
      CDibDoc* pDocDest=(CDibDoc*) pDocTemplate->CreateNewDocument();           \
      BeginWaitCursor();                                                        \
      HDIB hBmpSrc=pDocSrc->GetHDIB();                                          \
      HDIB hBmpDest = (HDIB)::CopyHandle((HGLOBAL)hBmpSrc);                     \
      if ( hBmpDest==0 ) {                                                      \
            pDocTemplate->RemoveDocument(pDocDest);                             \
            return;                                                             \
      }                                                                         \
      BYTE* lpD = (BYTE*)::GlobalLock((HGLOBAL)hBmpDest);                       \
      BYTE* lpS = (BYTE*)::GlobalLock((HGLOBAL)hBmpSrc);                        \
      int iColors = DIBNumColors((char *)&(((LPBITMAPINFO)lpD)->bmiHeader));    \
      RGBQUAD *bmiColorsDst = ((LPBITMAPINFO)lpD)->bmiColors;                   \
      RGBQUAD *bmiColorsSrc = ((LPBITMAPINFO)lpS)->bmiColors;                   \
      BYTE * lpDst = (BYTE*)::FindDIBBits((LPSTR)lpD);                          \
      BYTE * lpSrc = (BYTE*)::FindDIBBits((LPSTR)lpS);                          \
      DWORD dwWidth  = ::DIBWidth((LPSTR)lpS);                                  \
      DWORD dwHeight = ::DIBHeight((LPSTR)lpS);                                 \
      DWORD w=WIDTHBYTES(dwWidth*((LPBITMAPINFOHEADER)lpS)->biBitCount);        \
```

Comentarii:

// Accesul la obiectul document al vederii curente (asociate cu imaginea deschisă în fereastra activă)
CDibDoc* pDocSrc=GetDocument();

//Accesul la template-ul documentului
CDibDoc* pDocDest=(CDibDoc*) pDocTemplate->CreateNewDocument();

//Creează obiectul destinaţie având acelaşi template ca şi documentul sursă
CDibDoc* pDocDest=(CDibDoc*) pDocTemplate->CreateNewDocument();

//Obţine handle-ul la imaginea sursă
HDIB hBmpSrc=pDocSrc->GetHDIB();

//Creează o copie a handle-ului imaginii sursă în imaginea destinaţie
HDIB hBmpDest = (HDIB)::CopyHandle((HGLOBAL)hBmpSrc);

//Obţine pointer-ul în memorie la începutul imaginilor destinaţie respectiv sursă
BYTE* lpD = (BYTE*)::GlobalLock((HGLOBAL)hBmpDest);
BYTE* lpS = (BYTE*)::GlobalLock((HGLOBAL)hBmpSrc);

//Obţine numărul de intrări din LUT (pentru imagini indexate/cu paleta): iColors = 2^n-1
// n = 1, 4 or 8 (nr. de biţi/pixel)
//Pentru imagini fara paleta (RGB cu n = 16, 24 sau 32 biţi/pixel): iColors = 0 !!!
int iColors = DIBNumColors((char *)&(((LPBITMAPINFO)lpD)->bmiHeader));

```
//Obţine pointer-ul la începutul LUT
RGBQUAD *bmiColorsDst = ((LPBITMAPINFO)lpD)->bmiColors;
RGBQUAD *bmiColorsSrc = ((LPBITMAPINFO)lpS)->bmiColors;

//Obţine pointer-ul la începutul datelor din bitmap (pixeli) din imaginea destinaţie respectiv sursă
BYTE * lpDst = (BYTE*)::FindDIBBits((LPSTR)lpD);
BYTE * lpSrc = (BYTE*)::FindDIBBits((LPSTR)lpS);

// Obţine lăţimea şi înălţimea imaginii bitmap (datele imagine – pixeli)
DWORD dwWidth  = ::DIBWidth((LPSTR)lpS);
DWORD dwHeight = ::DIBHeight((LPSTR)lpS);

//Obţine lăţimea (în octeţi) a unei linii din imagine din memorie în număr întreg de dublu-cuvinte pentru o imagine bitmap (1 dublu-cuvânt = 4 octeţi = 32 biţi); biBitCount conţine numărul de biţi/pixel
DWORD w=WIDTHBYTES(dwWidth*((LPBITMAPINFOHEADER)lpS)->biBitCount);
```

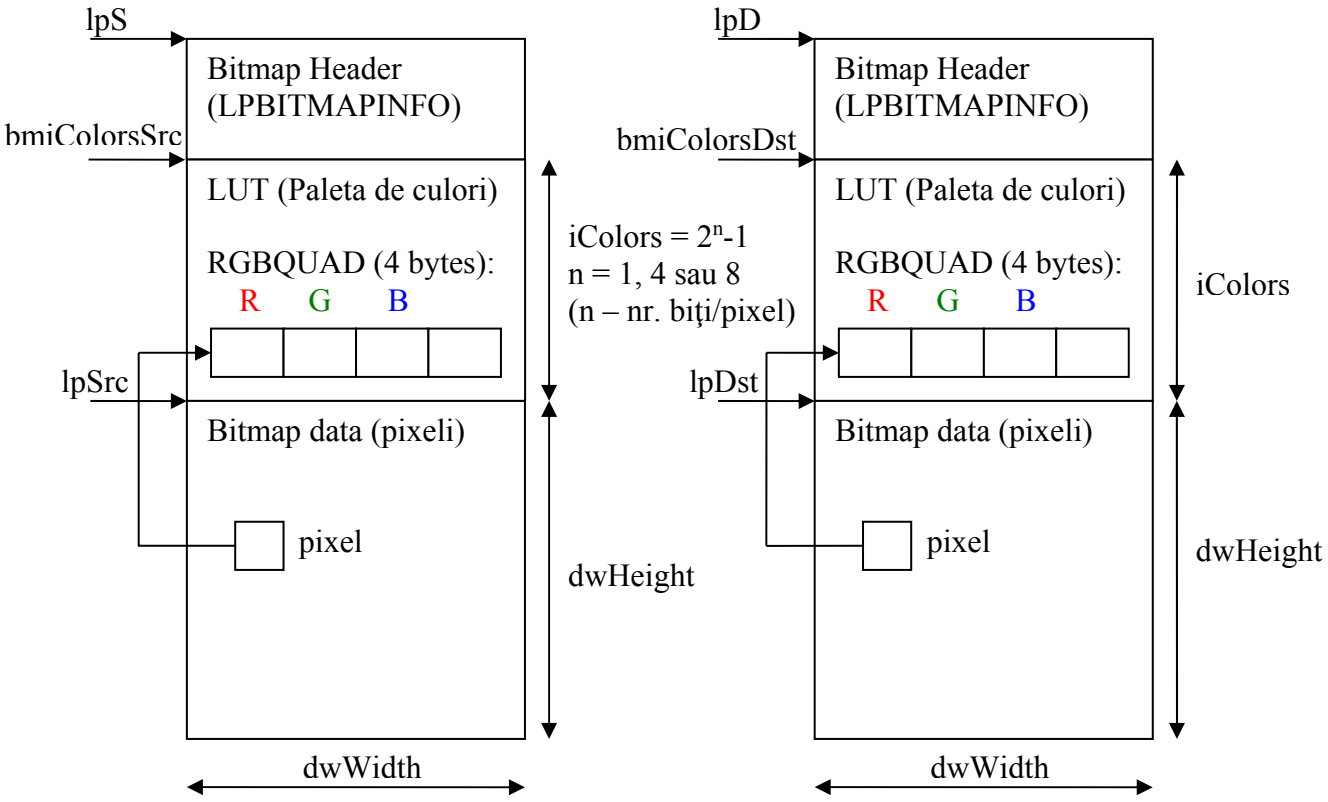

Fig. 1.7. Structura unei imagini bitmap (cu LUT – 1, 4 sau 8 biţi / pixel) în memorie (imaginea sursă şi imaginea destinaţie).

1.4.2. Definiţia macroului: `SFARSIT_PRELUCRARI("Operation name");`

```
CString Titlu;
#define SFARSIT_PRELUCRARI(Titlu)                                    \
      ::GlobalUnlock((HGLOBAL)hBmpDest);                             \
      ::GlobalUnlock((HGLOBAL)hBmpSrc);                              \
      EndWaitCursor();                                               \
```

```
pDocDest->SetHDIB(hBmpDest);                                                \
pDocDest->InitDIBData();                                                    \
pDocDest->SetTitle((LPCSTR)Titlu);                                          \
CFrameWnd* pFrame=pDocTemplate->CreateNewFrame(pDocDest,NULL);              \
pDocTemplate->InitialUpdateFrame(pFrame,pDocDest);
```

Observaţii:

//Eliberarea handle-urilor imaginilor bitmap destinaţie respectiv sursă
::GlobalUnlock((HGLOBAL)hBmpDest);
::GlobalUnlock((HGLOBAL)hBmpSrc);

//Setarea handle-ului imaginii destinaţie şi iniţializarea altor date în obiectul Document asociat
pDocDest->SetHDIB(hBmpDest);
pDocDest->InitDIBData();
pDocDest->SetTitle((LPCSTR)Titlu);

//Crearea unui frame pentru imaginea destinaţie (rezultate) şi actualizează conţinutul acestuia cu imaginea procesată
CFrameWnd* pFrame=pDocTemplate->CreateNewFrame(pDocDest,NULL);
pDocTemplate->InitialUpdateFrame(pFrame,pDocDest);

1.4.3. Accesul la paleta de culori (LUT)

LookUp Table-ul (Paleta de culori) poate fi accesată prin intermediul pointer-ului *bmiColorsSrc / bmiColorsDst*. Este un tabel cu intrări de 4 octeţi (structură RGBQUAD) conţinând un octet pentru fiecare culoare (R,G,B) şi unul rezervat.

În exemplul dat intrările din LUT ale imaginii destinaţie sunt egalate cu indexul lor, obţinând o imagine grayscale.

```
// Makes a grayscale image by equalizing the R, G, B components from the LUT
for (int k=0;  k < iColors ; k++)
     bmiColorsDst[k].rgbRed=bmiColorsDst[k].rgbGreen=
          bmiColorsDst[k].rgbBlue=k;
```

1.4.4. Accesul la pixelii unei imagini bitmap cu paletă

Pixelii unei imagini bitmap cu 8 biţi/pixel pot fi accesaţi ca şi în exemplul de mai jos:

```
// Goes through the bitmap pixels and performs their negative
for (int i=0;i<dwHeight;i++)
     for (int j=0;j<dwWidth;j++)
          lpDst[i*w+j]= 255 - lpSrc[i*w+j]; //makes image negative
```

Locaţia pixelului curent (*i,j*) din imaginea bitmap este la adresa *i*w+j* relativ la începutul zonei de date a bitmap-ului. *w* este lăţimea unei linii în octeţi astfel încât să fie un multiplu întreg de dublu-cuvinte (1 dublu-cuvânt = 4 octeţi = 32 biţi, din considerente de acces rapid la memorie).

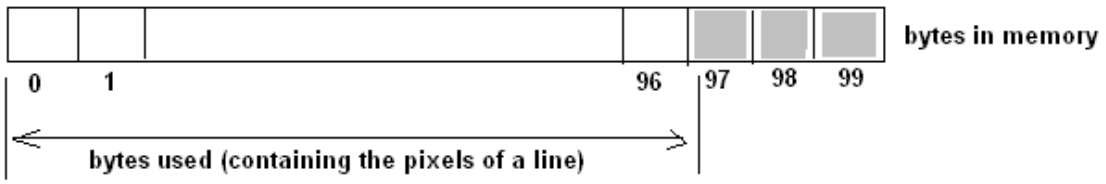

Fig. 1.8. Exemplu cum o linie de 97 de pixeli este reprezentată în memorie.

1.4.5. Accesul la pixelii unei imagini bitmap RGB (fară paletă)

Imaginile cu 16, 24, 32 biţi/pixel nu au paletă de culori. Informaţia de culoare (cele 3 componente: R, G, B) sunt conţinute în matricea de pixeli (bitmap data). În cele ce urmează (Fig. 1.9) se va exemplifica structura unei imagini bitmap cu 24 biţi/pixel (denumită şi RGB24).

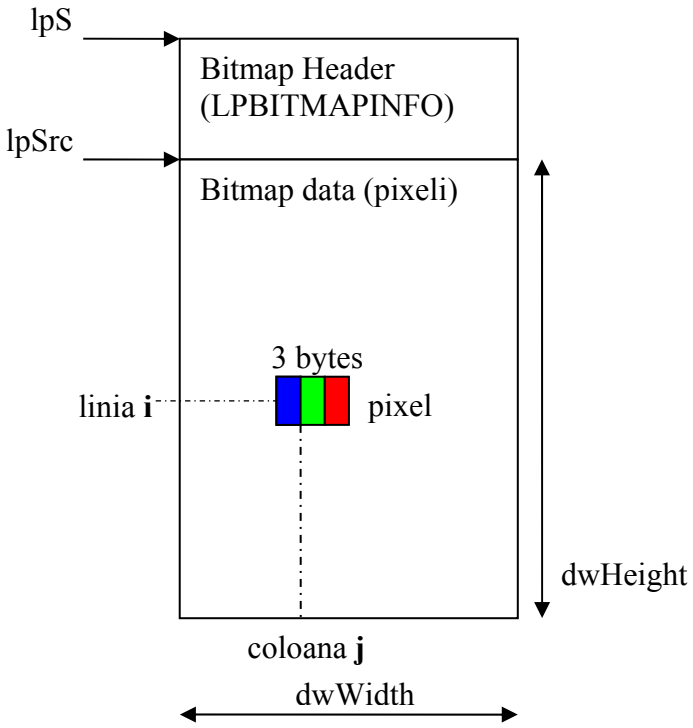

Fig. 1.9. Structura unei imagini bitmap cu 24 biţi/pixel (RGB24) în memorie.

Pixelii (componentele de culoare ale fiecărui pixel) pentru o imagine bitmap RGB24 pot fi accesaţi ca şi în exemplul de mai jos:

```
INCEPUT_PRELUCRARI();
BYTE red, green, blue;

for (int i=0;i<dwHeight;i++)
     for (int j=0;j<dwWidth;j++)
     {
          red =lpSrc[i*w+3*j+2];
          green = lpSrc[i*w+3*j+1];
          blue = lpSrc[i*w+3*j];
     }
...
```

1.5. Activităţi practice

1. Creaţi o copie a aplicaţiei *DIBLook* în directorul vostru local de lucru.
2. Deschideţi *diblook.sln* (fişierul corespunzător soluţiei) în Visual C++ 10.0.
3. Compilaţi şi executaţi aplicaţia.
4. Testaţi funcţia exemplu pusă la dispoziţie: *Processing->Parcurgere simpla*
5. Adăugaţi un meniu nou şi funcţia de procesare asociată (utilizând paşii din secţiunea 1.3 şi exemplul din secţiunea 1.4).
6. Aplicaţi operaţii aritmetice simple pe pixelii imaginii de intrare (adunare/scădere/înmulţire cu o constantă) şi puneţi rezultatul în pixelii corespunzători ai imaginii destinaţie. Adăugaţi condiţii suplimentare pentru a normaliza rezultatele (valorile pixelilor de ieşire/destinaţie) în domeniul BYTE (0…255).
7. Închideţi proiectul (mediul Visual Studio) şi apoi executaţi fişierul *clean.bat* aflat în directorul proiectului pentru a-l curăţa de fişierele rezultate în urma compilării.
8. **Salvaţi-vă ceea ce aţi lucrat. Utilizaţi aceeaşi aplicaţie în laboratoarele viitoare. La sfârşitul laboratorului de procesarea imaginilor va trebui să vă prezentaţi propria aplicaţie cu algoritmii implementaţi!!!**

Referinţe

[1] http://msdn2.microsoft.com/en-us/library/ms632591(VS.85).aspx
[2] http://www.functionx.com/visualc/Lesson05.htm
[3] http://msdn2.microsoft.com/en-us/library/4x1xy43a(VS.80).aspx
[4] http://msdn2.microsoft.com/en-us/library/d06h2x6e(VS.71).aspx