LUCRAREA DE LABORATOR 1

MEDIUL INTEGRAT C++ BUILDER

Obiectivele lucrării

a) Însuşirea modului de utilizare a celor mai importante componente ale mediului integrat C++ BUILDER . Realizarea unui program simplu care utilizează componente de tip *TButton, TEdit, Tlabel, RadioButton* etc.

b) Însuşirea modului de utilizare a componentei VCL **TTimer.** Însuşirea modului de utilizare a funcţiilor de lucru cu timpul sistem. Realizarea unor aplicaţii de gestionare a resursei timp.

c) Însuşirea modului de utilizare a componentelor VCL **TPaintBox** şi **TPanel.** Însuşirea modului de utilizare a principalelor funcţii grafice ale mediului C++BUILDER . Realizarea unor elemente pentru afişarea grafică a informaţiei (diagramă şi bargraf).

Indicatii teoretice

Facilităţile mediului C++Builder

Borland C++Builder este un mediu de programare vizual, orientat obiect, pentru dezvoltarea rapidă de aplicaţii (**RAD**) cu scop general şi aplicaţii client/server pentru Windows şi WindowsNT. Folosind C++Builder se pot crea aplicaţii Windows eficiente sciind un minim de cod. Facilităţile semnificative oferite de acestea sunt prezentate succint în cele ce urmează.

Înalta productivitate a mediului de dezvoltare

Aceasta este favorizată de principalele instrumente furnizate de mediul de dezvoltare integrat (**IDE**) C++Builder şi anume :

- *Visual Form Designer;*
- *Object Inspector;*
- *Component Palette;*
- *Project Manager;*
- *Code Editor;*
- *Debugger.*

Acestea dau posibilitatea utilizatorului să dezvolte rapid aplicaţii având totodată un control complet asupra codului şi resurselor.

Proiectare drag-and-drop

Utilizatorul poate crea aplicaţii prin simpla *tragere* (drag and drop) a componentelor din *Component Palette* pe *Form designer* urmată de setarea propietăţilor din *Object Inspector*. *Handler-ele* de evenimente sunt automat create, iar codul lor este complet accesibil. Acest mod de proiectare a unei aplicaţii nu restricţionează în nici un fel accesul programatorului la codul sursă, o aplicaţie putând fi scrisă şi fără a folosi componente vizuale.

Propietăţi, metode, evenimente

Dezvoltarea rapidă a aplicaţiilor înseamnă suport pentru propietăţile, metodele şi evenimentele obiectelor (*PME*). Propietăţile permit setarea uşoară a caracteristicilor componentelor. Metodele execută acţiuni asupra obiectelor. Evenimentele permit ca aplicaţia să răspundă la mesajele Windows, sau la schimbări de stare a obiectelor. Folosirea modelului PME furnizează un robust şi intuitiv mediu de dezvoltare pentru aplicaţiile Windows.

C++Builder Help

Mediul C++Builder oferă un ghid practic, care conţine peste 3000 de pagini de documentaţie despre IDE, VCL, baze de date şi tehnici de programare.

Codurile sursă pentru VCL

Mediul C++Builder pune la dispoziţie codurile sursă pentru *VCL – Visual Component Library*, furnizând astfel o unică privire înăuntrul modului în care lucrează C++Builder. VCL furnizează peste 100 de componente reutilizabile care ajută programatorul să construiească aplicaţii robuste într-un timp scurt. Aceste componente pot fi modificate pentru a corespunde necesităţilor din cele mai diverse. C++Builder –ul incude o suită completă de controale Windows: TreeView, Trackbars, ProgressBars, toolbars, Rich Edit, ListViews, ImageLists, StatusBars etc. Totodată C++Builder include suport pe 32 de biţi pentru numele lungi de fişiere, multi-threading şi Win API.

1 IDE (Mediul de Dezvoltare Integrat)

Elementele mediului integrat de dezvoltare sunt:

- Meniu principal (Main Menu);
- Forma (Form);
- Editorul de cod (Code Editor);
- Bara cu instrumente (Toolbar);
- Paleta cu componente (Component Palette);
- Tabelul cu proprietăţi ale obiectelor (Object Inspector);
- Administratorul de program (Program Manager).

MAIN MENU

În figura 1.1 se prezintă funcţiile specifice meniului principal.

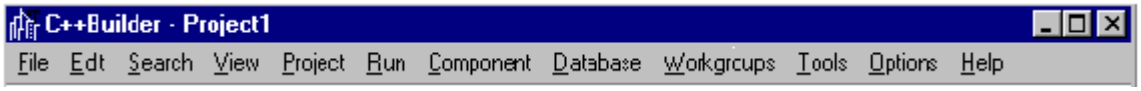

Fig. 1.1- Funcţiile specifice meniului principal

Semnificaţiile butoanelor din Main Meniu sunt următoarele:

File pentru a deschide, crea, salva, închide project-uri şi fişiere;
Edit pentru prelucrare de texte şi componente;
Search pentru localizare de text, erori, obiecte, variabile, unit-uri, ...în editorul de cod;
View pentru a afişa, sau ascunde elemente ale mediului;
Project pentru a compila o aplicaţie;
Run pentru a executa şi a depana o aplicaţie.
Component pentru a crea sau a instala o componentă.
DataBase pentru manipulare de baze de date.
Workgroups pentru manipularea proiectelor mari.
Tools pentru a rula programele utilitare disponibile, fără a părăsi mediul C++Builder;
Options pentru a controla comportamentul mediului de dezvoltare;
Help pentru a obţine ajutor în diversele faze de utilizare a mediului.

FORMA (Form)

Înreaga parte vizibilă a unei aplicaţii este construită pe un obiect special numit **formă**(ca cea din figura 1.2).
O formă liberă este creată de fiecare dată când este lansat în execuţie mediul C++Builder. O aplicaţie poate avea mai multe forme. Adăugarea de noi forme unei aplicaţii se face selectând comanda New Form din meniul File. Pe formă se pot aşeza şi aranja componente vizulale şi non-vizuale care alcătuiesc interfaţa cu utilizatorul. Fiecărei forme îi sunt asociate două fişiere cu exensiile **.cpp** respectiv **.h** (în cazul formei de mai sus **unit1.cpp şi unit1.h**)

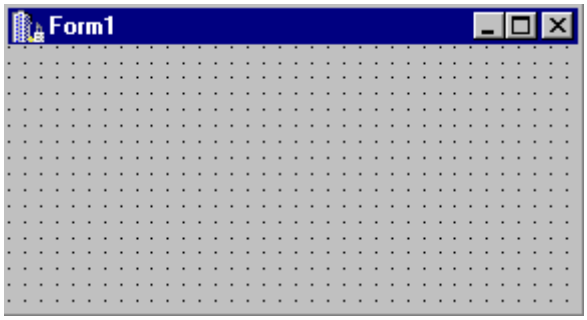

Figura 1.2 - Editorul de cod

Mediul C++Builder are o fereastră unde programatorul poate scrie codul unei aplicaţii.. Editorul de cod este un editor ASCII complet şi poate deschide mai multe fişiere simultan. Fiecărui fişier deschis îi este ataşat, în partea de sus a ferestrei, un buton cu numele lui. Trecerea de la un fişier la altul se face prin click pe butonul ataşat fişierului. La intrarea în C++Builder, sau la crearea unui nou project, editorul arată ca în figura 1.3.

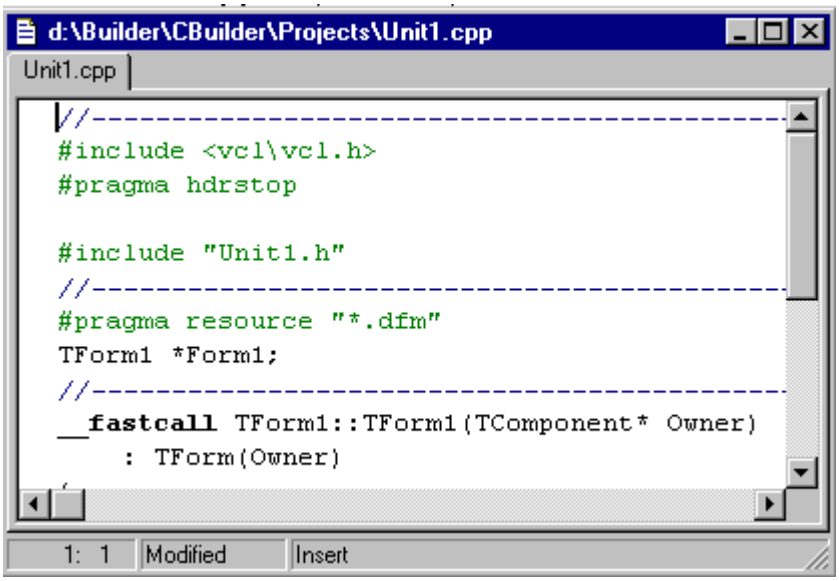

Fig 1.3 – Editor de cod

În momentul în care adăugăm un *unit*, sau o *formă* în editorul de cod se va creea o nouă fereastră.

Prin execuţia unui click dreapta în editorul de cod mediul C++Builder pune la dispoziţie un meniu cu comenzile cele mai des folosite la editarea, depanarea, execuţia a unei aplicaţii.

Prin selecţia opţiunii *Properties*, din acest meniu, avem acces la setările de bază ale editorului, şi anume la culori, fonturi etc. – figura 1.4..

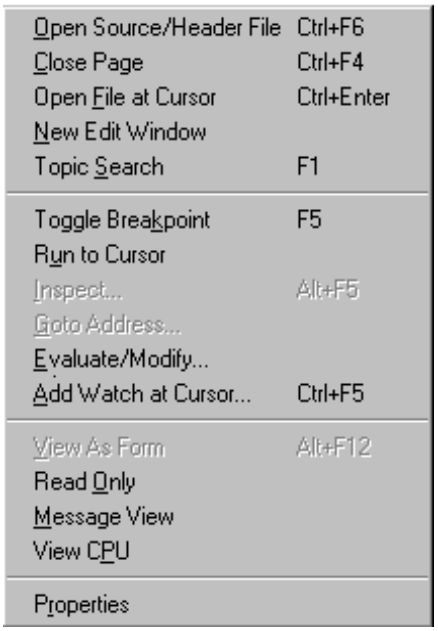

Fig. 1.4 – Setarile de baza a editorului

Bara cu instrumente

Aceasta reprezintă o scurtătură la comenzile aflate în MainMenu.

În varianta implicită comenzile pe care le conţine sunt cele specificate în figura 1.5.

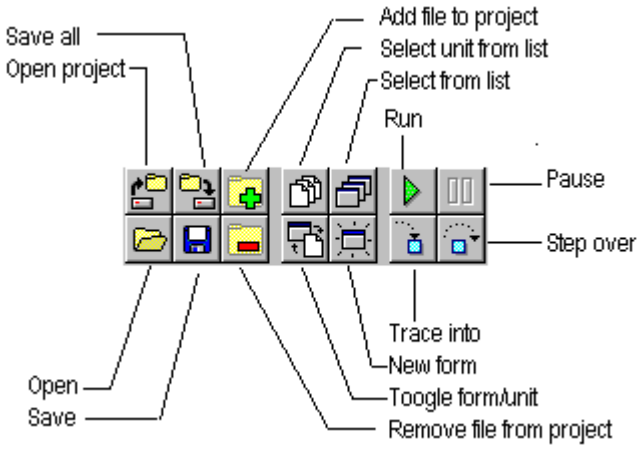

Fig. 1.5 – Comenzile din MainMenu

Paleta cu componente

Componentele sunt elemente utilizate pentru a crea aplicaţii C++Builder. O componentă este de fapt un element de tip *UI (user interface).* Pot fi vizuale (de exemplu butoanele, cutiile de dialog), sau pot fi non-vizuale (de exemplu timer-ul). Spunem despre o componentă că este *vizuală*, dacă ea este vizibilă, sau va fi vizibilă la momentul execuţiei, iar o componentă este *non-vizuală*, dacă la momentul proiectării aplicaţiei apare pe formă ca un desen, iar în momentul execuţiei aplicaţiei devine invizibilă (de exemplu TTimer din pagina System), sau este invizibilă până în momentul în care este apelată (de exemplu TOpenDialog sau TSaveDialog din pagina Dialogs). Fiecare componentă are atribute care permit controlul aplicaţiei. Componentele sunt grupate în pagini. În forma implicită paginile sunt:

Standard, Win95, Additional, Dat Access, Data Control, Win31, Internet, Dialogs, System, QReport, ActiveX – figura 1.6

Fig. 1.6 - Paleta de componente

De exemplu cele mai folosite componente sunt cele din pagina Standard, care conţine cutii de dialog, meniuri, butoane etc. Pentru a obţine help despre fiecare dintre ele, executaţi click pe componenta dorită, iar apoi apăsaţi pe F1.

O componentă specială este şi forma, care are la rândul ei ataşte propietăţi, metode, evenimente etc.

Aşezarea unei componente pe o formă se poate face în mai multe moduri:

- dacă dorim plasarea componentei în mijlocul formei atunci executăm dublu click pe forma respectivă;
- dacă dorim să plasăm componenta în alt loc decât centrul formei, atunci executăm un click pe onentei va coincide cu locul unde am executat cel de-al doilea click.

În aceste două cazuri dimensiunile componentei vor fi cele implicite. Se pot modifica aceste dimensiuni, fie din Object Inspector (vezi mai jos), fie cu ajutorul mouse-ului.

În cazul în care ştergem o componentă de pe formă, şablonul handler-ului de evenimente asociat componentei va rămâne (însă fără codul sursă existent înainte) deoarece acesta ar putea fi apelat de către alte metode. Dacă programatorul intervine în codul sursă şi şterge acest handler, compilatorul va da o eroare.

Tabelul cu proprietăţi ale obiectelor

Acest tabel (*Object Inspector*) care face legătura între interfaţa aplicaţiei şi codul scris de programator are două funcţii:

— setează propitetăţile componentelor aflate în formă;
— creează şi ajută la navigatul prin handler-ele de evenimente.
Un handler de evenimente se execută în momentul în care apare un eveniment (de exemplu apăsarea unei taste, a unui buton de mouse etc.).

Object Selector

În capătul de sus al lui se află Object Selector care conţine toate componentele de pe formă împreună cu tipul lor.

Object Inspector are două pagini – figurile 1.7, 1.8.:

1) Properties page (figura 1.7) permite stabilirea (setarea) propietăţilor unei componente, şi anume: dimensiunile ei, poziţia în cadrul formei, fonturile folosite, numele etc. Alte propietăţi pot fi setate la momentul execuţiei programului prin scrierea de cod sursă în cadrul handler-lor de evenimente.

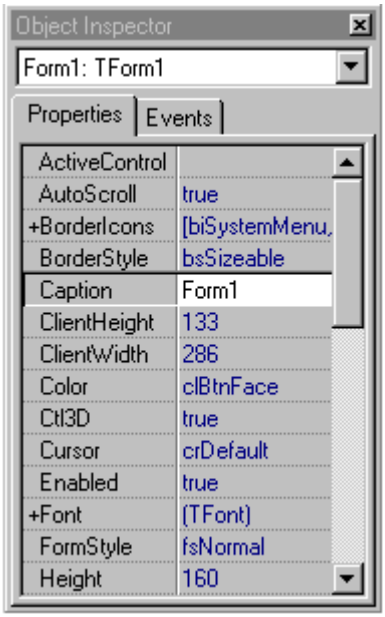

Fig. 1.7 - Properties page

2) Event page (figura 1.8) permite legarea unei componente la evenimentele programului. Prin executarea unui dublu click pe un eveniment, de exemplu pe *OnClik*, C++Builder creează un handler de evenimente, care este de fapt o metodă ataşată unei clase şi care se va executa când apare un eveniment particular (în cazul nostru executarea unui click pe buton).

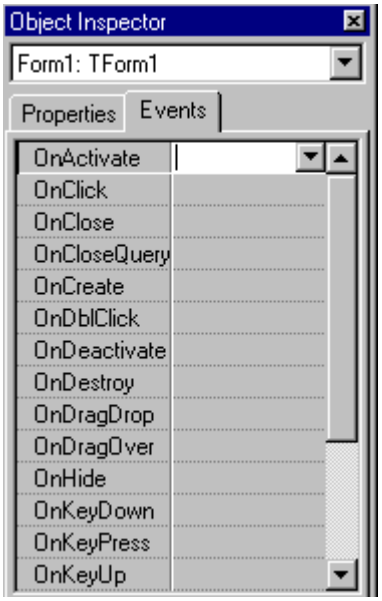

Fig 1.8 - Event page

Un handler de evenimente pentru componenta **TButton** din cadrul paginii Standard va arăta în felul următor:

```
void_fastcall Tform1::Button1Click(TObject *Sender)
{
        //Aici putem scrie cod sursă, care se va executa
```

```
        //în momentul apăsării butonului cu ajutorul mouse-ului
}
```

În cazul în care alegem evenimentul **OnDblClick** (OnDoubleClick) handler-ul de evenimente va arăta în felul următor (am ales în cazul acesta componenta **RadioButton**):

```
   void_fastcall Tform1::RadioButton1DblClick(TObject *Sender)
{
        //Aici putem scrie cod sursă, care se va executa
        //în momentul apăsării butonului cu ajutorul mouse-ului

}
```

Precun se vede şi din cele două exemple formele şi componentele sunt numerotate (*TForm1*, *RadioButton1*). Dacă am mai fi adăugat pe formă încă un RadioButton acesta ar fi fost denumit *RadioButton2* etc. Object Inspector denumeşte automat toate componentele şi handler-ele de evenimente.

Pagina cu evenimente va afişa la un moment dat doar evenimentele unei singure componente şi anume a aceleia selectate din formă (poate fi chiar forma însăşi).

Administratorul de program

Aplicaţiile C++Builder sunt denumite **project-uri**. Un project este un grup de mai multe fişiere care împreună alcătuiesc aplicaţia. Fiecare fişier reprezintă, pentru aplicaţia din care face parte, o "resursă" care necesită setări speciale pentru a putea fi legată la aplicaţiea finală (DLL, sau EXE). Pe măsură ce aplicaţia creşte, numărul de fişiere necesare va fi tot mai mare (de exemplu pot fi adăugate fişiere multimedia, baze de date, unit-uri Pascal etc.) şi deci se va face tot mai simţită nevoia unei manipulări cât mai uşoare a acestor fişiere. C++Builder se ocupă de manipularea tuturor acestor fişiere şi totodată pune la dispoziţia programatorului un instrument care să permită acestuia să navigheze uşor printre fişierele ce alcătuiesc un project. Acest instrument este denumit sugestiv *Project Manager*.

În structura unui *project* intră trei tipuri de fişiere (cu extensiile **.mak, .cpp şi .res).**

Fişierul .cpp (cu numele implicit *Project1.cpp*) este punctul central al unei aplicaţii C++ Builder. El conţine funcţia WinMain, deci de aici îşi începe programul execuţia. În continuare se prezintă un exemplu de program simplu care conţine doar acest fişier:

```
//---------------------------------------------------------------------
1 #include <vcl\vcl.h>
2 #programa hdstop
//---------------------------------------------------------------------
3 USERES ("Project. res");
//---------------------------------------------------------------------
4 WINAPI WinMain (HINSTANCE, HINSTANCE, LPSTR, int)
{
5           try
            {
6           Application->Initialize( );
7           Application->Run ( );
            }
8           catch (Exception &exception)
            {
9           Application->ShowException (&exception);
            }
            return 0;
     }
   //---------------------------------------------------------------
```

Am numerotat liniile programului pentru a face mai uşoară explicarea lor:

linia 1: fişierul ***vcl.h*** conţine definiţiile necesare pentru Visual Component Library (VCL). Acest fişier este inclusîn fiecare project C++Builder.

linia 2: atenţionează procesorul să nu mai adauge alte fişiere handler la lista celor deja existente (acestă directivă termină lista cu fişiere handler).

linia 3: USERES este un macrou folosit pentru a adăuge resurse la project. În acest context precizăm că mai există şi alte macrouri dintre care:

- USEFORM (pentru a adăuga forme la un project);
- USEOBJ (pentru a adăuga fişiere obj la project);
- USEUNIT (pentru a adăuga unit-uri Object Pascal la project);
- USEDATAMODULE;
- USEDATAMODULENS;
- USERC;
- USEFORMNS;
- USELIB;
- USEFILE.

C++Builder creează automat aceste macrouri, aşa că nu este necesar ca programatorul să intervină în ele.

linia 4: reprezintă apelul funcţiei WinMain (de aici începe execuţia programului).

liniile 5 şi 8: aceste instrucţiuni ţin de programarea cu excepţii.

linia 6: se iniţializează obiectul VCL Aplication pentru acest process.

linia 7: procesul este lansat în execuţie (de aici programul începe să se ruleze în mediul Windows).

În continuare se prezintă programul sursă demonstrativ Unit1.cpp :

```
//---------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------
#pragma resource "*.dfm"
TForm1 *Form1;
// Form1 este un  pointer care acceseaza aplicatia
// TForm este clasa care defineste fereastra aplicatiei
// TForm1 este clasa derivata din TForm1
//---------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
            : TForm(Owner)
// Constructorul clasei si al aplicatiei
// Constructorul este o metoda speciala de initializare
{
            Label1->Caption="";
// Se reseteaza proprietatea Caption pentru Label1 si anume egal ""
}
//---------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
```

```
// Numele    functiei (Button1Click) si apartenenta( la clasa TForm1) sunt
          generate automat
// Button1Click este generata automat la apsarea butonului Button1
// Button1 este butonul pe care este afisat textul GO
{
Edit1->Text="Bun venit !";
//Se modifica proprietatea Text a obiectului Edit1 si anume se incarca cu
          valoarea
// specificata (mesajul dintre ghilimele)
Label1->Caption="Specializarea Tehnologii Informationale";
//Se modifica proprietatea Caption a obiectului Label1 si anume se incarca cu
          valoarea
// specificata  (mesajul dintre ghilimele)
Button1->Enabled=false;
//  Se modifica proprieataea Enabled a obiectului Button1 si anumue se
          dezactiveaza
Button2->Enabled=true;
//  Se modifica proprieataea Enabled a obiectului Button2 si anumue se
          activeaza
}
//---------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
 // Button3Click este generata automat la apsarea butonului Button3
 // Button3 este butonul pe care este afisat textul Exit
 Close();
 // Se aplica metoda Close() care inchide aplicatia curenta

}
//---------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
//Au loc modifcari asemanatoare cu functia Button1Click
Edit1->Text="";
Label1->Caption="";
Button2->Enabled=false;
Button1->Enabled=true;
}
//---------------------------------------------------------
```

2 Utilizarea componentei TTimer

Componenta TTimer se găseşte în **Component Palette** (*pagina System*) .
Obiectul de acest tip permite execuţia in cadrul aplicaţiei a unor funcţii la intervale specificate. În context Windows obiectul TTimer lansează către aplicaţie mesaje la intervale prestabilite.

O particularitate faţă de componentele utilizate în lucrarea precedentă constă in faptul ca acest obiect nu are corespondent grafic pe formă în momentul execuţiei programului.

In figura 2.1 este reprezentat obiectul vizual Timer asa cum se găseşte în pagina System. Aducerea pe formă a acestui obiect se realizează in conformitate cu precizările din lucrarea precedentă.

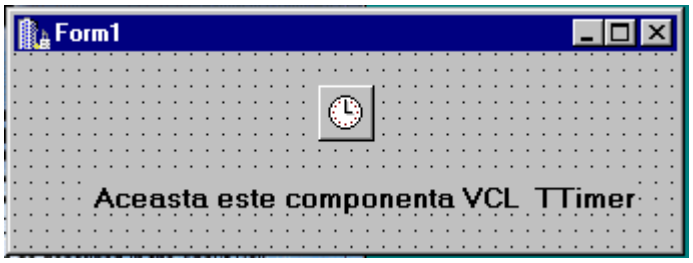

Fig. 2.1 - Obiectul vizual Timer

În figura 2.2 se prezintă Tabelul cu Proprietăţi (*Object Inspector*) paginile *Proprietăţi şi Evenimente* pentru componenta TTimer.

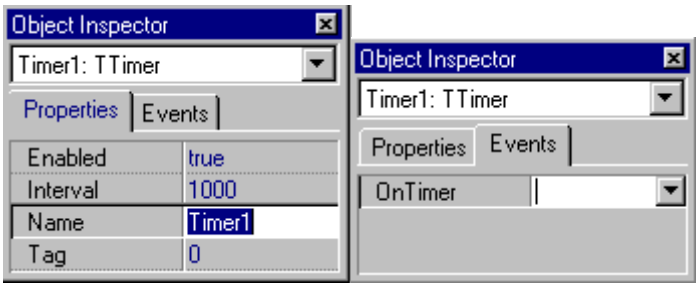

Fig 2.2 - Tabelul cu Proprietăţi

— Proprietatea *Enabled* stabileşte dacă obiectul TTimer răspunde la evenimentul *OnTimer* (dacă Enabled este *true* atunci se răspunde la eveniment – Timer-ul este activat) ;
— Proprietatea *Interval* specifica numărul de milisecunde dintre două mesaje tip TTimer consecutive (TTimer este apelat dupa fiecare trecere a intervalului specificat) ;
— Proprietatea *Name* specifică numele Timer-ului;
— Proprietatea *Tag* este utilizata pentru transferul de informatii suplimentare (variabile de tip int) ;
— Evenimentul *OnTimer* apare de fiecare dată când trece intervalul (*Interval)* specificat.

În continuare este prezentat codul programului **P1** care realizează:

— dezactivarea timer-ului Timer1 (in constructor si la apăsarea butonului *Dezactivare Timer) ;*

— activarea Timer-ului Timer1 (la apasarea butonului *Activare Timer*);

— transferul unei valori întregi , prin intermediul proprietăţii *Tag* in componenta Label1.

```
//-----------Programul-P1-----------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "ex_timer.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"

int i=0;

TForm1 *Form1;
//---------------------------------------------------------------------
void__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
    Label1->Caption="";
    Timer1->Enabled=false;
    DesBtn->Enabled=false;
    ActivBtn->Default=true;
}
//---------------------------------------------------------------------

void __fastcall TForm1::ActivBtnClick(TObject *Sender)
{
    Timer1->Enabled=true;
    ActivBtn->Enabled=false;
    DesBtn->Enabled=true;
}
//---------------------------------------------------------------------
void __fastcall TForm1::DesBtnClick(TObject *Sender)
{
    Timer1->Enabled=false;
    ActivBtn->Enabled=true;
    DesBtn->Enabled=false;
}
//---------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    i++;
    Timer1->Tag=i;
    Label1->Caption=Timer1->Tag;
}
//-----------Programul-P1-----------------------------------------------
```

In figura 2.3 se prezintă dispunerea pe formă a obiectelor din programul **P1**.

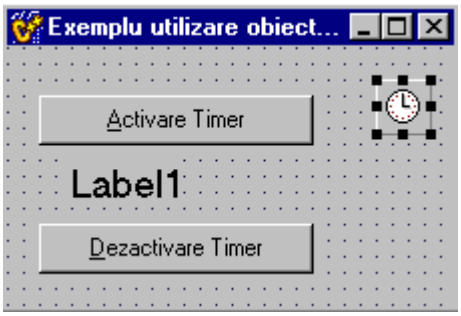

Fig. 2.3 – Rezultatul programului P1

Funcţii de gestionare a timpului

Prototipurile funcţiilor de gestionare a timpului sistem se găsesc in header-ele **dos.h , time.h , vcl\sysdefs.h.** Precizările următoare se referă la header-ul **dos.h**., în care se găsesc prototipurile urmatoarelor funcţii ce permit gestionarea resursei timp:

```
 void getdate(struct date *datep)
void gettime(struct time *timep)
void setdate(struct date *datep)
void settime(struct time *timep)
```

in care: ***datep** si ***timep** sunt pointeri la structurile **struct date** si **struct time**;

Structurile menţionate mai sus (aferente datei şi orei) sunt de forma:

```
struct date
          {
            int  da_year;     /* pentru an */
            char da_day:     /* pentru  zi */
            char da_ mon;    /* pentru luna */
          };
```

respectiv:

```
struct time
          {
            unsigned char ti_min;        /* pentru minute */
            unsigned char ti_hour;       /* pentru ore */
            unsigned char ti_hund;       /* pentru sec/100 */
            unsigned char ti-sec;   /* pentru secunde */
          };
```

Funcţiile **`getdate`** si **`gettime`** preiau data şi ora curente iar funcţiile **`setdate`** şi **`settime`** permit iniţializarea acestora.

Modurile de utilizare a functiilor **`getdate`** si ***g*`ettime`** sunt evidenţiate în programul **P2** ale cărui text sursă se prezinta in continuare.

```
//----------Programul-P2-------------------------------------------------------
#include <vcl\vcl.h>
#include <stdio.h>
#pragma hdrstop

#include "Unit3.h"
#include "dos.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TForm1 *Form1;
struct date d;
struct time t;
//---------------------------------------------------------------------------
void__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormActivate(TObject *Sender)
{
 Edit1->Clear();
 Timer1->Interval=1000;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
char buf[20];
getdate(&d);
gettime(&t);
sprintf(buf,"%02d-%02d-%4d %02d:%02d:%02d",d.da_day,d.da_mon,d.da_year,
t.ti_hour,t.ti_min,t.ti_sec);
Edit1->Text=(AnsiString)buf;
// Conversie din String in AnsiString AnsiString este o clasa care permite lucru cu
// // siruri de caractere in mediul C++Builder
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   Close();
}
//---------------------------------------------------------------------------
```

Programul **P2** afişează data şi ora cu periodicitate de o secundă (setarea proprietăţii *Interval* pentru Timer1).Ieşirea din program se face prin apăsarea butonului Exit.

3 Utilizarea componentei TPaintBox

Componenta TPaintBox se găseşte în **Component Palette** (*pagina System*) . Obiectul de acest tip furnizează o componentă *TCanvas* care permite desenarea în interiorul unui dreptunghi, prevenind depăşirea marginilor acestuia.

În figura 3.1 este prezentat obiectul vizual TPaintBox (asa cum apare pe formă) iar în figura 3.2 acelaşi obiect aşa cum apare în pagina *System*. Aducerea pe formă a acestui obiect se realizează in conformitate cu precizările din prima lucrare. Ca şi în cazul obiectului Ttimer, nici obiectul TPaintBox nu este vizibil în timpul execuţiei programului (după cum s-a menţionat acesta delimitează spaţiul de lucru pentru desenare).

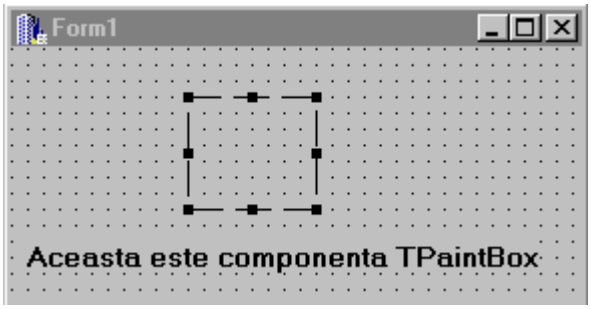

Fig 3.1 - Obiectul vizual TPaintBox

Fig. 3.2 - Componenta VCL TpaintBox

În figura 3.3 se prezintă Tabelul cu Proprietăţi (*Object Inspector*) paginile *Proprietăţi şi Evenimente* pentru componenta TPaintBox.

- Proprietatea *Name* specifică numele PaintBox-ului.
- Proprietatea *Tag* este utilizata pentru transferul de informatii suplimentare (variabile de tip int).

Restul elementelor din figura 3.3 sunt cunoscute de la lucrările de laborator anterioare.

Utilizarea componentei TPanel

Componenta TPanel se găseşte în **Component Palette** (*pagina Standard*) . Obiectul de acest tip poate fi utilizat pentru desenare dacă pe el se amplasează o componentă TPaintBox. În prezenta lucrare obiectul TPanel va fi utilizat pentru realizarea bargrafului.

În figura 3.4 este prezentat obiectul vizual TPanel (aşa cum apare pe formă) iar în figura 3.5 acelaşi obiect aşa cum apare în pagina *Standard*. Aducerea pe formă a acestui obiect se realizează in conformitate cu precizările din prima lucrare. Obiectul *TPanel* este vizibil în timpul execuţiei programului.

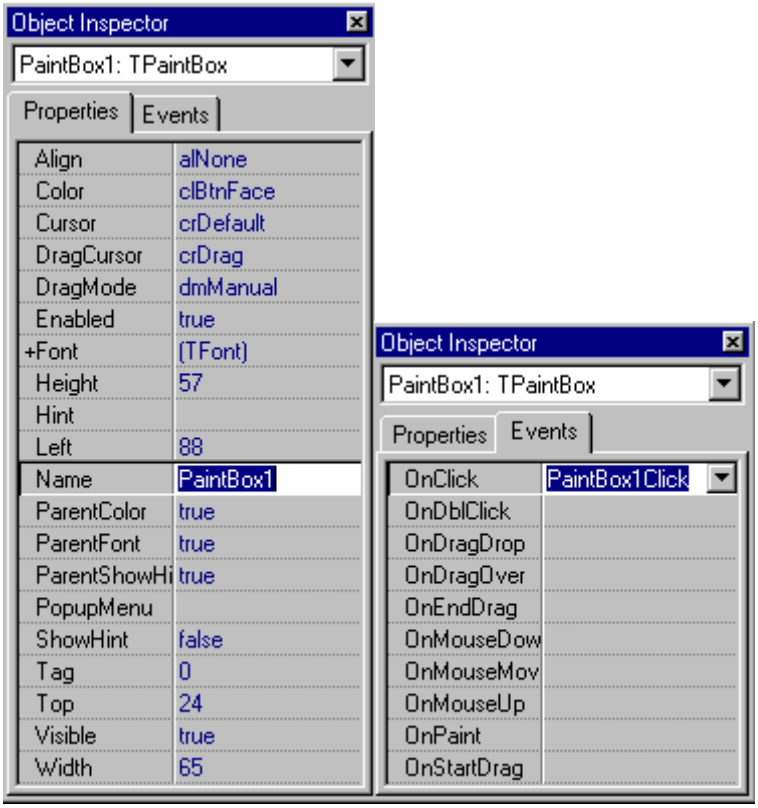

Fig 3.3 – Object inspector

Fig 3.4 - Obiectul vizual TPanel

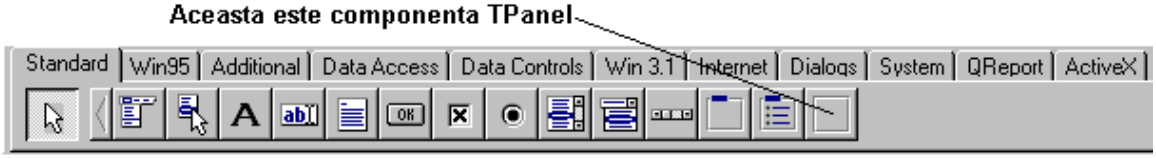

Fig 3.5 – Componenta Tpanel

În figura 3.6 se prezintă Tabelul cu Proprietăţi (*Object Inspector*) paginile *Proprietăţi şi Evenimente* pentru componenta TPanel.

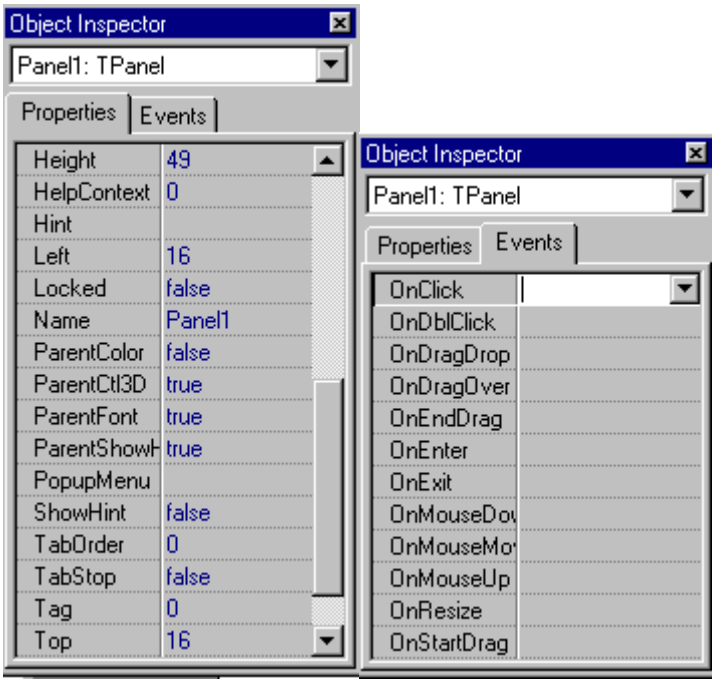

Fig 3.6 – Object Inspector

- Proprietatea *Name* specifică numele Panel-ului.
- Proprietatea *Tag* este utilizata pentru transferul de informatii suplimentare (variabile de tip int).
- Proprietăţile *Height* şi *Top* specifică înălţimea respectiv vârful (raportat la obiectul *Form*). Aceste proprietăţi se vor modifica prin program în cadrul aplicaţiei de construcţie a bargrafului.

Restul elementelor din figura 3.6 sunt fie cunoscute de la lucrările de laborator anterioare fie nesemnificative pentru prezenta lucrare.

Utilizarea principalelor obiectelor grafice ale mediului C++Builder

Obiectul TCanvas furnizează spaţiul de desenare pentru obiectele pe a căror suprafaţă se poate realiza un desen (de exemplu o linie pe un obiect *TPaintBox*). În continuare se vor prezenta proprietăţile *Brush* şi *Pen* ale obiectului *TCanvas* .

Brush - permite umplerea unui contur închis cu o culoare (proprietatea *Color*) sau cu un model de haşurare (proprietatea *Style*).

În tabelul 3.1 se prezintă codurile culorilor principale, care pot fi utilizate pentru toate obiectele grafice.

Tabelul 3.1- Codurile culorilor

Culoare	**Functie C++Builder**
negru	clBlack
albastru	clBlue
verde	clGreen
rosu	ClRed
galben	clYellow
mov	ClFuschia
maro	clMaroon
alb	ClWhite
argintiu	clSilver
gri	clGray

În tabelul 3.2 se prezintă codurile stilurilor de haşurare.

.

Tabelul 3.2 - Codificarea stilurilor de haşurare

Nume	**Descriere**
bsSolid	umple uniform toti pixelii cu culoarea selectată
bsHorizontal	haşură orizontală
bsVertical	haşură verticală
bsFDiagonal	haşură \\\ (diagonală stânga-dreapta)
bsBDiagonal	haşură /// (diagonală dreapta – stânga)
bsCross	haşură în cruce +++
bsDiagCross	haşură în cruce oblică xxx

Pen - permite fixarea unor atribute ale liniilor drepte sau curbe şi anume culoare (*color)*, grosime (*width)*, tip *(style).*

În tabelul 3.3 sunt prezentate codurile pentru stilurile de linie.

Tabelul 3.3 - odificarea stilurilor de linie

Nume	**Descriere**
psSolid	linie continuă
psDot	linie punctată
psDash	linie întreruptă
psDashDot	linie intreruptă (linie-punct)
psDashDot Dot	linie intreruptă (linie-punct-punct)

Pentru grosimi se foloseşte proprietatea *Width* a obiectului grafic *Pen* în care se specifică un număr întreg (implicit Width=1 pixel, iar alte grosimi se exprimă prin multipli 3 pixeli = 3*Width).

Pentru culori sunt valabile codurile din tabelul 3.1.

Obiectului *TCanvas* îi sunt asociate şi foarte multe metode (funcţii) dintre care în tabelul 3.4 se prezintă cele cu o frecvenţă mai mare de utilizare.

Tabelul 3.3 -Funcţii de desenare ale obiectului TCanvas

Prototip	Funcţie
MoveTo(int x,int y)	fixeaza pozitia curenta in (x,y)
LineTo (int x,int y);	linie din poz. curenta pina la (x,y)
Ellipse(int stinga,int sus, int dreapta, int jos)	desenează o elipsă tangentă la laturile dreptunghiului specificat
Rectangle(int stinga,int sus, int dreapta,int jos)	desenează un dreptunghi cu coordonatele colţurilor specificate
TextOut (int x, int y, AnsiString Text)	afişează textul *Text* începând cu punctul de coordonate x, z
FloodFill (int x, int y, TColor Culoare , TfillStyle Stil) Stil=fsSurface sau fsBorder	umple o suprafaţă,mărginită de un contur cu culoare *Culoare*, care conţine punctul de coordonate (x,y) cu stilul *Stil* (implicit *fsBorder)*
CopyRect (TRect *dest* TCanvas *Canvas*, TRect *sursa*)	copiază zona definită de dreptunghiul *sursă* în Canvas-ul *Canvas* în zona definită de dreptunghiul *dest*

Observaţii

1. Suprafaţa delimitată de un obiect TPaintBox este vazută ca o matrice de dimensiuni **Width * Height** puncte, conform figurii 3.7.

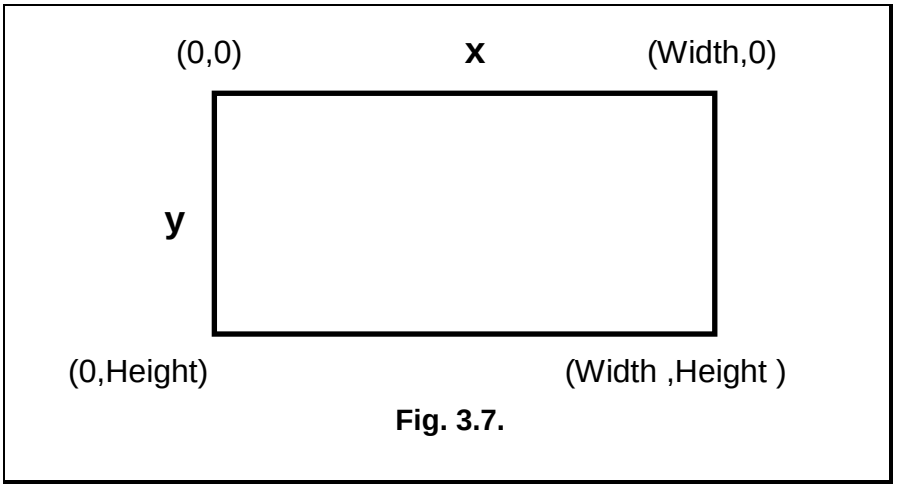

Fig. 3.7.

2. Referitor la utilizarea obiectului *CopyRect* (copiază o zonă dreptunghiulară dintr-un *Canvas* în altă zonă dreptunghiulară în acelaşi *Canvas*) se prezintă următorul exemplu edificator.

```
TRect sursa,destinatie; // variabilele  sursa si destinatie sunt de tipul TRect
sursa=Rect(stinga1, sus1, dreapta1, jos1) ; // se construieste dreptunghiul sursa
destinatie=Rect(stinga2, sus2, dreapta2, jos2) ; // se construieste dreptunghiul
destinatie
PaintBox1->Canvas->CopyRect(destinatie,PaintBox1->Canvas,sursa) ;
 //copiaza imagina incadrata de dreptunghiul sursa  in dreptunghiul destinatie
```

Se prezintă mai jos codul sursă al programului P3 care utilizează o mare parte din resursele grafice ale mediului C++Builder enumerate în indicatii teoretice (comentariile se referă întotdeauna la linia următoare).

```
//---------------------------------------------------------------------------
#include <vcl\vcl.h>
#pragma hdrstop

#include "P3.h"
//---------------------------------------------------------------------------
#pragma resource "*.dfm"
TForm1 *Form1;
int xs,ys;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
//Specifica culoarea liniei (negru)
PaintBox1->Canvas->Pen->Color=clBlack;
// Instructiune MoveTo muta cursorul grafic in pozitia X,Y
//Constructia Edit1->Text.ToInt() permite convertirea numarului care se introduce
in casuta //de editare dintr-un ansistring intr-un numar intreg
PaintBox1->Canvas->MoveTo(Edit1->Text.ToInt(),Edit2->Text.ToInt());
//Instructiunea LineTo deseneaza o linie din pozitia curenta in pozitia X,Y
PaintBox1->Canvas->LineTo(Edit3->Text.ToInt(),Edit4->Text.ToInt());
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
PaintBox1->Canvas->Pen->Color=clBlack;
//Instructiunea Rectangle deseneaza un dreptunghi care are coltul din stinga sus in
//punctul de coordonate  X1,Y1 iar  coltul  dreapta jos in  punctul de coordonate
X2,Y2
PaintBox1->Canvas->Rectangle(Edit1->Text.ToInt(),Edit2->Text.ToInt(),Edit3-
>Text.ToInt(),Edit4->Text.ToInt());
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  //dreptunghi este o variabila de tipul Trect
  TRect dreptunghi;
   //Se specifica culoarea de umplere  (culoarea pensulei = rosu)
  PaintBox1->Canvas->Brush->Color = clRed;
  // Se specifica stlilul de hasurare (bsHorizontal)
  PaintBox1->Canvas->Brush->Style = bsHorizontal;
  // Se defineste variabila dreptunghi
  //Este folosit cuvantul Rect si parametrii ca in exemplul de mai sus
  //Coltul din stigna sus dat de X1,Y1
  //Coltul din dreapta jos X2,Y2
```

```
   dreptunghi=Rect(Edit1->Text.ToInt(),Edit2->Text.ToInt(),Edit3-
>Text.ToInt(),Edit4->Text.ToInt());
  // Umple  dreptunghiul desenata cu hasura specificata mai sus prin intermediul
obiectului
  //Brush
   PaintBox1->Canvas->FillRect(dreptunghi);
   }
//---------------------------------------------------------------------------
void __fastcall TForm1::Button4Click(TObject *Sender)
{
//Se redeseneaza paintbox-ul (functia Repaint)
PaintBox1->Repaint();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button5Click(TObject *Sender)
{
    //Specifica culoarea liniei
 PaintBox1->Canvas->Pen->Color=clBlack;
 //Specifica grosimea liniei
 PaintBox1->Canvas->Pen->Width=3;
 //Specifica stilul liniei
 PaintBox1->Canvas->Pen->Style=psDash;
//Se deseneaza o elipsa care este incadrata in dreptunghiul de coordonate X1,Y1 si
X2,Y2
//Coltul din stinga sus (coordonate  X1,Y1)
//Coltul din dreapta jos (coordonate X2,Y2)
 PaintBox1->Canvas->Ellipse(Edit1->Text.ToInt(),Edit2->Text.ToInt(),Edit3-
>Text.ToInt(),Edit4->Text.ToInt());
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button6Click(TObject *Sender)
{
 //se declara variabilelele sursa si destinatie de tipul TRect
 TRect sursa,destinatie;
 //Se construiesc  dreptunghiurile sursa si destinatie
 sursa=Rect(0,0,100,150);
 destinatie=Rect(100,0,200,150);
 //Copiaza imagina incadrata de dreptunghiul sursa  in dreptunghiul destinatie
  PaintBox1->Canvas->CopyRect(destinatie,PaintBox1->Canvas,sursa);
 }
//---------------------------------------------------------------------------
void __fastcall TForm1::Button7Click(TObject *Sender)
{
   //Se modifica culoare de hasura (in clAqua - albastru deschis)
  PaintBox1->Canvas->Brush->Color = clAqua;
  //Se modifica stilul de hasura (cruce in diaonala)
  PaintBox1->Canvas->Brush->Style = bsDiagCross;
//Instructiunea FloodFill  umple o suprafata marginita de  conturul de culoare
neagra, care conţine punctul de coordonate (x,y) cu  stilul  fsBorder (umple pana
la contur)
  PaintBox1->Canvas->FloodFill(Edit3->Text.ToInt()-3,Edit4->Text.ToInt()-
3,clBlack,fsBorder);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button8Click(TObject *Sender)
{
Close();
}
```

3 Sarcina lucrarii

1) Vor fi examinate toate componentele prezentate în indicatii teoretice;

2) Se modifică programul din *Project1.cpp* astfel încât să se obţină forma cu obiecte din figura 4.1 ;

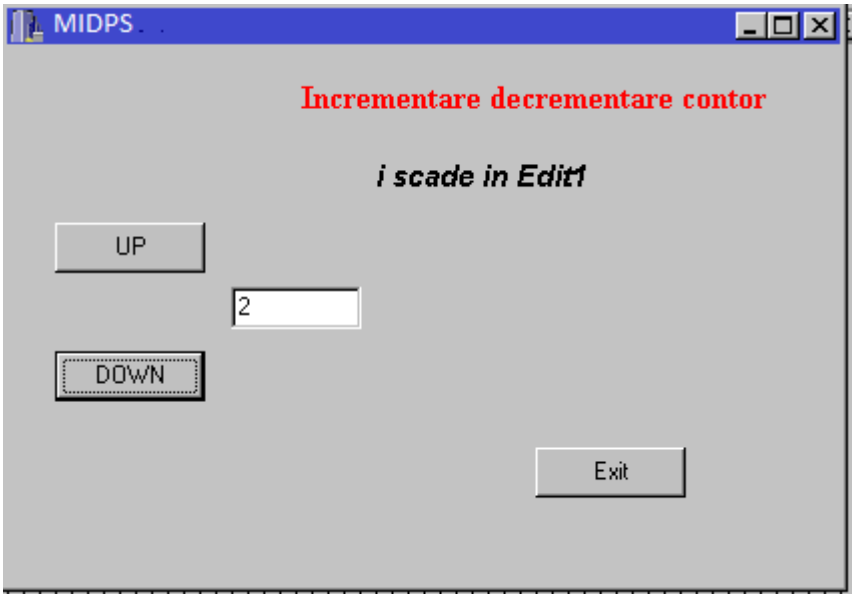

Fig. 4.1 – Realizarea 1

Se vor utiliza următoarele obiecte (în afara formei):

- două butoane (Button 1 şi 2) pentru incrementarea (UP) respectiv decrementarea (DOWN) a unei variabile întregi **i** ;
- un buton (Button 3) pentru ieţirea din program (Exit);
- o casetă de editare (Edit1) unde se va afişa valoarea variabilei **i**;
- două etichete (Label1 şi 2) pentru afişarea textului „**Incrementare decrementare contor.”** Respectiv a **sensului de variaţie a variabilei i din caseta Edit1;**
- în caption-ul formei se va afişa textul **„ MIDPS 1- A”;**
- fiecare obiect va avea hint-ul activ completat corespunzător .

3) Se elaborează un program pentru realizarea unui cronometru.

Se vor utiliza următoarele obiecte, evidenţiate în figura 4.2:

- o formă (*Form1*) pe care sunt dispuse celelalte obiecte şi în *Caption*-ul căreia se va afişa textul **„MIDPS”;**
- patru butoane (*Button 1, 2, 3 , 4*) cu următoarele funcţii:
  - Button1 – pornirea cronometrului(Caption **Start**);
  - Button2 – oprirea cronometrului(Caption **Stop**);
  - Button3 – iniţializarea cronometrului(Caption **Zero**);
  - Button4 – ieşirea din program (Caption **Exit**).
- două timere (*Timer1* şi *Timer2*) cu următoarele funcţii
  - Timer1 (*Interval=1000 ms*) utilizat la afişarea timpului curent;

- Timer2 (*Interval=100 ms*) utilizat pentru cronometru;

- două casete de editare (*Edit1* si *Edit2*) utilizate pentru :
  - Edit1 - afisarea datei si orei curente;
  - Edit2 - afişarea timpului cronometrat;
- două etichete (*Label1* si *Label2*) cu Caption-ul conform figurii 2.4

Observaţii:

- din primele trei butoane, la un un moment dat va fi activ unul singur;
- fiecare obiect va avea *hint*-ul activ completat corespunzător;

În timpul execuţiei programului forma va avea aspectul din figura 4.3

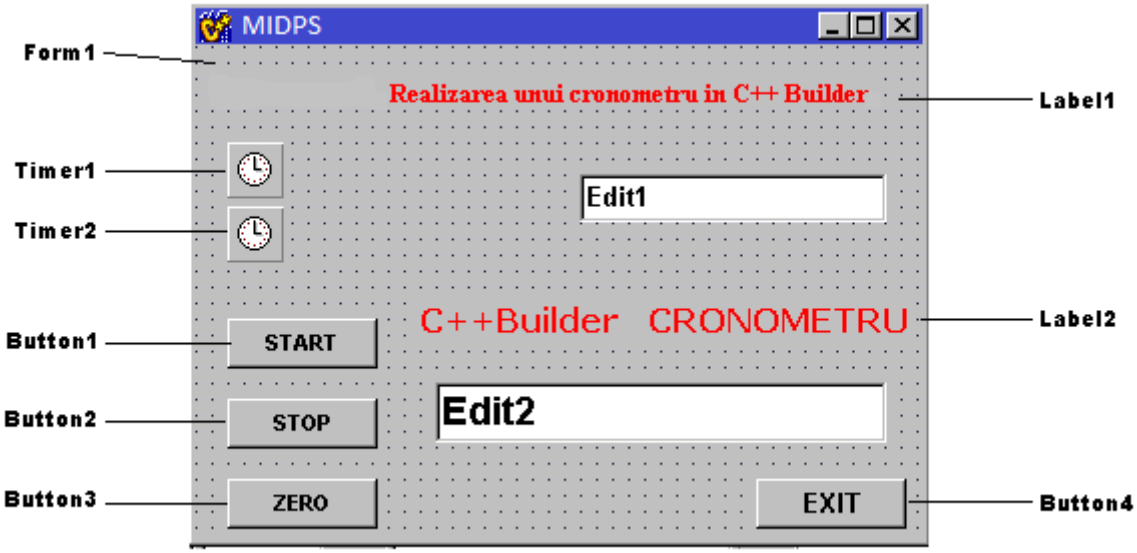

Fig 4.2 – Realizarea 2

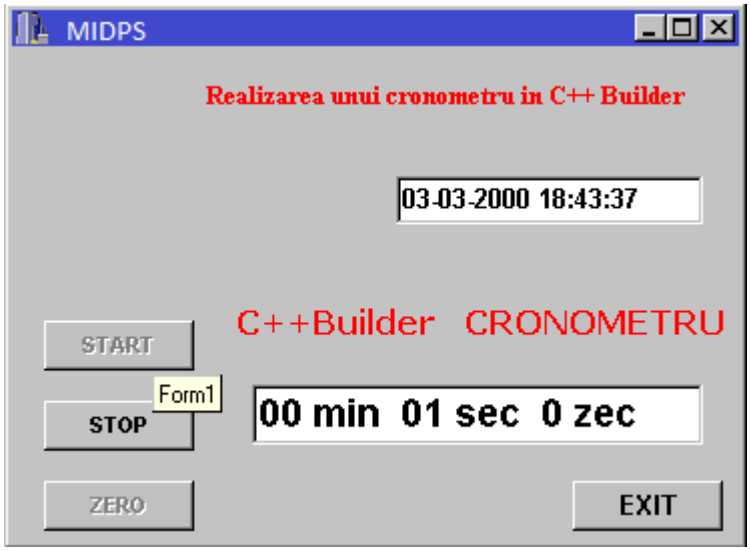

Fig.4.3 – Rezultatul aplicatiei 2

4) Se elaborează un program pentru realizarea a două elemente de afişare (bargraf şi diagramă cu avans continuu) pentru care forma arată ca în figura 4.4 pe care sunt dispuse următoarele obiecte:

- o formă (*Form1*) în *Caption*-ul căreia se va afişa textul **„MIDPS;**
- trei butoane (*Button 1, 2, 3*) cu următoarele funcţii:
  - Buton1 – activarea afişării în diagramă şi în bargraf (Caption **Start**);
  - Buton2 – oprirea afişării în diagramă şi în bargraf (Caption **Stop**);
  - Buton3 – ieşirea din program (Caption **Exit**).

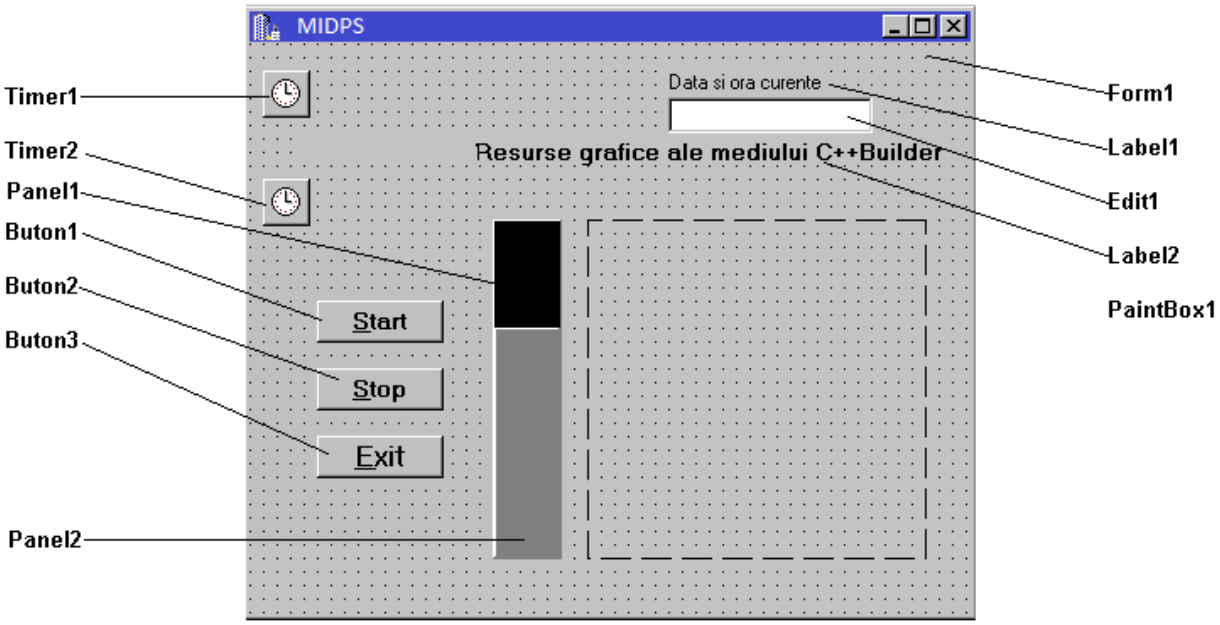

Fig 4.3 – Aplicatia 3

- două timere (*Timer1* şi *Timer2*) cu următoarele funcţii
  - Timer1 (*Interval=1000 ms*) utilizat la afişarea timpului curent;
  - Timer2 (*Interval=500 ms*) pentru intervalul de afişare în diagramă şi în bargraf;
- o casetă de editare (*Edit1*) utilizată pentru afişarea datei si orei curente;
- două etichete (*Label1* si *Label2*) cu Caption-ul conform figurii 4.4

Observaţii:
- din primele două butoane, la un un moment dat va fi activ unul singur;
- fiecare obiect va avea *hint*-ul activ completat corespunzător;
- valoarea numerică ce se va afişa în cele două elemente grafice se obţine cu funcţia *random* după care numărul generat se va converti în pixeli ţinându-se cont de înălţimea comună a graficului şi bargrafului
- pentru realizarea bargrafului se vor utiliza două obiecte de tip TPanel de culori diferite care se vor suparpune;
- pentru desenarea graficului se vor utiliza funcţiile *MoveTo, LineTo* iar pentru avansul acestuia funcţia *CopyRect.*

În timpul execuţiei programului forma va avea aspectul din figura 4.4.

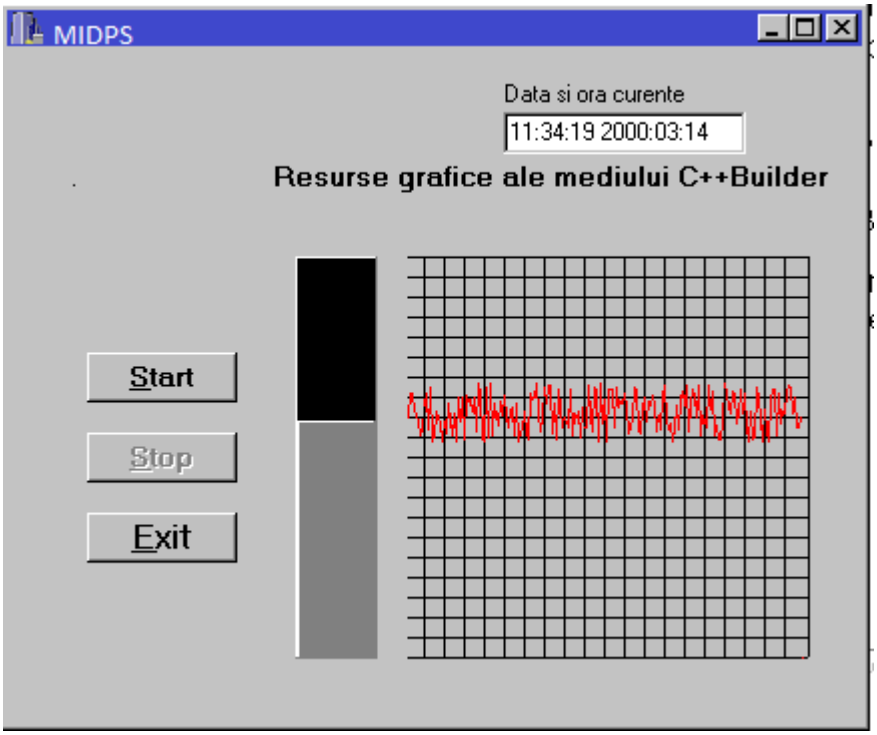

Fig.4.4- Rezultatul aplicatiei 3