**Universitatea Pedagogică de Stat “Ion Creangă”**

**Marina Bostan** **Nicolae Balmuș**

# TEHNOLOGII DE PROGRAMARE VIZUALĂ

**Chișinău – 2020**

Ghidul metodologic **„Tehnologii de programare vizuală”** a fost elaborat în cadrul proiectului de cercetare și inovare **„Elaborarea și implementarea manualelor digitale interactive în învățământul preuniversitar”**, implementat în baza contractului de finanțare încheiat între **Agenția Națională pentru Cercetare și Dezvoltare** și **Universitatea Pedagogică de Stat „Ion Creangă”** și aprobat prin Decizia Colegiului Autorității contractante nr. 01-PC din 10.01.2020 din cadrul Priorității strategice IV. Provocări societale, înscris în Registrul de stat al proiectelor din sfera științei și inovării cu cifrul 20.80009.0807.25.

Aprobat în ședința proiectului proces-verbal nr.6 din 29.10.2020 și ședința Senatului Universității Pedagogice de Stat „Ion Creangă” din Chișinău, proces verbal nr. 4 din 26.11.2020.

**Bostan, Marina.**
Tehnologii de programare vizuală : Note de curs / Marina Bostan, Nicolae Balmuş ; Universitatea Pedagogică de Stat "Ion Creangă". – Chişinău : S. n., 2020 (Tipogr. UPS "Ion Creangă"). – 104 p. : fig., tab.
Bibliogr.: p. 104 (16 tit.). – 100 ex.
ISBN 978-9975-46-502-1.
004.9(075.8)
B 71

**Cuprins**

## Cuvânt înainte

Prin programarea vizuală se subînțelege utilizarea mediilor de programare în gestionarea informației într-o manieră vizuală, ce oferă utilizatorului posibilitatea de a elabora și dezvolta aplicații, bazate pe paradigma programării orientate pe obiecte.

Un limbaj vizual administrează informația vizuală, suportă interacțiune vizuală sau permite programarea prin folosirea expresiilor vizuale. Programarea vizuală este în mod uzual definită ca fiind utilizarea expresiilor vizuale (cum ar fi grafică, desene, animații sau iconițe), ca mijloace prin care programele sunt create si modificate.

Odată cu apariția limbajelor de programare orientate pe obiecte, evoluția produselor software a cunoscut un salt important. Anume în baza acestei concepții sunt elaborate o bună parte din produsele software. Aceasta va permite adaptarea mai rapidă a cadrului didactic la noi platforme de dezvoltare.

Notele de curs se adresează cursanților pentru studierea tehnologiilor de programare orientată pe obiect în scopul implementării unor interfețe grafice în format vizual (crearea și dezvoltarea aplicațiilor interactive), cât și poate fi utilă cadrelor didactice care urmează să implementeze tehnologia programării vizuale în predarea informaticii în acord cu necesitățile de adaptare a curriculumului disciplinar la nevoile elevilor.

Lucrarea cuprinde lucrări de laborator care includ atât descrierea teoretică, cât și demersul aplicativ cu scopul deprinderii stilului de programare specific limbajelor de programare vizuală prin elaborarea aplicațiilor interactive. Elementele de programare vizuală din acest suport se bazează pe limbajul de programare Object Pascal, implementat în mediul de programare Delphi Seattle în versiunea 10.

Sunt descrise conținuturi teoretice atât din Elementele limbajului de programare Object Pascal atât și din Mediului de programare vizuală Delphi 10 Seattle, necesare pentru elaborare, dezvoltare și respectiv implementare aplicațiilor interactive cu interfață grafică.

Instrucțiunile metodologice conțin o scurtă referință a proprietăților unora dintre cele mai frecvent utilizate componente ale mediului Delphi în aplicații, precum și un set de lucrări de laborator. Cititorul ar trebui să fie familiarizat cu principiile programării orientate obiect, cu limbajul de programare Object Pascal și cu elementele de bază ale programării vizuale.

## Ce este Delphi?

Delphi a fost dezvoltat inițial de Borland ca un instrument pentru dezvoltarea rapidă a aplicațiilor Windows în calitate de succesor al limbajului Turbo Pascal. Prima versiune a lui Delphi a fost lansată la 14 februarie, anul 1995. În prezent Delphi este dezvoltat și întreținut de Embarcadero Technologies [1]. La momentul actual Delphi este un produs software care folosește limbajul de programare Object Pascal și oferă un mediu de dezvoltare integrat IDE (Integrated Development Environment) pentru dezvoltarea rapidă a aplicațiilor RAD (Rapid Application Development).

Ultimele versiuni Delphi oferă două cadre pentru dezvoltarea aplicațiilor vizuale: VCL și FireMonkey. Biblioteca de componente vizuale VCL (Visual Component Library)- este cadrul pentru dezvoltarea aplicațiilor Windows pure iar FireMonkey (FMX) este cadrul pentru dezvoltarea aplicațiilor vizuale multiplatforme Windows, macOS, iOS, Android și Linux (x64).

La realizarea versiunilor noi Delphi se respectă principiul compatibilității descendente (codul aplicațiile realizate în versiunile precedente, cu mici excepții, este recunoscut și se execută în versiunile noi).

FMX nu este compatibil cu VCL; sunt două cadre separate. Cu toate acestea, aplicațiile FireMonkey permit partajarea ușoară a unităților de cod non-vizual cu aplicațiile VCL. Toate

Delphi este cunoscut pentru viteza sa rapidă de compilare, codul nativ și productivitatea dezvoltatorului.

## Descărcarea și instalarea Mediului de Programare Delphi.

Începând cu anul 2020 versiunile comunitare Delphi (fără restricții funcționale) pot fi descărcate și instalate gratuit la adresa oficială a companiei Embarcadero https://www.embarcadero.com/products/delphi/starter/free-download, completând în prealabil un formular de înregistrare.

La finalizarea procesului de instalare în lista de aplicații Windows apare meniul - Embarcadero Rad Studo10 Seattle(fig.1a),iar pe masa de lucru iconița aplicației (fig.1b). Iconița aplicației poate fi fixată și pe panelul de sarcini (fig1c).

Mediul de programare Delphi se lansează realizând dublu click pe iconițele respective (fig.1,b,c) sau accesând meniul Delphi 10 Seattle (fig.1a)

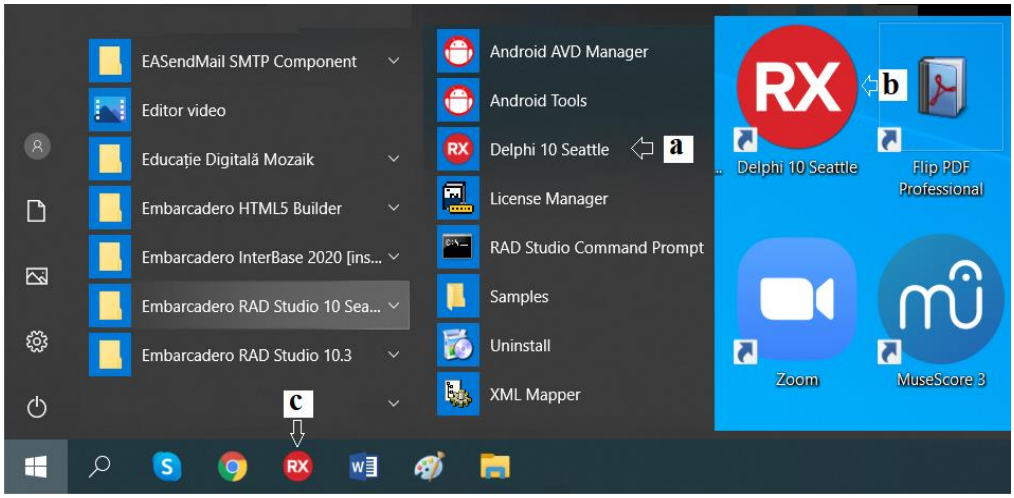



Figura 1. Metode de lansare a mediului de programare Delphi 10 Seattle.

După lansarea mediului de programare Delphi 10 Seattle apare fereastra din figura 2.

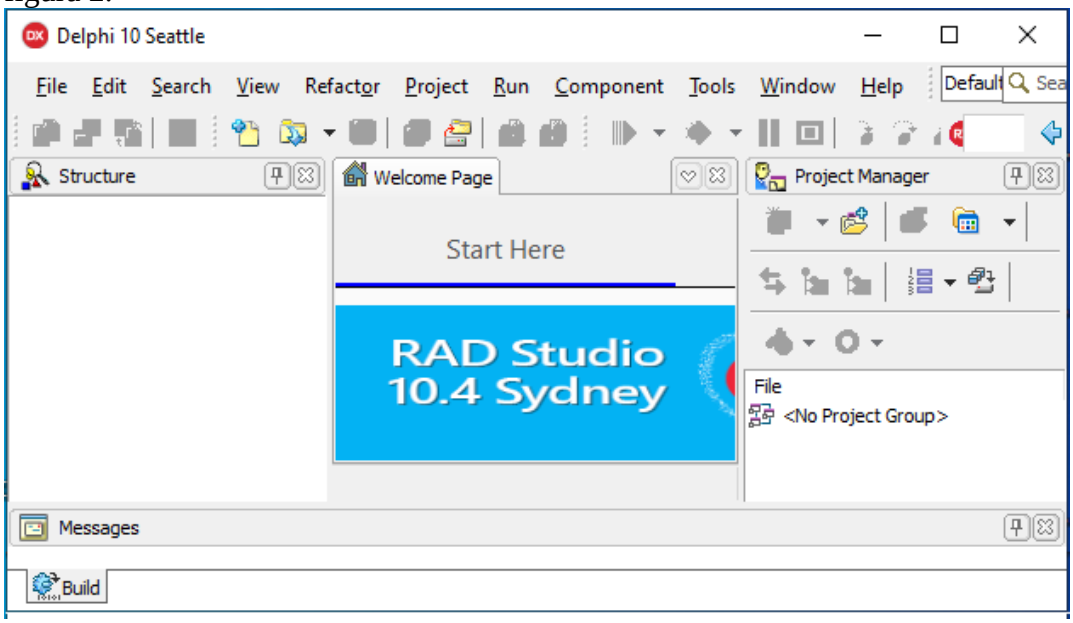



Figura 2. Fereastra de lansare a meniului de programare Delphi 10 Seattle

## Crearea aplicațiilor Delphi

În mediul de programare Delphi 10 Seattle pot fi realizate mai multe tipuri de aplicații. Pentru accesarea lor, în fereastra din figura 2, selectăm File=>New=>Other...(figura 3)

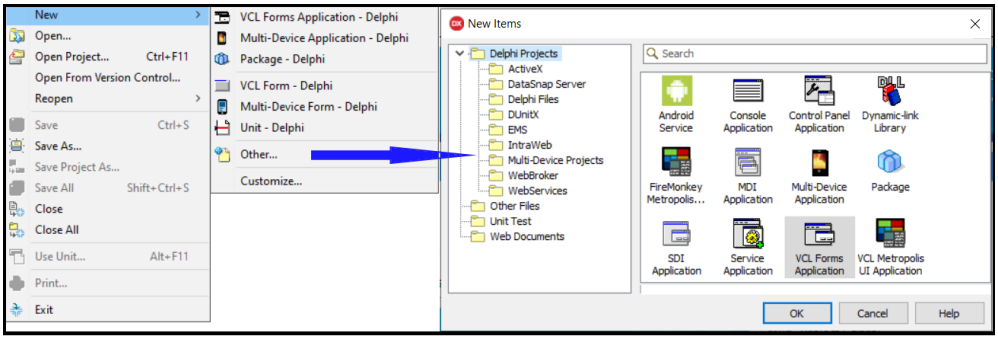



Figura 3. Tipuri de aplicații Delphi 10 Seattle.

În această lucrare sunt descrise două tipuri de aplicații Delphi 10 Seattle:

- Console Application (aplicații consolă);
- Multi-Device Application (apicalii multe-funcționale);

## Aplicații consolă (Console Application).

Pentru realizarea unei aplicații consolă, în fereastra (Search, figura 3), cu ajutorul mouse-lui se selectează opțiunea ”Console Application”. Ca urmare apare fereastra din figura 4.

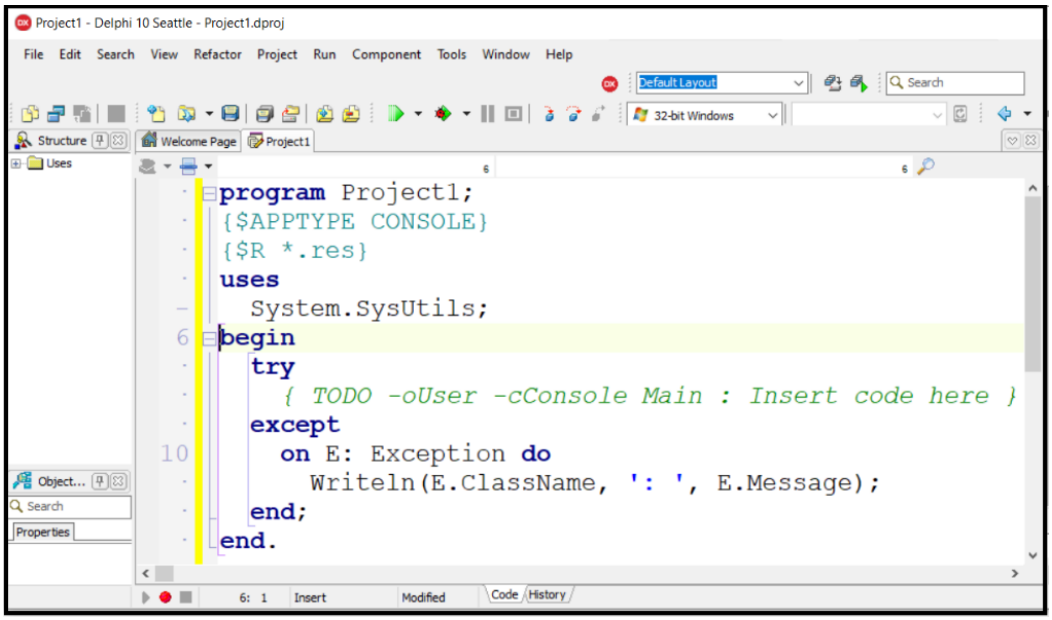



Figura 4. Fereastra aplicației consolă Delphi 10 Seattle.

Menționăm că codul din figura 4 a fost creat în mod predefinit. În baza lui, la executare, se obține fereastra aplicației consolă (o fereastră Windows de culoare neagră) care clipește și se închide momentan. Obligatoriu în acest cod sunt instrucțiunile begin end. Restul codului poate fi șters sau modificat în conformitate

cu algoritmul pe care programatorul dorește să-l implementeze, respectând alfabetul, vocabularul, sintaxa, conceptele procedură și obiect (clasă) ale limbajului de programare **Object Pascal**.

Pentru realizarea unui program elementar care va afișa un mesaj, tradițional de salut este necesar să cunoaștem sintaxa și modul de funcționare a instrucțiunilor de scriere/citire (**writeln/readln**) cu ajutorul căreia se afișează pe ecran mesaje (șiruri de caractere delimitate de apostrofe) și rezultate ale calculelor expresiilor matematice care se scriu și se evaluează conform regulilor cunoscute în matematică. În figura 5 este reprezentat codul și rezultatul executării aplicației consolă în care se afișează mesajul ”Bună ziua Consolă Delphi” și câteva calcule matematice.

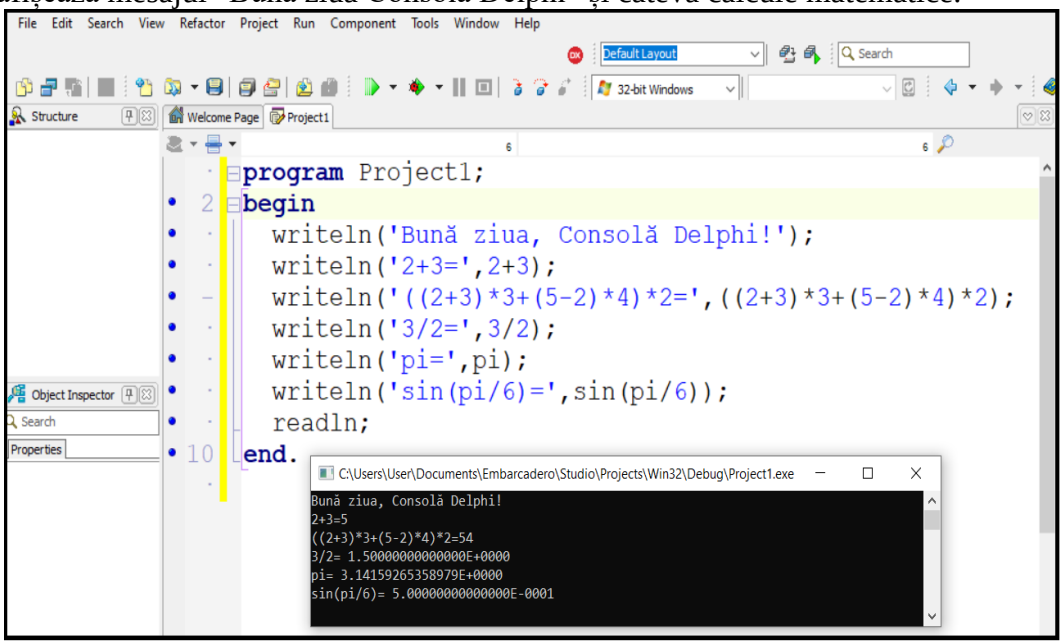



Figura 5. Exemplu de aplicație consolă Delphi.

Fereastra în care se afișează rezultatele unui program consolă Delphi ne este una simplă, analogică cu fereastra de vizualizare a rezultatelor TPascal. Aceasta este o fereastră Windows cu toate actele în regulă care, în particular, permite selectarea și copierea informației cu ajutorul mouse-lui și a tastelor Ctrl C.

## Aplicații Delphi cu interfață grafică.

Aplicația consolă Delphi, prin atașarea în lista uses a unut-urilor necesare, are posibilitate să acceseze rutine pentru crearea și gestionarea obiectelor, în baza cărora se creează interfețe grafice. Acest mod de realizare a aplicațiilor cu interfață grafică necesită cunoștințe avansate de Programare Orientată pe Obiecte în baza cărora, prin cod de programare, relativ voluminos, se creează interfață grafică.

Mediul integrat de dezvoltarea al aplicațiilor Delphi (fig.2) permite crearea aplicațiilor cu interfață grafică la nivel vizual, utilizând componente (obiecte) disponibile în paleta de instrumente, pentru care, prin cod Object Pascal, se programează evenimentele necesare. Acest stil de programare se numește Programare Orientată pe Evenimente și este mult mai accesibil pentru programatorii începători care în baza tehnicilor de programare procedurală realizează relativ repede aplicații interactive cu interfață grafică.

Deoarece aplicațiile Multi-Device permit compilarea codului Object Pascal pentru diferite sisteme de operare, în această lucrare vom analiza numai al doilea tip de aplicații care se mai numesc FireMonkey (FMX).

Pentru realizarea unei aplicații vizuale (FMX) în fereastra de lansare a mediului de programare Delphi (fig.2) selectăm File=>New=>Multi-Device Application=>Blank Applicaion. Ca urmare apare fereastra reprezentată în figura 6 în care utilizatorul.

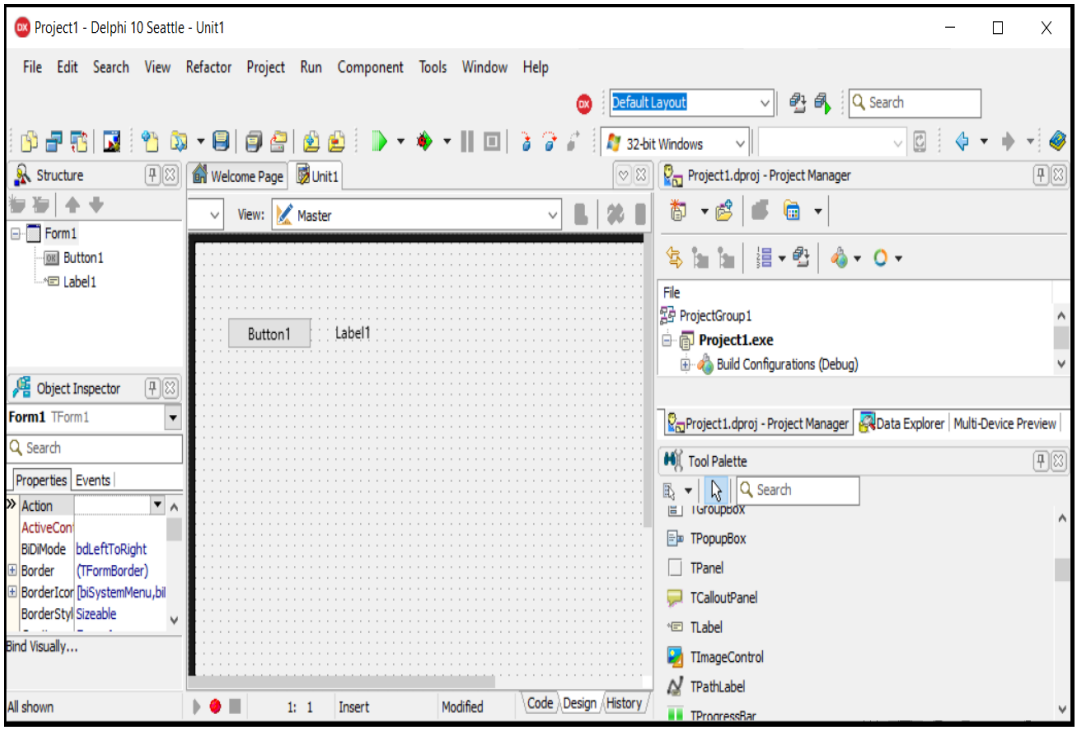



Figura 6. Exemplu de aplicație vizuală Delphi FMX

Pentru realizarea aplicației din figura 6 utilizatorul realizează următorii pași:

- cu ajutorul mouse-lui, preia din paleta de instrumente (Tool Palette) componentele Button1 și Label1 și le plasează pe forma aplicației în locul dorit;
- pentru programarea evenimentului OnClick al butonului Button1, realizează dublu click pe acest buton. Ca urmare apare fereastra editorului de cod în care programatorul completează porțiunea de cod rezervată pentru procedura Form1.Button1Click(Sender: TObject);

În figura 7 este reprezentat codul și rezultatul executării programelor consolă (a) și vizuală (b) cu același efect grafic.

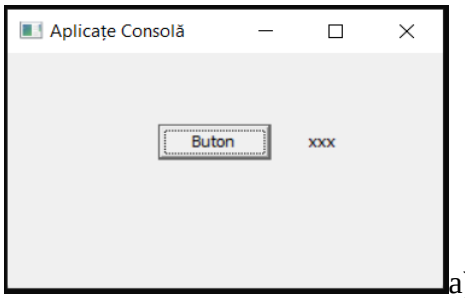
Menționăm că în cazul programului consolă programatorul trebuie să scrie de la tastatură tot codul (a) iar în cazul aplicației vizuale numai două rânduri ale evenimentului Button1.OnClick (evidențiate cu **bold**). Restul codului din listingul aplicației vizuale (b) a fost generat în mod automat. Acest moment este înalt apreciat de programatorii care realizează aplicații vizuale în mediul de programare Delphi. Pe site-ul Embacadero se menționează că productivitatea de realizare a aplicațiilor vizuale Delphi se mărește de 10 ori în comparație cu soluțiile propuse de alte limbaje concurente.

```
program Project1;
{$APPTYPE CONSOLE}
uses Vcl.Forms,Vcl.StdCtrls;
var form1:TForm;
button1:TButton;
label1:TLabel;
type TEventBClick = class
procedure BClick(Sender :
  TObject);
end;
procedure TEventBClick.
BClick(Sender : TObject);
begin
 form1.Caption:='Aplicațe
 vizuală clasică';
 label1.Caption:='xxx';
```

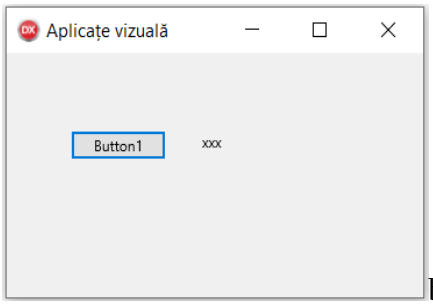
```
unit Unit1;
interface
uses
System.SysUtils,
System.Types,
System.UITypes,
System.Classes,
System.Variants,
FMX.Types,     FMX.Controls,
FMX.Forms,     FMX.Graphics,
FMX.Dialogs, FMX.StdCtrls,
FMX.Controls.Presentation;
type
 TForm1 = class(TForm)
 Button1: TButton;
 Label1: TLabel;
 Procedure
 Button1Click(Sender:
 TObject);
```

```
end;
var BtClick: TEventBClick;
begin
BtClick :=
TEventBClick.Create();
Application.CreateForm(TForm,
Form1);
button1:=Tbutton.Create(Form1)
 button1.Parent:=Form1;
 button1.Left:=100;
 button1.Top:=50;
 button1.Caption:='Buton';
 Label1:=Tlabel.Create(Form1);
 Label1.Parent:=Form1;
 Label1.Caption:='Label1';
 Label1.Left:=200;
 Label1.Top:=55;
button1.OnClick:=
BtClick.BClick;
Application.Run;
end.
```

```
 private
 { Private declarations }
 public
 { Public declarations }
 end;
 var
 Form1: TForm1;
implementation
{$R *.fmx}
procedure
TForm1.Button1Click(Sender:
TObject);
begin
 form1.Caption:=
'Aplicație vizuală
 Delphi';
label1.Text:='xxx';
end;
end.
```



a)



b)

Figura 7. Codul și rezultatul aplicațiilor Delphi: a) consolă și b) vizuală

Pentru realizarea rapidă a aplicațiilor vizuale în mediul de programare Delphi, utilizatorul trebuie să cunoască:

1. Interfața (meniurile, ferestrele), proprietățile, metodele și evenimentele componentelor utilizate pentru realizarea aplicațiilor vizuale;

2. Alfabetul, sintaxa, instrucțiunile de bază ale limbajului Pascal (în foarte mare măsură moștenite de la Turbo Pascal);
3. Noțiuni elementare de Programare Orientată pe Obiecte (Object Pascal).

## Interfața mediului de dezvoltare a aplicațiilor vizuale Delphi 10 Seattle

Interfața mediului de dezvoltare a aplicațiilor Delphi 10 Seattle depinde de tipul aplicației vizuale selectate de utilizator la etapa inițială File=>New=>. În cazul selectării tipului de aplicație multiplatform (Multi-Device Application), varianta Blank Application, interfața conține trei pagini: Design (figura 8). Code (figura 9) și Histori. Paginile Design și Code sunt principale: în ele se dezvoltă design-ul și se scrie de la tastatură codul Object Pascal al aplicației. Prezentăm succint, în continuare aceste pagini:

Pagina **Design** (figura 8) conține:

- Bara de meniuri: File, Edit, Search, View, Project, Run, etc;
- Bara de instrumente care conține un șir de iconițe cu ajutorul cărora se accesează rapid unele opțiuni ale meniurilor: -Save, -Save All, -Run, etc;
- Fereastra **Structure,** în care se vizualizează structura de componente ale formei;
- Fereastra **Object Inspector** care conține două pagini: **Properties** și **Events**, în care pot fi consultate și modificate unele proprietăți și evenimente ale Formei și componentelor ei;
- Fereastra **Project Manager** în care se vizualizează structura proiectului;
- Fereastra **Tool Palette** în care se caută și se selectează componentele necesare pentru realizarea design-ului aplicației;
- Fereastra formei în care se realizează design-ul aplicației la nivel vizual.

Pagina **Code** (figura 9) conține:

- Fereastra **Structure** cu informație completă despre structura codului Object Pascal al aplicației (clase, variabile, constante, proceduri, funcții, etc).
- Fereastra/ ferestrele codului în care se redactează de la tastatură codul aplicației divizat în unit-uri.
- Fereastra **Messages** în care se vizualizează informații importante cu privire la realizarea procesului de compilare. Dacă în procesul de compilare a aplicației se depistează erori, atunci în această fereastră se indică tipul erorii, rândul și unitatea în care sa produs eroarea.

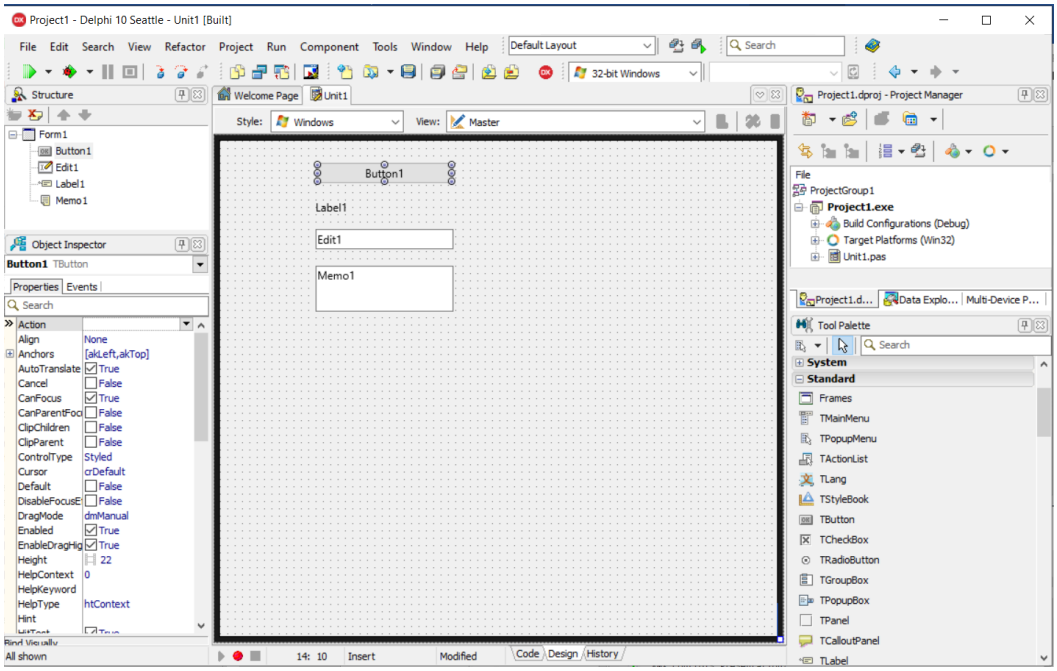



Figura 8. Pagina Design a aplicației FMX- Delphi 10 Seattle.

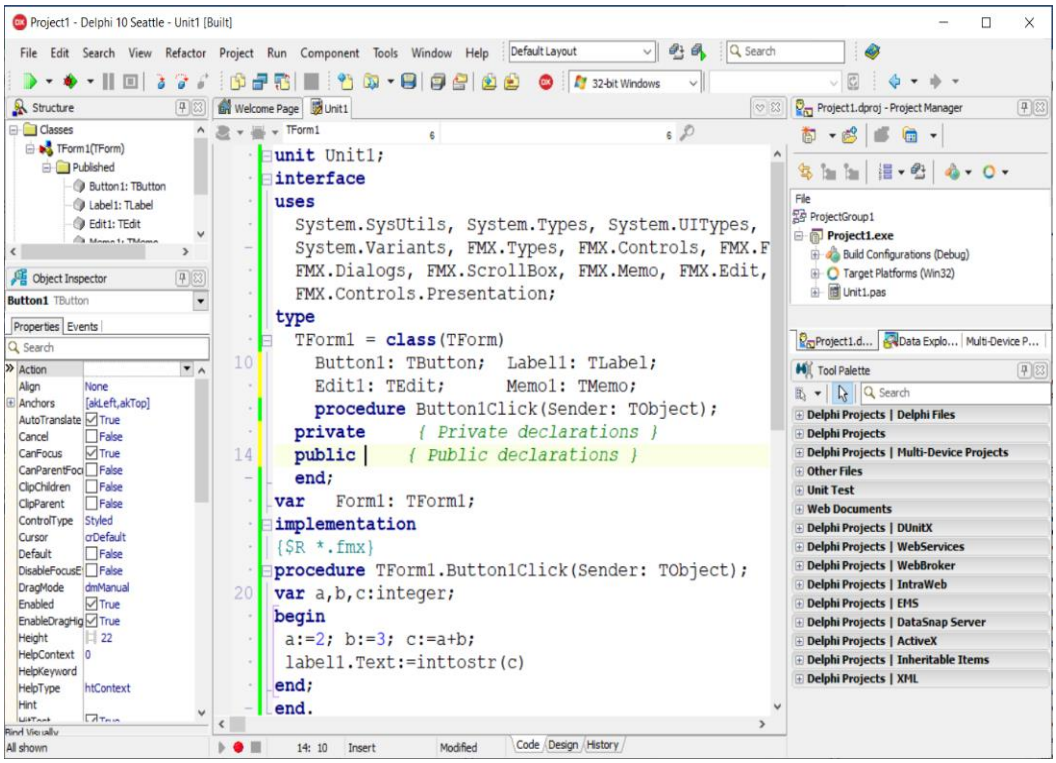



Figura 9. Pagina Code a aplicației FMX- Delphi 10 Seattle.

Menționăm că codul unitului Unit1 (figura 9), inclusiv structura procedurii Button1Click a fost generat în mod automat de mediul de programare Delphi. De la tastatură au fost scrise numai:

- declarația Pascal a variabilelor: **var** a,b,c: integer;
- instrucțiunile de atribuire: a:=2; b:=3; c:=a+b;
- instrucțiunea label1.Text:=inttostr(c); în baza căreia în proprietatea Text a componentei Label1 se afișează rezultatul calculelor (valoarea expresiei a+b;

Programul se trimite la executare:

- cu ajutorul opțiunii **Run** din meniul **Run;**
- Prin apăsarea butonului sau tastei F9.

Pentru dezvoltarea unor aplicații consolă și vizuale mai avansate utilizatorul trebuie să cunoască suficient de profund limbajul de programare Pascal și noțiunile principale utilizate în Programarea Orientată pe Obiecte (POO), incorporate în Object Pascal – limbajul de programare a aplicațiilor Delphi.

## Salvarea și recuperarea aplicațiilor Delphi

În procesul de dezvoltare a aplicațiilor Delphi se creează mai multe fișiere. Din această cauză se recomandă ca fiecare aplicație să fie salvată într-o mapă separată. Salvarea aplicației se realizează cu ajutorul opțiunii **Save All** a meniului **File** care salvează în regim de dialog fișierele Unit1.pas, Project1.dpr și Project1.dproj.

**Notă** La etapa inițială se recomandă ca numele acestor fișiere să nu fie modificate de utilizator.

Fișierul executabil al aplicației Project1.exe și se află în mapa Win32=>Debug.

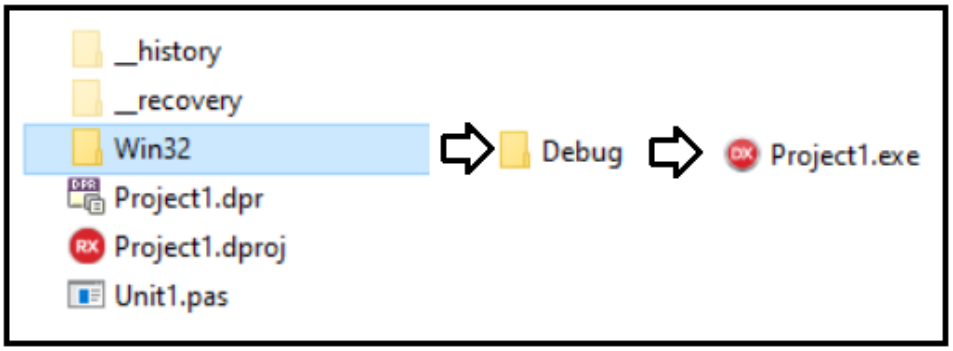



Figura 10. Structura fișierelor unei aplicații Delphi.

Pentru recuperarea unei aplicați Delphi, în mapa unde a fost salvată aplicația se realizează dublu click pe fișierul Project1.dproj. Ca urmare se lansează mediul de programare Delphi, în ferestrele căruia se află codul și designul aplicației încărcate pe care utilizatorul poate să le dezvolte în continuare. Această activitate poate fi realizată și la etapa când mediul de programare este lansat. În acest caz se accesează meniul File=>Open, după care, în regim de dialog se caută fișierul respectiv *.dproj.

## Limbajul de programare Object Pascal (descriere succintă)

Limbajul de programare Object Pascal se dezvoltă în continuu, începând cu lansarea primei versiuni a mediului de programare Delphi. Acesta este o extensie a limbajului de programare Turbo Pascal 5.5 care oferă caracteristici de Programare Orientată pe Obiecte (OOP), cum ar fi clase și metode. Limbajul continuă să se dezvolte în fiecare versiune nouă a lui Delphi, respectând politica de compatibilitate cu versiunile precedente.

## Elemente de sintaxă.

**Alfabetul (**caracter fundamentale**)**. Object Pascal utilizează setul de caractere ASCII, inclusiv literele de la A la Z și de la a la z, cifrele de la 0 la 9 și alte caractere standard _ # $ & ' ( ) * + , – ./ : ; < = > @ [ ] ^ { } .(punctul). Nu este sensibil la litere mari și mici.

Caracterele fundamentale se combină pentru a forma identificatori, declarații, expresii și instrucțiuni:

**Identificatorii** sunt cuvinte cu ajutorul cărora se identifică constante, variabile, câmpuri, tipuri, proprietăți, proceduri, funcții, programe, unități, biblioteci etc. Un identificator poate avea orice lungime, dar numai primele 255 de caractere sunt semnificative și trebuie să înceapă cu o literă sau o subliniere (_) și nu poate conține spații (literele, cifrele și sublinierile sunt permise după primul caracter.

Cuvintele rezervate precum (**begin, end, procedure,** etc) nu se pot utiliza în calitate de identificatori.

**Identificatori calificați** sunt structuri care indică apartenența ierarhică a identificatorilor. Sintaxa unui identificator calificat este identificator1.identifier2 unde identificatorul1 califică identificatorul2. De exemplu, dacă două unități declară fiecare o variabilă numită V atunci Unit2.V va identifica variabila V definită în Unit2; Form1.Button1.Click apelează metoda Clic a butonului Button1 din Form1.

(unitate sintactică care apare într-o instrucțiune și denotă o valoare), () și instrucțiuni: O expresie este o unitate sintactică care apare într-o instrucțiune și denotă o valoare.

O instrucțiune descrie o acțiune algoritmică care poate fi executată în cadrul unui program. O declarație definește un identificator (cum ar fi numele unei funcții sau al unei variabile) care poate fi utilizat în expresii și declarații și, dacă este cazul, alocă memorie pentru identificator.

**Șiruri de caractere**

Un șir de caractere, denumit și șir literal sau șir constant, constă dintr-un șir citat, un șir de control sau o combinație de șiruri de date citate și de control. Separatoarele pot apărea numai în cadrul șirurilor citate. Un șir citat este o secvență de până la 255 de caractere din setul extins de caractere ASCII, scris pe o singură linie și închis de apostrofări. Un șir citat cu nimic între apostrofuri este un șir nul. Două apostrofuri secvențiale dintr-un șir citat denotă un singur caracter, și anume un apostrof. Un șir de control este o secvență de unul sau mai multe caractere de control, fiecare dintre acestea constă din simbolul # urmat de o constantă întreagă nesemnată de la 0 la 255 (zecimală sau hexazecimală) și denotă caracterul ASCII corespunzător. Se permite combina șirurile citate cu șirurile de control pentru a forma șiruri de caractere mai mari. De exemplu, puteți utiliza 'Linia 1'#13#10'Line 2' pentru a separa textul în - două linii: 'Linia 1' și 'Linia 2'.

**Comentarii și directive de compilare** Comentariile sunt ignorate de compilator, cu excepția cazului în care acestea funcționează ca separatori (delimitarea token-uri adiacente) sau directive compilator. Există mai multe moduri de a construi comentarii:

- { Textul dintre o acoladă stângă și o acoladă dreaptă constituie un comentariu. }

- (* Textul dintre o paranteză stângă plus asterisc și un asterisc plus paranteză dreapta este, de asemenea, un comentariu *)

// Orice text între o bară oblică dublă și sfârșitul liniei constituie un comentariu.

Comentariile care sunt la fel nu pot fi imbricate. De exemplu, {{}} nu va funcționa, pe când (*{}*)- funcționează. Acest lucru este util pentru comentarea secțiunilor de cod care conțin, de asemenea, comentarii. Un comentariu care conține un semn dolar

($) imediat după deschiderea { sau (* este o directivă compilator. De exemplu, {$WARNINGS OFF} îi spune compilatorului să nu genereze mesaje de avertizare.

**Declarații și instrucțiuni**. Numele variabilelor, constantelor, tipurilor, câmpurilor, proprietăților, procedurilor, funcțiilor, programelor, unităților, bibliotecilor și pachetelor sunt numite identificatori. Identificatorii trebuie declarați înainte de ai putea utiliza; singurele excepții sunt câteva tipuri predefinite, rutine și constante pe care compilatorul le înțelege automat (maxint, variabila Rezult atunci când apare în interiorul unui bloc de funcții și variabila Self atunci când apare în interiorul unei implementări de metodă). O declarație definește unul sau mai mulți identificator separați prin virgulă și, dacă este cazul, alocă memorie pentru acesta. De exemplu:

**var** x:integer;- declară variabila x de tip integer;

**Function** Suma(X, Y: integer):integer; - declară o funcție numită Suma care utilizează două valori de tip integer și returnează un integer.

Fiecare declarație se termină cu punct și virgulă.

Când se declară mai multe variabile în același bloc cuvântul rezervat **var** corespunzător se scrie o singură dată:

**var** a,b,c:extended; - declară variabilele a,b,c te tip extended (real);

În general, declarațiile pot apărea numai la începutul blocului de instrucțiuni ale unui subprogram, la începutul secțiunilor de interfață (**interface**) (după clauza **uses**) sau de implementare **(implemenation**) a unei unități.

Mai detaliat convențiile specifice pentru declararea variabilelor, constantelor, tipurilor, funcțiilor și așa mai departe sunt explicate în documentația pentru aceste subiecte.

**Instrucțiunile** (statement) definesc acțiuni algoritmice în cadrul unui program. Instrucțiunile sunt de următoarele tipuri:

- simple cum ar fi apelurile proceduri sau funcții;
- de atribuire care are următoarea structură **v:=expresie**; (v- variabilă declarat; **:=** simbolul operației de atribuire; **expresie** este o structură care conține variabile, constante și funcții predefinite sau definite de utilizator asamblate, de regulă, conform regulilor din matematică cu paranteze și operații predefinite. Tipul expresiei în mod obligatoriu trebuie să

coincidă cu tipul variabilei **v** din partea stângă a instrucțiunii de atribuire. Exemple: x:=x+1; y:=sin(pi/2)+cos(pi/4);

Instrucțiunile de atribuirile și apelurile de procedură, se pot combina pentru a forma instrucțiuni compuse de tipul:

```
if (x>5) and (x<10) then y:=x+2 else y:=x-5;
```

```
repeat s:=s+i; i:=i+1; until i<=100;
```

Mai detaliat regulile de scriere a instrucțiunilor și expresiilor sunt explicate în documentația pentru aceste subiecte.

**Instrucțiuni de conversie a tipurilor de date**. Foarte frecvent, în aplicațiile interactive Delphi, apare necesitate de a transforma valoarea unei variabile dintr-un tip în altul. Pentru aceste scopuri există multe funcții și proceduri predefinite. In această introducere prezentăm câteva:

| Funcția | Semnificația |
|---|---|
| IntToStr(i:integer):string | Transformă valoarea lui i din format **integer** în **string** |
| StrToInt(s:string):integer | Transformă valoarea lui s din format **string** în **integer.** |
| FloatToStr(x:extended):string | Transformă valoarea lui x din format **real** în **string** |
| StrToFloat(s:string):extended | Transformă valoarea lui s din format **string** în **extended** |

## Tipuri de date predefinite

Object Pascal este un limbaj de programare stric tipizat. Un tip de date definește o mulțime finită de valori și o mulțime finită de operații, funcții și proceduri asociate. În memoria calculatorului, la nivel de cod mașină, datele se reprezintă ca șiruri de cifre binare. Trecerea de la datele de intrare la această reprezentare binară și invers, trecerea de la reprezentarea internă a datelor la cea a datelor de ieșire, nu ne interesează în detaliu. Aceste transformări se realizează în mod automat în procesul de compilare a programului și în procedurile de intrare/ieșire a datelor.

## Tipuri de date numerice.

În Delphi, există 3 grupuri de date de tip numeric:

- numere întregi;
- numere cu virgulă mobilă (au o fracție zecimală);
- numere cu virgulă zecimală fixată (pentru calcule financiare);

Pentru declararea variabililor de tip numeric sunt predefinite următoarele tipuri:

| **Tip** | **Domeniu de valori** |
|---|---|
| **Integer** types | |
| byte | 0 … 255 |
| shortint | -127 ... 127 |
| word | 0 ... 65535 |
| smallint | -32768 ... 32767 |
| cardinal | 0 ... 4429967295 |
| integer | -2147483648 ... 2147483647 |
| int64 | -9223372036854775808 ... 9223372036854775807 |
| **Floating point** types | |
| single | 7 cifre semnificative exponent-38 ... 38 |
| double | 15 cifre semnificative exponent -308 ... 308 |
| extended | 19 cifre semnificative exponent -4932 ... 4932 |
| real identic cu double | 15 cifre semnificative exponent -308 ... 308 |
| **Fixed point** types | |
| Currency | se utilizează în calcule financiare 4 zecimale fixe<br>MaxCurrency =922337203685477.5807 |

**Operații predefinite pentru tipurile de date numerice.**

| Operația | Denumire/ tip | Descriere | Tipul operanzilor | Tipul rezultatului |
|---|---|---|---|---|
| + | adunarea/ binar | întoarce suma operanzilor | întreg, real | întreg, real |

| | | | | |
|---|---|---|---|---|
| - | scăderea/ binar | întoarce diferența operanzilor | întreg, real | întreg, real |
| * | înmulțirea/ binar | întoarce rezultatul înmulțirii operanzilor | întreg, real | întreg, real |
| / | împărțirea/ binar | întoarce rezultatul împărțirii operanzilor | întreg, real | real |
| div | împărțirea/ binar | întoarce câtul împărțirii operanzilor | întreg | întreg |
| mod | câtul (restul)/ binar | întoarce restul împărțirii operanzilor | întreg | întreg |
| - | minus/ unar | întoarce valoarea opusă a operandului | întreg, real | întreg, real |
| + | plus/ unar | indică în mod explicit semnul plus | întreg, real | întreg, real |

Funcții și proceduri predefinite pentru datele de tip numeric.

| Sintaxa | Tip argument/ rezultat | Descriere |
|---|---|---|
| abs(x) | integer, real/ integer/real | Calculează valoarea absolută a lui x. |
| sqr(x) | integer, real/ integer/real | Calculează $x^2$ |
| sqrt(x) | integer, real/ real | Calculează $\sqrt{x}$ |
| sin(x) | integer, real/ | Calculează $\sin(x)$ |

| | | |
|---|---|---|
| | real | |
| cos(x) | integer, real/ real | Calculează cos $(x)$ |
| arctan(x) | integer, real/ real | Calculează arctg $(x)$ |
| ln(x) | integer, real/ real | Calculează ln $(x)$ |
| exp(x) | integer, real/ real | Calculează $e^x$ |
| random | | Generează un număr aleatoriu real în intervalul (0,1) |
| random(x) | integer | Generează un număr aleatoriu întreg în intervalul [0,1-x] |
| inc(x) | integer/ integer | incrementează valoarea lui x cu 1 |
| dec(x) | integer/ integer | decrementează valoarea lui x cu 1 |
| round(x) | real/ integer | întoarce valoarea rotunjită a parametrului real x |
| trunc(x) | real/ integer | întoarce valoarea întreagă a parametrului real x (taie tot ce e după virgulă) |
| frac(x) | real/ real | întoarce partea de după virgulă a unui număr real. |
| odd(x) | Integer/ boolean | Întoarce rezultatul **true** dacă x este impar și **false** dacă x este par |

**Proceduri predefinite**

| Sintaxa | Tipul argumentului | Descriere |
|---|---|---|
| inc(x,n) | x,n:integer | incrementează valoarea lui x cu n |
| inc(x) | x:integer | incrementează valoarea lui x cu 1 |
| dec(x,n) | x,n:integer | decrementează valoarea lui x cu n |
| dec(x) | x:integer | decrementează valoarea lui x cu 1 |

**Expresii:**

O expresie reprezintă o formulă ce definește calculul unei valori prin aplicarea unor operatori asupra unor operanzi: constante, variabile, funcții, mulțimi. Evaluarea unei expresii se face de la stânga la dreapta respectând reguli stricte de prioritate ale operatorilor. În Pascal există 4 nivele de prioritate și anume:

| Operatori | Prioritate |
|---|---|
| apel funcție, not | prima (cea mai înaltă) |
| _ , / , div, mod, and, shl, shr | a doua |
| +, -, or, xor | a treia |
| =, <>, <, >, <=, >=, in | a patra |

Există trei reguli de bază pentru prioritate Într-o expresie și anume:

- un operand aflat între doi operatori de priorități diferite este legat de operatorul de prioritate mai înaltă.
- un operand aflat între doi operatori de priorități egale este legat de operatorul din stânga sa.
- expresiile din paranteze se evaluează prioritar fiind tratate ca un singur operand.
- într-o expresie se folosesc numai parantezele ronde, ( ), pentru a schimba prioritatea operațiilor.
- expresiile se scriu pe un singur nivel și pot fi divizate în mai multe rânduri.
- operațiile se efectuează conform priorității operatorilor;
- în cazul priorităților egale, operațiile se efectuează de la stânga spre dreapta;
- mai întâi se calculează expresiile dintre paranteze.

La scrierea expresiilor se iau următoarele precauții:

- să nu se omită operatorul înmulțirii, * , între doi operanzi.
- să nu apară doi operatori consecutive: x*-y e scrisă greșit iar x*(-y) e scrisă corect.
- să fie sigur că toți operanzii reprezentați prin variabile au valori definite anterior evaluării expresiei.
- expresiile se scriu pe un singur nivel și pot fi divizate în mai multe rânduri

Exemple:

| Format matematic | Format Pascal |
|---|---|
| $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ | x:=(-b+sqrt(sqr(b)-4*a*c))/(2*a) |
| $x = \frac{\left(1 - e^{-(y+2z)}\right)}{\sqrt{\frac{1}{2 + z^4}}}$ | x:=(1-exp(-(y+2*z)))sqrt(1/(2+sqr(z)*sqr(z))) |

**Tipul de date boolean (logic)**

Tipul de date boolean reprezintă o mulțime cu două elemente (false, true). Pentru acest tip de date sunt predefinite următoarele operații:

- **not** negația (inversia logică, operația logică *Nu*);
- **and** conjuncția (produsul logic, operația logică *ȘI*);
- **or** disjuncția (suma logică, operația logică *SAU*).
- **xor** disjuncția excluzivă (operația logică *SAU exclusiv*).

Tabelul de adevăr al operațiilor logice.

| x | y | **not** x | x **and** y | x **or** y | x **xor** y |
|---|---|---|---|---|---|
| False | false | true | false | false | false |
| false | true | true | false | true | true |
| true | false | false | false | true | true |
| true | true | false | true | true | false |

Operatori de relație: = (egal), <> (neegal), < (mai mic), > (mai mare), <= (mai mic sau egal), >= (mai mare sau egal). Acești operatori sunt binari, necesită operanzi de tip compatibili și întorc rezultat de tip boolean.

Cu ajutorul operanzilor, operatorilor de relație și operațiilor logice se alcătuiesc expresii logice. Evaluarea lor se realizează în baza regulilor de prioritate menționate mai sus. Exemplu: a:=7; b:=3; i:=5;

1) (abs(a-5)>0)and((b-5)<0);
2) not (odd(i)) or (a>b)

Operanzii logici (**not, and, or, xor**) acționează și asupra datelor de tip **integer** acționând asupra fiecărui bit din reprezentarea binară numărului întreg. Suplimentar mai sunt definiți operanzii (**shl** - deplasare stânga, **shr** - deplasare dreapta).

**Tipul de date char (caracter)**

- reprezintă mulțime finită și ordonată de caractere din setul ASCII ( American Standard Code for Information Interchange ) extins. Sintaxa de declarare a unei variabile de tip **char** este: **var** c:char. Variabililor de tip char li se pot atribui valori de caractere ASCII cuprinse între apostrofe. Exemplu: c:='A';

Pentru tipul de date **char** este predefinită funcția chr(n) care întoarce caracterul cu numărul de ordine n.

Tipurile de date **integer**, **boolean** și **char** sunt mulțimi ordonate. Pentru ele sun predefinite următoarele funcții: **ord**(x)-întoarce numărul de ordine al elementului în mulțime; **pred**(x) și **succ**(x) -întoarce predecesorul și respectiv, succesorul elementului x.

În Pascal, utilizatorul poate să declare propriile tipuri de ordinale respectând următoarea sintaxă:

**TYPE** culoare=(alb, rosu, galben, verde, albastru);

**Submulțimi.** Din elementele unei mulțimi ordinale pot fi create intervale (domenii) și submulțimi.

Exemple de intervale: **var** a:5..10; b: 'a'..'z';

Prin aceste declarații variabila a reprezintă mulțimea ordonată a numerelor întregi de la 5 până la 10 iar b mulțimea ordonată a caracterelor de la 'a' până la ..'z'.

Din elementele unei tip ordinal, cu numărul maximal de elemente 256, pot fi create mulțimi (seturi) cu următoarea sintaxă de declarare: **var** s:**set** of **char**; b:**set** of **byte**; Elementele unei mulțimi sunt delimitate de parantezele [ ]. Exemplu:

s:=[1,4,6..9]; – este mulțimea numerelor întregi (1,4,6,7,8,9); b:=[‘D’,’e’,’l’,’p’,’h’,’i’] – este submulțimea caracterelor din cuvântul Delphi.

Notă: elementele unei mulțimi aparțin tipului de bază ordinal dar nu sunt ordonate în interiorul ei.

Pentru mulțimi este predefinită operația de includere **in**.

Exemplu: ’e’ **in** [‘D’,’e’,’l’,’p’,’h’,’i’] este o expresie logică care întoarce rezultatul **true**; 2 **in** [1,4,6..9] întoarce rezultatul **false.**

**Atenție**! Submulțimile cu caracter (diacritice) nu sunt suportate.

**Constante.**

În Pascal constantele se declară cu ajutorul cuvântului rezervat **const**. Se utilizează două tipuri de declarare a constantelor (fără tip (constante adevărate) și cu tip (constante cu valori modificabile)

Exemple1: const a=5; b=1.53; c=’A’; Tipul acestor constante se determină la etapa de compilare a programului reieșind din tipul expresiei care se atribuie. Deci a va fi de tip întreg, b de tip real, c de tip caracter. Valorile acestor constante nu pot fi modificate în codul programului.

Exemplu2: const a:integer=5; b:extended=1.53; c:char=’A’; În acest caz valorile constantelor pot fi modificate în codul programului. Aceste constante se mai numesc variabile inițializate.

## Instrucțiunile de bază

**Instrucțiunea de atribuire**. Sintaxa:

<identificator de variabilă> := <expresie> Exemplu: a:=5; b:=b+7;

**Notă!** În mod obligatoriu tipul expresie trebuie să coincidă cu tipul variabile.

**Instrucțiunea compusă** reprezintă o secvență de instrucțiuni considerată ca un tot și executată în ordinea în care sunt scrise instrucțiunile în secvență. Instrucțiunile din secvență sunt separate prin **;** și încadrate în parantezele de instrucțiune **begin** și **end**.

**Instrucțiuni condiționale (if, case)**. O instrucțiune condițională selectează pentru execuție o singură instrucțiune ( sau nici o instrucțiune ) din instrucțiunile sale componente.

## Instrucțiunea If.

Sintaxa instrucțiunii If are două variante:

Varianta1. **If** <expresie logică> **Then** <instrucțiune1>

Dacă rezultatul evaluării expresiei logice este TRUE se execută instrucțiune1. În caz contrar se trece la următoarea instrucțiune din program.

Varianta 2. **If** < expresie logică> **Then** <instrucțiune1> **else** <instrucțiune2>

Dacă rezultatul evaluării expresiei logice este TRUE se execută instrucțiune1. În caz contrar se execută instrucțiune2.

**Atenție!**: delimitatorul **;** nu trebuie pus înaintea lui ELSE deoarece duce la terminarea deciziei **If** fără a considera și cea de a doua alternativă

## Instrucțiunea Case

Instrucțiunea servește pentru alegerea și execuția unei singure instrucțiuni dintre mai multe instrucțiuni componente în funcție de valoarea unei expresii scalare de tip ordinal, numită expresie selector. Are următoarea sintaxă:

**case** < expresie selector> **of**
    < listă de constante1>: < instrucțiune1>;
    < listă de constante2>: < instrucțiune2>;
.....
    < listă de constanteN>: < instrucțiuneN>;
**else** < instrucțiuneN+1>;
**end**;
< listă de constante> este o listă de constante de același tip ca și expresia selector.

**Notă1:** Constantele în listă trebuie să fie unice. Listele de constante, de asemenea, trebuie să fie unice.

**Notă2:** Dacă tipul selectorului este **char** atunci în lista de constante pot fi incluse caractere cu diacritice.

## Instrucțiunea For

**Instrucțiunea For** servește pentru organizarea ciclurilor. Are două variante:

**Sintaxa variantei1**.
**for** v:= < expresie1> **to** < expresie2 > **do** < instrucțiune simplă sau compusă>
Sintaxa variantei 2.
**for** v:= < expresie1> **downto** < expresie2 > **do** < instrucțiune simplă sau compusă>
**v** este variabila de control a ciclului. În mod obligatoriu trebuie să fie declarată de tip ordinal. Expresiile 1 și 2 trebuie, în mod obligatoriu, să întoarcă rezultate compatibile cu tipul variabilei ciclului.

Când se intră într-o instrucțiune **for** se evaluează o singură dat expresiile 1 și 2 prin care se determină valorile inițială și finală a variabilei ciclului. instrucțiunea ciclului se execută o dată pentru fiecare valoare a variabilei de control.

Variabila de control începe întotdeauna cu valoarea inițială. când instrucțiunea **for** folosește **to** valoarea variabilei de control crește cu unu la fiecare repetare. dacă valoarea inițială este mai mare decât valoarea finală, instrucțiune nu se execută.

Când instrucțiunea **for** folosește **downto**, valoarea variabilei de control scade cu unu la fiecare repetare. dacă valoarea inițială este mai mică decât valoarea finală, instrucțiune nu se execută.

**Notă**: se interzice schimbarea forțată a valorii variabilei de control a ciclului. Aceast[ eroare se semnalează la etapa de compilare;

**Atenție!** Includerea după **do** a simbolului **;** va provoca o eroare de calcul care nu este semnalată la etapa de compilare aprogramului.

## Instrucțiunea Repeat

Instrucțiunea repeat...until servește pentru organizarea ciclurilor și are următoarea sintaxă:

**repeat** <succesiune de instrucțiuni> **until** < expresie logică>

Succesiunea de instrucțiuni între cuvintele rezervate**, repeat** și **until**, se execută în mod repetat până când expresia logică întoarce rezultatul **true**. Succesiunea de instrucțiuni este executată cel puțin o dată deoarece expresia logică este evaluată după execuția succesiunii de instrucțiuni.

## Instrucțiunea While

Instrucțiunea **while..do** servește pentru organizarea ciclurilor și are următoarea sintaxă:

**while** < expresie logică> **do** < instrucțiune simplă sau compusă a ciclului>

Expresia logică între cuvintele rezervate **while** și **do** este evaluată înainte ca instrucțiunea ciclului să fie executată. Instrucțiunea ciclului este executată repetat cât timp expresia logică dă rezultatul **true**; dacă expresia logică dă **false** la început, instrucțiunea ciclului nu se execută nici o dată.

**Atenție!** După **do** nu se admite instrucțiunea vidă **;**.

| Exemplul 1: Calcularea sumei numerelor întregi de la 1 până la 100 |
|---|

| for | repeat | until |
|---|---|---|
| `var i,s:integer;`<br>`begin s:=0;`<br>`for i:=1 to 100 do`<br>`s:=s+i;`<br>`end;` | `var i,s:integer;`<br>`begin s:=0; i:=0;`<br>`repeat`<br>`i:=i+1;s:=s+i;`<br>`until i>=100`<br>`end;` | `var i,s:integer;`<br>`begin s:=0; i:=0;`<br>`while (i<100) do`<br>`begin`<br>`i:=i+1;s:=s+i;`<br>`end; end;` |

Exemplul 2. Calcularea sumei tuturor numerelor de la 1 până la 100 care se împart exact la 3,5,7,9. Variantele de implementare **case** și **if**.

```
var i,s,n:integer;
begin
 s:=0;
 for I := 0 to 100 do
 begin
   n:=i mod 10;
   case n of
   3,5,7,9:s:=s+i;
   end;
 end;
end;
```

```
var i,s,n:integer;
begin
s:=0;
for I := 0 to 100 do
 begin
   n:=i mod 10;
   if n=3 then s:=s+i;
   if n=5 then s:=s+i;
   if n=7 then s:=s+i;
   if n=9 then s:=s+i;
end;
end;
```

## Subprograme (proceduri și funcții)

Procedurile și funcțiile permit structurarea programelor complexe, fiecare procedură sau funcție realizând complet o sarcină concretă în cadrul programului în care apare.

Sintaxa de declararea a procedurilor:

**procedure** <identificator procedură> (<listă de parametri formali>);
<declarații locale>
Begin
<instrucțiuni>
End;

Sintaxa de declararea a funcțiilor:

**function** <identificator funcție> (<listă de parametri formali>):tipF;
<declarații locale>
Begin
<instrucțiuni>

End;

(<listă de parametri formali> este o listă de parametri formali valoare și variabilă pentru care se indică prin : tipul. Parametrii formali variabilă sunt precedați de cuvântul rezervat **var** și un spațiu.

**Notă:** În blocul de instrucțiuni al funcției, în mod obligatoriu identificatorului funcției i se atribuie un rezultat calculat, tipul căruia va coincide cu tipul declarat tipF al funcției.

**Sintaxa de apel a subprogramelor**.

**Procedurile** se apelează într-o instrucțiune care conține identificatorul procedurii urmat de lista parametrilor actuali, cuprinsă în paranteze. În mod obligatoriu tipul parametrilor actuali trebuie să coincidă cu tipul parametrilor formali. Cu ajutorul parametrilor formali și actuali se realizează comunicarea între subprograme și programul de bază în care se apelează procedura. Cu ajutorul parametrilor formali valoare se transmit datele în subprogram iar cu ajutorul parametrilor formali variabilă se recuperează rezultatele calculate în subprogram.

**Funcțiile** pot fi apelate în expresii, la fel ca și cele predefinite, respectând tipul rezultatului pe care îl întoarce procedura.

Exemple: Se dă un număr de tip integer cu multe cifre. Să se scrie un subprogram care ava calcula suma tuturor cifrelor.

| Subprogram funcție | Subprogram procedură |
|---|---|
| `function FSC(n:integer):integer;`<br>`var s:integer;`<br>`begin`<br>`s:=0;`<br>`repeat`<br>`s:=s+n mod 10;n:=n div 10;`<br>`until n=0 ;`<br>`FSC:=s;end;` | `Procedure PSC(n:integer;`<br>`var s:integer);`<br>`begin`<br>`s:=0;`<br>`repeat`<br>`s:=s+n mod 10;n:=n div 10;`<br>`until n=0 ;`<br>`end;` |
| În secvențele următoare prezentăm codul care utilizează aceste subprograme pentru calcularea sumei cifrelor tuturor numerelor de la 1 până la 100. | |
| `var i,s:integer;`<br>`begin` | `var i,s,r:integer;`<br>`begin` |

```
 s:=0;
 for i:=1 to 100 do
 s:=s+FSc(i);
 label1.Text:=inttostr(s);
end;
```

```
s:=0;
for i:=1 to 100 do
begin PSc(i,r);
 s:=s+r;
 label1.Text:=inttostr(s);
end;end;
```

## Exemple de aplicații vizuale.

Pentru studierea limbajului de programare Pascal în baza interfeței vizuale Delphi, la etapa inițială, utilizatorul trebuie să cunoască un set foarte limitat de componente Delphi și noțiuni foarte superficiale de Programare Orientată pe Obiecte (POO).

O componentă este un exemplar al unei clase predefinite în mediul de programare Delphi. În momentul plasării pe forma a unei componente, mediul de programate îi atribuie în mod automat un nume (identificator) și îl declară ca parte componentă a tipului de date TForm1.

Din POO, utilizatorul trebuie să înțeleagă că componentele sunt obiecte în care sunt unite într-un tot întreg datele (câmpuri) și subprogramele (metode) care prelucrează aceste date. Accesul la câmpurile și metodele unui obiect se realizează prin indicarea numelui și identificatorul câmpului sau metodei separate prin (punct). O parte din câmpurile (proprietățile) și metodele (evenimentele) componentelor pot fi consultate și modificate la etapa de design a aplicației vizuale.

În baza acestei informații utilizatorul poate crea aplicații vizuale, suficient de atractive, utilizând componente predefinite și programând diverse evenimente disponibile în inspectorul de obiecte. Acest stil de programare se mai numește **Programare Orientată pe Evenimente**.

Exemplu: Dacă dorim să realizăm o aplicație vizuală în care la realizarea unui click pe componenta Button1 să se calculeze suma tuturor numerelor de la 1 până la 100 și rezultatul să fie afișat pe bara de titlu a formei trebuie să scriem următorul cod pentru evenimentul OnClick al butonului Button1. Structura acestei proceduri se afișează la etapa de design când realizăm dublu-click pe componenta Button1 sau în inspectorul de obiecte la activarea câmpului OnClick.

| Structura evenimentului OnClick creată în mod automat | Evenimentului OnClick completat de utilizator |
|---|---|
| **procedure** TForm1.Button1Click (Sender: TObject);<br><br>**begin**<br><br><br><br>**end;** | **procedure** TForm1.Button1Click (Sender: TObject);<br><br>**var** i,s:integer;<br><br>**begin** s:=0;<br><br>**for** i:=1 **to** 100 **do** s:=s+i;<br><br>form1.caption:=inttostr(s);<br><br>**end;** |

Listingul integral al aplicației este reprezentat în figura 11. Menționăm că de la tastatură utilizatorul a introdus numai codul încadrat cu culoare roșie.

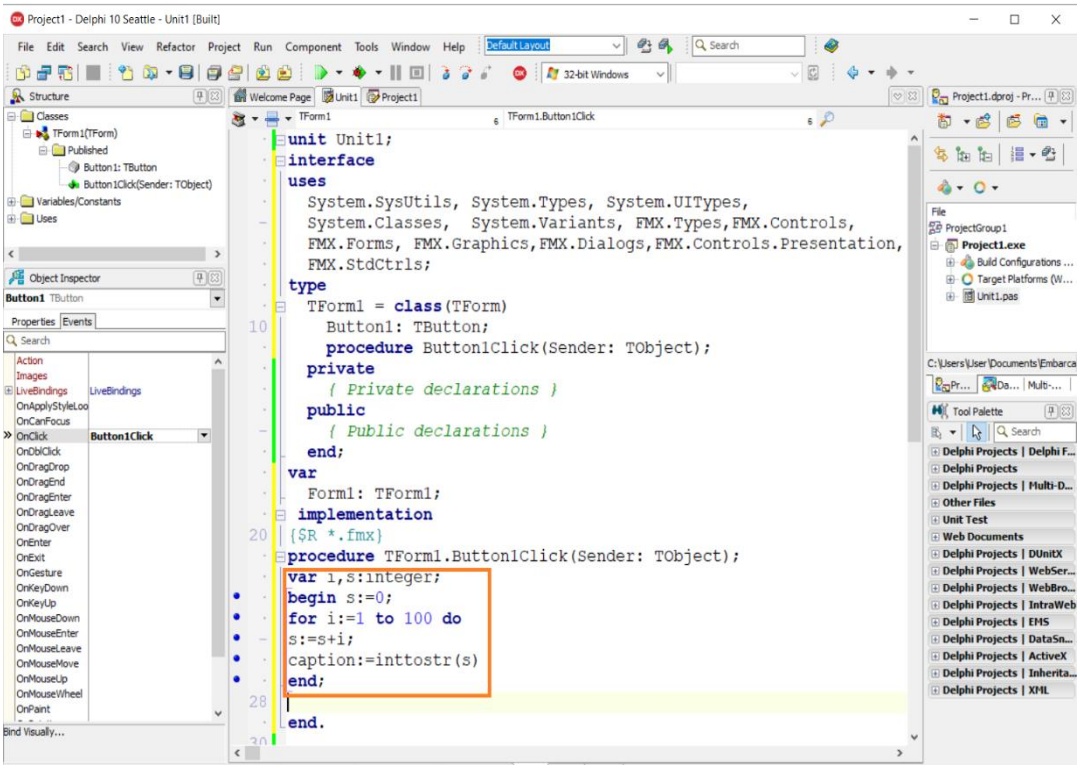



Figura 11. Listingul aplicației vizuale Delphi.

În continuare prezentăm succint câteva componente frecvent utilizate în aplicațiile vizuale Delphi:

**TButton** (paleta Standard) – este un obiect de formă rectangulară 

pentru care este implementat efectul vizual de apăsare. Proprietatea Text:String poate fi modificată la etapa de design și reprezintă stringul care se afișează pe suprafața butonului. De regulă se utilizează pentru programarea evenimentului OnClick care poate fi deschis din inspectorul de obiecte sau prin dublu click pe componentă.

**TLabel** (paleta Standard) – este un obiect de formă rectangulară fără contur pe suprafața căruia poate fi afișat un șir de caractere care se specifică în proprietatea text accesibila la etapa de design și de programare. De regulă se utilizează pentru afișarea unei informații statice (un rezultat obținut prin cod de programare).

**TEdit** (paleta Standard) – este un obiect de formă rectangulară cu contur, în interiorul căruia la etapa de executare a programului poate fi redactat, de la tastatură, un șir de caractere. Acest șir de caractere se asociază cu proprietatea Text a obiectului TEdit și poate fi utilizat pentru introducerea interactivă a datelor în programul care se execută.

**Notă. Cu ajutorul unei componente TButton și a câtorva componente TLabel și TEdit poate fi realizată, în regim vizual interactiv, orice aplicație de tip consolă.** Întregul cod al aplicației se scrie în evenimentele OnClick ale butoanelor. Variabilele, constantele declarate în partea de interfață

## Tipuri de date structurate.

Spre deosebire de tipurile de date simple (predefinite), instanțele de tip structurat dețin mai multe valori. Tipurile structurate includ seturi, tablouri, înregistrări și fișiere, precum și tipuri de clase, referințe de clasă și interfețe. Cu excepția seturilor, care dețin numai valori ordinale, tipurile structurate pot conține alte tipuri structurate; un tip poate avea nivele nelimitate de structurare.

## Tipul de date Set

Un set este o colecție de valori de același tip ordinal. Valorile nu au o ordine inerentă și nici nu este semnificativ ca o valoare să fie inclusă de două ori într-un set.

Gama unui tip de set este setul de valori al unui tip ordinal specific, numit tip de bază; adică valorile posibile ale tipului de set sunt toate subseturile tipului de bază, inclusiv setul gol. Tipul de bază nu poate avea mai mult de 256 de valori posibile. Sintaxa de declarare are forma:

**set of** <tip de bază ordinal>

Datorită limitărilor de dimensiune pentru tipurile de bază, seturile sunt de obicei definite cu subintervale. Exemple:

var s1:set of 1..10; s2:set of 'a'..'z';

În baza acestor declarații putem crea următoarele construcții:

s1:=[1,5,3..7]; s2:=[‘b’,’d’, ‘k’..’z’];

## Tipul de date array (tablou).

Pot fi declarate tablouri unidimensionale și multidimensionale.

**Tablourile unidimensionale** sunt structuri omogene formate dintr-un număr fix de componente de același tip, numit tip de bază. Sintaxa de declarare:

Var <identificator> :array[domeniu de tip ordinal] of <tip de bază>

Exemplu: **var** T:**array**[1..10] **of** integer; - definește un tablou cu 10 elemente de tip integer.

Se admite declararea constantelor de tip tablou. Exemplu:

**const** T1D:**array**[0..4] **of** integer=(1,2,3,4,5);

Accesarea unei componente a tabloului se face prin identificatorul tabloului și valoarea indicelui sau o expresie al cărui rezultat trebuie să se afle în domeniul definit de tipul indicelui.

**Tablouri multidimensionale.** Componentele unui tablou pot fi de orice tip, inclusiv de tip tablou. Astfel dacă tipul de bază este un tip tablou se poate trata rezultatul ca un tablou de tablouri sau ca un tablou cu mai multe dimensiuni.

Exemplu: **var** T2D:**array**[1..3] of **array** [1..4] **of** integer;

**var** T2D:**array**[1..3,1..4] **of** integer; sunt două moduri de definire a unui tablou 2D (matrice) cu doi indici. Primul indice indică linia iar al doilea coloana.

Exemplu de constantă de tip tablou 2D:

**const** T2D:**array**[1..2,1..3]**of** integer=((1,2,3),(4,5,6));

**Tablouri dinamice**. Tablourile dinamice se declară la fel ca și cele statice dar fără indicarea domeniului de valori a indicilor.

Exemple: **var** TD1D:**array of** integer; TD2D:**array of array of** integer;

Declararea tablourilor dinamice nu alocă memorie pentru componente. Memoria se alocă dinamic prin apelul procedurii **setlength**(IDT,N) unde IDT este identificatorul tabloului iar N este numărul de elemente. În cazul tablourilor dinamice multidimensionale procedura **setlength** are mai mulți parametri, de exemplu alocarea memoriei pentru un tablou 2D se realizează prin instrucțiunea **setlength**(T2D,L,C) unde L și C sunt numărul de linii și respectiv, coloane.

Funcțiile **high**(IDT) și **low**(IDT) determină valoarea maximală și, respectiv minimală a indicelui tabloului.

Tablourile dinamice pot fi utilizați în calitate de parametri formali și actuali pentru proceduri și funcții.

**Notă**: Indicii tablourilor dinamice sunt numerotează de la 0.

## Tipul de date Record (înregistrare)

O **înregistrare** reprezintă un set eterogen de elemente. Fiecare element este numit câmp; declarația unui tip de înregistrare specifică un nume și un tip pentru fiecare câmp. Sintaxa unei declarații de tip înregistrare este:

type Nume=record

<lista campuri1>:tip;

……

<lista campuriN>:tip;

end;

unde Nume este un identificator valid; record-cuvânt rezervat; <lista campuri1> -, este un identificator valid sau o listă de identificatori delimitați prin virgule și tip declarat în final prin :.

Un câmp al unei variabile de tip record se accesează prin idicarea numelui variabilei și a câmpului separate prin punct.

**Constantele de tip Record** se declară conform următoarei sitaxe:

**Const** <NumeConst>: <tip record> =(<câmp1>:valoare; (<câmp2>:valoare;...)

În următorul cod de program se declară tipul TData cu 3 câmpuri, o constantă și două variabile de tipul TData.

```
procedure TForm1.BClick(Sender:TObject);
type Tdata=record
   A: integer;
   L:(ian,feb,mar,apr,mai,iun,iul,aug,sep,oct,noi,dec);
   Z:1..31;
 end;
```

```
 const cd:TData=(A:2020;L:Dec;z:19);
 var d1,d2:TData;
begin  d1.A:=2019; d1.L:=ian; d1.Z:=15;
       d2:=cd;
end;
```

## Tipul de date string (șir de caractere)

Un string reprezintă un șir de caractere. Compilatoarele pentru ultimele versiuni Delphi recunosc următoarele tipuri de șiruri de caractere: **ShortString** (compatibil string în versiunile precedente Pascal și Delphi), **AnsiString**, **Unicod( UnicodeString** și **WideString**).

Tipul de date predefinit în Delphi pentru gestionarea șirurilor de caractere are identificatorul **string** și este identic cu **UnicodeString** pentru care numărul maximal de caractere gestionate este ~$2^{31}$=2147483648 caractere. (în pascal tipul de date string gestionează 255 caractere)

Pentru tipul de date string este predefinită o singură operație, + (concatenarea). Exemplu; s1:='abc'; s2:='def'; s:=s1+s2; s='absdef'.

Pentru tipurile de date string sunt predefinite diverse funcții și proceduri. În următorul tabel prezentăm succint, doar, câteva:

| Unitatea | Sintaxa de declarare / descriere succintă |
|---|---|
| System | **function** Length(S:string):Integer; returnează numărul de caractere în șirul S. S[0] întoarce același rezultat |
| System | **function** Pos(SS,S: String): Integer;<br>returnează prima poziția a subșirului SS în șirul S. |
| System | **function** Copy ( Source : string; StartChar, Count : Integer ): string; |
| System | **procedure** Delete ( var Source : string; StartChar : Integer; Count : Integer ) ; |
| System | **procedure** Insert ( const InsertStr : string; var TargetStr : string; Position : Integer ) ; |
| System | **function** StringReplace(const S, OldS, NewS: string; Flags: TReplaceFlags): string;<br>**type** TReplaceFlags = set of (rfReplaceAll, rfIgnoreCase); |
| System. StrUtils | **function** StuffString(const AText: string; AStart, ALength: Cardinal; const ASubText: string): string; |

| | |
|---|---|
| System. SysUtils | **function** Trim(const S: string): string; |
| System. SysUtils | **function** TrimLeft(const S: string): string; |
| System. SysUtils | **function** TrimRight(const S: string): string; |
| System. SysUtils | **function** LowerCase(const S: string): string; |
| System. SysUtils | **function** Upper Case(const S: string): string; |
| System. SysUtils | **function** CompareStr(const S1, S2: string): Integer; |
| | Utilizând sistemul de ajutor online Delphi, http://docwiki.embarcadero.com/Libraries/Sydney/en/System.SysUtils descoperiți alte funcții utile |

## Funcții și proceduri pentru conversia datelor

| **Funcții și proceduri pentru conversia datelor numerice** (System.SysUtils) |
|---|
| function CurrToStr(Value: Currency): string; |
| function CurrToStrF(Value: Currency; Format: TFloatFormat; Digits: Integer): string; |
| function FloatToStr(Value: Extended): string; |

| |
|---|
| function FloatToStrF(Value: Extended; Format: TFloatFormat; Precision, Digits: Integer): string; |
| function StrToCurr(const S: string): Currency; |
| function StrToFloat(const S: string): Extended; |
| function StrToFloatDef(const S: string; const Default: Extended): Extended; |
| function StrToInt(const S: string): Integer; |
| function StrToInt64(const S: string): Int64; |

TFloatFormat (descriere)

| TFloatFormat = (ffGeneral, ffExponent, ffFixed, ffNumber, ffCurrency); | |
|---|---|
| ffGeneral | Format de număr general. Valoarea este convertită în cel mai scurt șir zecimal posibil utilizând formatul științific. Zerourile de sfârșit sunt eliminate din șirul rezultat. |
| ffExponent | Format științific. Valoarea este convertită într-un șir al formularului "-d.ddd... E+dddd". Șirul rezultat începe cu semnul minus dacă numărul este negativ, iar o cifră precede întotdeauna virgula zecimală. Numărul total de cifre înaintea exponentului din șirul rezultat (inclusiv cel dinaintea punctului zecimal) este dat de parametrul Preciziei. Caracterul exponent "E" din șirul rezultat este întotdeauna urmat de un semn plus sau minus și de până la patru cifre. Parametrul Cifre specifică numărul minim de cifre din exponent (între 0 și 4). |
| ffFixed | Format punct fix. Valoarea este convertită într-un șir al formularului "-ddd.ddd...". Șirul rezultat începe cu semnul minus dacă numărul este negativ și cel puțin o cifră precede întotdeauna virgula zecimală. Numărul de cifre după virgula zecimală este dat de parametrul Cifre -- acesta trebuie să fie între 0 și 18. Dacă numărul de cifre din stânga punctului zecimal este mai mare decât precizia specificată, valoarea rezultată va utiliza formatul științific. Valorile rezultate sunt căptușite cu zerouri atunci când parametrul Cifre este mai mare decât numărul de cifre dictate de precizie. De exemplu, având în vedere ffFixed cu precizie setat la 5 și cifre setate la 3, fomatting 345.6789 |

| | |
|---|---|
| | dă șirul "345.680", capitonarea cu zerouri după ce precizia este îndeplinită. |
| ffNumber | Format de număr. Valoarea este convertită într-un șir al formularului "-d,ddd,ddd.ddd...". Formatul ffNumber corespunde formatului ffFixed, cu excepția faptului că șirul rezultat conține separatoare de miime. Valoarea rezultată este căptușită cu zerouri atunci când parametrul Cifre este mai mare decât numărul de cifre dictate de precizie. De exemplu, având în vedere ffNumber cu precizie setat la 5 și cifre setate la 3, fomatting 345.6789 dă șirul "345.680", capitonare cu zerouri după ce precizia este îndeplinită. |
| ffCurrency | Format monetar. Valoarea este convertită într-un șir care reprezintă o sumă monetară. Conversia este controlată de variabilele globale CurrencyString, CurrencyFormat, NegCurrFormat, ThousandSeparatorși DecimalSeparator, toate acestea sunt inițializate din Format monedă în secțiunea Internațională din Panoul de control Windows. Numărul de cifre după virgula zecimală este dat de parametrul Cifre care trebuie să fie între 0 și 18. Valoarea rezultată este completată cu zerouri atunci când parametrul Cifre este mai mare decât numărul de cifre dictate de precizie. De exemplu, având în vedere ffCurrency cu precizie setat la 5 și cifre setate la 3, formatarea nunărului 345.6789 dă șirul "345.680", completat cu zerouri după ce precizia este îndeplinită. |

## Unitatea Math

Math este o unitate predefinită în Delphi FMX în care sunt implementate multe funcții și proceduri cu ajutorul cărora se mărește substanțial productivitate de scriere a aplicațiilor în care sunt necesare diverse calcule cu date de tip numeric. În tabelul următor prezentăm câteva din aceste rutine

| Funcții trigonometrice |
|---|
| function ArcCos(Var X : Extended): Extended;<br>function ArcSin(Var X : Extended): Extended;<br>function ArcTan2(Var Y, X:Extended):Extended;<br>procedure SinCos(Var Theta: Extended; var Sin, Cos: Extended);<br>function Tan(Var X: Extended): Extended;<br>function Cotan(Var X: Extended): Extended; |

| |
|---|
| function Secant(Var X: Extended): Extended;<br>function Cosecant(Var X: Extended): Extended;<br>function Hypot(Var X, Y: Extended): Extended; { Sqrt(X**2 + Y**2) } |
| Funcții pentru conversia unghiurilor |
| function RadToDeg(const Radians: Extended): Extended;<br>function DegToRad(const Degrees:Extended): Extended; { Radians := Degrees * PI / 180} |
| Funcții hiperbolice |
| function Cot(const X: Extended): Extended;<br>function Sec(const X: Extended): Extended;<br>function Csc(const X: Extended): Extended;<br>function Cosh(const X: Extended): Extended;<br>function Sinh(const X: Extended): Extended;<br>function Tanh(const X: Extended): Extended;<br>function CotH(const X: Extended): Extended;<br>function SecH(const X: Extended): Extended;<br>function CscH(const X: Extended): Extended;<br>function ArcCot(const X: Extended): Extended;<br>function ArcSec(const X: Extended): Extended;<br>function ArcCsc(const X: Extended): Extended;<br>function ArcCosh(const X: Extended): Extended;<br>function ArcSinh(const X: Extended): Extended;<br>function ArcTanh(const X: Extended): Extended;<br>function ArcCotH(const X: Extended): Extended;<br>function ArcSecH(const X: Extended): Extended;<br>function ArcCscH(const X: Extended): Extended; |
| Funcții logaritmice |
| function LnXP1(const X: Extended): Extended;<br>function Log10(const X: Extended): Extended;<br>function Log2(const X: Extended): Extended;<br>function LogN(const Base, X: Extended): Extended; |
| Funcții exponențiale |
| function IntPower(const Base: Extended; const Exponent: Integer): Extended;<br>function Power(const Base, Exponent: Extended): Extended; |
| Funcții statistice |
| function Mean(const Data: array of Extended): Extended;<br>(calculează valuarea medie a unui tablou de date)<br>function Sum(const Data: array of Extended): Extended;<br>function SumInt(const Data: array of Integer): Integer;<br>function SumOfSquares(const Data: array of Extended): Extended;<br>procedure SumsAndSquares(const Data: array of Extended;<br>var Sum, SumOfSquares: Extended); |

| |
|---|
| function MinValue(const Data: array of Extended): Extended;<br>function MinIntValue(const Data: array of Integer): Integer;<br>function Min(const A, B: Integer): Integer;<br>function Min(const A, B: Int64): Int64;<br>function Min(const A, B: UInt64): UInt64;<br>function Min(const A, B: Extended): Extended;<br>function MaxValue(const Data: array of Extended): Extended;<br>function MaxIntValue(const Data: array of Integer): Integer;<br>function Max(const A, B: Integer): Integer;<br>function Max(const A, B: Int64): Int64;<br>function Max(const A, B: UInt64): UInt64;<br>function Max(const A, B: Extended): Extended;<br>function StdDev(const Data: array of Extended): Extended;<br>procedure MeanAndStdDev(const Data: array of Extended; var Mean, StdDev: Extended); |
| Funcții Rnndom (aleatorii) |
| function RandomRange(const AFrom, ATo: Integer): Integer;<br>function RandomFrom(const AValues: array of Integer): Integer;<br>function RandomFrom(const AValues: array of Int64): Int64;<br>function RandomFrom(const AValues: array of UInt64): UInt64;<br>function RandomFrom(const AValues: array of Extended): Extended;<br>function RandG(Mean, StdDev: Single): Single; |

## Mesaje și dialoguri

Pentru comunicarea cu utilizatorul în aplicațiile vizuale se utilizează ferestre cu mesaje și dialoguri. În unitatea **FMX.Dialogs** sunt definite funcții și proceduri pentru afișarea ferestrelor cu mesaje și dialoguri.

**Function** ShowMessage('Mesaj Simplu')

**function** MessageDlg(Var AMessage: string; Var ADialogType: TMsgDlgType; Var AButtons: TMsgDlgButtons; Var AHelpContext: LongInt): Integer;

În AMessage se indică șirul de caractere pe care utilizatorul îl va vedea în fereastra de dialog.

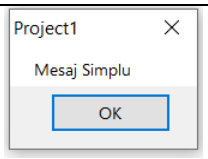
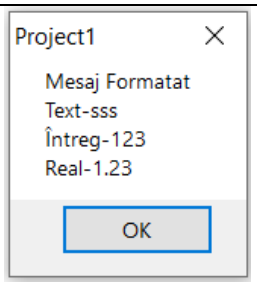
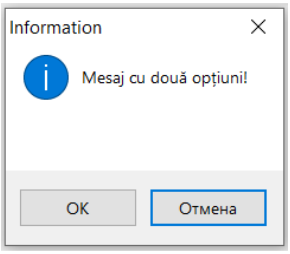
TMsgDlgType=(mtWarning, mtError, mtInformation, mtConfirmation, mtCustom); Elementele acestei mulțimi permit selectarea iconiței și a tipului de mesaj care se afișează în fereastra de dialog.

TMsgDlgBtn =(mbYes, mbNo, mbOK, mbCancel, mbAbort, mbRetry, mbIgnore, mbAll, mbNoToAll, mbYesToAll, mbHelp, mbClose);

TMsgDlgButtons = set of TMsgDlgBtn;

În Delphi FMX fereastra de dialog este de tip modal (blochează accesul la alte ferestre) și poate să conțină unul sau două butoane. Valoarea de tip întreg pe care o întoarce funcția MessageDlg depinde de tipul butonului care a fost apăsat (1:mbOK; 2:mbCansel; 3:mbAbort; 4:mbRetry; 5:mbIgnore; 6:mbYes; 7:mbNo; 8:mbClose). În baza acestor valori se programează finalitatea dialogului.

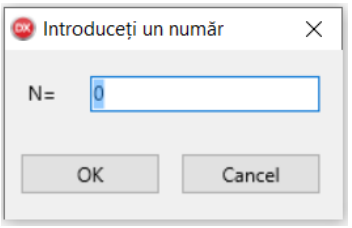
În continuare prezentăm trei exemple care generează ferestre de dialog.

| Exemple de implementare | Rezultate |
|---|---|
| **procedure** TForm1.Button1Click(Sender: TObject);<br>**begin**<br>ShowMessage('Mesaj Simplu')<br>**end;** |  |
| **procedure** TForm1.Button3Click(Sender: TObject);<br>**var** s,msg:string;<br>i:integer;r:extended;<br>**begin**<br>msg:='Mesaj Formatat'+#13+'Text-%s'+#13+<br>'Întreg-%d'+#13+'Real-%f';<br>s:='sss'; i:=123; r:=1.23456;<br>ShowMessageFMT(msg,[s,i,r] )<br>**end;** |  |
| **procedure** TForm1.Button5Click(Sender: TObject);<br>**var** m:integer;<br>**begin**<br>Label1.Text:=' ';<br>m:=MessageDlg('Mesaj cu două opțiuni!',<br>TMsgDlgType.mtInformation,<br>[TMsgDlgBtn.mbOK,TMsgDlgBtn.mbCancel],<br>1, TMsgDlgBtn.mbCancel);<br>**case** m of<br>1:Label1.Text:='A fost selectat Butonul 1 ';<br>2:Label1.Text:='A fost selectat Butonul 2 '; |  |

```
 end;
end;
```

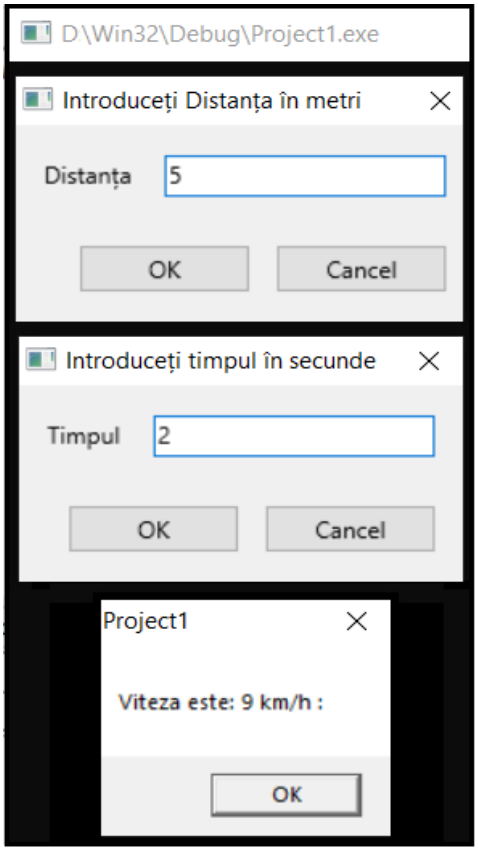
Funcția InputBox(const ACaption, APrompt, ADefault: string): string este declarată în unitatea Dialogs și servește pentru afișarea unei ferestre de dialog cu o castă pentru editarea unui text (de tipul TEdit) și două butoane (OK și Cansel). ACaption reprezintă eticheta situată în stânga TEdit-ului. ADefault reprezintă tectul predefinit afișat în TEdit. Dacă utilizatorul face clic pe butonul **OK** , InputBox returnează textul introdus în controlul TEdit al casetei de dialog. Dacă utilizatorul face clic pe butonul **Cancel**, InputBox returnează valoarea ADefault. Exemplu:

```
procedure TForm1.
Button5Click(Sender: TObject);
var n:integer;
begin
n:=strtoint(InputBox('Introduceți un
număr', 'N=', '0'));
end;
```




Menționăm că instrucțiunile pentru afișarea mesajelor pot fi utilizate ți în aplicațiile consolă. Pentru aceasta ăn compartimentul Uses se adaugă unitatea FMX.Dialogs. Exemplu:

```
program Project1;

{$APPTYPE CONSOLE}

{$R *.res}

uses
System.SysUtils,FMX.Dialogs;

Var T,D,Vms,Vkh:extended;

 Begin

   D := StrTofloat(InputBox('
Introduceți    Distanța    în
metri', 'Distanța', '0'));

   T := StrToFloat(InputBox('
Introduceți     timpul     în
secunde', 'Timpul', '0'));

   Vms := D /  T;

   Vkh:=round((Vms * 3600) /
1000);

   Showmessage('Viteza este:
' + FloatToStr(Vkh) + ' km/h
: ');

end.
```




## Componente Delphi

## Componente pentru gestionarea textelor.

Biblioteca FireMonkey oferă un spectru larg de componente specializate în afișarea și procesarea datelor de tip text. Componenta principală pentru afișarea unui text explicativ este TLabel (etichetă). Componenta principală care se utilizează pentru introducerea și redactarea unui texte de la tastatură este TEdit. Componente asemănătoare mai sunt (TNumberBox, TSpinBox și TComboTrackBar), care oferă diferite variații de introducere și redactare de la tastatură a datelor de tip numeric. O componentă importantă care se utilizează pentru introducerea de la tastatură a unor texte mari, divizate în linii este TMemo.

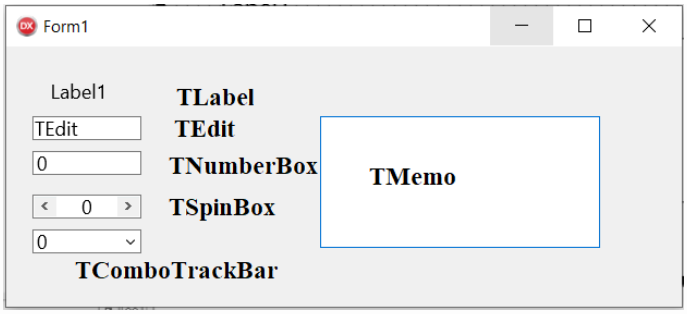



Figura 12.Exemple de componente pentru gestionarea textelor.

Majoritatea proprietăților acestor componente, afișate în Inspectorul de Obiecte, sunt suficient de intuitive întrucât utilizatorul poate să le studieze de sine stătător. Le menționăm pe cele mai importante:

**Text: String** – specifică informația care se afișează în controlul componentei;

**TextSettings** – proprietate compusă cu ajutorul căreia se setează: Fontul, Culoarea, Alinierea(verticală/orizontală)). Pentru TMemo **TextSettings** are câmpul **WordWrap** care permite setarea opțiunii de divizare a liniilor de text**.**

**ReadOnly:Boolean**- interzice (true) redactarea textului. Valoarea predefinită este false.

**Max/Min**- valoarea maximală/minimală a valorilor numerice gestionate.

**ValueType**- Tipul valorilor numerice gestionate (Integer/Float).

Pentru aceste componente sunt implementate metodele (**CopyToClipboard** (Ctrl-C), **PasteFromClipboard**(Ctrl-V), **CutToClipboard**(Ctrl-X)).

Din mulțimea de evenimente a le acestor componente menționăm:

**Evenementul OnChenge** care se declanșează în momentul apăsării pe tasta Enter, după modificarea textului din controlul componentei.

Pentru componenta TMemo, principala proprietate este Lines:TStrings în care se află informația despre șirurile de caractere din fiecare rând a componentei:

Memo1.Lines.Strings[i]:String- stringul din linia i;

memo1.Lines.Count:Integer – numărul de linii;

memo1.Lines.Append(S:String) – adaugă un rând la sfârșit stringul S;

memo1.Lines.Clear – șterge toate rândurile din Memo1;

memo1.Lines.Delete(i:integer)-șterge rândul i;

memo1.Lines.Insert(i:integer, s:String) -inserează după rândul i un rând care conține stringul s;

memo1.Lines.LoadFromFile(nume fișier .txt) - încarcă în memo1 textul din fișierul *.txt;

memo1.Lines.SaveToFile(nume fișier .txt); - salvează textul din memo1 în fișierul *.txt;

## Componenta TStringGrid.

TStrizngGrid este o componentă cu ajutorul cărora pot fi create și gestionate tabele cu celule omogene de string-uri redactabile. și TGrid cu celule neomogene de tipuri diferite. Acestă componentă se află în pagina Grids. Elementele principale ale acestei componente sunt coloanele de tip TStringColumn.

Pentru completarea unei componente TStringGrid cu coloane, la etapa de design se utilizează Editorul de coloane (Items Editor) care se afișează realizând dublu click pe componenta TStringGrid (fig.13).

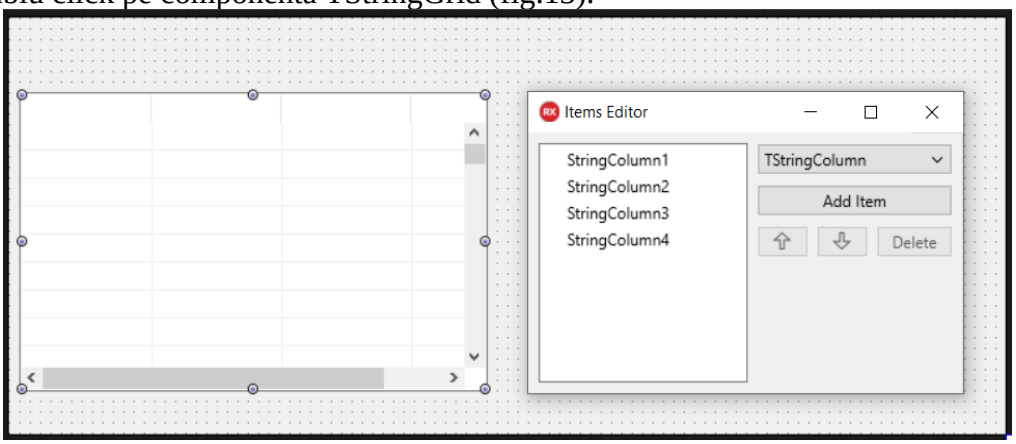



Figura 13. Crearea tabelelor TStringGrid.

La fiecare apăsare pe butonul Add Item se adaugă coloane noi de celule în stringgrid. Din mulțimea de proprietăți a componentei TStringgrid menționăm următoarele:

- **cells[i,j]** de tip string. Această proprietate nu este afișată în Inspectorul de obiecte dar este accesibila la etapa de scriere a codului de programare. În ea se află matricea de stringuri pe care le gestionează componenta TStringgrid.

**Atenție! primul indice (i) este coloana iar al doilea (j) rândul**.

- **Options** conține opțiuni de tip boolean cu ajutorul cărora se setează permisiunile de: editare a celulelor (**Edit**), vizibilitate a liniilor (**RowLines**), coloanelor ( **ColLines**) și antetelor de coloane(**Header**).
- **ShowScrollBars** – permite/interzice afișarea barelor de defilare.
- **TextSettings** – setează opțiuni pentru afișarea textului în celule (culoarea, alinierea, fontul ...).
- **RowCount** – setează numărul de linii

Proprietatea **ColCount** nu este accesibilă la etapa design, iar la etapa de programare această proprietate este accesibilă numai în regim de citire. Modificarea dinamică a numărului de coloane poate fi realizată, doar prin cod de programare.

În continuare prezentăm codul unei aplicații vizuale în care, la apăsarea butonului OK, se creează un tabel cu numărul de coloane și linii specificate în componentele **SiinBox1** și **SiinBox2** (fig.14). Pentru ca la etapa de executare a programului să nu se vadă antetul coloanelor, pentru câmpul **Header** al proprietate Options a componentei Stringgrid1 a fost selectată valoare **false.**

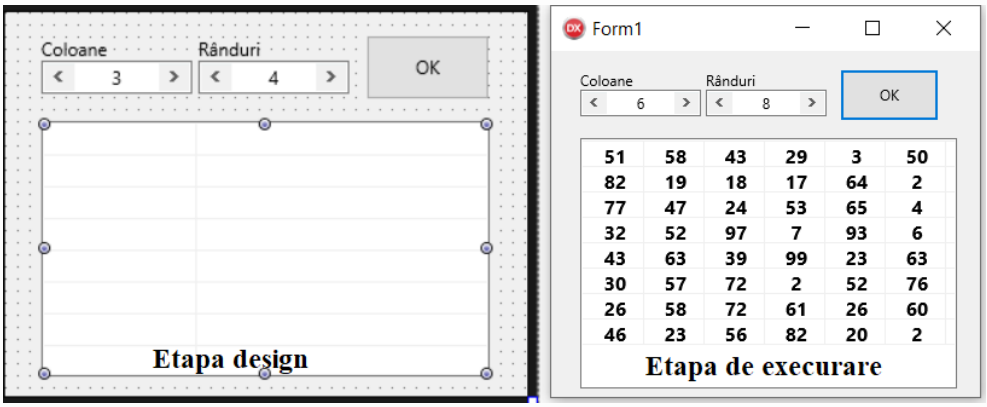



Figura 14. Designul și executarea aplicației.

Codul de programare pentru butonul OK este următorul:

```
procedure TForm1.Button1Click(Sender: TObject);
Var nc,nr,i,j:integer; ww:extended;
begin
 while   stringgrid1.ColumnCount<>0   do   stringgrid1.Columns[0].
Destroy;
 nc:=round(spinbox1.Value); nr:=round(spinbox2.Value);
 stringgrid1.RowCount:=nr; ww:=0;
 for i:=0 to nc-1 do
 begin
  StringGrid1.AddObject(TStringColumn.Create(stringgrid1));
  StringGrid1.Columns[i].Width:=50;
  ww:=ww+StringGrid1.Columns[i].Width+2
 end;
 for i:=0 to nc-1 do
 begin
  For j:=0 to nr-1 do
  StringGrid1.Cells[i,j]:=inttostr(random(100));//Celule se
                          // completează cu numere întregi
 end;

StringGrid1.Height:=(stringgrid1.RowCount+1)*(StringGrid1.RowHeight
+2);
 StringGrid1.Width:=ww;
end;
```

## Componente de tip Button.

Butoanele sunt componente care pot fi apăsate, activate și dezactivate. De regulă toate au proprietatea **Text** care specifică șirul de caractere afișat pe suprafața butonului.

**TButton** - este un simplu buton care se utilizează cel mai frecvent în aplicațiile vizuale. De regulă se programează evenimentul OnClick care se accesează în pagina **Events** a Inspectorului de obiecte (**Object Inspector**) sau prin dublu click pe suprafața butonului

```
procedure TForm1.Button1Click(Sender: TObject);
begin


end;
```

Codul scris în această procedură se va realiza numai în cazul când, la etapa de rulare a aplicației se apasă cu mouse-ul pe acest buton.

**TCornerButton** – este un buton care are un aspect grafic mai complex având forma colțurilor personalizată. Proprietatea **CornerType** (**Bevel, InterLine, InterRound, Round, xRadius, yRadius**) setează forma colțurilor și raza curburilor.

**TSpeedButton** – De regulă se utilizează pentru realizare accesului rapid la unele funcții ale aplicației vizuale incluse în opțiuni mai complexe. Are proprietatea **GroupNam**e (butoanele cu același **GroupName** nu pot să se afle în stare apăsată concomitent). Starea de apăsare a butonului poate fi consultată în proprietatea **IsPressed**:boolean.

**TRadioButton** este un buton în formă de cerc. În stare selectată cercul este gol iar în stare selectată are în interior un cerculeț vopsit. Cu ajutorul proprietății **GroupName** aceste butoane se grupează pentru ca starea de apăsare să se excludă reciproc. Starea de selectare a butonului poate fi consultată în proprietatea **IsChecked**:boolean.

**TCheckBox** este un buton care prin click cu ajutorul mouse-ului se bifează/debifează. Starea de bifare poate fi consultată în proprietatea **IsChecked**:boolean. **TCheckBox** nu pot fi grupate deoarece nu au proprietatea **GroupName.**

## Componente de tip container.

Există o serie de componente care special au fost concepute pentru oferirea de spațiu de stocare pentru alte componente. Descriem în continuare câteva:

**TPanel -** reprezintă un panou generic de uz general utilizat pentru a ține mai multe controale în scopuri de organizare. Dacă proprietatea Scale a panelului se modifică atunci toate componente din panel se scalează la fel.

**TCalloutPanel** este un panel care are un indicator în formă de triunghi pe un a din laturile sale. Poziția și dimensiunile indicatorului sunt specificate în proprietățile (**CalloutPosition**, **CalloutLength**, **CalloutWidth)**.

**TGroupBox**. De cele mai multe ori este utilizat în calitate de container pentru gruparea componente de același tip care dețin proprietatea de a fi grupate (**TRadioButton**, **TSpeedButton)** . Proprietatea **Text** conține șirul de caractere care etichetează acest obiect.

**TToolBar** este o componentă cu ajutorul căreia, în aplicațiile vizuale, se creează bare de instrumente în care se includ diverse butoane. De regulă se ajustează la marginea de sus a aplicației vizuale.

**TStatusBar** este o componentă asemănătoare cu **TTooBar** dar se ajustează la marginea de jos a aplicației vizual.

**TTabControl** este un container care conține mai multe file **TTabItems** în care pot fi stocate diverse componente. Filele (TabItems) se accesează prin selectarea cu ajutorul mouse-ului a etichetelor filelor. Gestionarea filelor (adăugare/ștergere) se realizează cu ajutorul editorului de itemi (**Items Editor**) care se afișează realizând dublu click pe suprafața **TabControl**-ului. Informația afișată pe eticheta filei se află în proprietatea text a obiectelor **TabItens**.

Eticheta unei file TabControl poate fi modificată prin proprietatea **Text**.

În fig.15 este prezentat designul unei aplicații care conține componentele prezentate mai sus.

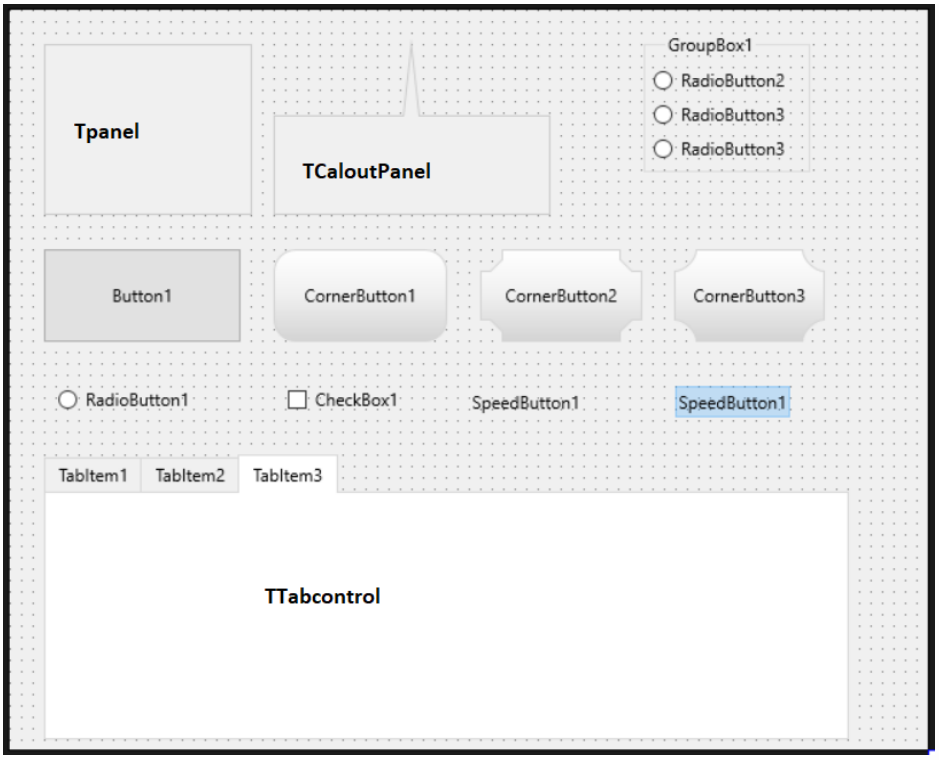



Figura 15. Exemple de componente vizual Delphi FMX.

**Notă**. Controalele TPanel, TCaloutPanel etc nu au publicate proprietăți care le modifică culoare. Prin cod de programare acest efect poate fi realizat pin instrucțiunea:

**(panel1.Controls[0] as TRectangle).Fill.Color:=TAlphacolors.Red**;

## Pagina de componente Layouts (aspecte)

Această pagină conține 12 componente care joacă rolul de containere pentru alte componente vizibile. La etapa de executare conturul acestor componente nu este vizibil.

| | Proprietăți |
|---|---|
| TLayout Se utilizează atunci când trebuie să organizați mai multe controale grafice sub același părinte | |
| TGridLayout Este un control care aranjează componentele copil într-o grilă de celule de dimensiuni egale. | **ItemWidth, ItemHeight** – setează dimensiunile celulelor grilei. **ControlsCount** (numai citire) – conține numărul de controale în GridLayout. |
| TGridPanelLayout Este o grilă de controale Panel în celule căreia por fi | **ColumnCollection și RowCollection** – sunt proprietăți compuse cu ajutorul |

| | |
|---|---|
| plasate c\te un control copil de diverse tipuri. | cărora se setează numărul de coloane și rândur. |
|  Este un control care afișează controalele copil în rând unul după altul, la fel ca și cuvintele într-un text. Controalele copil pot avea dimensiuni diferite. | **Justify (center, justify, left, right)** - specifică alinierea controalelor copil în rând. **JustifyLastLine** - specifică alinierea ultimei linii de controale. **VerticalCap**-specifică intervalul dintre liniile cu controale. |
|  Sunt containere cu contur invizibil la executare în interiorul cărora pot fi plasate diverse controale copil. Dacă un control copil parțial sau complet se află în afara containerului apar bare de defilare. | |
|  Sunt containere cu contur vizibil la executare în interiorul cărora pot fi plasate diverse controale copil. Dacă un control copil parțial sau complet se află în afara containerului apar bare de defilare. Aceste componente sunt util în cazul este necesar de vizualizat controale (ex. imagini) cu dimensiuni mari. | |

## Componente de tip ScrollBar.

**TScrollbar -** reprezintă o bară de defilare standard cu cursor și butoane care este utilizată pentru a derula conținutul unei ferestre, a unui formular sau a unui control cu ajutorul mouse-lui. Are predefinite următoarele proprietăți:

- **Value** – reprezintă poziția cursorului pe bara de defilare;
- **Min** și **Max** – poziția minimală/maximală a cursorului;
- **CanFocus**: boolean – dacă are valoarea **true** tastele de direcție sunt asociate cursorului.
- **Orientation** (Horizontal/Verical) – Setează orientarea orizontală sau verticală barei de defilare.

**TSmallScrolbar** – este o bară de defilare analogică cu TScrollbar cu dimensiuni reduse și fără butoane la capete.

**TArcDial** este o componentă de defilare cu cursor rotativ. Proprietatea Value indică poziția cursorului în grade. Dacă proprietatea ShowValue are valoarea true atunci poziția cursorului este indicată în centrul componentei.

De regulă pentru aceste componente se programează evenimentul OnChange în care se utilizează proprietate Value pentru realizarea efectelor dorite.

```
procedure TForm1.ArcDial1Change(Sender: TObject);
begin
.....:= ArcDial1.Value
end;
```

**TProgressBar** – se utilizează în aplicații în care trebuie să informați utilizatorul despre progresul uneia sau mai multor sarcini efectuate de aplicație. Proprietatea **Value** afișează progresul aplicației în intervalul definit de proprietățile **Min** și **Max**.

**TAniIndicator** este un indicator animat care se utilizează pentru a ilustra un proces nedeterminat de așteptare. Cu ajutorul proprietății **Enabled**:boolean se declanșează sau se finalizează.

În fig. 16 sunt reprezentate aceste componente la etapa de rulare a aplicației.

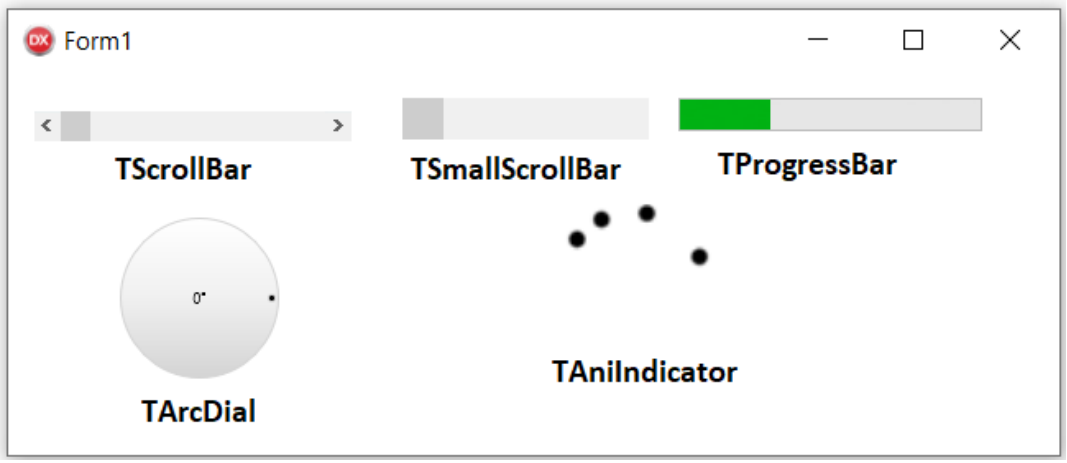



Figura 16. Exemple de componente de tipul Scroll.

## Componente de tip Listă

**TCombobox -** este un buton cu o listă ascunsă de itemi predefiniți. Lista se afișează în momentul când se apasă pe buton. După selectarea itemului lista se ascunde iar itemul selectat apare pe buton. Lista predefinită de itemi este stocată în proprietatea **Items:TSrings** care poate fi redactată în Inspectorul de Obiecte.

Valoarea itemului selectat se păstrează în proprietatea **Selected.Text**

**TComboEdit** - este o casetă de editare cu o listă ascunsă de itemi predefiniți. Lista se afișează în momentul când se apasă pe butonul casetei. După selectarea itemului lista se ascunde iar itemul selectat apare în fereastra de editare. Lista predefinită de itemi este stocată în proprietatea **Items:TSrings** care poate fi redactată în Inspectorul de Obiecte.

Valoarea itemului selectat se păstrează în proprietatea **Text**.

**TListBox** – afișează o listă predefinită de itemi într-o fereastră cu bară de defilare. Lista predefinită de itemi este stocată în proprietatea **Items:TSrings** care poate fi redactată în Inspectorul de Obiecte. Itemul selectat își modifică culoarea.

Proprietatea **Sorted:boolean** setează opțiunea de ordonare a elementelor listei în ordine alfabetică.

Valoarea ultimului element selectat se păstrează în proprietatea **Selected.text**

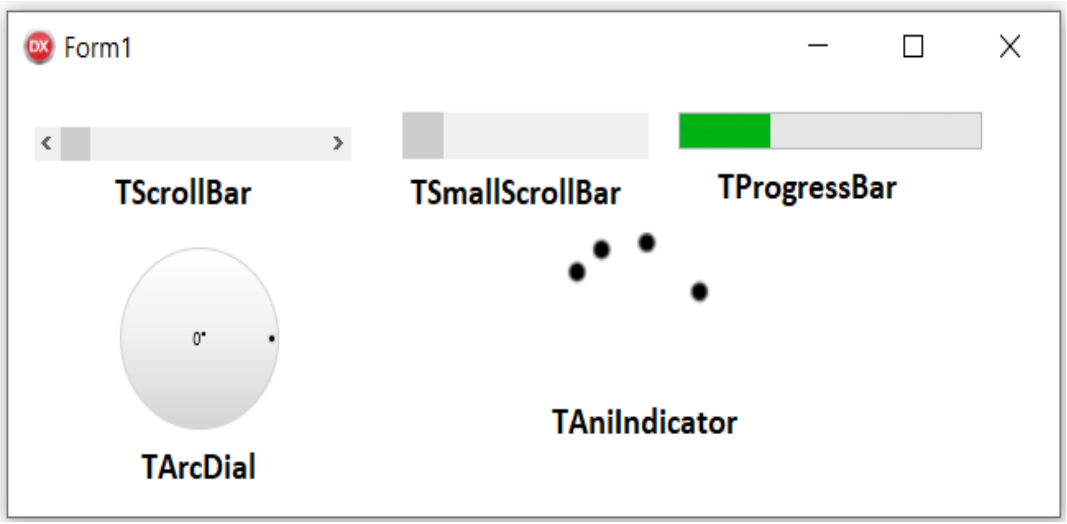



Figura 17. Exemple de componente de tip listă.

## Pagina de componente Shapes (obiecte grafice 2D predefinite)

În această pagină sunt definite componente cu ajutorul cărora ușor se trasează forme geometrice (linii, cecuri, etc) în interiorul unei regiuni rectangulare cu lungimea Width și înălțimea Height.

| |
|---|
| TLine Desenează un segment de linie. Proprietatea LineType (Diagonal, Top, Left, Bottom, Right) determină orientarea segmentului în interiorul controlului |
| TRectangle Desenează un dreptunghi. XRadius, YRadius – determină raza de rotunjire a colțurilor. |
| TRoundRect Desenează un dreptunghi cu colțurile rotunjite. Proprietatea Corners definește care colțuri trebuie să fie rotunjite. Raza de rotunjire a colțurilor este definită de formula R=min(width, heigth)/2. |
| TEllipse Desenează o elipsă în interiorul controlului. |
| TCircle Desenează o circumferință cu raza definită de formula R=min(width, heigth)/2. |

| |
|---|
| TArc Desenează un arc de elipsă. StartAngle și EndAngle determină începutul ș sfârșitul arcului în grade. |
| TPie Desenează un sector de elipsă. StartAngle și EndAngle determină începutul ș sfârșitul arcului în grade |
| TText Desenează textul specificat în proprietatea **Tex** în interiorul controlului. AutoSize:boolean setează proprietatea de ajustare a dimensiunilor controlului la dimensiunile textului; TextSetings – setează proprietățile textului (Fontul, culoarea etc) |
| Pentru personalizarea stilurilor de trasare a liniilor de contur și de umplere conturilor închise se utilizează proprietățile **Stroke** și **Fill** |

## Gestionarea culorilor.

În Delphi FMX pentru vizualizarea culorilor se utilizează modelul ARGB compus din patru componente a câte 8 biți (în total 32 biți): Canalul A- (transparența 0-255 ($FF); canalul R (256 nuanțe de roșu); canalul G (256 nuanțe de verde); canalul B (256 nuanțe de albastru); Pentru gestionarea culorilor Este predefinit tipul de date TAlphaColor, identic cu tipul TCardinal. Unei variabile de tip C:TAlphaColor putem săi atribuim valori direct (C:=$FFFF0000- roșu absolut intransparent; (C:=$7FFF0000- roșu semitransparent;) sau prin atribuirea uneia din cele 141 Varante predefinite în TAlphaColors (C:=TAlphaColors.Red).

Pentru gestionarea interactivă a culorilor se utilizează componentele din pagina **Colors**. Cele mai utilizate sunt: TColorListBox, TColorComboBox, TColorPanel, TColorPicker, TComboColorBox. Toate aceste componente au proprietatea Color care se selectează în mod interactiv și poate fi atribuită variabililor de tip TAlphaColor prin cod de programare.

În fig.18 este reprezentată o captură de ecran în care culorile obiectelor Rectangle1 și Ellipse1 sunt modificate interactiv prin intermediul evenimentului **OnChange**.

```
procedure TForm1.ColorListBox1Change(Sender: TObject);
begin
     Rectangle1.Fill.Color:=ColorListBox1.Color;
     Ellipse1.Stroke.Color:=ColorListBox1.Color;
end;
```

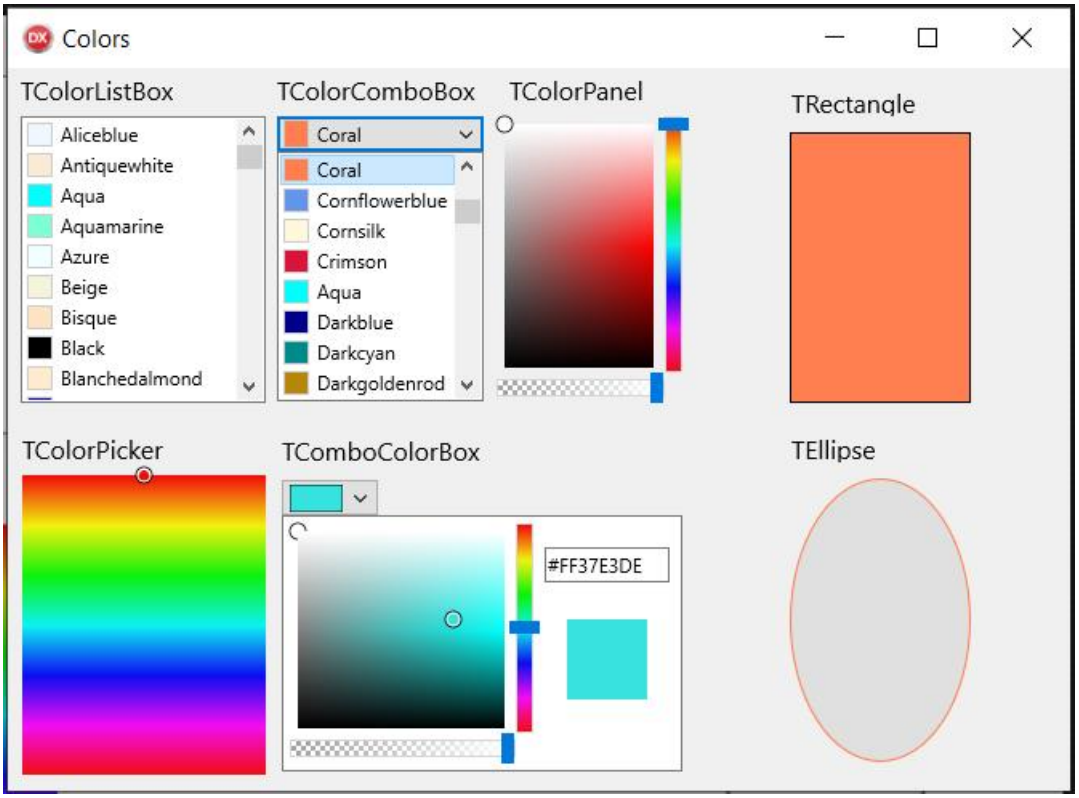



Figura 18. Exemple de utilizare a componentelor pentru gestionarea culorilor.

## Gestionarea fișierelor.

În aplicațiile interactive Delphi foarte frecvent se apelează la subprogramele cu ajutorul cărora se gestionează informația stocată în fișiere, mape, discuri. Prezentăm în continuare cele mai frecvent utilizate funcții și proceduri:

| | |
|---|---|
| **function** GetCurrentDir: string | Întoarce directoriul curent |
| function CreateDir(Var Dir: string): Boolean; | Creează directoriul cu numele Dit |
| function RemoveDir(Var Dir: string): Boolean; | Distruge directoriu cu numele Dir |
| function FileExists(Var FileName: string): Boolean; | Verifică dacă fișierul cu numele FileName există |
| function DeleteFile(Var FileName: string): Boolean; | Distruge fișierul cu numele FileName |
| **function** FileSizeByName(Var AFilename: TIdFileName): Int64; | Unit **IdGlobalProtocols** Determină dimensiunea fișierului |
| TOpenDialog și TSaveDialog sunt componente invizibilă pentru care metoda **Execute** afișează o casetă de dialog în care utilizatorul selectează unul sau mai | |

| | |
|---|---|
| multe fișiere, numele cărora se utilizează în codul programelor pentru realizarea operațiilor asupra fișierelor.<br>Proprietăți: **Option[ofAllowSelect]** permite selectarea mai multor fișiere; **InitialDir** – setează directoriul inițial de căutare; **FileName** – conține numele și extensia fișierului care a fost selectat în caseta de dialog; **Files:TStrings** conține lista de fișiere selectate; **Filter:string** – Setează filtrul de vizualizare a fișierelor în caseta de dialog. Exemplu ('Delphi Files\|*.pas;*.dpr')- vor fi selectate numai fișierele cu extensiile \|*.pas;*.dpr. | |
| **Procedure** assignfile (var F: TextFile; Var FileName: string); | Asociază variabila F cu FileName – nume fișier de tip TextFile în care urmează să se scrie sau din care se va citi informație de tip text. |
| **Procedure** Assignfile (var FileHandle: File; Var FileName: string); | Asociază variabila F cu FileName – nume fișier de tip File în care urmează să se scrie sau din care se va citi informație de tip binar. |
| **Procedura** ReWrite(**var**FileHandle: TextFile); | Deschideți un fișier text sau binar pentru acces la scriere |
| **Procedure** Reset(var FileHandle: TextFile); | Deschide un fișier text pentru citire sau un fișier binar pentru citire / scriere |
| **Procedure** Append(var FileHandle: TextFile); | Deschide un fișier text pentru a permite adăugarea textului la final |
| **Procedura** Write (var FileHandle: File; var Value1 {, var Value2 ...});); | Scrieți date într-un fișier binar |
| **Procedura** Write (var FileHandle: TextFile; Expression1 {options} {, Expression2 {options} ...}); | Scrieți date într-un fișier binar sau text |
| **Procedura** WriteLn (var FileHandle: TextFile; Expression1 {options} {, Expression2 {options} ...}); | Scrieți o linie completă de date într-un fișier text |
| **Procedure** CloseFile(var FileHandle: TextFile); | Închide fișierul deschis |

## Gestionarea evenimentelor

Toată activitatea unui sistemului de operare cu interfață grafică și periferiile calculatorului se Varruiește în jurul evenimentelor cum ar fi clic pe tastatură, mișcări și clicuri cu ajutorul mouse-ului etc. Fiecare din aceste evenimente produc mesaje pe care le prelucrează sistemul de operare. Marea majoritate de mesaje care se produc în aplicațiile care rulează pe calculator se gestionează, în mod automat de sistemul de operare. În Delphi toate componentele de tip TContol posedă proceduri și funcții

cu ajutorul cărora utilizatorul poate să creeze evenimente cu conținut personalizat. Mesajele produse de aceste evenimente se prelucrează în procesul de rulare a aplicației executabile Delphi, realizând diverse efecte de interacțiune cu utilizatorul. În fig.19 sunt reprezentate secvențe din fereastra Object Inspector pentru 3 componente (TForma, TButton, TEdit).

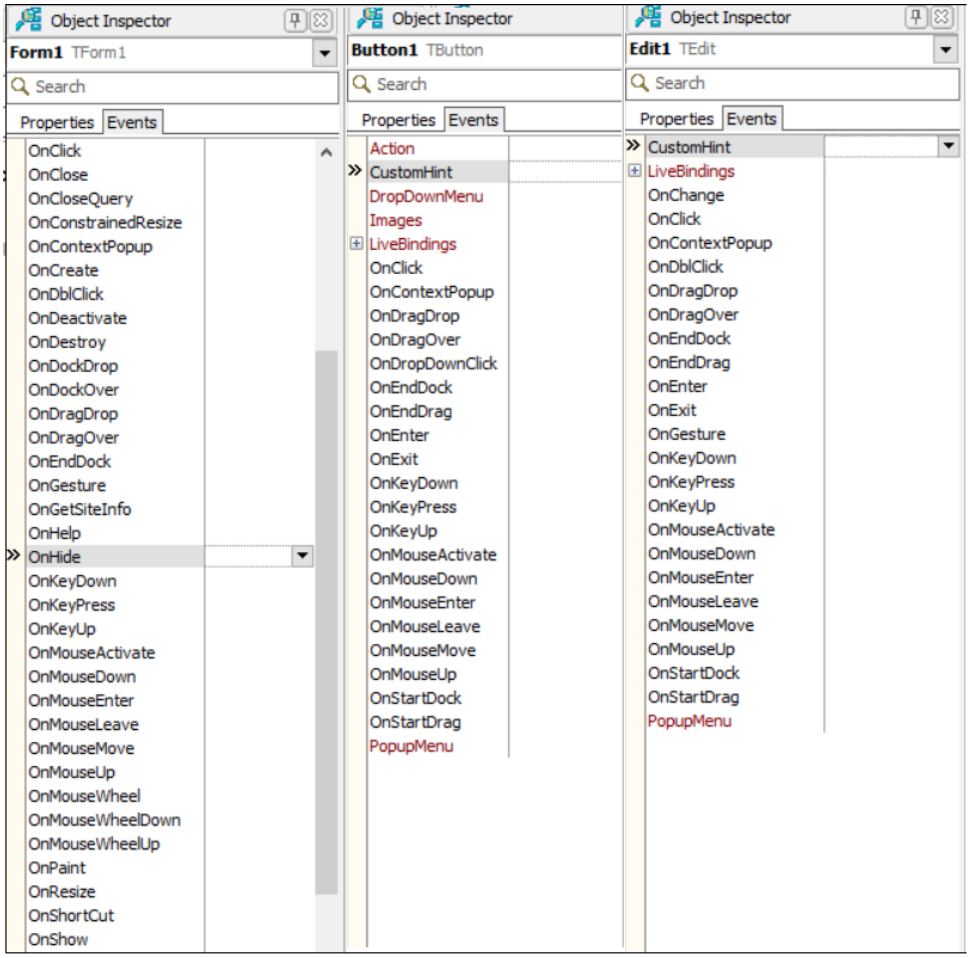



Figura 19. Secvențe cu evenimente din fereastra Object Inspector.

Evenimentele, în dependență de esența lor, se divizează în câteva tipuri care se întâlnesc la diferite componente. Cel mai frecvent se întâlnesc evenimentele de tipul **TNotifyEvent** care sunt proceduri **procedure** (Sender: TObject), cu un singur parametru formal Sender, căruia la apelarea evenimentului îi corespunde obiectul care a declanșat evenimentul.

Cu ajutorul operatorului **is** se poate de determinat dacă o clasă predefinită în Delphi face parte din declarația directă sau moștenită a obiectului Sender. Cunoscând

clasa obiectului Sender cu ajutorul operatorului **as** pot fi accesate proprietățile și metodele lui. În fig.20 sunt prezentate exemple de expresii formate cu operatorul **as**.

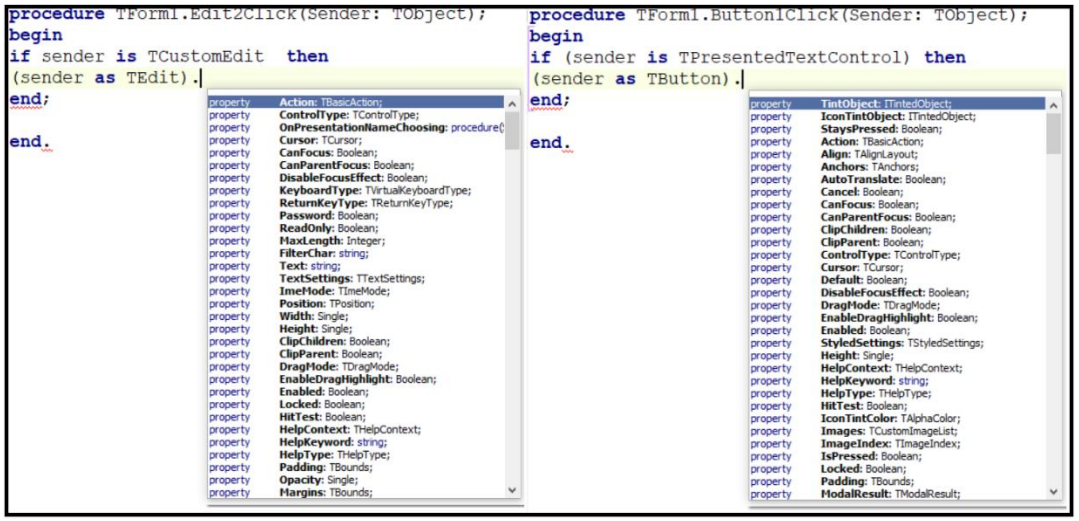



Figura 20. Exemple de utilizare a operatorilor **is** și **as**.

Prezentăm în continuare o descriere succintă a celor mai utilizate evenimente:

| Evenient | Descriere |
|---|---|
| OnCreate | Este de tipul TNotifyEvent și se declanșează atunci când componenta se creează. |
| OnClick | Este de tipul TNotifyEvent și de obicei, se declanșează atunci când indicatorul mouse-ului se află deasupra componentei și utilizatorul apasă și eliberează butonul stâng al mouse-ului. |
| OnMouseDown | Este de tipul TMouseEvent=procedure(Sender:TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Single); și se declanșează atunci când indicatorul mouse-ului se află deasupra componentei și utilizatorul apasă unul din butoanele mouse-ului.<br>TMouseButton=(mbLeft, mbRight, mbMiddle); (mbLeft, mbRight, mbMiddle);<br>Type TShiftState= set of (ssShift, ssAlt, ssCtrl, ssLeft, ssRight, ssMiddle, ssDouble, ssTouch, ssPen, ssCommand, ssHorizontal); |

| | |
|---|---|
| | X, Y: Single – coordonatele mouse-lui în momentul declanșării evenimentului, relativ la colțul stânga sus a componentei. |
| OnMouseUp | Este de tipul TMouseEvent și se declanșează atunci când indicatorul mouse-ului se află deasupra componentei și utilizatorul eliberează butonul mouse-ului. |
| OnMouseMove | Este de tipul TMouseMove=procedure(Sender: TObject; Shift: TShiftState; X, Y: Single); și se apelează repetat atunci când indicatorul mouse-lui se deplasează deasupra componentei. În variabilile X,Y se returnează coordonatele curente ale mouse-lui relativ la colțul stânga sus al componentei. |
| OnValidate | Este de tipul TValidateTextEvent=procedure(Sender: TObject; var Text: string); Se realizează după ce utilizatorul apasă tasta ENTER sau focalizarea părăsește controlul. |
| OnKeyDown / OnKeyUp | Sunt de tipul TKeyEvent=procedure (Sender: TObject; var Key: Word; var KeyChar: Char; Shift: TShiftState); Se realizează când se apasă /se eliberează o tastă; Key-întoarce codul numeric al tastei; KeyChar – caracterul care corespunde tastei; Shift – starea tastelor auxiliare și a mouse-lui. |

Exemplu: În secvența următoare de cod este prezentată partea de implementare a unei aplicații Delphi FMX pe forma căreia este plasată componenta Panel1:TPanel care la etapa de executare se deplasează cu ajutorul mouse-lui pe suprafața formei. Pentru realizarea acestui efect au fost programate evenimentele OnMouseDown, OnMouseMove și OnMouseUp.

```
implementation

{$R *.fmx}
var xp,yp:single; b:boolean;
procedure TForm1.Panel1MouseDown(Sender: TObject; Button:
TMouseButton;
  Shift: TShiftState; X, Y: Single);
begin
xp:=x;yp:=y;b:=true;
end;

procedure TForm1.Panel1MouseMove(Sender: TObject; Shift:
TShiftState; X, Y: Single);
begin
   if b then
   begin
      panel1.Position.x:=panel1.Position.x+x-xp;
      panel1.Position.y:=panel1.Position.y+y-yp;
   end;
```

```
end;

procedure TForm1.Panel1MouseUp(Sender: TObject; Button:
TMouseButton;  Shift: TShiftState; X, Y: Single);
begin
   b:=false;
end;
end.
```

## Gestionarea timpului

Pentru gestionarea timpului ( ani luni zile..... secunde milisecunde) în Delphi se utilizează tipul de date **TdateTime**. Acest tip de date este definit în unitatea SysUtils care se atașează în mod automat la în aplicațiile DelphiFMX.

**type TDateTime≡typeDouble;**

Partea întreagă a unei valori de tip TDateTime în Delphi reprezintă numărul de zile care au trecut începând cu data 30/12/1899. Partea fracționară a acestei valori reprezintă partea respectivă din 24 ore care a trecut.

Iată câteva exemple de valori de tipul TDateTime transformate în data și ora corespunză ...are.

| | | | |
|---|---|---|---|
| 0 | 30/12/1899 12:00 am | -1.25 | 29/12/1899 6:00 am |
| 2.75 | 1/1/1900 6:00 pm | 35065 | 1/1/1996 12:00 am |

Pentru măsurarea timpului în ...ate sistemele de operare este incorporat un ceas de sistem care de regulă măsoară timpul în milisecunde. Ceasul de sistem de regulă se pornește în momentul când sistemul de operare sa instalat pe calcula ...r. Accesul la valorile ceasului de sistem în Delphi FMX se realizează cu aju ...rul funcției **Function.TThread.GetTickCount:Cardinal;** definită în unitatea System.Classes

**Procedure.TThread.Sleep(M:integer); se utilizează pentru realizarea pauzelor în executarea programului.**

**Rutine pentru obținerea datei și orei curente ( System.SysUtils)**

| Rutină | Descriere |
|---|---|
| **Function** CurrentYear:Word ; | Returnează anul curent. |
| **Function** Data: TDateTime ; | Returnează data curentă. |

| | |
|---|---|
| **Function** DayOfWeek ( Var DateTime: TDateTime ): Word ; | Returnează ziua săptămânii pentru o dată specificată. |
| **Function** GetTime: TDateTime ; | Returnează ora curentă. |
| **Function** Now: TDateTime ; | Returnează data și ora curente. |
| **Function** Time: TDateTime ; | Returnează ora curentă. |

**Rutine pentru crearea valorilor de tip DateTime (System.SysUtils)**

| Rutină | Descriere |
|---|---|
| **Function** EncodeDate ( An Luna Zi: Word ): TDateTime ; | Returnează o valoare TDateTime care reprezintă un an o lună și o zi specificate. |
| **Function** EncodeDateDay ( Var AYear ADayOfYear: Word ): TDateTime ; | Returnează o valoare TDateTime care reprezintă numărul de ordine al zilei în anul specificat |
| **Function** EncodeDateTime ( Var AYear AMonth ADay AHour AMinute ASecond AMilliSecond: Word ): TDateTime ; | Returnează o valoare TDateTime care reprezintă anul, luna, ziua, ora, minutele, secundele și milisecundele specificate. |
| **Function** EncodeTime(Hour,Min,Sec, MSec:Word):TDateTime ; | Returnează o valoare TDateTime pentru ora, minutele, secundele și milisecundele specificate. |

**Rutine pentru extragerea informațiilor din valorile de tip TDateTime**

| Rutină | Descriere |
|---|---|
| **procedure** DecodeDate ( Var DateTime: TDateTime ; var Year Month Day: Word); | Returnează anul, luna și ziua pentru o valoare de tipul TDateTime . |
| **procedure** DecodeDateDay ( Var AValue: TDateTime ; out AYear ADayOfYear: Word ) ; | Returnează anul și ziua anului pentru o valoare TDateTime specificată . |

| | |
|---|---|
| **Function** DecodeDateFully ( Var DateTime: TDateTime ; var Anul, Luna, Ziua, DOW: Word ): Boolean ; | Returnează valorile anul, luna, ziua și ziua săptămânii pentru o valoare TDateTime . |
| **procedure** DecodeDateMonthWeek ( Var AValue: TDateTime ; out AYear, AMonth AWeekOfMonth, ADayOfWeek: Word ) ; | Returnează anul, luna săptămâna lunii și ziua săptămânii pentru o valoare TDateTime specificată . |
| **procedure** DecodeDateTime ( Var AValue: TDateTime ; out AYear AMonth, ADay, AHour, AMinute, ASecond AMilliSecond: Word ) ; | Returnează anul, luna, ziua, ora, minutele, secundele și milisecundele pentru valoarea TDateTime specificată . |
| **procedure** DecodeTime ( Var DateTime: TDateTime ; var Hour, Min, Sec, MSec: Word ) ; | Returnează ora, minutele, secundele și milisecundele pentru valoarea TDateTime specificată .. |
| **Function** NthDayOfWeek ( Var AValue: TDateTime ): Word ; | Returnează ziua săptămânii pentru valoarea TDateTime specificate . |

**Conversia valorilor de tip TDateTime(avansat)a**

| Rutină | Descriere |
|---|---|
| **Function** DateTimeToStr(Var DateTime: TDateTime): string; **Function** DateTimeToStr(Var DateTime: TDateTime; Var AFormatSettings: TFormatSettings): string; | Convertește o valoare TDateTime într-un șir de caractere utilizând setările locale sau formatul specificat în Varanta AFormat pentru afișarea datei și timpului |
| **Function** TimeToStr(Var DateTime: TDateTime): string; **Function** TimeToStr(Var DateTime: TDateTime; Var AFormatSettings: TFormatSettings): string; | Convertește o valoare TDateTime într-un șir de caractere utilizând setările locale sau formatul specificat în Varanta AFormat pentru afișarea timpului |

| | |
|---|---|
| **Function** DateToStr(Var DateTime: TDateTime): string;<br>**Function** DateToStr(Var DateTime: TDateTime; Var AFormatSettings: TFormatSettings): string; | Convertește o valoare TDateTime într-un șir de caractere utilizând setările locale sau formatul specificat în Varanta AFormat pentru afișarea datei. |
| **Function** FloatToDateTime(Var Value: Extended): TDateTime; | Convertește o valoare de tip real în rezultat de tipul TDateTime |
| **Procedure** DateTimeToString(var Result: string; Var Format: string; DateTime: TDateTime); | Convertește valoare TDateTime în șirul de caractere Result utilizând Formatul specificat. |

**Specificarea formatului pentru date de tip TDateTime**

Șirurile Format Data și Ora sunt compuse din specificatori care reprezintă valori care trebuie inserate în șirul formatat. Unii specificatori (cum ar fi „d”) formatează pur și simplu numere sau șiruri. Alți specificatorii (cum ar fi "/") se referă la șiruri specifice locației din variabilele globale.

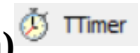
În tabelul următor specificările sunt date cu litere mici. Carcasa este ignorată în formate cu excepția specificatorilor „am / pm” și „a / p”.

| Specificator | Afișează |
|---|---|
| c | Afișează data utilizând formatul dat de variabila globală ShortDateFormat urmat de ora utilizând formatul dat de variabila globală LongTimeFormat . Ora nu este afișată dacă valoarea dată-oră indică cu precizie miezul nopții. |
| d | Afișează ziua sub forma unui număr fără un zero de pornire (1-31). |
| dd | Afișează ziua ca un număr cu un zero în prim-plan (01-31). |

| | |
|---|---|
| ddd | Afișează ziua ca o abreviere (Sun-Sat) folosind șirurile date de variabila globală ShortDayNames . |
| dddd | Afișează ziua ca nume complet (duminică-sâmbătă) folosind șirurile date de variabila globală LongDayNames . |
| ddddd | Afișează data utilizând formatul dat de variabila globală ShortDateFormat . |
| dddddd | Afișează data utilizând formatul dat de variabila globală LongDateFormat . |
| e | (Numai Windows) Afișează anul din perioada / epoca actuală ca număr fără zero zero (numai în regiunile japoneze coreene și taiwaneze). |
| ee | (Numai Windows) Afișează anul din perioada / epoca actuală ca număr cu zero zero (numai în regiunile japoneze coreene și taiwaneze). |
| g | (Numai Windows) Afișează perioada / era ca o abreviere (numai localuri japoneze și taiwaneze). |
| gg | (Numai Windows) Afișează perioada / epoca ca nume complet (numai localuri japoneze și taiwaneze). |
| m | Afișează luna sub forma unui număr fără zero inițial (1-12). Dacă specifica ...rul m urmează imediat un specifica ...r h sau hh este afișat minutul mai degrabă decât luna. |
| mm | Afișează luna sub forma unui număr cu zero zero (01-12). Dacă specifica ...rul mm urmează imediat un specifica ...r h sau hh este afișat minutul mai degrabă decât luna. |
| mmm | Afișează luna ca prescurtare (ianuarie-decembrie) folosind șirurile date de variabila globală ShortMonthNames . |

| | |
|---|---|
| mmmm | Afișează luna ca nume complet (ianuarie-decembrie) folosind șirurile date de variabila globală LongMonthNames . |
| yy | Afișează anul ca număr din două cifre (00-99). |
| yyyy | Afișează anul ca număr din patru cifre (0000-9999). |
| h | Afișează ora fără zero inițial (0-23). |
| hh | Afișează ora cu zero zero (00-23). |
| n | Afișează minutul fără zero inițial (0-59). |
| nn | Afișează minutul cu zero zero (00-59). |
| s | Afișează al doilea fără un zero inițial (0-59). |
| ss | Afișează al doilea cu zero zero (00-59). |
| z | Afișează milisecunda fără zero zero (0-999). |
| zzz | Afișează milisecunda cu zero zero (000-999). |
| t | Afișează timpul utilizând formatul dat de variabila globală ShortTimeFormat . |
| tt | Afișează timpul utilizând formatul dat de variabila globală LongTimeFormat . |
| am pm | Folosește ceasul de 12 ore pentru specifica ...rul h sau hh precedent și afișează „am” pentru orice oră înainte de prânz și „pm” pentru orice oră după prânz. Specifica ...rul am / pm poate utiliza minuscule majuscule sau minuscule iar rezultatul este afișat corespunză ...r. |
| a / p | Folosește ceasul de 12 ore pentru specifica ...rul h sau hh precedent și afișează „a” pentru orice oră înainte de prânz și |

| | |
|---|---|
| | „p” pentru orice oră după prânz. Specifica ...rul a / p poate utiliza minuscule majuscule sau minuscule iar rezultatul este afișat corespunză ...r. |
| am pm | Folosește ceasul de 12 ore pentru specifica ...rul h sau hh precedent și afișează conținutul variabilei globale TimeAMString pentru orice oră înainte de prânz și conținutul variabilei globale TimePMString pentru orice oră după prânz. |
| / | Afișează caracterul separa ...r de date dat de variabila globală DateSepara ...r . |
| : | Afișează caracterul separa ...r de timp dat de variabila globală TimeSepara ...r . |
| „xx” / „xx” | Caracterele cuprinse între ghilimele simple sau duble sunt afișate ca atare și nu afectează formatarea. |

## Componenta TTimer (pagina System) 


La etapa de executare a aplicației este o componentă invizibilă. Se utilizează pentru organizarea repetărilor în timp a unor activități de programare cu o perioadă Varantă. Are două proprietăți importante:

- **Interval**: integer (setează intervalul de repetare în milisecunde);
- **Enabled**: boolean(valoarea **true** activează iar **false** dezactivează timer-ul).

Evenimentul OnTimer este o procedură de tip **TNotifyEvent** în care se specifică codul de program care se repetă cu perioada setată în proprietatea **Interval**.

## Cronometru digital. Aplicație vizuală DelphiFMX.

Procesul de creare a unei aplicații vizuale interactive, în linii mari poate fi divizat în trei etape: scenariul, designul și scrierea codului.

Scenariul, în cazul dat, este simplu: dorim să realizăm un cronometru digital care vizual și funcțional să corespundă unui cronometru digital real, care în varianta

cei mai simplă reprezintă o cutie care conține două butoane (start și stop) și un indicator al intervalului de timp măsurat.

Etapa designului, de asemenea este simplă. Paleta de instrumente Delphi conține multe componente care pot fi folosite în calitate de cutie (TPanel, TTabControl, TGroupBox), butoane (TButton, TSpeedButton, TCornerButton), indicator (TLabel, TEdit ...). În cazul dat, din paleta de instrumente (ToolPalette), alegem în calitate de container componenta GroupBox și o plasăm pe formă. În mod predefinit i se atribuie numele GroupBox1. Adăugăm în acest container din pagina de componente Standard două butoane (Button1, Button2) și eticheta Label1. Din pagina System adăugăm componenta Timer1. Pentru aceste componente setăm proprietățile indicate în următorul tabel:

| | |
|---|---|
| GroupBox1 | Width=200; Text='Cronometru Digital' |
| Label1 | Position.Y=20; Align:=Horizontal;<br>TexrSetings=>HorizAlign=Center;<br>TexrSetings=>Font=>Size=20; Text='00 : 00 : 00 :000' |
| Button1 | Position.X=15; Position.Y=60; Text='Start' |
| Button2 | Position.X=105; Position.Y=60; Text='Stop' |
| Timer1 | Enabled=false; Interval=1 |

Ca urmare, designul aplicației are forma reprezentat în fig. 21a.

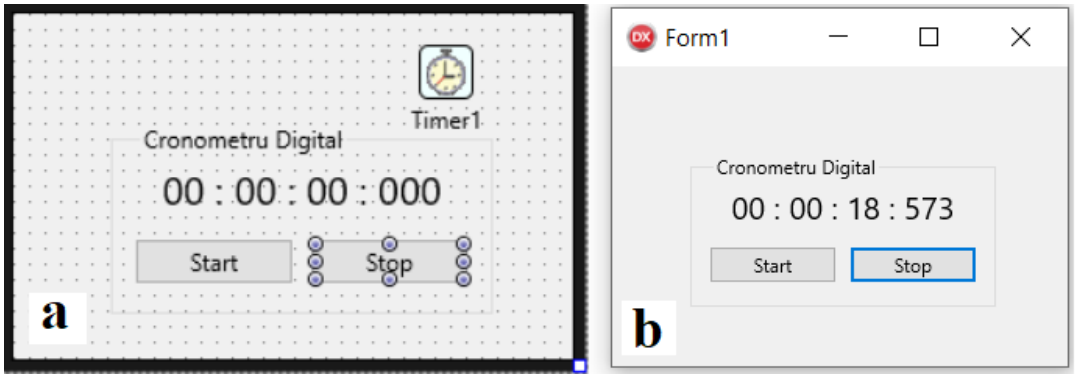



Figura 21. Designul (a) și executabilul (b) aplicației "Cronometru Digital".

La etapa de scriere a codului, în inspectorul de obiecte selectăm și programăm următoarele evenimente: OnClick (Button1, Button2) și OnTimer(Timeer1). În continuare prezentăm partea **implementation** a listingului Unit1.

```
implementation
{$R *.fmx}
var ti:tdatetime;
procedure TForm1.Button1Click(Sender: TObject);
```

```
begin
   ti:=now;
   timer1.Enabled:=true;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
   timer1.Enabled:=false;
end;

procedure TForm1.Timer1Timer(Sender: TObject);
var s:string;
begin
   datetimetostring(s,'hh : nn : ss : zzz',(now-ti));
   label1.Text:=s;
end;

end.
```

La executare ”Cronometrul Digital” are formaa reprezentată în fig. 21b.

## Introducere în programarea orientată pe obiecte

La etapa inițială calculatoarele se programau în baza tehnologiilor de **programare liniară** în care elementele principale erau datele (variabile, constante) și instrucțiunile: de atribuire, de ramificare și de organizare a repetărilor (ciclurile). Acest stil de programare nu a existat mult timp deoarece nu avea prevăzut un mecanism de utilizarea repetată a porțiunilor similare de cod. Programele aveau un număr mare de instrucțiuni și era foarte complicat de depistat erorile de programare (depanare). Pentru înlăturarea acestui neajuns a fost introdusă noțiunea de subprogram (proceduri și funcții) care se implementează la nivel formal în partea de declarații a programului și se apelează de multipli ori în corpul programului de bază pentru prelucrarea datelor. Această tehnologie de programarea a fost numită orientată pe proceduri și a fost utilizată cu succes pentru dezvoltarea aplicațiilor consolă (fără interfață grafică) și continuă să fie utilizată în prezent, preponderent în scopuri educaționale, pentru instruirea elevilor și studenților la etapa inițială de studiere a tehnicilor de programare.

În prezent se dezvoltă o nouă tehnologie de programare numită **Programarea Orientată pe Obiecte** (POO). Elementul principal în această tehnologie este noțiunea de clasă - o structură complexă care gestionează ca un tot întreg ansamblul de date și subprograme prin care se definește clasa. Un exemplar al clasei în POO se numește obiect.

Noțiunile principale în POO sunt: **moștenirea, polimorfismul, încapsularea**.

**Moștenirea** este o proprietate prin care o clasa de obiecte poate fi obținuta din derivarea unei clase sau a mai multor clase anterior definite. O clasă derivată poate constitui, la rândul ei, clasa de baza pentru derivarea altor clase. Când se definește o clasa se începe cu caracteristicile ei generale. Definirea unei alte clase prin derivarea clasei de baza presupunea definirea caracteristicilor specifice acesteia in raport cu cea de baza. Nivelurile de derivare ale claselor pot creste ca un arbore genealogic. Conceptul de moștenire este deosebit de important in POO. Posibilitatea de a crea o arborescenta de clase, de la cea mai generala la cea specificată, permite un control mai bun al programelor, permițând, totodată, lizibilitatea facilă a programelor de către utilizatorii textelor sursa sau dezvoltatorii de programe. Moștenirea si proprietățile acesteia reprezentă adevărata forță in programarea obiectată pe obiecte.

**Încapsularea** reprezintă altceva proprietatea claselor de obiecte de a grupa sub aceeași structura datele si metodele aplicabile asupra datelor. In sensul programării orientata pe obiecte, încapsularea definește, de asemenea, modalitatea în care diversele obiecte si restul programului se pot referi la datele specifice obiectelor. De asemenea încapsularea permite, cu ajutorul directivelor (**private** și **public**) separarea datelor in date private si date publice. Programele utilizatorilor pot accesa si utiliza datele unui obiect, declarate private, numai prin utilizarea unor metode definite public. Separarea datelor in secțiuni de date publice si private garantează protejarea datelor fata de utilizarea lor eronata in programe.

**Polimorfismul** în traducere înseamnă "a avea mai multe forme" si se realizează prin **overloading** (supraîncărcare) si **overriding** (suprascriere) in momentul in care folosim moștenirea. Practic polimorfismul ne oferă posibilitatea de a folosi metodele moștenite sub mai multe forme, in funcție de rezultatul pe care îl dorim (astfel codul nostru devenind mai flexibil). **Încapsularea** permite ascunderea prin atributul **private** de către utilizatori a unor elemente (variabile, metode) din care se compun clasele.

## Declararea claselor.

De regulă clasele se declară și se construiesc în unit-uri separate care au două compartimente **inreface** și **implementation**. Declarațiile se realizează în partea de interfață (**interface**) și au următoarea structură:

**Type** <nume (identificator)>=**class** (<numele clasei părinte>)

<declarații de câmpuri și metode >
**private**
<declarații de câmpuri și metode >
**public**
<declarații de câmpuri și metode >
<declarații ale constructorului (**constructor** create) >
<declarații ale destructorului (**destructor** destroy) >
**end**;

Metodele incluse în declarația clasei, în mod obligatoriu trebuie să fie descrise în partea de implementare unde se specifică numele clase și numele metodei separate prin . (punct). În compartimentul public se mai declară două metode care au statut special **constructor** cu numele **create** și **destructor** cu numele **destroy** care au structură analogică cu procedurile. În constructor se specifică modul cum se creează și cum se inițializează clasa iar în destructor se specifică modul cum se distruge clasa. Dacă clasa se creează prin moștenire atunci în declararea constructorului și destructorului se adaugă directiva **override** iar în partea de implementare a constructorului prima instrucțiune va fi **inherited.** În partea de implementare a destructorului instrucțiunea **inherited** va fi ultima.

Prezentăm în continuare conținutul integral al unit-ului UPanelM în care se implementează clasa TPanelM prin moștenire de la clasa predefinită TPanel. Clasa TPanelM se deosebește de clasa predefinită TPanel prin adăugarea unui unui buton cu numele b.

```
unit UPanelM;
interface
uses FMX.StdCtrls, // TPanel, TButton
System.Classes;   // TComponent
type TPanelM=class(Tpanel)
   b:TButton;
  private
      { Private declarations }
  public
  constructor Create(Owner: TComponent); override;
  destructor destroy; override;
    { Public declarations }
  end;
implementation
constructor TPanelM.Create(Owner: TComponent);
begin
   inherited Create(Owner);
   width:=100; height:=50;
   b:=TButton.Create(Self);
```

```
   b.Parent:=self;
end;
destructor TPanelM.Destroy;
begin
   b.Destroy;
   inherited Destroy;
   end;
end.
```

## Crearea dinamică a componentelor.

Componentele în mediul de programare vizuală Delphi sun exemplare ale claselor predefinite. Forma este o componentă specifică. Ea se declară și se creează în mod automat la selectarea opțiunii File>New>Multi-DeviceApplication.

Unitul aplicației Delphi la acest moment are forma:

| | |
|---|---|
| `unit Unit1;`<br>`interface`<br>`uses`<br>`System.SysUtils.....;`<br>`type  TForm1 = class(TForm)`<br><br>`  private`<br>`    { Private declarations }`<br>`  public`<br>`    { Public declarations }`<br>`  end;`<br>`var Form1: TForm1;`<br>`implementation`<br>`{$R *.fmx}`<br>`end.` | `unit Unit1;`<br>`interface`<br>`uses`<br>`System.SysUtils.....;`<br>`type TForm1 = class(TForm)`<br>`    Button1: TButton;`<br>`  private`<br>`    { Private declarations }`<br>`  public`<br>`    { Public declarations }`<br>`  end;`<br>`var Form1: TForm1;`<br>`implementation`<br>`{$R *.fmx}`<br>`end.` |
| Figura ..... | Figura ..... |

În unit1 se include în mod automat declarația clasei **type** `TForm1` și se declară variabila **var** `Form1: TForm1;` (clasa `TForm1` este identică cu clasa predefinită `TForm`). În momentul când din paleta de componente se adaugă pe suprafața formei componenta Button1, codul de declarare a clasei TForm1 se completează cu câmpul **`Button1: TButton`**. La acest moment clasa TForm1 diferă de clasa predefinită TForm prin câmpul **`Button1: TButton`**.

Dacă, la etapa de executare, dorim că adăugăm în formă o componentă nouă se parcurg următorii pași:

1. Se declară o variabilă de tipul respectiv (exemplu var p:TPanel).
2. În evenimentul (exemplu Button1.Click) cu ajutorul căruia dorim să creăm componenta se scrie următorul cod:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
     p:=TPanel.create(self);
     p.Parent:=self;
```

```
......// în continuare componenta p este creată și devine
//vizibilă în forma aplicației/
end;
```

Notă. self este o constată predefinită de tip TComponent care permite obiectului să facă referire la el însuși. În cazul dat p este o variabilă declarată în Form1, deci self va face referire la Form1.

3. Dacă la executarea programului se apasă pe button1 în colțul stânga sus va apărea panelul pe care l-am creat dinamic.
4. Dacă dorim să distrugem dinamic obiectul creat atunci se apelează instrucțiunea destroy (în cazul dat p.destroy;)

Notă. În mod analogic poate fi creată dinamic orice componentă predefinită Delphi sau declarată prin cod de programare (exemplul TPanelM descris mai sus).

## Crearea componentelor de concepție proprie.

DgjhaD

**Ceas static**

```
unit UCeasA;
interface
uses
FMX.Objects,//obiecte FMX,TCircle...
FMX.Types,//TTimer
System.Classes, // TComponent
System.UITypes, // TAlphaColors
System.SysUtils;//inttostr
type TCeasS=class(TCircle)
  ach,acm,acs:TRoundRect;
  private
   FOra:Word; FMin:Word; FSec:Word;
   procedure SetOra(Value:Word);
   procedure SetMin(Value:Word);
   procedure SetSec(Value:Word);
  public
   procedure DrawAce;
   property Ora:word read FOra Write SetOra;
   property Min:word read FMin Write SetMin;
   property Sec:word read FSec Write SetSec;
   constructor Create(Owner: TComponent); override;
 end;
implementation

constructor TCeasS.Create(Owner: TComponent);
```

```
var r,rshm:extended;
  I: Integer;
  shm:array[0..59] of TCircle;
  shc:array[1..12] of TText;
begin
  inherited Create(Owner);
  position.X:=50; position.Y:=50;
  width:=400;
  height:=400;
  r:=(width/2);
  Fora:=7;
  Fora:=8;Fmin:=25;Fsec:=30;
  ach:=TRoundRect.Create(Self);
  ach.Parent:=self;
  ach.Width:=r*0.75;ach.Height:=3; ach.Stroke.Thickness:=2;
  ach.Position.X:=r;ach.Position.y:=r;
  ach.RotationCenter.X:=0; ach.RotationCenter.Y:=0.5;
  acm:=TRoundRect.Create(Self);
  acm.Parent:=self;
  acm.Width:=r*0.9;acm.Height:=3;acm.Stroke.Thickness:=2;
  acm.Position.X:=r;acm.Position.y:=r;
  acm.RotationCenter.X:=0; acm.RotationCenter.Y:=0.5;
  acs:=TRoundRect.Create(Self);
  acs.Parent:=self;
  acs.Width:=r*0.9;acs.Height:=3;acs.Stroke.Thickness:=2;
  acs.Position.X:=r;acs.Position.y:=r;
  acs.RotationCenter.X:=0; acs.RotationCenter.Y:=0.5;
  for I := 0 to 59 do
  begin
    if i mod 5 =0 then rshm:=6 else rshm:=4 ;
    shm[i]:=TCircle.Create(self);shm[i].Parent:=self;
    shm[i].Width:=2*rshm;shm[i].Height:=2*rshm;;
    shm[i].Position.X:=r+r*sin(2*i*pi/60)-rshm;
    shm[i].Position.Y:=r-r*cos(2*i*pi/60)-rshm;
    end;
  for I := 1 to 12 do
  begin
    shc[i]:=TText.Create(self);shc[i].Parent:=self;
    shc[i].Width:=30;shc[i].Height:=30;;
    shc[i].Position.X:=r+(r-20)*sin(2*i*pi/12)-15;
    shc[i].Position.Y:=r-(r-20)*cos(2*i*pi/12)-15;
    shc[i].Font.Size:=22; shc[i].Text:=inttostr(i);
  end;
```

```
  DrawAce;
end;
procedure TCeasS.DrawAce;
begin
  ach.RotationAngle:=-90+(ora+min/60)*30;
  acm.RotationAngle:=-90+(min+sec/60)*6;
  acs.RotationAngle:=-90+sec*6;
end;
procedure TCeasS.SetOra(Value:Word);
begin
  if FOra<>Value then begin Fora:=Value; DrawAce;end;
end;
procedure TCeasS.SetMin(Value:Word);
begin
  if FMin<>Value then begin FMin:=Value; DrawAce;end;
end;
procedure TCeasS.SetSec(Value:Word);
begin
  if FSec<>Value then begin FSec:=Value; DrawAce;end;
end;
end.
```

**Vizualizarea exemplarelor clasei TCeasS**

Pentru crearea în fereastra formei Form1 a unui exemplar al clasei TCeasS se parcurg următorii pași:

1. În compartimentul **uses** al unit-ului unit1 adăugăm numele unit-ului unde a fost creată clasa (uCeasA);
2. Declarăm global o variabilă **var** Ceas:TCeasS;
   Adăugăm pe formă un buton (Button1) și programăm evenimentul
   **procedure** TForm1.Button1Click(Sender: TObject);
   **begin**
   Ceas:=TCeasS.Create(self);
   Ceas.Parent:=Self;
   Ceas.Ora:=11; Ceas.min:=23; Ceas.Sec:=42;
   **end;**
3. Compilăm și executăm aplicația. Ca urmare apare ceasul care va indica timpul specificat în evenimentul Button1Click (ora 11, 23 minute, 42 secunde. Dimensiunile ceasului vor fi cele predefinite în constructorul clasei.

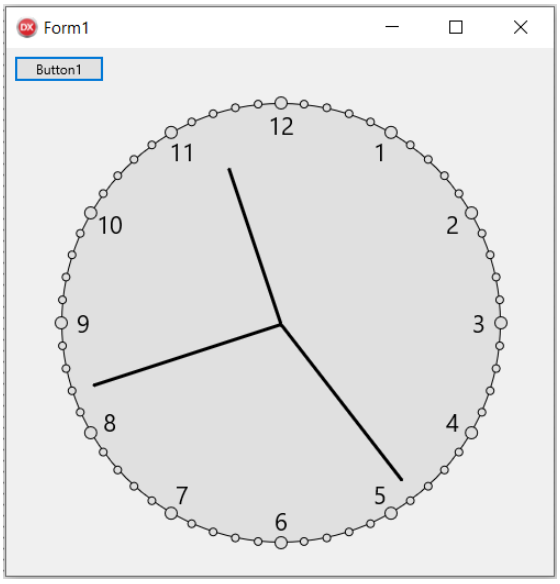



Figura . Ceas Static

**Ceas animat**

În baza clasei TCeasS tot în unit-ul uCeasA în partea de implementare (**implementation**) delarăm clasa TCeasA cu următorul cod:

```
type TCeasA=class(TCeasS)
  private
  tim:TTimer;
  procedure ontim(Sender: TObject);
  public
  constructor Create(Owner: TComponent); override;
 end;
```

În partea de implementare (implementation) adăugăm următorul cod:

```
constructor TCeasA.Create(Owner: TComponent);
begin
  inherited Create(Owner);
  tim:=TTimer.Create(self); tim.Enabled:=true;
  tim.Interval:=10; tim.OnTimer:=ontim;
  end;
procedure TCeasA.ontim(Sender: TObject);
var h,m,s,ms:word;
begin
  decodeTime(now,h,m,s,ms);
  ach.RotationAngle:=-90+(h+m/60)*30;
  acm.RotationAngle:=-90+(m+s/60)*6;
```

```
  acs.RotationAngle:=-90+s*6;
end;
```

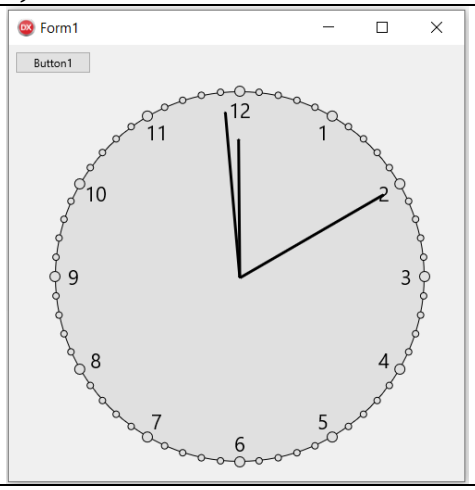



Fig Captură ecran la ora 11:59:10

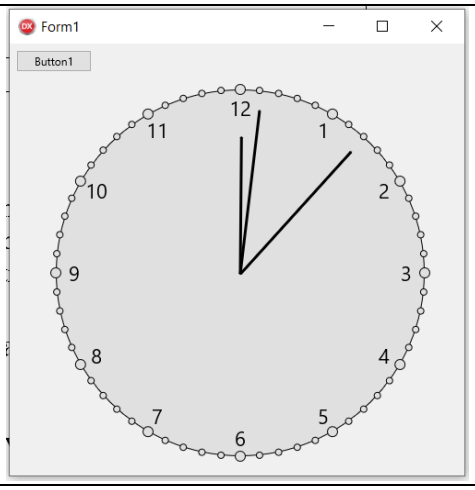



Fig Captură ecran la ora 12:01:07

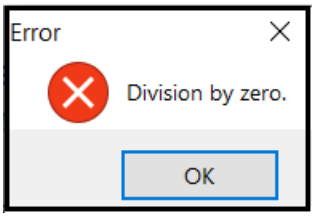
## Gestionarea excepțiilor

O excepție se declanșează atunci când o eroare sau un alt eveniment întrerupe derularea normală a unui program. Excepția transferă controlul către un gestionar (handler) de excepție, care vă permite să separați logica normală de execuție a programului de gestionarea erorilor. O excepție poate transmite informații, cum ar fi un mesaj de eroare, de la punctul unde a fost depistată până la punctul în care este tratată.

Când o aplicație folosești unitatea SysUtils , majoritatea erorilor rutine sunt convertite automat în excepții. Un număr de erori care altfel ar determina oprirea unei aplicații (memorie insuficientă, divizare la zero, erori generale de protecție) pot fi astfel capturate și gestionate. Exemplu:

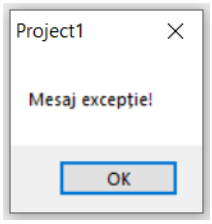
```
procedure TForm1.Button1Click(Sender: TObject);
var a,b:integer;
begin
  a:=0; b:=1 div a;
  label1.Text:=inttostr(b);
end;
```

La executare se afișează următorul mesaj de eroare



În Delphi excepțiile sunt considereate obiecte. Clasa de bază are numele **Exeption** și este declarată în unitatea SysUtils. Constructorul acestei clase are un singur argument de tipul string în care de regulă se indică mesajul exepției declanșate. Exemplu:

```
procedure TForm1.Button7Click(Sender:
TObject);
var E:Exception;
begin
   E:=Exception.Create('Mesaj excepție!');
   ShowMessage(E.Message);
end;
```




Pentru gestionarea exceșiilor în Delphi se utilizează instrucțiunea **try**…**except** cu următoarele sintaxă:

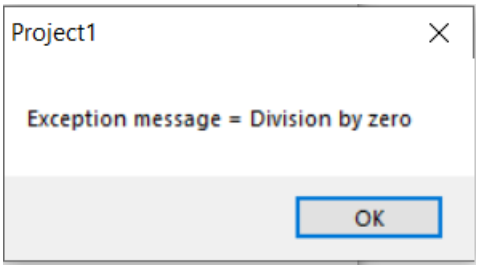
**try** <instrucțiuni, care potential pot provoca erori>

**exept** [on <clasa excepției> do <instrucțiune>];

**end**;

Exemplu:

```
procedure
TForm1.Button9Click(Sender:
TObject);
 var   n1, n2 ,r: integer;
 begin
   try
    n1 := 4; n2 := 0; r := n1
div n2;
   except
     on E : Exception do
       ShowMessage('Exception
message = '+E.Message);
   end; end;
```

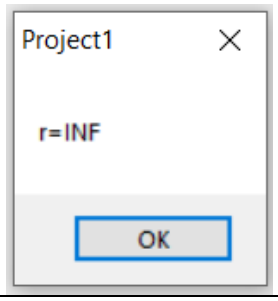


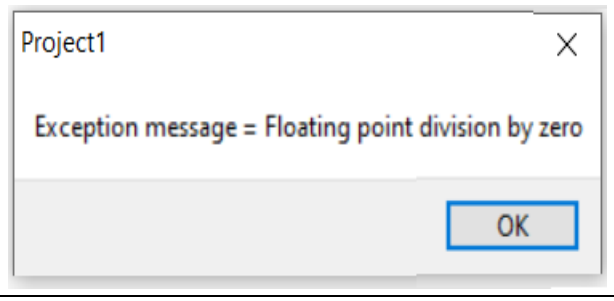

Atenție! În Delphi FMX și Delphi VCL unele excepțiile sunt tratate în mod diferit. Exemplu:

| Delphi FMX | Delphi VCL |
|---|---|

```
var   n1, n2 ,r: real;
 begin
   try
     n1 := 4;      n2 := 0;      r:=n1/n2;
     ShowMessage('r='+floattostr(r));
   except
     on E : Exception do
       ShowMessage('Exception message = '+E.Message);
   end;
```

```
end;
```




Instrucțiunnea **try…**finally are următoarea sintaxă:

**Try** <blocul de instrucțiuni1> **finally** <blocul de instrucțiuni2> **end**;

De regulă se utilizează în blocurile de instrucțiuni unde se creează, se utilizează și în final se distrug diverse obiecte dinamice. Exemplu:

| Exemplul 1 | Exemplul 2 |
|---|---|
| `Reset(F);`<br>`try`<br>`  ... // process file F`<br>`finally`<br>`  CloseFile(F);`<br>`end;` | Var SL: TStringList;<br>begin<br>SL := TStringList.Create<br>try<br>SL.Text := aString;<br>SL.SaveToFile(aFilename);<br>finally<br>SL.Free<br>end;<br>end; |

## Subprograme utile pentru dezvoltarea aplicațiilor educaționale.

## Conversia datelor dintr-un format în altul.

Probleme cu privire la conversia datelor (numerelor naturale, timpului, unităților de măsură etc) dintr-un format în altul se formulează frecvent la orele de matematică și alte discipline școlare Exemplu:

Se dă un număr natural în format arab. Să se realizeze transformarea în format: a) roman; b) litera (cu cuvinte).

```
Function NArabToCharRoman(N:int64):String;
```

```
{transformă N în șir de caractere conform ortografiei limbii
române. N<1000 }
Const
  Unitati: Array[1..19] of String=('unu','doi','trei',
'patru','cinci','șase','șapte','opt','nouă','zece',
'unsprezece','doisprezece','treisprezece','paisprezece',
'cinsprezece','șaisprezece','șaptesprezece',
'optsprezece','nouăsprezece');
  zeci: Array[2..9] of String=('douăzeci','treizeci',
'patruzeci','cincizeci','șaizeci','șaptezeci','optzeci',
'nouăzeci');
Var  Temp: String;    S,Z,U: Byte;
begin
  If N = 0 then   begin  Result := 'zero'; Exit;  end;
    U := N mod 10;{unitati}
    N := N div 10; {zeci}
    Z := N mod 10; {unitati de zeci}
    N := N div 10; {zeci de zeci}
    If Z =1 then     begin   Dec(Z);   Inc(U, 10);   end;
    Temp := '';
    If Z > 1 then
    begin
      Temp := ' ' + Zeci[Z];
      If (Z <=9) then
      if U=0 then  Temp := Temp else   Temp := Temp + ' și';
    end;
    If U > 0 then Temp := Temp + ' ' + Unitati[U];
    Result := Temp;
    S := N mod 10; {Sute}
    If S > 0 then
    begin
      Temp := '';
      If S= 1 then Temp := ' ' + 'o  sută' + Temp;
      If S= 2 then Temp := ' ' + 'două sute' + Temp;
      If S> 2 then Temp := ' ' + Unitati[S] + ' sute'+Temp;
      Result := Temp + Result;
    end;
    result:=trim(result);
end;// NArabToCharRoman

function ArabToRoman(num:Integer): string;
//num<3999
Const   Nvals = 13;
  vals: array [1..Nvals] of word =(1, 4, 5, 9, 10, 40, 50,
90, 100, 400, 500, 900, 1000);
  roms: array [1..Nvals] of string[2] =('I', 'IV', 'V',
'IX', 'X', 'XL', 'L', 'XC', 'C','CD', 'D', 'CM', 'M');
var  b: 1..Nvals;
```

```
begin
  result := '';
  b := Nvals;
  while num > 0 do
  begin
    while vals[b] > num do dec(b);
    dec (num, vals[b]);
    result := result + roms[b]
  end;
end; // ArabToRoman

function RomanToArab(const aRoman: string): Integer;
  function DecodeRomanDigit(aChar: Char): Integer;
  begin
    case aChar of
      'M', 'm': Result := 1000;
      'D', 'd': Result := 500;
      'C', 'c': Result := 100;
      'L', 'l': Result := 50;
      'X', 'x': Result := 10;
      'V', 'v': Result := 5;
      'I', 'i': Result := 1
    else
      Result := 0;
    end;
  end;// DecodeRomanDigit

var i: Integer;  lCurrVal: Integer;  lLastVal: Integer;
begin
  Result := 0;
  lLastVal := 0;
  for i := Length(aRoman) downto 1 do
  begin
    lCurrVal := DecodeRomanDigit(aRoman[i]);
    if lCurrVal < lLastVal then
      Result := Result - lCurrVal
    else
      Result := Result + lCurrVal;
    lLastVal := lCurrVal;
  end;
end; // RomanToArab
```

## Subprograme utilizate în elaborarea softurilor educaționale.

## Sortarea datelor. Metoda bulelor.

```
Procedure MBule(Var Tab:Array Of Integer);
```

```
Var i,j,t:Integer;
Begin
  For i:=Low(Tab) To High(Tab)-1 Do
  For j:=i+1 To High(Tab) Do If Tab[i]>Tab[j] Then
  Begin
    t:=Tab[i];
    Tab[i]:=Tab[j];
    Tab[j]:=t;
  End;End;// MBule
```

## Sortarea datelor. Metoda quicksort.

```
procedure quicksort(Var Tab:Array Of Integer);
procedure sort(l,r: integer);
var
  i,j,x,y: integer;
begin
  i:=l; j:=r; x:=Tab[(l+r) DIV 2];
  repeat
    while Tab[i]<x do i:=i+1;
    while x<Tab[j] do j:=j-1;
    if i<=j then
    begin
      y:=Tab[i]; Tab[i]:=Tab[j]; Tab[j]:=y;
      i:=i+1; j:=j-1;
    end;
  until i>j;
  if l<j then sort(l,j);
  if i<r then sort(i,r);
end;// sort

begin {quicksort};
```

```
  sort(Low(Tab),High(Tab));
end;
```

## Calculare determinanților

```
Function det(x:mat):extended;
//Type Mat=Array of array of extended;
var i,j,k,t:integer; s:real; m:mat;//m-minorul
begin
t:=high(x); //
setlength(m,t,t); //
if t=0 then det:=x[0,0] else
  begin
   s:=0;
   for k:=0 to t do
   begin
    for i:=0 to t-1 do for j:=0 to t-1 do
    if j<k then m[i,j]:=x[i+1,j] else
m[i,j]:=x[i+1,j+1];
    if odd(k) then s:=s-x[0,k]*det(m)else
s:=s+x[0,k]*det(m);
   end;
   det:=s;
  end;
end;
```

## Rezolvarea numerică a sistemelor de ecuații liniare

```
procedure gauss(a:mat;b:vnr; var x:vnr);
 {Procedura serveste la rezolvarea sistemelor de
ecuatii liniare
  a-coeficientii de pe linga x a[i,j];  b-coeficientii
liberi b[i];
  x-solutiile ecuatiilor liniare x[i];}
Var k,i,j,h,n:integer;s,maxa,p:extended;
begin
 n:=high(b);
for k:=0 to n-1 do
begin
   maxa:=abs(a[k,k]);  {---se afla locul ecuatiei cu
coeficientul "a" maxim--}
   h:=k;
   for i:=k to n do
   begin
      if abs(a[i,k])>maxa then
      begin
        maxa:=abs(a[i,k]);    h:=i;
      end;
   end;
{-------------------------------------------------------}
   if h<>k then
   begin
     for i:=k to n do
         begin
            p:=a[h,i];  {ecuatia primul coeficient al
}
```

```
            a[h,i]:=a[k,i];{careea este maximal se
aduce pe}
            a[k,i]:=p;  {primul loc
}
         end;
     p:=b[h];
     b[h]:=b[k];
     b[k]:=p;
   end;
 for i:=k+1 to n do
   begin
    for j:=k+1 to n do a[i,j]:=(a[i,j]-
a[i,k]*a[k,j]/a[k,k]);
    b[i]:=(b[i]-a[i,k]*b[k]/a[k,k]);
   // se imparte la a[k,k] pentru ca matricea sa ramana
echivalenta
    for j:=0 to k do a[i,j]:=0;
   end;
 end; {k}
 x[n]:=b[n]/a[n,n];
 for i:=n-1 downto 0 do
 begin
   s:=0;
    for j:=i+1 to n do
    s:=s+a[i,j]*x[j];
   x[i]:=(b[i]-s)/a[i,i];
 end;
end;{gauss}
procedure cramer(a:mat;b:vnr; var x:vnr);
```

```
 {Procedura serveste la rezolvarea sistemelor de
ecuatii liniare
  a-coeficientii de pe linga x a[i,j];
  b-coeficientii liberi b[i];
   x-solutiile ecuatiilor liniare x[i];}
Var k,i,j,h,n,l:integer;s,maxa,p,f,d0:extended; mp:mat;
begin
 n:=high(b);
 setlength(mp,n+1,n+1);  //
 d0:=det(a);//
 for j:=0 to n do
 begin
   for k:=0 to n do for l:=0 to n do mp[k,l]:=a[k,l];
    for i:=0 to n do mp[i,j]:=b[i];
   x[j]:=det(mp)/d0;
 end;
end;{Cramer}
```

## Calcularea perimetrului, ariei și coordonatelor centrului geometric al unui poligon

```
procedure CalcPPlg(p:TPolygon;scl:extended;var
prm,splg,xcc,ycc:extended) ;
var p1,p2,p3:TPointF;
    i,n:integer;
    xc,yc,st,l1,l2,l3,pp2:single;
begin
  n:=length(p);
 // p1.X:=100+random(200);p1.Y:=100+random(200);;
```

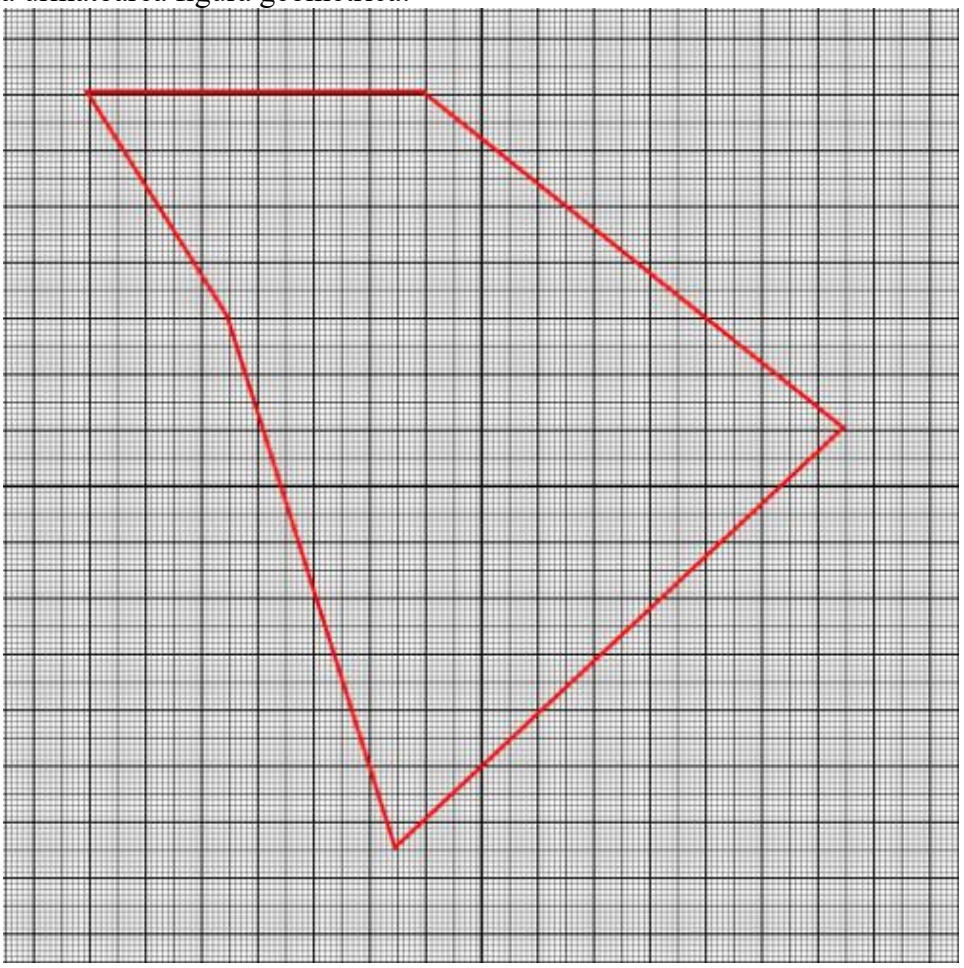
```
 p1.X:=p[0].X;   p1.y:=p[0].y;
  prm:=0;
  for I := 1 to n-1 do prm:=prm+sqrt(sqr(p[i].X-p[i-1].X)+
sqr(p[i].Y-p[i-1].Y) );
  prm:=prm+sqrt(sqr(p[n-1].X-p[0].X)+ sqr(p[n-1].Y-p[0].Y)
);
  prm:=prm/scl;
  splg:=0; xcc:=0;ycc:=0;
  for i:=0 to n-2 do
  begin
    p2:=p[i]; p3:=p[i+1];
    l1:=sqrt(sqr(p2.X-p1.X)+sqr(p2.y-p1.y));
    l2:=sqrt(sqr(p3.X-p2.X)+sqr(p3.y-p2.y));
    l3:=sqrt(sqr(p3.X-p1.X)+sqr(p3.y-p1.y));
    pp2:=(l1+l2+l3)/2;
    xc:= (p1.X+p2.X+p3.X)/3; yc:=(p1.Y+p2.Y+p3.Y)/3;
    st:=sqrt(pp2*(pp2-l1)*(pp2-l2)*(pp2-l3));
    xcc:=xcc+st*xc;  ycc:=ycc+st*yc;
    splg:=splg+st;
  end;
    p2:=p[0]; p3:=p[n-1];
    l1:=sqrt(sqr(p2.X-p1.X)+sqr(p2.y-p1.y));
    l2:=sqrt(sqr(p3.X-p2.X)+sqr(p3.y-p2.y));
    l3:=sqrt(sqr(p3.X-p1.X)+sqr(p3.y-p1.y));
    pp2:=(l1+l2+l3)/2;
    xc:= (p1.X+p2.X+p3.X)/3; yc:=(p1.Y+p2.Y+p3.Y)/3;
    st:=sqrt(pp2*(pp2-l1)*(pp2-l2)*(pp2-l3));
    xcc:=xcc+st*xc;  ycc:=ycc+st*yc;
    splg:=splg+st ;
    xcc:=xcc/splg;  ycc:=ycc/splg;
    splg:=splg/sqr(scl);
end;
```

**Subiecte randomizate pentru activități de laborator**

## Probleme cu poligoane.

Varianta 1.

Se dă următoarea figură geometrică:



1. Denumiți laturile poligonului și aflați lungimea lor.

1._____2_____3_____4_____5_____6_____

2. Calculați perimetrul poligonului.

P=_____

3. Aflați aria figurii.

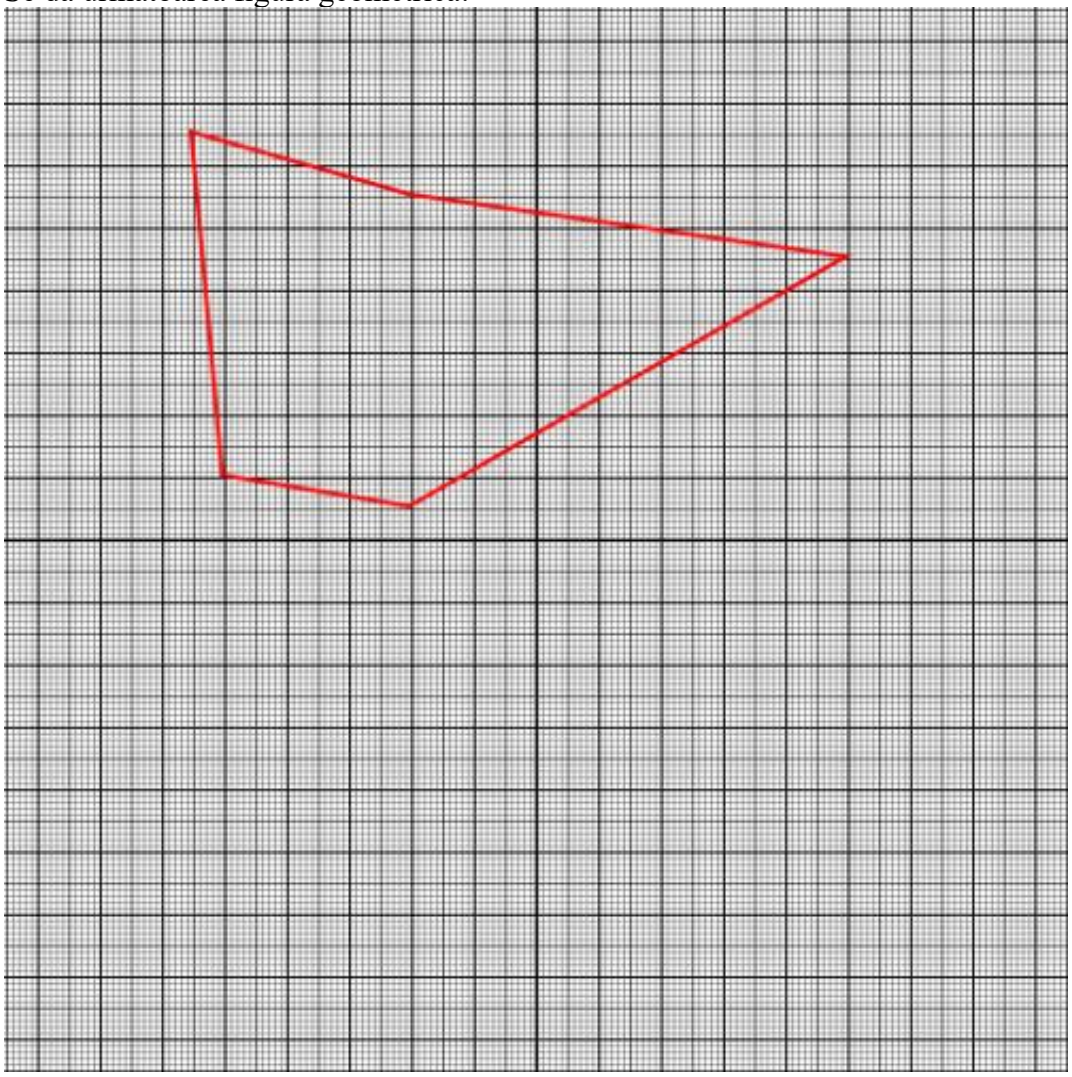
S=_____

4.Calculați coordonatele centrului geometric (de greutate/ masă)

Xc=_____Yc=_____

Varianta 2

Se dă următoarea figură geometrică:



1. Denumiți laturile poligonului și aflați lungimea lor.

1._____2_____3_____4_____5_____6_____

2. Calculați perimetrul poligonului.

P=_____

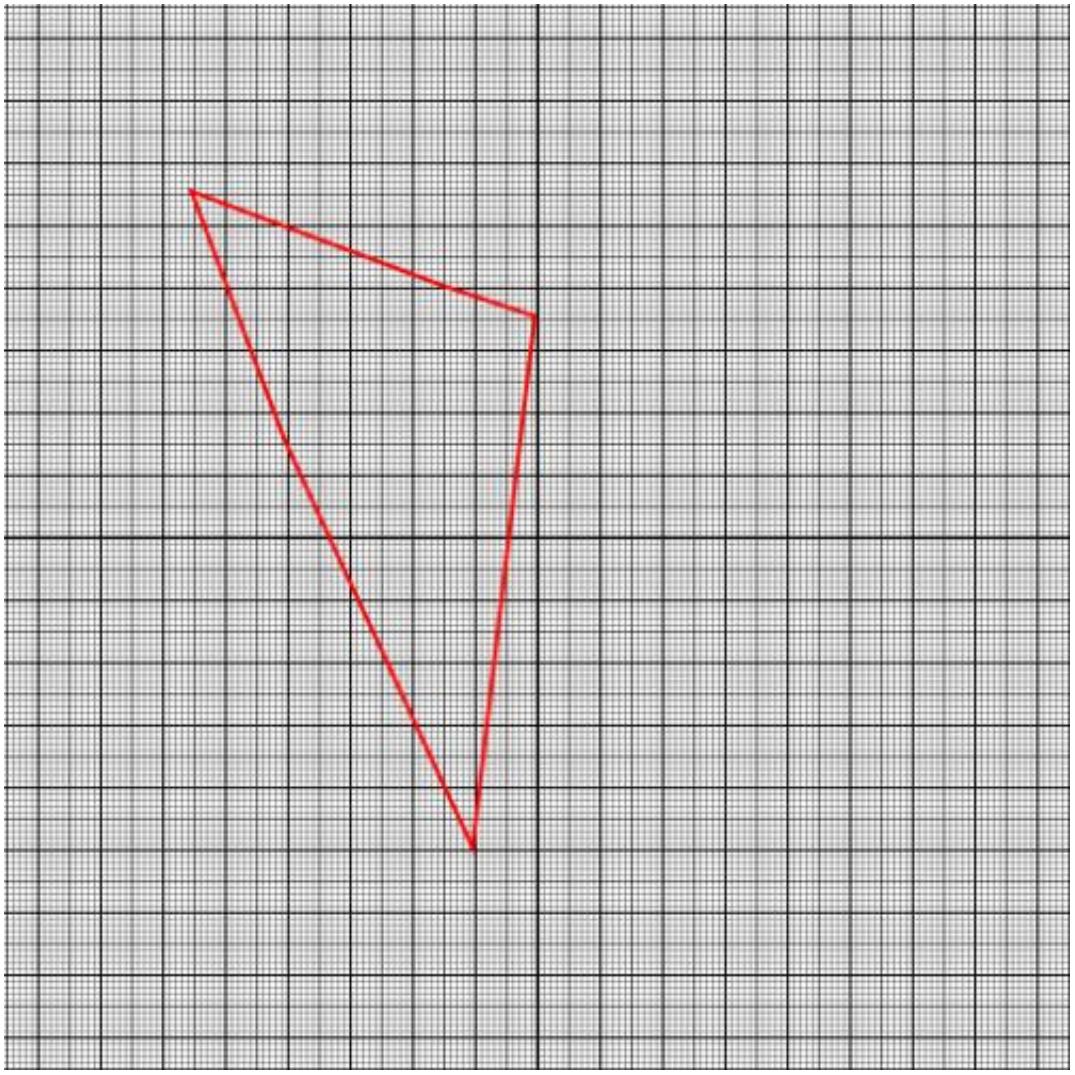
3. Aflați aria figurii.

S=_____

4. Calculați coordonatele centrului geometric (de greutate/ masă) Xc=_____Yc=_____

Varianta 3

Se dă următoarea figură geometrică:



1. Denumiți laturile poligonului și aflați lungimea lor.

1._____2_____3_____4_____5_____6_____

2. Calculați perimetrul poligonului.

P=_____

3. Aflați aria figurii.

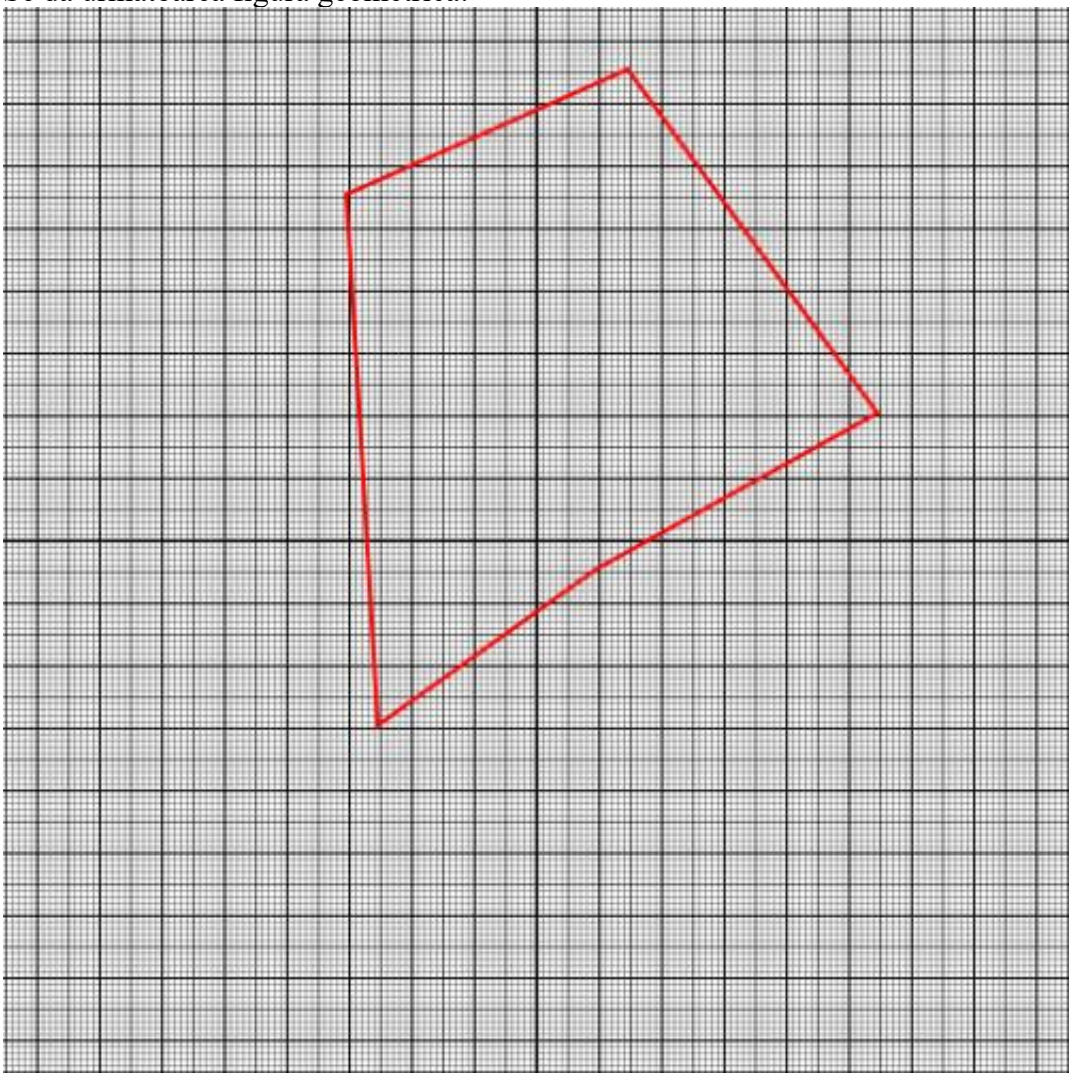
S=_____

4.Calculați coordonatele centrului geometric (de greutate/ masă)

Xc=_____Yc=_____

Varianta 4

Se dă următoarea figură geometrică:



1. Denumiți laturile poligonului și aflați lungimea lor.

1._____2_____3_____4_____5_____6_____

2. Calculați perimetrul poligonului.

P=_____

3. Aflați aria figurii.

S=_____

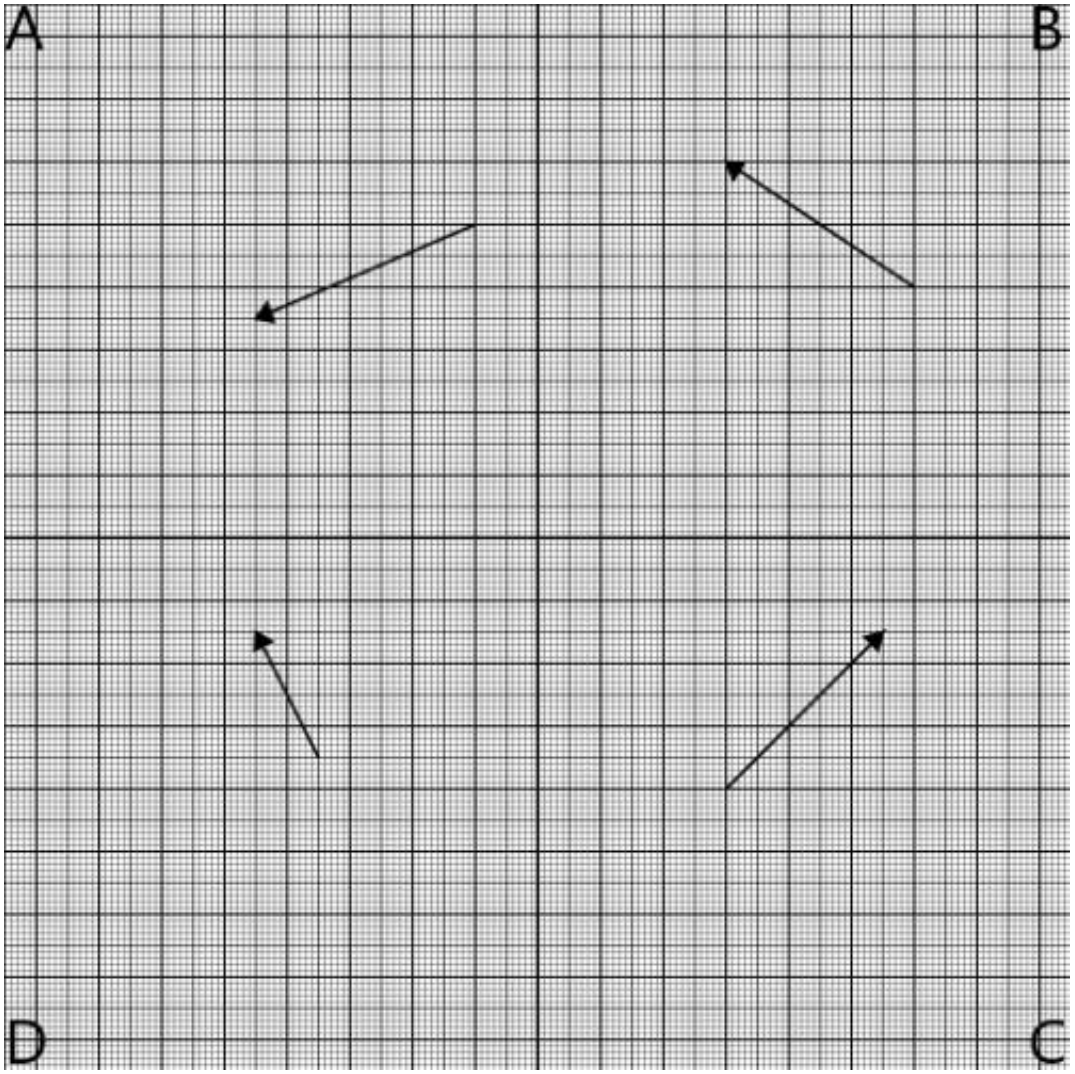
4.Calculați coordonatele centrului geometric (de greutate/ masă)

Xc=_____Yc=_____

## Probleme cu vectori.

Varianta 1

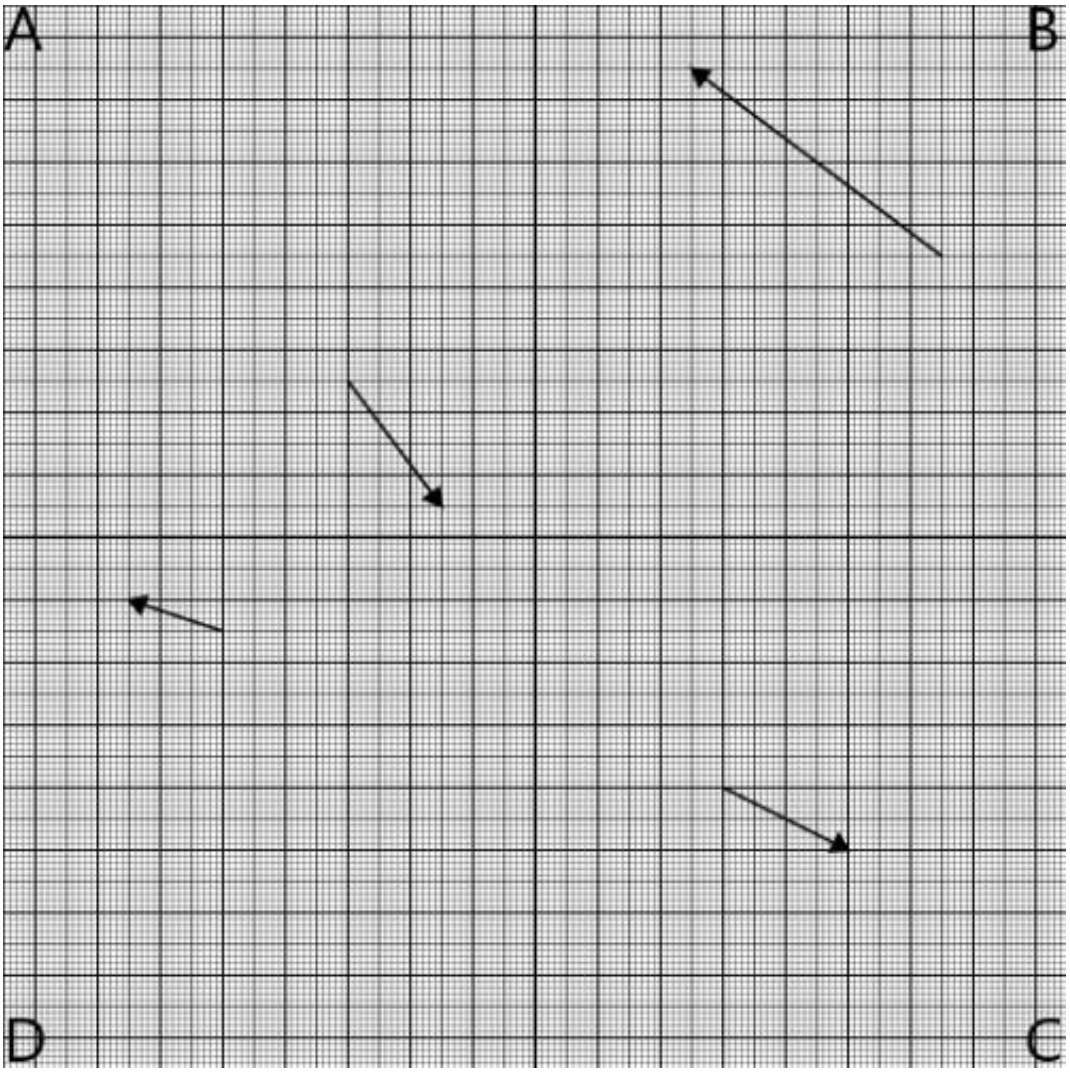
Se dau următorii vectori:




1. Calculați modulul vectorului R=-A-B-C-D și unghiul (în grade) format cu axa Ox
   |R|=_____, Alfa=_____
2. Calculați produsul scalar P=(A,B); P=_____
3. Calculați proiecțiile produsului vectorial V=[C,D];
   Vx=_____; Vy=_____; Vz=_____
4. Calculați proiecțiile produsului vectorial dublu D=[A[B,C]];
   Dx=_____; Dy=_____; Dz=_____

Varianta 2

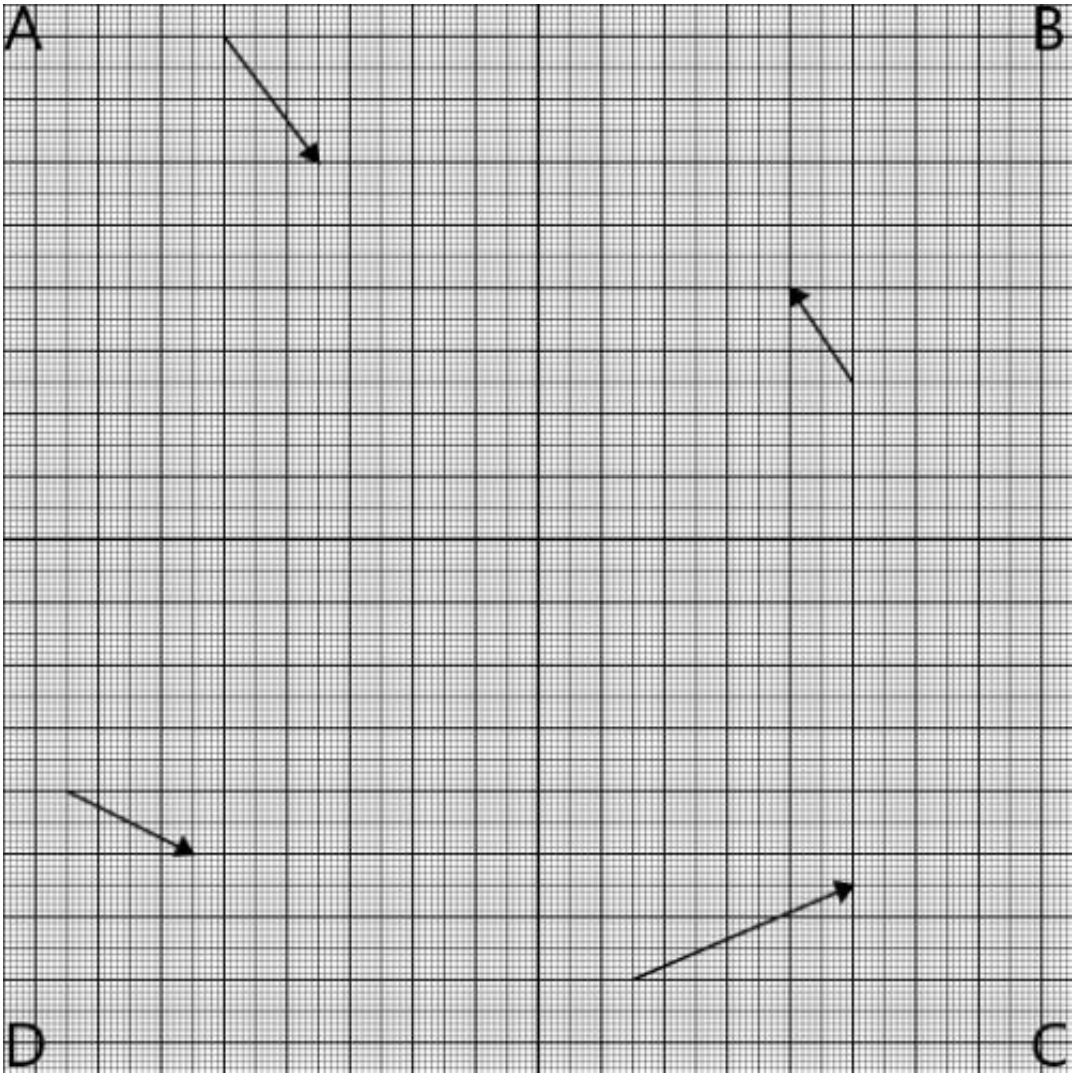
Se dau următorii vectori:




1. Calculați modulul vectorului R=-A-B-C-D și unghiul (în grade) format cu axa Ox
   |R|=_____, Alfa=_____
2. Calculați produsul scalar P=(A,B); P=_____
3. Calculați proiecțiile produsului vectorial V=[C,D];
   Vx=_____; Vy=_____; Vz=_____
4. Calculați proiecțiile produsului vectorial dublu D=[A[B,C]];
   Dx=_____; Dy=_____; Dz=_____

Varianta 3

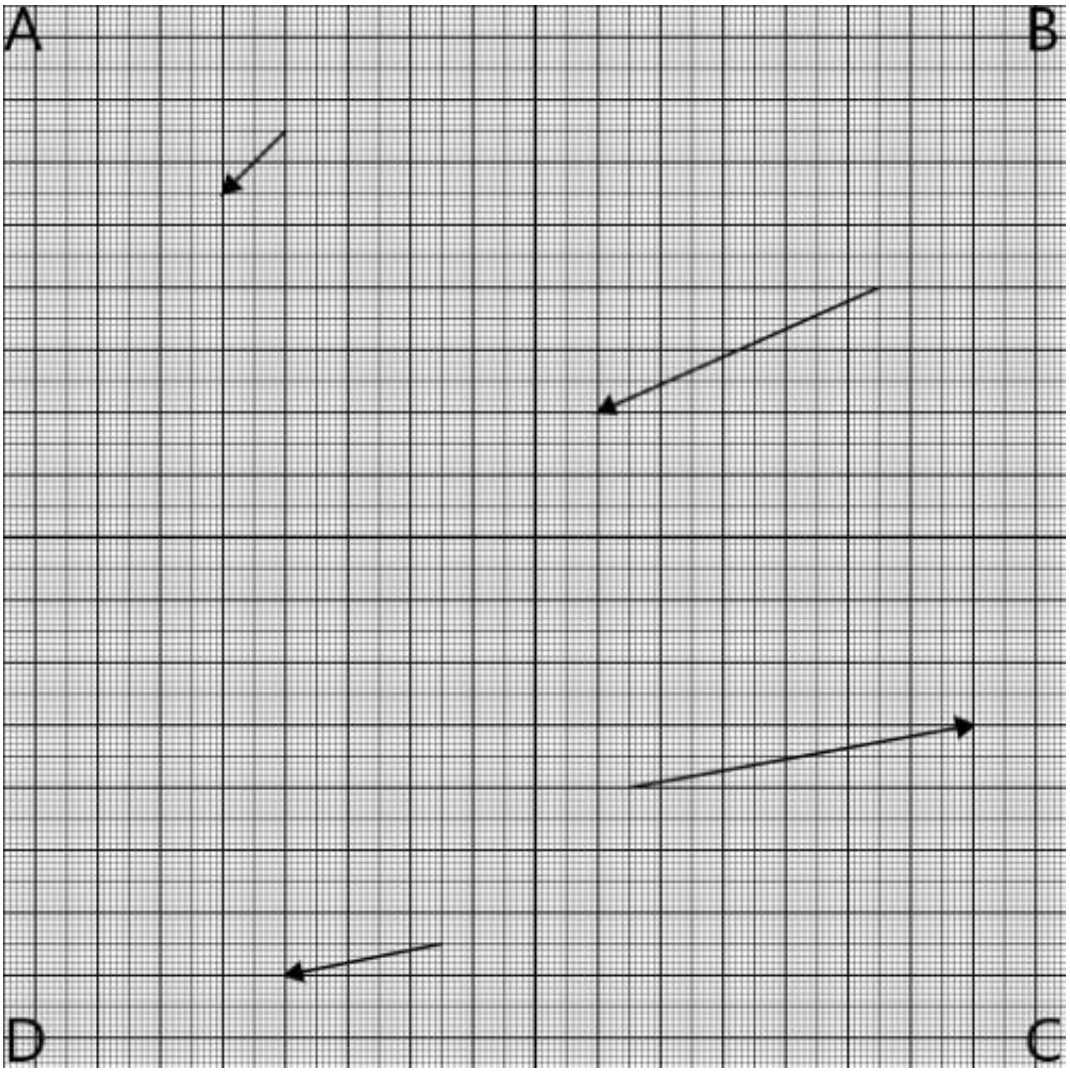
Se dau următorii vectori:




1. Calculați modulul vectorului R=-A-B-C-D și unghiul (în grade) format cu axa Ox
   |R|=_____, Alfa=_____
2. Calculați produsul scalar P=(A,B); P=_____
3. Calculați proiecțiile produsului vectorial V=[C,D];
   Vx=_____; Vy=_____; Vz=_____
4. Calculați proiecțiile produsului vectorial dublu D=[A[B,C]];
   Dx=_____; Dy=_____; Dz=_____

Varianta 4

Se dau următorii vectori:

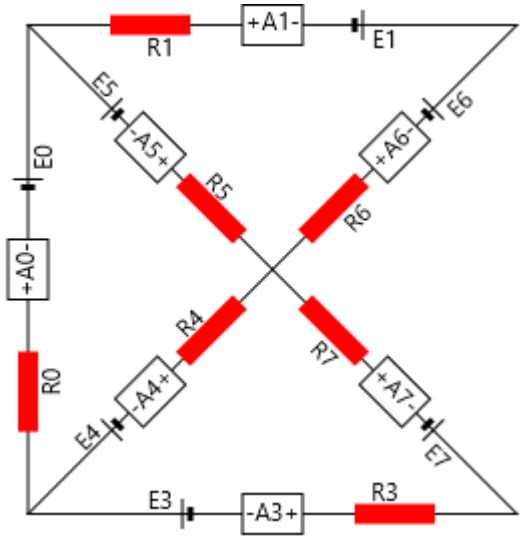



1. Calculați modulul vectorului R=-A-B-C-D și unghiul (în grade) format cu axa Ox
   |R|=_____, Alfa=_____
2. Calculați produsul scalar P=(A,B); P=_____
3. Calculați proiecțiile produsului vectorial V=[C,D];
   Vx=_____; Vy=_____; Vz=_____
4. Calculați proiecțiile produsului vectorial dublu D=[A[B,C]];
   Dx=_____; Dy=_____; Dz=_____

## Circuite de curent continuu

Varianta 1

Se dă următoarea schemă de curent continuu:



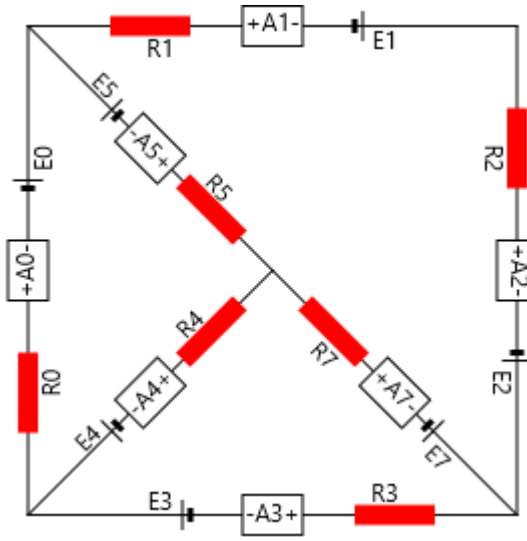

R0=8Ω E0=8V
R1=6Ω E1=6V
R3=2Ω E3=1V
R4=7Ω E4=5V
R5=5Ω E5=9V
R6=9Ω E6=9V
R7=1Ω E7=3V

1. Aflați intensitățile curenților În ramurile circuitlui.

---

Varianta 2

Se dă următoarea schemă de curent continuu:

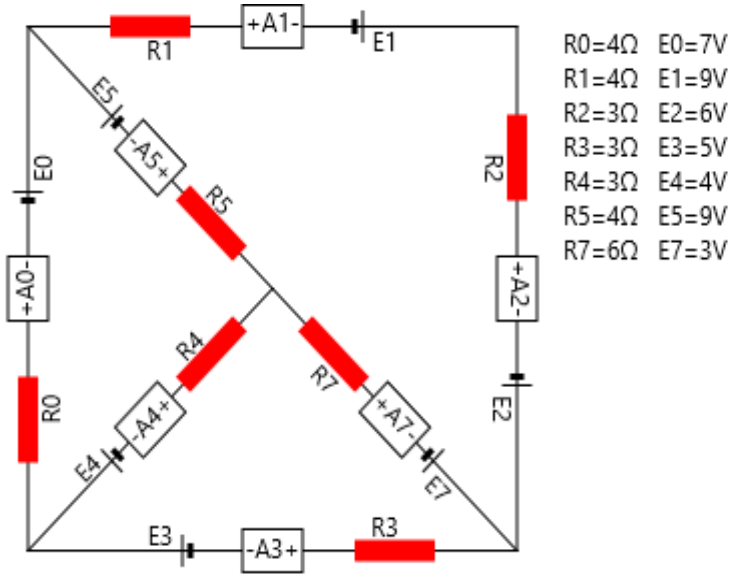



R0=6Ω E0=5V
R1=6Ω E1=6V
R2=3Ω E2=10V
R3=7Ω E3=10V
R4=4Ω E4=8V
R5=5Ω E5=6V
R7=1Ω E7=1V

1. Aflați intensitățile curenților În ramurile circuitlui.

Varianta 3

Se dă următoarea schemă de curent continuu:

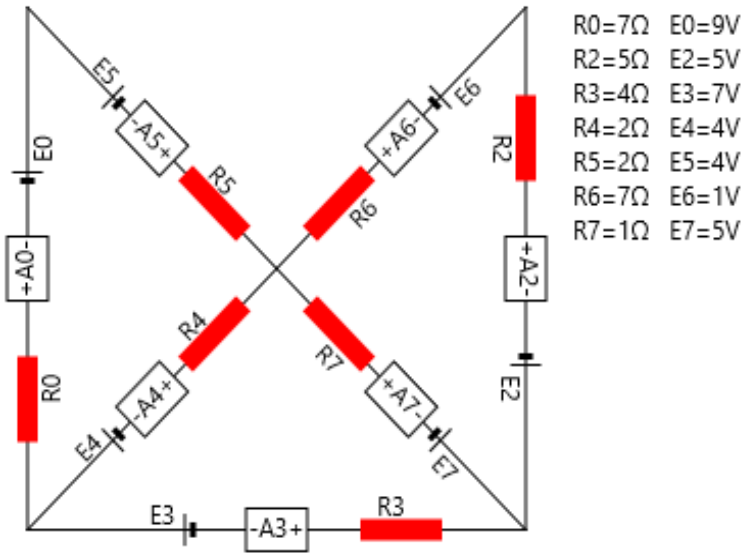



R0=4Ω E0=7V
R1=4Ω E1=9V
R2=3Ω E2=6V
R3=3Ω E3=5V
R4=3Ω E4=4V
R5=4Ω E5=9V
R7=6Ω E7=3V

1. Aflați intensitățile curenților În ramurile circuitului.

Varianta 4

Se dă următoarea schemă de curent continuu:

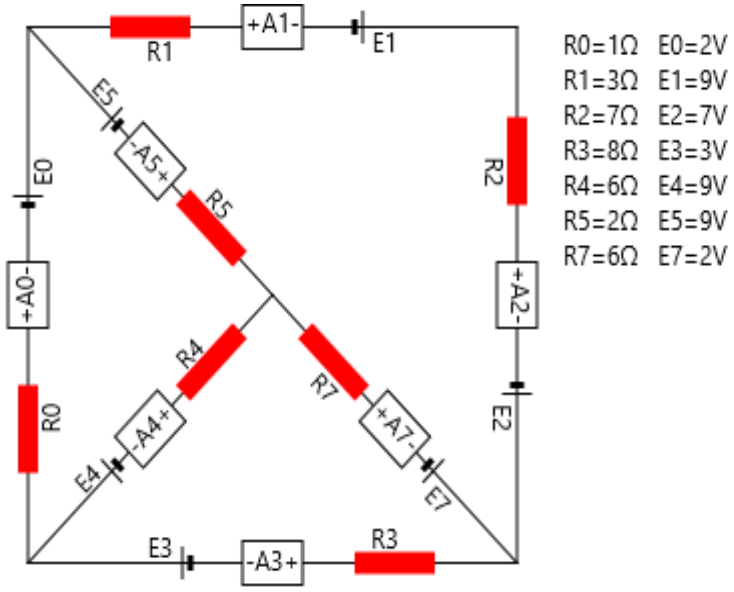



R0=7Ω E0=9V
R2=5Ω E2=5V
R3=4Ω E3=7V
R4=2Ω E4=4V
R5=2Ω E5=4V
R6=7Ω E6=1V
R7=1Ω E7=5V

1. Aflați intensitățile curenților În ramurile circuitului.

Varianta 5

Se dă următoarea schemă de curent continuu:



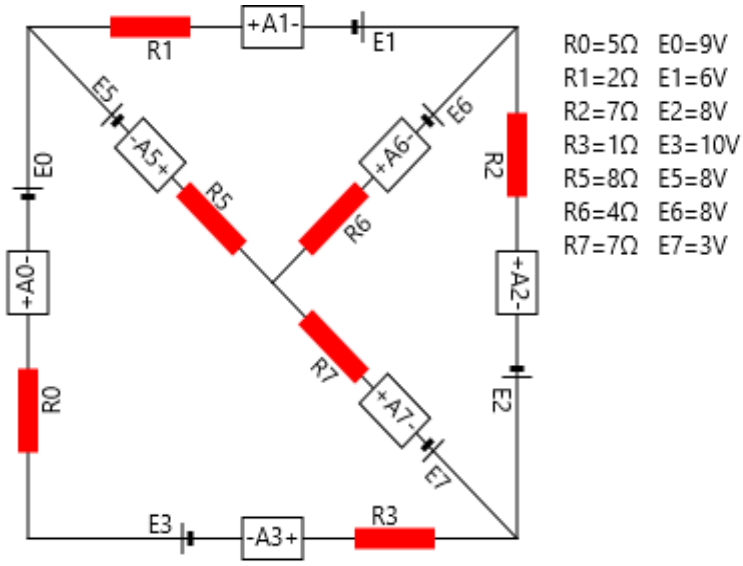

R0=1Ω E0=2V
R1=3Ω E1=9V
R2=7Ω E2=7V
R3=8Ω E3=3V
R4=6Ω E4=9V
R5=2Ω E5=9V
R7=6Ω E7=2V

1. Aflați intensitățile curenților În ramurile circuitului.

---

Varianta 6

Se dă următoarea schemă de curent continuu:



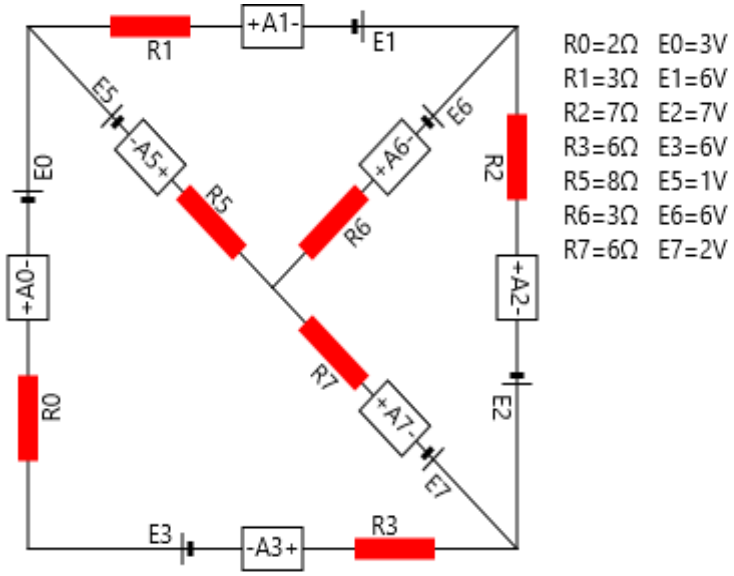

R0=5Ω E0=9V
R1=2Ω E1=6V
R2=7Ω E2=8V
R3=1Ω E3=10V
R5=8Ω E5=8V
R6=4Ω E6=8V
R7=7Ω E7=3V

1. Aflați intensitățile curenților În ramurile circuitului.

---

Varianta 7

Se dă următoarea schemă de curent continuu:



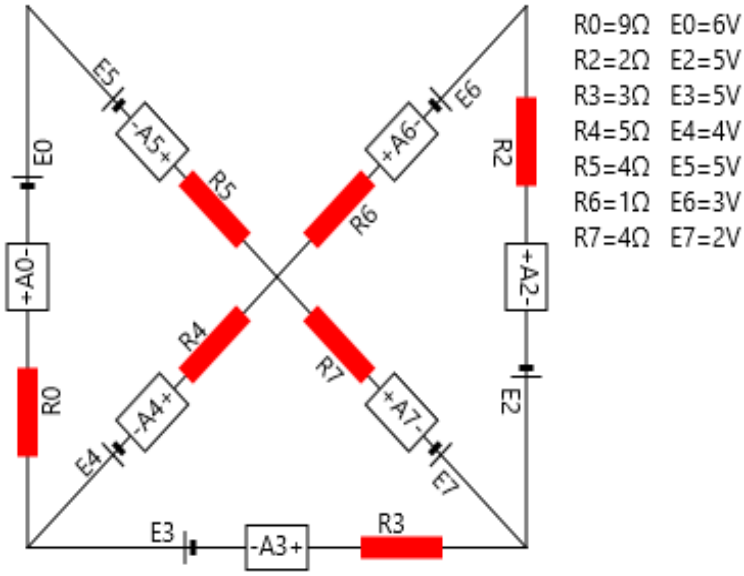

R0=2Ω E0=3V
R1=3Ω E1=6V
R2=7Ω E2=7V
R3=6Ω E3=6V
R5=8Ω E5=1V
R6=3Ω E6=6V
R7=6Ω E7=2V

1. Aflați intensitățile curenților În ramurile circuitlui.

---

Varianta 8

Se dă următoarea schemă de curent continuu:



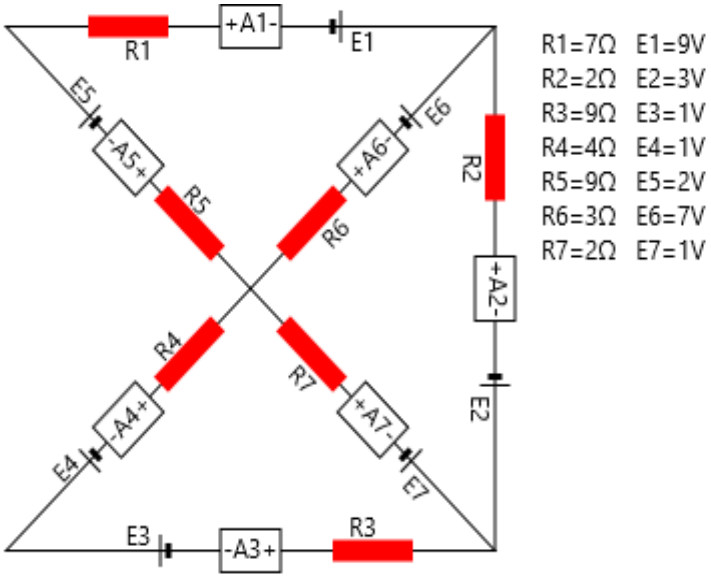

R0=9Ω E0=6V
R2=2Ω E2=5V
R3=3Ω E3=5V
R4=5Ω E4=4V
R5=4Ω E5=5V
R6=1Ω E6=3V
R7=4Ω E7=2V

1. Aflați intensitățile curenților În ramurile circuitlui.

---

Varianta 9

Se dă următoarea schemă de curent continuu:



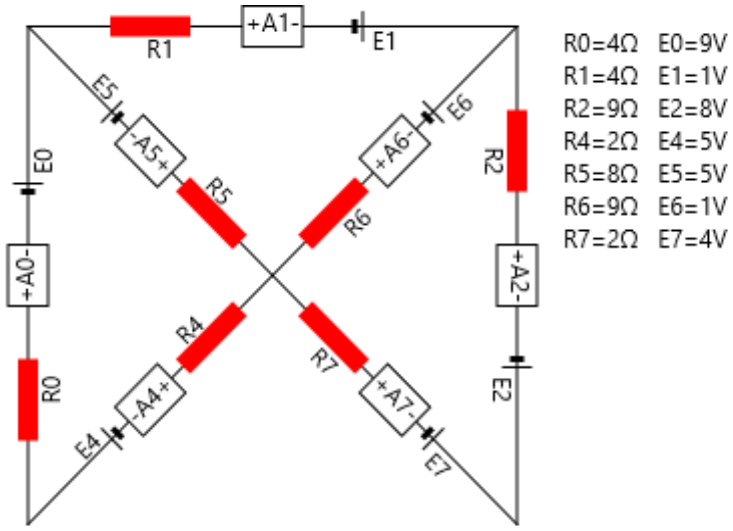

R1=7Ω E1=9V
R2=2Ω E2=3V
R3=9Ω E3=1V
R4=4Ω E4=1V
R5=9Ω E5=2V
R6=3Ω E6=7V
R7=2Ω E7=1V

1. Aflați intensitățile curenților În ramurile circuitlui.

---

Varianta 10

Se dă următoarea schemă de curent continuu:




R0=4Ω E0=9V
R1=4Ω E1=1V
R2=9Ω E2=8V
R4=2Ω E4=5V
R5=8Ω E5=5V
R6=9Ω E6=1V
R7=2Ω E7=4V

1. Aflați intensitățile curenților În ramurile circuitlui.

## Probleme cu matrici

Varianta 1

Se dau matricele:

$$M1=\begin{pmatrix} -3 & 9 & 2 \\ -8 & 2 & -7 \\ 3 & -6 & 6 \end{pmatrix} \quad \text{şi } M2=\begin{pmatrix} -5 & 5 & -3 \\ 7 & -3 & 9 \\ -2 & 8 & -2 \end{pmatrix}$$

Calculaţi:

R1) 4* M1 +4* M2;

R2) M1 x M2;

R3) M1 x M2- 4*M1;

R4) $M_1^3$;

R5) Determinantul M1;

R6) Matricea M1 transpusă ;

R7) Adjuncta matricei M1;

R8) Inversa matricei M1;

R9) Să se determine matricea X din ecuaţia M1*X=M2;

R10) Să se determine matricea X din ecuaţia 2*X+5*M1=-5*M2;

Varianta 2

Se dau matricele:

$$M1=\begin{pmatrix} 3 & -1 & -9 \\ 3 & -6 & 7 \\ -7 & 10 & -1 \end{pmatrix} \quad \text{şi } M2=\begin{pmatrix} -8 & 4 & -6 \\ 4 & -6 & 7 \\ -4 & 6 & -4 \end{pmatrix}$$

Calculaţi:

R1) 2* M1 +3* M2;

R2) M1 x M2;

R3) M1 x M2- 2*M1;

R4) $M_1^3$;

R5) Determinantul M1;

R6) Matricea M1 transpusă ;

R7) Adjuncta matricei M1;

R8) Inversa matricei M1;

R9) Să se determine matricea X din ecuaţia M1*X=M2;

R10) Să se determine matricea X din ecuaţia 4*X-4*M1=-2*M2;

Varianta 3

Se dau matricele:

$$M1=\begin{pmatrix} 3 & 4 & 4 \\ 2 & 6 & -2 \\ -4 & -7 & -8 \end{pmatrix} \quad \text{şi } M2=\begin{pmatrix} 10 & 8 & 7 \\ 5 & 2 & -2 \\ -5 & -10 & 6 \end{pmatrix}$$

Calculaţi:

R1) 2* M1 +4* M2;

R2) M1 x M2;

R3) M1 x M2- 2*M1;

R4) $M_1^2$;

R5) Determinantul M1;

R6) Matricea M1 transpusă ;

R7) Adjuncta matricei M1;

R8) Inversa matricei M1;

R9) Să se determine matricea X din ecuaţia M1*X=M2;

R10) Să se determine matricea X din ecuaţia -1*X+5*M1=-2*M2;

Varianta 4

Se dau matricele:

$$M1=\begin{pmatrix} -7 & -9 & 4 \\ -2 & 9 & 5 \\ -4 & -9 & 3 \end{pmatrix} \quad \text{şi } M2=\begin{pmatrix} -3 & 8 & 4 \\ -6 & -8 & 3 \\ -3 & 7 & 4 \end{pmatrix}$$

Calculaţi:

R1) 3* M1 +5* M2;

R2) M1 x M2;

R3) M1 x M2- 2*M1;

R4) $M_1^3$;

R5) Determinantul M1;

R6) Matricea M1 transpusă ;

R7) Adjuncta matricei M1;

R8) Inversa matricei M1;

R9) Să se determine matricea X din ecuaţia M1*X=M2;

R10) Să se determine matricea X din ecuaţia -4*X-1*M1=1*M2;

## Bibliografie


1. https://www.embarcadero.com/ru/products/delphi/starter/free-download
2. http://docwiki.embarcadero.com/RADStudio/Sydney/en/Tutorials
3. Delphi7. Developer's Guide. http://docs.embarcadero.com/products/rad_studio/delphi7/D7_DevelopersGuide.pdf
4. Осипов Д. Л. 0-74 Delphi. Программирование для Windows, OS X, IOS и Android.- СПб.: БХВ-Петербург, 2014. - 464 ISBN 978-5-9775-3289-1
5. В. Леонов Обучение мобильной разработке на Делфи. https://drive.google.com/file/d/1NVFI6uwJq0DxnfOi7_kVdgY9lhiKkurc/view
6. ObjectPascal Guide http://docs.embarcadero.com/products/rad_studio/cbuilder6/EN/CB6_ObjPascalLangGuide_EN.pdf
7. Modern Object Pascal Introduction https://castle-engine.io/modern_pascal_introduction.html
8. Программирование на языке Delphi https://www.bsuir.by/m/12_100229_1_90135.pdf
9. Никита Культин Основы программирования в Delphi 2010 БХВ-Петербург ISBN 978-5-9775-0519-2 2010 https://books.google.md/books?id=bR79n7NJYOcC&printsec=frontcover&hl=ru#v=onepage&q&f=false
10. Mihai Oltean Crina Groșan Delphi 7.0 în 200 aplicații editura Albastră 2004 p.458.
11. S. Gîncu Medologia rezolvării problemelor de informatică înstilul orientat pe obiecte Chișinău 2012. https://en.calameo.com/read/002801569ce96f41ef59f
12. Programarea orientată pe obiect și programarea vizuală. http://colegiulbratianu.ro/wpcontent/themes/theme53309/documente/software/DotNet.pdf
13. Архангельский А. Я. Програмирование в C++Builder
14. Москва: Изд. БИНОМ 2003. http://a1308.ru/books/id324 6. Культин Н. Б. C++ Builder в задачах и примерах. СПб.: БХВПетербург 2005. ftp://ftp.kng.ignix.ru/BOOKS/LANG/C%23/C++_Builder_v_zadachakh_i_prim erakh.pdf
15. A. Braicov S. Gîncu Borland C++ Builder. Chișinău 2009. http://en.calameo.com/read/002801569838bdc5a9164
16. Уроки Делфи http://www.delphi-manual.ru/lesson1.php