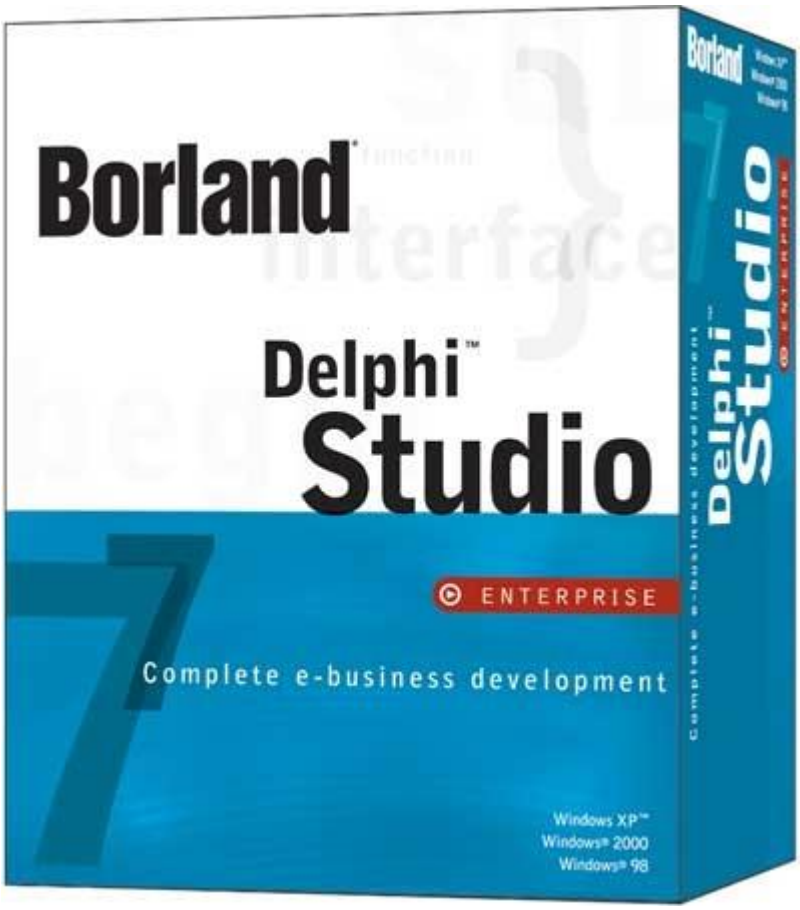
Modul Praktikum

Pemrograman Visual 1

Antarmuka Borland Delphi 7



SEKOLAH TINGGI INFORMATIKA & KOMPUTER INDONESIA

2009

Winner Vs Looser

Winner is always a part of solutions

Looser is always a part of problems

Winner sees answer in every problem

Looser sees problem in every answer

Winner always has a program

Looser always has an excuse

Winner always says, "It's difficult, but it's possible."

Looser always says, "It's possible, but it's difficult."

Kata Pengantar

Pascal merupakan salah satu bahasa pemrograman yang terkenal dengan kekuatan strukturnya. Selain itu didukung pula dengan kemudahannya untuk digunakan, sehingga menjadi pilihan yang patut diperhitungkan dalam dunia pemrograman. Kecepatan penjalanan program yang dihasilkannya juga menjadi salah satu daya saing mengapa Pascal terus berkembang hingga saat ini dengan berbagai bentuk, sampai terakhir pada bentuk OOP (object oriented programming) dan visual. Delphi merupakan versi visual dari Pascal, yang mana merupakan topik utama yang dibahas dalam modul praktikum ini.

Modul praktikum ini merupakan yang pertama disusun setelah perubahan kurikulum STIKI untuk tahun ajaran 2009 / 2010. Pada terbitan ini penyusun telah memperbaiki susunan, bahasa, gaya, dan jenis huruf yang dipakai sehingga meningkatkan derajat keterbacaan modul praktikum ini.

Penyusun berharap semoga Modul Praktikum Pemrograman Delphi ini dapat bermanfaat bagi semua pihak, terutama bagi mahasiswa yang mengambil mata kuliah Pemrograman Visual 1 di Sekolah Tinggi Informatika & Komputer Indonesia.

Namun tentunya modul ini masih jauh dari sempurna. Oleh karena itu tinjauan dan saran yang bersifat membangun tetaplah sangat diharapkan demi peningkatan kesempurnaan modul praktikum ini.

STIKI, September 2009

Penyusun

Daftar Isi

Modul 1

1

Properties & Events

Persiapan

- Baca buku pendukung / referensi mengenai Delphi
- Alat tulis untuk mengerjakan laporan praktikum yang pada sampul depan tertulis Nama, NRP dan Kelas.

Pekerjaan

- Pelajari dengan seksama teori yang terdapat pada pertemuan kali ini.
- Kerjakan percobaan dan latihan secara mandiri. Tanyakan kepada asisten jika mengalami kesulitan.
- Laporan dikerjakan dirumah untuk dikumpulkan pada pertemuan berikutnya.

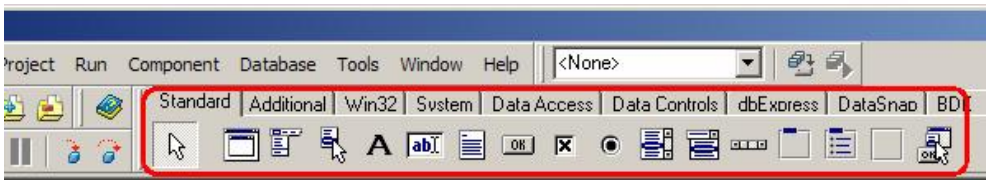
Hasil

- Praktikan diharapkan mengenal property & event dari beberapa komponen dasar pada Tab Standard.

Teori

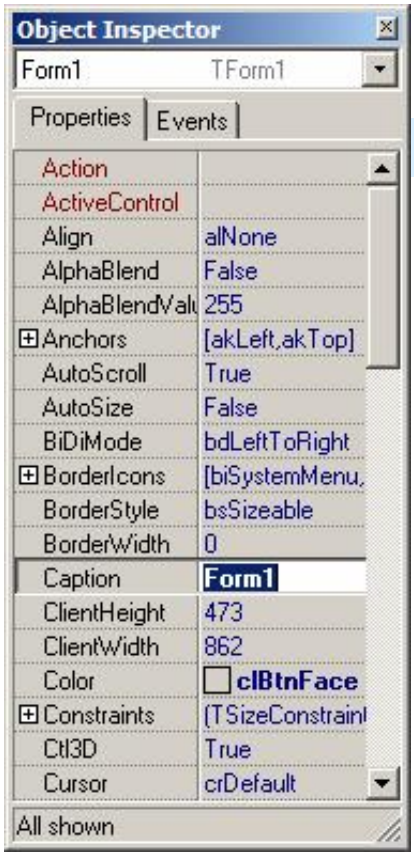
Delphi merupakan sebuah bahasa pemrograman visual yang berbasis Windows. Dalam perancangan program, Delphi menyediakan berbagai macam komponen yang memudahkan programmer dalam proses developing khususnya ketika men-develop User Interface.

Gambar 1.1: Component Pallete



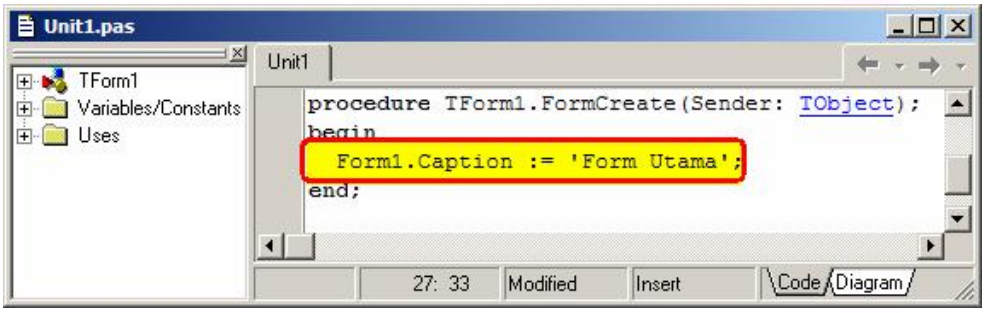
Untuk merubah atribut komponen yang digunakan, masing – masing komponen tersebut memiliki property yang mungkin saja tidak sama antara komponen satu dengan lainnya. Property tersebut dapat diubah melalui window Object Inspector yang terletak di sebelah kiri bawah User Interface Delphi.

Gambar 1.2: Object Properties Window



Untuk merubah property – property diatas tidak harus melalui Window Object Inspector, tetapi dapat juga diubah melalui Code Editor. Keuntungan merubah property melalui Code Editor adalah property dapat diubah ketika program sedang berjalan.

Gambar 1.3: Code Editor Window



Hal yang tidak kalah pentingnya untuk diketahui bagi para pengguna Delphi bahwa Delphi merupakan sebuah bahasa pemrograman dengan sifat *event-driven*. Artinya Delphi akan menjalanakan perintah ketika sebuah *event* terjadi. *Event* sendiri merupakan suatu property yang memiliki nilai sebuah pointer yang menunjuk ke sebuah prosedur. Sedangkan prosedur yang ditunjuk tadi disebut dengan *event-handler*. Tugas pemrogram yang sifatnya mengetik atau sering juga disebut dengan coding dilakukan untuk mengisi event-handler tersebut.

Untuk melihat *event* yang ada pada sebuah komponen, dapat dilihat pada Object Inspector pada tab Events (ada dua tab pada Object Inspector, yang pertama adalah tab Properties dan yang ke dua tab Events).

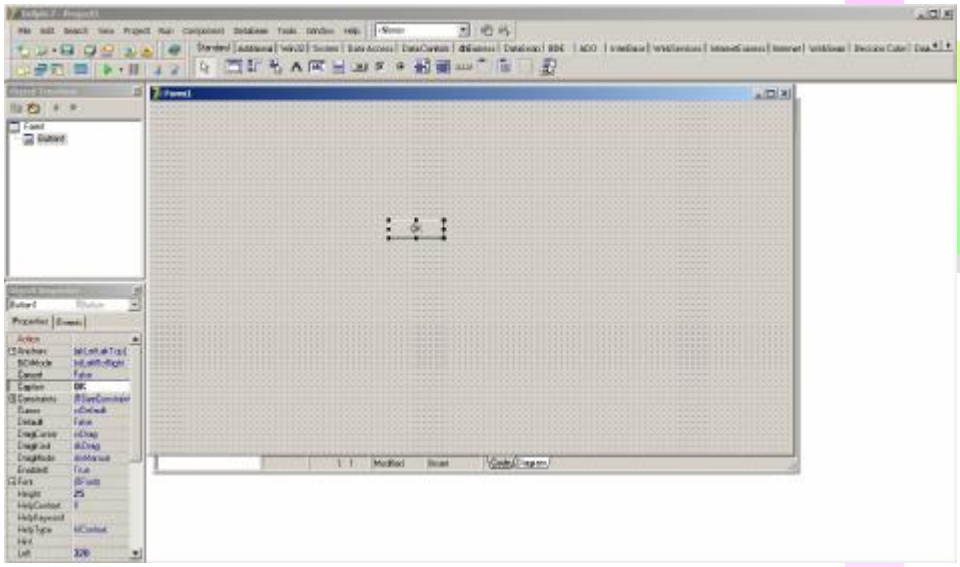
Beberapa *event* yang dimiliki komponen :

- OnClick, timbul ketika komponen diklik dengan mouse.
- OnDblClick, timbul ketika komponen diklik ganda dengan mouse.
- OnChange, timbul ketika sebuah komponen yang memi-liki properti Text (seperti TEdit) diubah isinya (baik oleh pemakai atau oleh program).
- OnEnter, timbul ketika sebuah komponen menerima fo-kus.
- OnExit, timbul ketika sebuah komponen kehilangan fo-kus.
- OnKeyDown, timbul ketika sebuah komponen menerima penekanan tombol papan ketik.
- OnKeyUp, timbul ketika sebuah komponen yang telah menerima penekanan tombol papan ketik, di mana tombol tersebut sedang dilepaskan oleh pemakai.
- OnKeyPress, timbul ketika sebuah komponen menerima sebuah karakter dari penekanan dan pelepasan sebuah tombol papan ketik.
- OnMouseDown, timbul ketika tombol mouse ditekan (dan ditahan) pada sebuah komponen.
- OnMouseUp, timbul ketika tombol mouse dilepas (setelah ditekan) pada sebuah komponen.
- OnMouseMove, timbul ketika kursor mouse berpindah posisi pada sebuah komponen.

Latihan

Tujuan dari Project Pertama ini adalah bila komponen *Button1* diklik, maka akan menampilkan pesan “HELLO WORLD”. Untuk itu maka Buatlah sebuah project baru dan letakkan sebah komponen TButton pada Form yang ada. Ubahlah property *Caption Button1* menjadi OK.

Gambar 1.4: Tampilan Project Pertama.

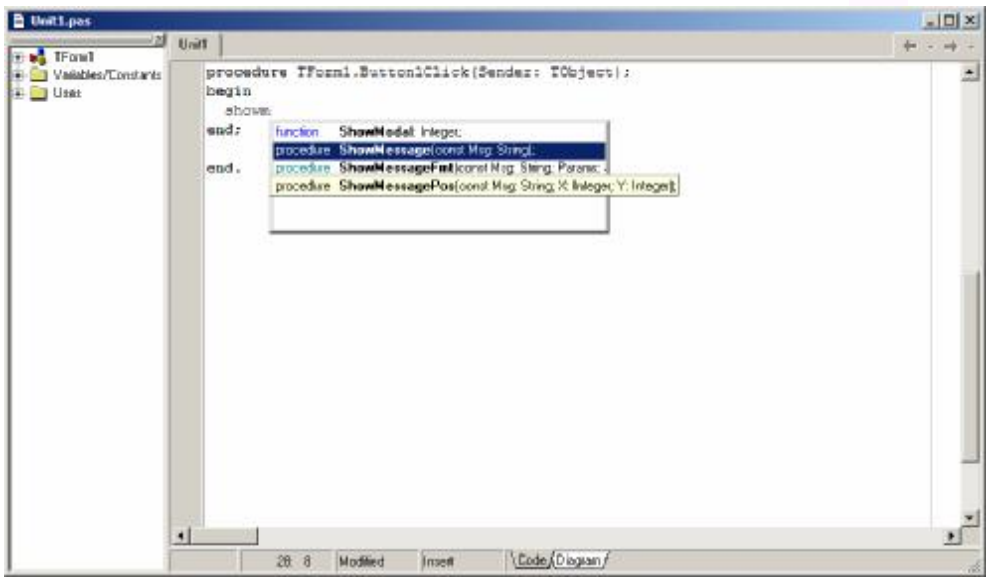


Kemudian pada event OnClick (klik 2x pada Button1 sehingga muncul *Code Editor*) ketikkan perintah berikut :

```
ShowMessage('HELLO WORLD');
```

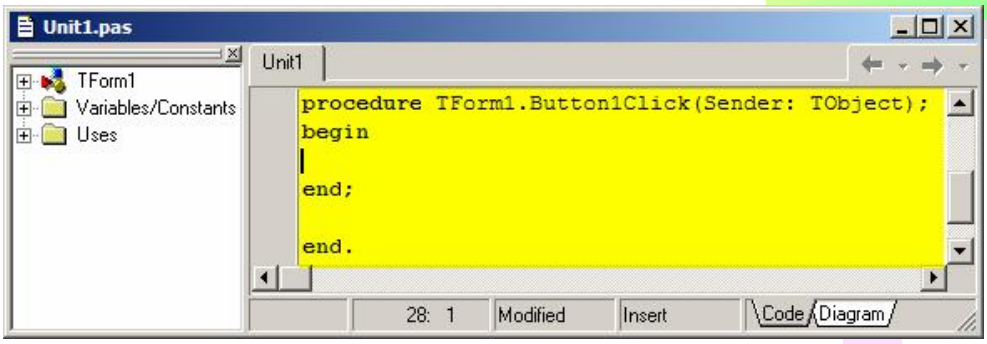
Untuk memudahkan programmer, Delphi menyediakan fasilitas berupa *Drop Down Command List* dengan cara menekan tombok *Ctrl* + *Space*. Ketika tombol *Ctrl* + *Space* ditekan maka akan muncul daftar dari perintah – perintah yang dapat digunakan. Untuk mempercepat, jangan pilih dari daftar tersebut melainkan lanjutkan mengetik perintah diatas, maka secara otomatis Delphi akan menyaring perintah yang sesuai dengan karakter yang diketikkan.

Gambar 1.5: Drop Down Command List



Selain melakukan Klik 2x pada Button1, penggunaan event juga dapat melalui *Object Inspector*. Untuk itu, klik pada Button1, dan tekan F11. Maka akan muncul *Window Object Inspector*, kemudian pilih pada tab *Events*, dan cari *event OnClick*. Lakukan klik 2x pada tempat kosong di sebelah kanan tulisan "*OnClick*". Maka programmer akan dihadapkan langsung pada code editor dengan tampilan sebagai berikut:

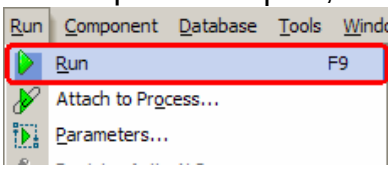
Gambar 1.6: Event-handler ButtonOKClick.



Maka secara otomatis akan disediakan event-handler seperti pada gambar di atas. Programmer dapat langsung mengetikkan kode program untuk memerintahkan komputer melakukan sesuatu bila Button1 tadi diklik.

Setelah kode program dimasukkan, kemudian cobalah jalankan program tersebut dengan salah satu cara berikut ini:

- Pilih pilihan Run|Run, atau



- Klik pada toolbar yang bertanda  , atau Tekan F9.

Maka hasil kompilasi program akan tampil seperti gambar dibawah ini :

Gambar 1.7: Hasil eksekusi Project1

Hal-hal yang harus Diperhatikan

Beberapa hal berikut ini harus Anda perhatikan di saat melakukan *coding* (menuliskan kode program). Bila Anda melanggar hal-hal berikut ini, maka akan terjadi masalah pada waktu kompilasi.

- Jangan ubah baris antara deklarasi kelas form **`(TForm1 = class(TForm))`** sampai dengan klausa private, karena ba-gian tersebut secara otomatis akan diubah oleh Delphi sesuai dengan korespondensi form yang disunting.
- Bila ingin menambahkan metode pada kelas form Anda, lakukan hanya pada bagian private atau public.
- Anda juga bisa menambahkan bagian protected dan pu-blished pada kelas form Anda.
- Jangan mengubah deklarasi peubah global form Anda (var Form1: TForm1;).
- Sebelum Anda melakukan kompilasi, pada waktu mela-kukan coding, ingat jumlah blok yang ada. Semua blok harus berakhir dengan benar.

Selain itu beberapa tip berikut bisa dilakukan untuk mempermudah:

- Lakukan penyimpanan secara berkala untuk menghindari kehilangan data kalau sewaktu-waktu aliran listrik terganggu.
- Untuk sekedar memeriksa kesahihan kode program, bi-sa dilakukan syntax check dengan memilih pilihan Pro-ject|Syntax check.
- Untuk mengkompilasi bisa dilakukan dengan pilihan Project|Compile atau Project|Build. Perbedaan compile de-ngan build adalah jika compile hanya mengkompilasi ba-ris-baris yang diubah, sedangkan build mengkompilasi seluruh project.

Tugas

Tugas 1.1

Buatlah form dengan menggunakan beberapa komponen sehingga tampilannya seperti gambar dibawah ini :

Gambar 1.8: TugasPertama.

Jelaskan komponen apa saja yang digunakan serta property apa saja yang diubah

Tugas 1.2

Buatlah rangkuman tentang fungsi dari (minimal) 5 property & 5 event dari komponen yang anda gunakan pada Tugas 1.1

Modul 2

2

Type Data & Variable

Persiapan

- Baca buku pendukung / referensi mengenai Delphi
- Pelajari tentang Type Data & Variable pada Delphi
- Bawa file program pertemuan sebelumnya
- Alat tulis untuk mengerjakan laporan praktikum yang pada sampul depan tertulis Nama, NRP dan Kelas.

Pekerjaan

- Mengenal Type Data pada Delphi
- Mengenal Variabel pada Delphi
- Mengenal multiple form

Hasil

- Praktikan diharapkan mengenal type data dan variable pada Delphi sekaligus mengimplementasikan kedalam multiple form

Teori

Variabel

Untuk dapat menggunakan Delphi dengan optimal, maka perlu untuk mengenal variabel yang berfungsi untuk menampung data dari user ataupun dari program itu sendiri. Agar dapat digunakan, maka variabel harus mempunya nama. Ada beberapa aturan penamaan variabel antara lain :

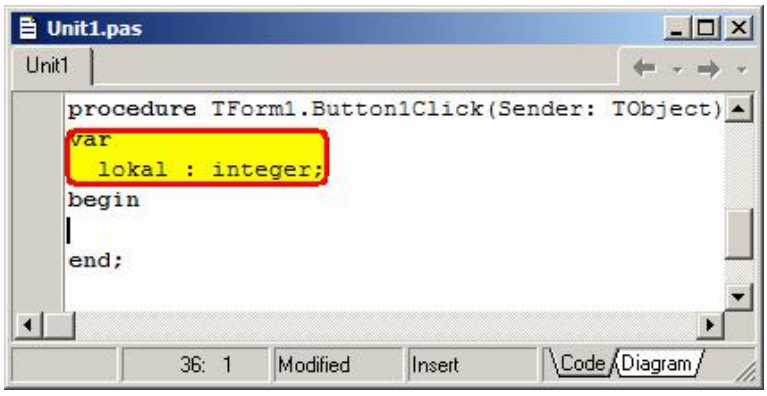
- Nama variabel harus diawali dengan huruf atau garis bawah (tidak boleh diawali dengan angka)
- Nama variabel harus terdiri dari 1 kata (tidak boleh menggunakan spasi, koma ataupun titik)
- Nama variabel tidak boleh sama dengan reserverd words (kata – kata yang telah digunakan oleh Delphi misalnya Begin)

Dalam penggunaannya, variabel dibedakan menjadi 2 jenis yaitu variabel lokal & variabel global.

Variabel lokal

Adalah variabel yang dideklarasikan di dalam sebuah procedure / function sehingga procedure / function itu sendirilah yang dapat menggunakannya. Untuk membuat variabel lokal maka harus dideklarasikan dulu perintah Var.

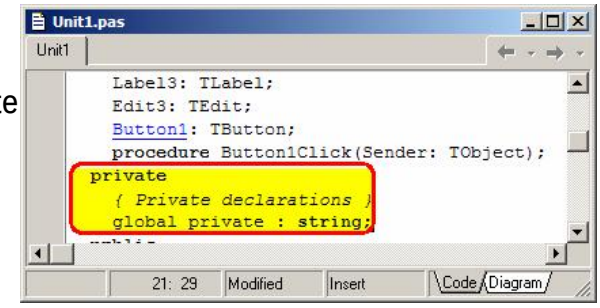
Gambar 2.1: Variabel lokal

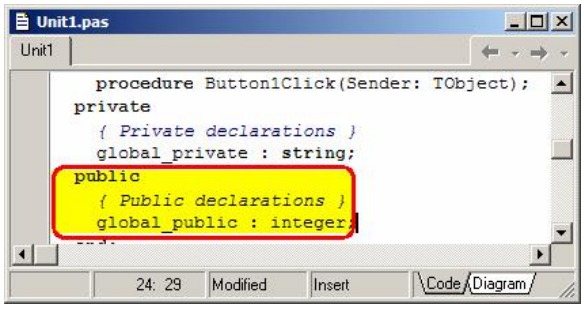


Variabel global

Adalah variabel yang dideklarasikan pada bagian public / private. Perbedaan denga variabel lokal adalah variabel global dapat digunakan oleh procedure / function yang ada di dalam unit tersebut (ingat setiap event pasti akan membuat sebuah procedure baru). Jika variabel dituliskan pada bagian private, maka variabel tersebut tidak dapat digunakan / diakses oleh unit lain.
Jika variabel dituliskan pada bagian public, maka variabel tersebut dapat digunakan / diakses dari unit lain.

Gambar 2.2: Variabel global private & public





Type Data

Type data tidak dapat dipisahkan dari variable karena type data lah yang akan menentukan jenis data yang akan ditampung di dalam sebuah variabel. Delphi mengenal beberapa type data seperti pada tabel dibawah ini :

Bilangan bulat (integer):

Type Data	Jangkauan
Byte	0 ... 255
Word	0 ... 65535
ShortInt	-128 ... 127
SmallInt	-32768 ... 32767
Integer	-2147483648 ... 2147483647
Cardinal	0 ... 4294967295
LongInt	-2147483648 ... 2147483647
Int64	-2^63 ... 2^63-1

Bilangan Pecahan (Real) :

Type Data	Jangkauan
Real	2.9 x 10^-39 ... 1.7 x 10^38
Single	1.5 x 10^-45 ... 3.4 x 10^38
Double	5.0 x 10^-324 .. 1.7 x 10^308
Extended	3.6 x 10^-4951 .. 1.1 x 10^4932

Boolean (True / False) :

Type Data	Jangkauan
Boolean	True / False
ByteBool	True / False
WordBool	True / False
LongBool	True / False

Character & String :

Type Data	Jangkauan
Char	1 karakter
ShortString	0 ... 255 karakter
String / ANSI String	0 ... 2^31 karakter
WideString	0 ... 2^30 karakter

Setelah mengetahui beberapa jenis type data diatas, hal yang perlu diperhatikan adalah bahwa setiap tampilan yang muncul di layar bertype string. Baik itu angka maupun huruf. Jadi jangan terkejut ketika menjumlahkan 1 + 2 = 12 karena 1 & 2 bertype string yang mana type data string tidak dapat dioperasikan secara matematis.

Agar permasalahan diatas dapat diselesaikan, maka dibutuhkan fungsi untuk mengkonfersi (merubah) type data dari satu type data ke type data yang lainnya.

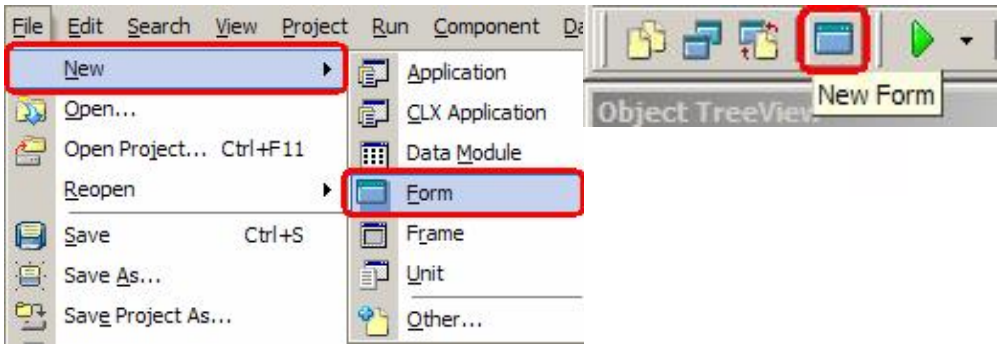
Konversi Type Data :

Fungsi	Kebalikan Fungsi	Kegunaan
StrToInt	IntToStr	Merubah string ke integer
StrToFloat	FloatToStr	Merubah string ke real
StrToDateTime	DateTimeToStr	Merubah string ke waktu

Multiple Form

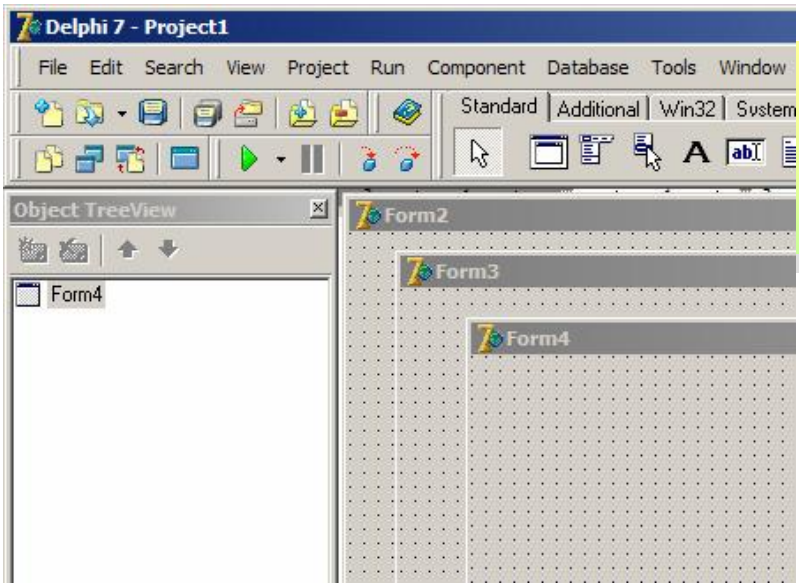
Sebuah program dapat terdiri dari 1 atau lebih form New Form tergantung kebutuhan. Untuk menambahkan form baru, dapat melalui menu File **à** New **à** Form atau melalui *speed buttons (short cut)* yang ada disebelah tombok Run.

Gambar 2.3: New Form



Setiap form akan membuat sebuah unit (file) baru. Untuk dapat memindahkan data dari satu form ke form yang lainnya maka data tersebut harus diletakkan pada variabel global public sehingga dapat diakses oleh file (form) lain.

Gambar 2.3: Multiple Form

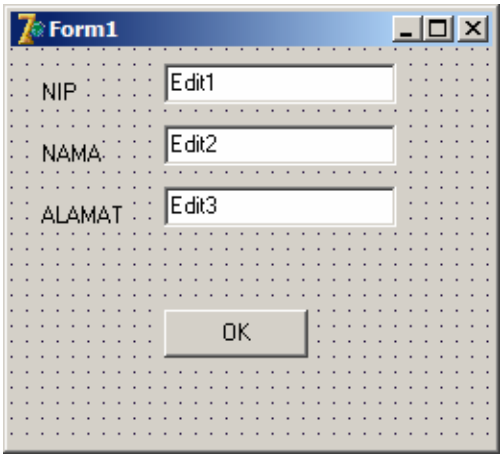


Latihan

Tujuan dari program ini adalah NIP, NAMA & ALAMAT yang di inputkan di Form1 dapat ditampilkan di Message Box.

Untuk itu maka buatlah sebuah form dengan tampilan seperti gambar berikut :

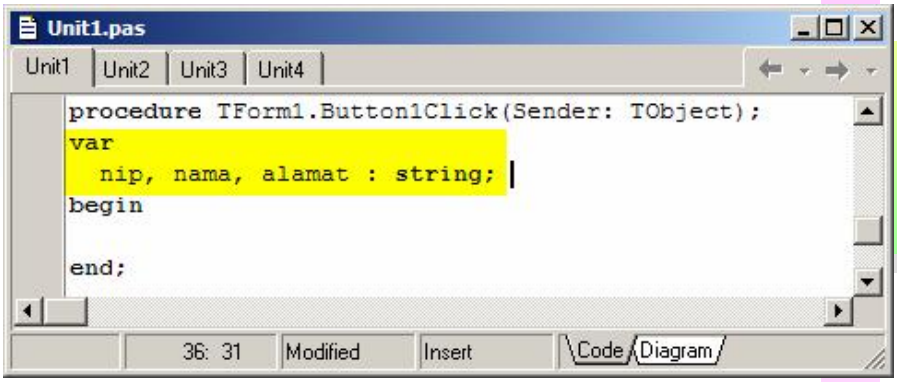
Gambar 2.4: Tampilan Project Kedua



Kemudian buatlah variabel lokal dan type data pada event OnClick Button1 seperti berikut :

```
Var
  nip, nama, alamat : string;
```

Gambar 2.5: Variabel lokal

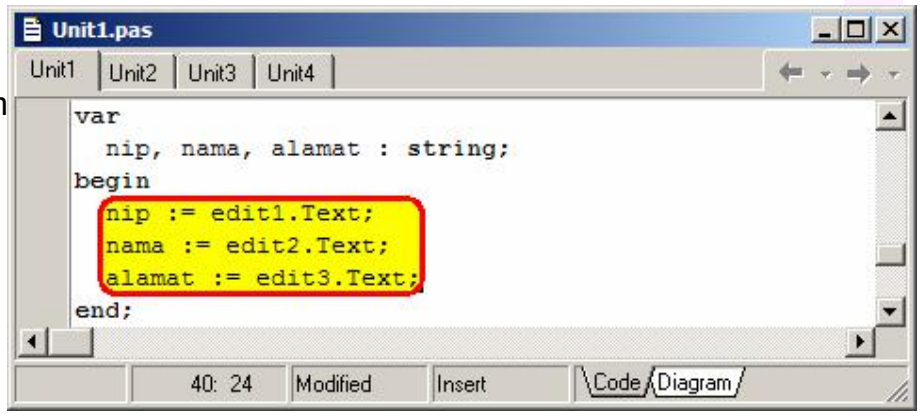


Setelah membuat variabelnya, sekarang ketikkan kode berikut diantara Begin & End pada procedure Button1Click :

```
nip := edit1.Text;
nama := edit2.Text;
alamat := edit3.Text;
```

Maksud dari kode program diatas adalah inputan user yang ada pada edit1, edit2 & edit3 dimasukkan kedalam variabel nip, nama & alamat. Karena kebetulan type data nya sama (string) maka tidak diperlukan konversi type data.

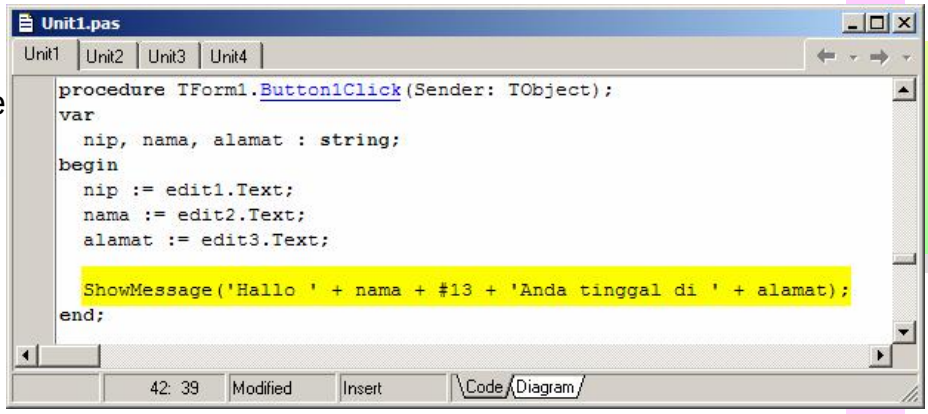
Gambar 2.6: Memasukkan data kedalam variabel



Setelah data masuk kedalam variabel, maka data tersebut akan ditampilkan kembali didalam message box. Untuk itu ketikkan perintah dibawah ini setelah perintah diatas :

```
ShowMessage('Hallo ' + nama + #13 + 'Anda tinggal di ' + alamat);
```

Gambar 2.7 : Menampilkan isi variabel ke message box

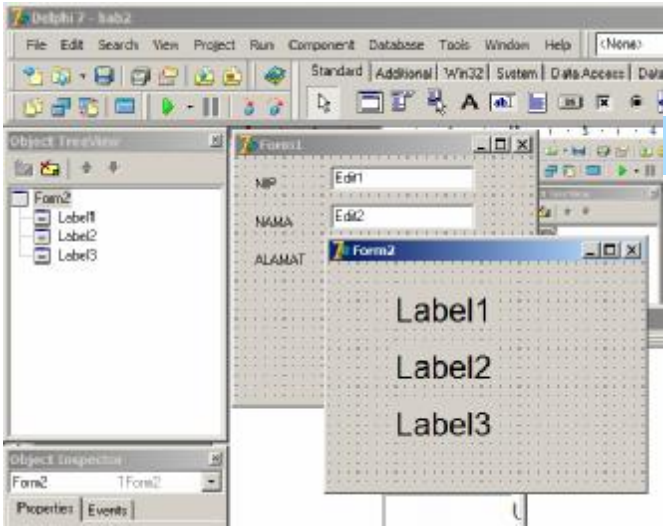


Jika berhasil, maka nama & alamat yang di-inputkan oleh user akan tampil pada message box.

Gambar 2.8: Menampilkan isi variabel ke message box

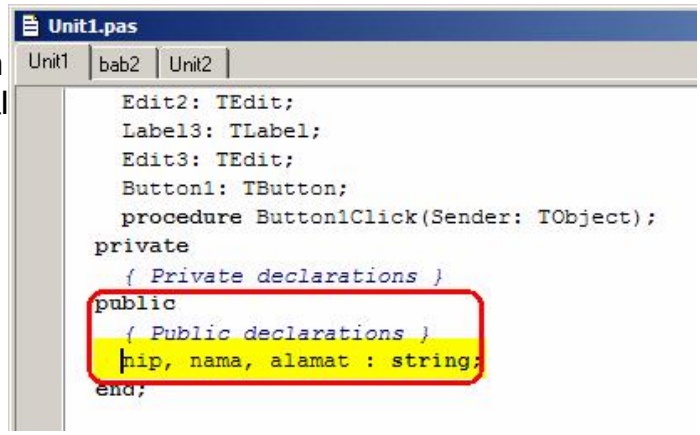
Jika telah berhasil menampilkan di *message box*, maka sekarang data akan ditampilkan di form lain. Untuk itu tambahkan sebuah form baru pada Project anda (cara lihat diatas). Kemudian tambahkan 3 buah komponen Label yang secara otomatis akan bernama Label1, Label2 & Label3. Ubahlah property Font pada Label1, Label2 & Label3 dengan ukuran font 20 serta jenis font Arial.

Gambar 2.9: Multiple form



Sekarang pindahkan variabel nip, nama, alamat yang ada di dalam procedure *Button1Click* ke bagian Public.

Gambar 2.10: Menggunakan variabel global public

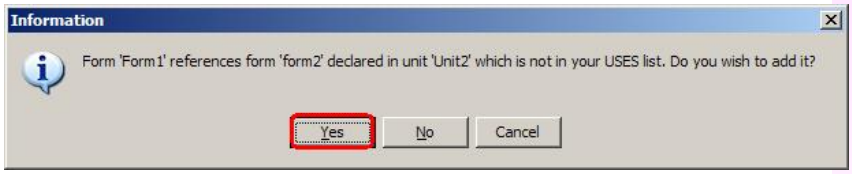


Langkah terahir adalah mengubah kode program yang ada di procedure Button1Click menjadi seperti berikut :

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  nip := edit1.Text;
  nama := edit2.Text;
  alamat := edit3.Text;

  form2.Label1.Caption := nip;
  form2.Label2.Caption := nama;
  form2.Label3.Caption := alamat;

  form2.Show;
end;
```

Jika referensi dengan Form2 belum terbentuk, maka komponen yang ada di Form2 (Label1, Label2, Label3) tidak dapat diakses dari Form1. Untuk itu tambahkan kode Uses Unit2; dibawah bagian implementation. Jika ada pertanyaan tentang pembentukan referensi, jawablah "Yes".

Gambar 2.11: Referensi dengan form 2



Setelah melakukan referensi dengan Form 2, maka jalankan (Run) project anda, isi datanya dan tekan tombol OK sehingga menghasilkan seperti gambar berikut :

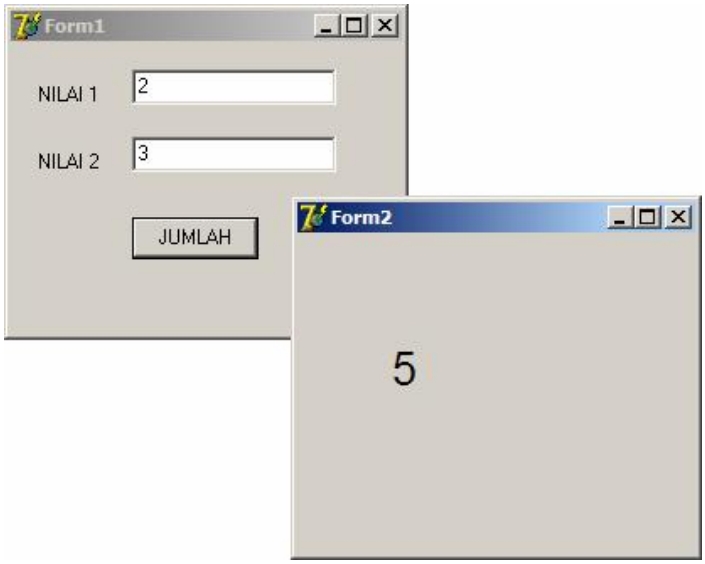
Gambar 2.12: Multiple form project

Tugas

Tugas 2.1

Buatlah sebuah program penjumlahan 2 bilangan yang hasilnya ditampilkan di form lain.

Gambar 2.12: Hasil Tugas



Ket :
Untuk dapat mengolah Inputan secara matematis, terlebih dahulu ubahlah type datanya (baca teori bab 2).

Modul 3

3

Conditional Statements

Persiapan

- Baca buku pendukung / referensi mengenai Delphi
- Pelajari tentang Conditional Statements pada Delphi
- Bawa file program pertemuan sebelumnya
- Alat tulis untuk mengerjakan laporan praktikum yang pada sampul depan tertulis Nama, NRP dan Kelas.

Pekerjaan

- Mengenal IF pada Delphi
- Mengenal IF Else pada Delphi
- Mengenal Nested IF pada Delphi
- Mengenal Case pada Delphi

Hasil

- Praktikan diharapkan mengenal dan dapat menggunakan Conditional Statements

Teori

Dalam kehidupan sehari – hari terkadang banyak kasus yang penyelesaiannya membutuhkan sebuah kondisi tertentu. Misalnya jika total pembelian lebih dari Rp. 100.000 makan mendapatkan diskon sebesar 10%. Dalam hal ini permasalahan cukup mudah dipecahkan karena hanya menggunakan sebuah kondisi. Pada kasus yang lain mungkin saja permasalahan menjadi lebih kompleks dengan kondisi lebih dari satu. Untuk menyelesaikan permasalahan semacam ini, Delphi telah menyediakan beberapa fungsi.

IF ... THEN ...

IF adalah sebuah perintah untuk menyatakan sebuah kondisi. Perintah ini akan menghasilkan nilai TRUE jika kondisi yang dimasukkan ternyata benar, dan sebaliknya akan menghasilkan FALSE jika kondisi yang dimasukkan ternyata salah.
Format penulisan IF adalah :

```
IF (kondisi) THEN (perintah)
```

Perintah akan dijalankan jika **kondisi** bernilai benar atau TRUE. Sedangkan jika **kondisi** salah, maka **perintah** tidak dijalankan dan Delphi akan menjalankan baris program berikutnya (jika ada).

Jika perintah yang diketikkan didalam kondisi IF lebih dari 1 baris, maka programmer wajib menuliskan perintah **Begin** dan **End;** pada awal & ahir kode program sebagai penanda awal perintah dan ahir perintah.

Sebagai contoh kasus pada permasalahan diatas yaitu jika total belanja lebih dari Rp. 100.000 maka mendapatkan diskon sebesar 10%. Jika total belanja ternyata kurang dari Rp. 100.000 maka tidak mendapatkan diskon. Jika diterjemahkan kedalam bahasa Delphi maka perintah nya akan menjadi seperti dibawah ini :

```
IF total >= 100000 THEN diskon := total * 0.1;
```

IF ... THEN ... ELSE ...

Terkadang adakalanya perlu untuk meng-*handle* juga jika ternyata kondisi yang terjadi adalah FALSE. Jika perintah pada bagian THEN dijalankan ketika kondisi bernilai TRUE, maka kondisi pada bagian ELSE dikerjakan ketika kondisi bernilai FALSE. Jadi dalam sekali waktu, Delphi hanya akan menjalankan salah satu dari kedua kondisi diatas.
Format penulisannya :

```
IF (kondisi) THEN
   …
ELSE
  …
```

Jika perintah yang diketikkan didalam kondisi IF atau ELSE lebih dari 1 baris, maka programmer wajib menuliskan perintah **Begin** dan **End;** pada awal & ahir kode program sebagai penanda awal perintah dan ahir perintah.

Contoh kasus nya, jika nilai lebih dari 60 maka statusnya dinyatakan LULUS dan sebaliknya jika nilai kurang dari 60 maka statusnya dinyatakan GAGAL. Jika kasus diatas diterjemahkan kedalam bahasa Delphi maka penulisannya akan menjadi seperti berikut :

```
IF nilai > 60 THEN
BEGIN
  Status := 'LULUS';
  Showmessage (Status);
END
ELSE
  Status := 'GAGAL';
```

NESTED IF

Nested IF (IF bersarang) adalah jika ada perintah IF didalam perintah IF yang lain. Kondisi ini dibutuhkan ketika kriteria yang dimasukkan lebih dari 2. Prinsip dasarnya tetap menggunakan IF ... THEN ... ELSE ... yaitu Delphi hanya akan mengerjakan salah satu perintah sesuai dengan hasil dari kondisi yang ditemukan (mengerjakan bagian THEN jika kondisi TRUE atau mengerjakan bagian ELSE jika kondisi salah). Penulisannya dalam bahasa Delphi adalah :

```
IF (kondisi1) THEN
   …
ELSE
```

```
  IF (kondisi2) THEN
     …
   ELSE
      IF (kondisi3) THEN
         …
      ELSE
         …
      END
   END
END;
```

Penjelasan :
Jika kondisi1 bernilai FALSE, maka Delphi akan menjalankan bagian ELSE. Pada bagian ELSE terdapat IF dengan kondisi2. Jika kondisi 2 bernilai FALSE, maka Delphi akan menjalankan bagian ELSE dan seterusnya. Jika kondisi2 bernilai TRUE maka Delphi akan menjalankan bagian THEN kemudian langsung keluar dari IF (END) tanpa mengerjakan bagian ELSE sehingga kondisi3 tidak pernah tersentuh.

Jika perintah yang diketikkan didalam kondisi IF lebih dari 1 baris, maka programmer wajib menuliskan perintah **Begin** dan **End;** pada awal & ahir kode program sebagai penanda awal perintah dan ahir perintah.

Contoh kasus jika nilai lebih dari 80 maka grade = A, jika nilai 60 $^{s}/_{d}$ 79 maka grade = B, jika nilai 60 $^{s}/_{d}$ 79 kurang dari 59 maka grade = C. Jika kasus diatas diterjemahkan kedalam bahasa Delphi maka penulisannya akan menjadi seperti berikut :

```
IF nilai >= 80 THEN
  grade := 'A'
ELSE
  IF nilai >= 60 THEN
    grade := 'B'
  ELSE
    grade := 'C'
  END;
END;
```

Penjelasan :
Jika nilai berisi lebih dari atau sama dengan 80 maka grade akan berisi A.
Jika nilai berisi lebih dari atau sama dengan 60, maka grade akan berisi B (karena pada IF pertama dinyatakan FALSE sehingga menjalankan bagian ELSE)
Jika nilai berisi kurang dari 60, mak agrade akan berisi C (karena pada IF pertama & IF kedua dinyatakan FALSE sehingga menjalankan bagian ELSE)

Banyaknya IF yang ada di dalam IF tidak terbatas. Yang perlu di perhatikan adalah meskipun pada Delphi proses dikerjakan berdasarkan event yang terjadi (*event driven programming*) tetapi kode program yang ada di dalam *procedure (event)* tetap dikerjakan secara *sequential* (berurutan).

CASE ... OF

Statement (perintah) CASE diciptakan untuk mempermudah statement NESTED IF diatas. Pada dasarnya cara kerja statement CASE sama dengan NESTED IF (sequential) tetapi dengan format penulisan yang berbeda sehingga lebih memudahkan programmer.

```
CASE variabel OF
  Nilai1 : …
  Nilai2 : …
  Dst
ELSE
  …
END;
```

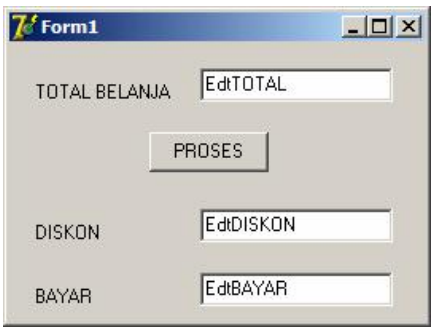
Jika contoh kasus pada NESTED IF diatas diterjemahkan kedalam CASE maka formatnya akan menjadi seperti berikut ini :

```
CASE nilai OF
  80 .. 100 : grade := 'A';
  60 .. 79 : grade := 'B';
ELSE
  Grade := 'C';
END;
```

Latihan

Untuk dapat lebih memahami tentang fungsi IF maka buatlah program berikut ini. Tujuan dari program ini adalah untuk memberikan diskon sebesar 10% dari total pembelian jika total pembelian lebih dari Rp. 100.000.

Gambar 3.1: Form Latihan 1



COMPONENT	PROPERTY	NILAI
Edit1	Name	EdtTOTAL
Edit2	Name	EdtDISKON
Edit3	Name	EdtBAYAR
Button1	Name	BtnPROSES
Button1	Caption	PROSES

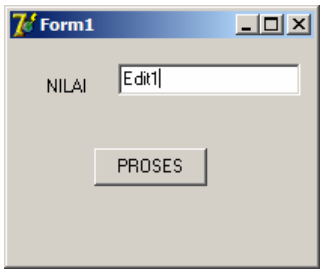
Setelah tampilan terselesaikan, maka sekarang ketikkan program berikut pada *event OnClick BtnPROSES* :

```
var
  total, diskon, bayar : real;
begin
  diskon := 0;
  total := StrToFloat(EdtTOTAL.Text);
  if total > 100000 then
    diskon := total * 0.1;
  EdtDISKON.Text := FloatToStr(diskon);
  EdtBAYAR.Text := FloatToStr(total - diskon);
end;
```

Setelah memasukkan kode diatas, maka jalankan program diatas dengan menekan tombol F9. Masukkan 200000 pada EdtTOTAL kemudian tekan tombol PROSES. Amati apa yang terjadi, kemudian ubah masukan pada EdtTOTAL menjadi 50000 lalu tekan tombol PROSES. Bandingkan hasilnya lalu amati kode program yang telah dituliskan diatas.

Jika telah memahami fungsi / perintah IF, maka sekarang buatlah project baru untuk memahami fungsi IF ... THEN ... ELSE dengan tampilan seperti berikut ini :

Gambar 3.2: Form Latihan 2



Tujuan dari program ini adalah jika nilai lebih dari 60 maka dinyatakan lulus, dan sebaliknya jika nilai kurang dari 60 maka dinyatakan tidak lulus. Untuk itu pada *event OnClick Button1* ketikkan kode program berikut ini :

```
var
  nilai : integer;
  status : string;
Begin
  nilai := StrToInt(Edit1.Text);
  IF nilai > 60 THEN
  BEGIN
    Status := 'LULUS';
    Showmessage ('ANDA ' + Status);
  END
  ELSE
  BEGIN
    Status := 'GAGAL';
    Showmessage ('ANDA ' + Status);
  END;
End;
```

Setelah memasukkan kode diatas, jalankan program tersebut dengan menekan tombol F9 kemudian masukkan sembarang nilai dan amati hasilnya. Masukkan nilai yang lain kemudian amati hasilnya kembali. Dari hasil yang ditampilkan, pelajari kembali kode program diatas.

Setelah memahami fungsi IF ... THEN ... ELSE diatas, maka ubahlah kode program diatas dengan kode program seperti dibawah ini :

```
var
  nilai : integer;
  grade : char;
begin
  nilai := StrToInt(Edit1.Text);
  IF nilai >= 80 THEN
   grade := 'A'
  ELSE
  BEGIN
   IF nilai >= 60 THEN
     grade := 'B'
   ELSE
     grade := 'C';
  END;
  ShowMessage('GRADE ANDA = ' + grade);
End;
```

Tujuan dari program diatas adalah jika nilainya lebih dari 80 maka grade = A, jika nilai 60 $^{s}/_{d}$ 79 grade = B sedangkan jika nilai dibawah 60 maka grade = C. Jalankan program diatas, masukkan sembarang nilai dan amati hasilnya. Ulangi lagi dengan memasukkan nilai yang berbeda dan amati hasilnya. Berdasarkan hasil dari program tersebut, maka pelajari kembali program yang telah diketikkan.

Setelah memahami program diatas, maka sekarang ubahlah kode program diatas dengan menggunakan bentuk CASE ... OF. Ketikkan kode program berikut ini :

```
var
  nilai : integer;
  grade : char;
Begin
  nilai := StrToInt(Edit1.Text);
  CASE nilai OF
    80 .. 100 : grade := 'A';
    60 .. 79 : grade := 'B';
  ELSE
    grade := 'C';
  END;
  ShowMessage('GRADE ANDA = ' + grade);
End;
```

Jika berhasil, maka tampilan program akan tampak seperti berikut ini :

Gambar 3.3: Form Latihan 4

Tugas

Tugas 3.1

Buatlah program untuk menghitung Take Home Pay seorang pegawai dengan kriteria sebagai berikut :

Golongan	A	B
Gaji Pokok	2.000.000	1.000.000
Insentif	Masa kerja > 5 tahun = 500.000 Masa kerja <= 5 tahun = 200.000	
Tunjangan	Anak 0 = 0 Anak 1 = 200.000 Anak 2 = 400.000 Anak >= 3 = 500.000	
Gaji Total	Gaji Pokok + Insentif + Tunjangan	
Pajak	Gaji Total >= 2.750.000 pajak 10% Gaji total >= 2.250.000 pajak 5% Gaji total < 2.250.000 pajak 0%	
Take Home Pay	Gaji Total - Pajak	

Tentukan sendiri bentuk form-nya, komponen yang digunakan serta inputan yang diperlukan sehingga menghasilkan Take Home Pay sesuai dengan kriteria.

Modul 4

4 Looping

Persiapan

- Baca buku pendukung / referensi mengenai Delphi
- Pelajari tentang Looping staements pada Delphi
- Bawa file program pertemuan sebelumnya
- Alat tulis untuk mengerjakan laporan praktikum yang pada sampul depan tertulis Nama, NRP dan Kelas.

Pekerjaan

- Mengenal FOR ... TO ... DO ... pada Delphi
- Mengenal WHILE ... DO ... pada Delphi
- Mengenal REPEAT UNTIL ... pada Delphi

Hasil

- Praktikan diharapkan dapat menggunakan Looping Statements serta menggunakannya secara bersamaan dengan Conditional Statements yang telah dipelajari pada pertemuan sebelumnya.

Teori

Dalam kondisi tertentu, terkadang perlu mengulang perintah beberapa kali untuk dapat memberikan hasil sesuai dengan yang diinginkan. Sebagai contoh kasus menghitung nilai rata – rata kelas. Untuk dapat menghitung nilai rata – rata kelas, maka nilai mahasiswa harus dimasukkan untuk kemudian di rata – rata.

Looping statements memungkinkan untuk mengulang perintah yang diketikkan sebanyak beberapa kali sampai kondisi yang diinginkan tercapai. Delphi menyediakan beberapa *Looping statements* yang masing – masing memiliki ciri khas.

FOR ... TO ... DO

Perintah perulangan ini adalah yang paling sederhana. Disini *programmer* harus mengisikan nilai awal dan nilai ahir. Format perulangan FOR ... TO ... DO :

```
FOR counter := (nilai awal) TO (nilai ahir) DO
BEGIN
   ……
END;
```

Selain format FOR ... TO ... DO ... aja juga format FOR ... DOWNTO ... DO untuk menghitung secara menurun.

Sebagai contoh kasus, jika ingin menghitung nilai 1 s/d 10, maka kode programnya adalah sebagai berikut :

```
Var
  Nilai, x, awal, akhir : integer;
Begin
  Nilai := 0;
  Awal := 1;
  Akhir := 10;
  FOR x := awal TO akhir DO
  Begin
    Nilai := Nilai + 1;
  End;
  ShowMessage (IntToStr(Nilai));
End;
```

WHILE ... DO

Pada dasarnya semua perintah perulangan bertujuan sama yaitu mengulang perintah yang diberikan sampai kondisi terpenuhi. Pada format perintah WHILE ... DO, perintah akan terus dijalankan sampai kondisi yang ada pada WHILE ... DO masih memenuhi syarat. Jika kondisinya sudah tidak memenuhi syarat, maka perulangan akan berhenti dan kemudian melanjutkan mengerjakan perintah pada baris berikutnya di luar perulangan.

Format perintah WHILE ... DO adalah :

```
WHILE (kondisi) DO
Begin
  …
  …
End;
```

Hal yang perlu di ingat adalah pada perulangan WHILE ... DO tidak ada penambahan nilai secara otomatis. Jadi *programmer* harus menambahkan nilai pada variabel sehingga dapat memenuhi kondisi yang ditentukan. Contoh kasus seperti diatas jika diubah kedalam bentuk WHILE ... DO maka akan jadi seperti berikut ini :

```
Var
  Nilai, x : integer;
Begin
  Nilai := 0;
  x := 1;
  WHILE x <= 10 DO
  Begin
    Nilai := Nilai + 1;
    inc (x);
  End;
  ShowMessage (IntToStr(Nilai));
End;
```

Penjelasan kode program :

- Variabel Nilai & x harus diberikan nilai awal agar tidak berisi nilai *random* (acak).
- Perintah **`Inc (x)`** adalah perintah untuk menambah isi dari variabel **`x`** dengan **`1`** atau sama dengan perintah **`x := x + 1`**.
- Jika perintah **`Inc (x)`** ini tidak diberikan maka isi dari variabel **`x`** akan tetap **`1`** sehingga kondisi **`x <= 10`** akan selalu terpenuhi yang berakibat Delphi akan mengulang terus perintah **`Nilai := nilai + StrToInt(Edit1.Text);`** sampai program di matikan.

REPEAT ... UNTIL

Perintah ini merupakan kebalikan dari perintah WHILE ... DO diatas. Jika pada perintah WHILE ... DO akan mengulang selama kondisi terpenuhi, maka pada REPEAT ... UNTIL justru perulangan akan berhenti ketika kondisi terpenuhi.

Format perintah REPEAT ... UNTIL adalah :

```
REPEAT
  …
  …
UNTIL (kondisi);
```

Jika program diatas diubah kedalam bentuk REPEAT ... UNTIL, maka bentuknya akan jadi seperti berikut ini :

```
Var
  Nilai, x : integer;
Begin
  Nilai := 0;
  x := 1;
  REPEAT
    Nilai := Nilai + 1;
    inc (x);
  UNTIL x > 10;
  ShowMessage (IntToStr(Nilai));
End;
```

Agar lebih efektif, perulangan juga dapat digabungkan dengan IF / CASE. Misalnya pada contoh kasus mencari bilangan genap pada deret bilangan 1 s/d 10.

Latihan

Latihan 4.1

Untuk dapat lebih memahami tentang perulangan, maka buatlah program dengan tampilan seperti gambar dibawah ini :

Gambar 4.1: Form Latihan 1

Setelah membuat tampilan di form, sekarang ketikkan kode program berikut kedalam event Button1Click :

```
Var
  maks, bil : integer;
Begin
  maks := StrToInt(Edit1.Text);
  FOR bil := 1 TO maks DO
  Begin
    IF bil MOD 2 = 0 THEN
    Begin
      Memo1.Lines.add (IntToStr(bil));
    End;
  End;
End;
```

Penjelasan kode program :

- Variabel **maks** diisi sebuah nilai yang didapat dari **Edit1.Text**. Karena perbedaan type data maka harus dikonversi terlebih dahulu.
- Pada putaran pertama variabel bil akan berisi 1 kemudian di cek pada kondisi apakah sisa bagi (MOD) dari nilai yang ada di dalam variabel bil adalah 0. Jika benar maka nilai tersebut akan ditampilkan di Memo1.
- Kemudian bil akan bernilai 2 secara otomatis dan kembali melakukan proses diatas.

Latihan 4.2

Jika telah memahami cara kerja FOR ... TO ... DO, maka sekarang ubahlah kode program diatas kedalam bentuk WHILE ... DO.

```
Var
  maks, bil : integer;
```

```
Begin
  maks := StrToInt(Edit1.Text);
  bil := 1;
  WHILE bil <= maks DO
  Begin
    IF Bil MOD 2 = 0 THEN
      Memo1.Lines.add (IntToStr(bil));
    inc(bil);
  End;
End;
```

Penjelasan kode program :

- Variabel **maks** diisi sebuah nilai yang didapat dari **Edit1.Text**. Karena perbedaan type data maka harus dikonversi terlebih dahulu.
- Variabel **bil** diberikan nilai default 1, jika tidak diberikan maka variabel **bil** akan berisi nilai acak.
- Selama isi dari variabel **bil <= maks** maka dilakukan pengecekan apakah nilai yang ada di dalam variabel **bil** memiliki sisa bagi (MOD). Jika tidak maka nilai tersebut akan ditampilkan di **Memo1**.
- Kemudian variabel **bil** ditambahkan dengan 1 agar dapat memenuhi kondisi diatas.

Latihan 4.3

Setelah dapat memahami cara kerja WHILE ... DO maka ubahlah kode program diatas kedalam bentuk REPEAT ... UNTIL seperti berikut ini :

```
Var
  maks, bil : integer;
Begin
  maks := StrToInt(Edit1.Text);
  bil := 1;
  REPEAT
    IF Bil MOD 2 = 0 THEN
        Memo1.Lines.add (IntToStr(bil));
    inc(bil);
  UNTIL bil > 10;
End;
```

Penjelasan kode program :

- Jika pada WHILE ... DO, kode program yang ada didalamnya selama kondisi yang ditentukan benar, maka pada REPEAT ... UNTIL merupakan kebalikannya. Selama kondisi yang diinginkan belum tercapai (FALSE) maka kode program yang ada didalamnya akan dikerjakan sampai kondisi yang diingikan tercapai (TRUE).

Tugas

Tugas 4.1

Buatlah 3 buah program untuk menghitung faktorial dari sebuah bilangan dengan ketiga cara perulangan diatas.

Modul 5

5 Array

Persiapan

- Baca buku pendukung / referensi mengenai Delphi
- Pelajari tentang variabel Array pada Delphi
- Bawa file program pertemuan sebelumnya
- Alat tulis untuk mengerjakan laporan praktikum yang pada sampul depan tertulis Nama, NRP dan Kelas.

Pekerjaan

- Mengenal variabel Array pada Delphi

Hasil

- Praktikan diharapkan dapat menggunakan variable Array untuk memudahkan dalam memasukkan data

Teori

Array adalah sekumpulan variabel yang memiliki nama dan type data yang sama. Karena semua variabel Array memiliki nama serta type data yang sama, maka untuk membedakan antara array satu dengan yang lainnya yaitu dengan menggunakan nomor (index). Dan karena array memiliki index yang unik (tidak boleh ada nomor index yang sama) maka array dapat berisi data yang sama.

Aturan penamaan variabel array serta type data yang digunakan sama dengan variabel pada bab 1 modul ini. Hanya saja yang membedakan adalah variabel array memiliki nomor (index) yang tidak boleh kembar. Pendeklarasian variable array adalah :

```
Var
  X : array [1..10] of integer;
```

Jadi artinya sekarang variabel X berjumlah 10 (**X[1], X[2], X[3] … X[10]**) yang kesemuanya bertype integer. Begitu juga untuk menggunakan variabel array ini, *programmer* harus menyebutkan indexnya misal :

```
X[1] := 10;
```

ARRAY 1 DIMENSI

Variabel array 1 dimensi adalah variabel array yang memiliki 1 buah nomor index. Batas nomor index maksimal array adalah dari 0 sampai dengan 255. Contoh array 1 dimensi :

```
Var
  Harga : array [1 .. 10] of real;
Begin
  Harga[1] := 10000;
  Harga[2] := 25000;
  Harga[3] := 47500;
End;
```

ARRAY MULTI DIMENSI

Array multi dimensi adalah array yang memiliki nomor index lebih dari 1. Contoh array 2 dimensi :

```
Var
  Matrix : array [1 .. 10, 1..10] of Integer;
Begin
  Matrix [1,1] := 1;
  Matrix [1,2] := 2;
  Matrix [2,1] := 3;
End;
```

ARRAY STATIS

Yang disebut dengan *array statis* adalah *array* yang panjangnya (jumlahnya) telah ditentukan di awal oleh programmer. Panjang (jumlah) dari array ini sekali ditetapkan tidak dapat diubah lagi (*fixed*). Contoh array statis :

```
Var
  nilai : array [1..20] of integer;
```

ARRAY DINAMIS

Array dinamis adalah variabel array yang panjangnya (jumlahnya) belum ditentukan ketika variabel tersebut di-deklarasi-kan. Untuk menentukan panjanganya, digunakan perintah **`SetLength(nama_array, panjangnya);`** . Contoh array dinamis :

```
Var
   Nama_mhs : array of string;
Begin
  SetLength (nama_mhs, 10);
  Nama_mhs[1] := 'Bunali';
  Nama_mhs[2] := 'Wonokairun'
  Nama_mhs[3] := 'Mat Pithi';
End;
```

Keunggulan variabel array dengan variabel biasa adalah pada nomor index-nya. Karena variabel array memiliki nama & type data yang sama, maka nomor index dapat dimanfatkan untuk digunakan pada perulangan sehingga lebih menghemat penggunaan variabel.

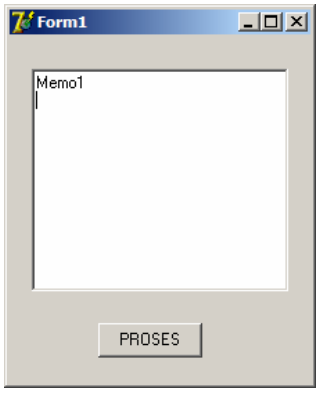
Misal untuk memasukkan data 20 mahasiswa, maka *programmer* hanya perlu menggunakan variabel array sebanyak 20 buah.

```
Var
  Nama_mhs : array [1 .. 20] of string;
  x := byte;
Begin
  FOR x := 1 TO 20 DO
  Begin
    Nama_mhs[x] := Edit1.Text;
  End;
End;
```

Latihan

Latihan 1

Untuk lebih memahami tentang variabel array maka buatlah tampilan program seperti berikut ini :



Gambar 5.1: Form Latihan 1

Tujuan dari program ini adalah menampilkan angka 1 s/d 20 pada Memo1 dengan menggunakan variabel array. Untuk itu ketikkan kode program berikut ini :

```
var
  angka : array [1..20] of integer;
  x : integer;
Begin
  FOR x := 1 TO 20 DO
  Begin
    angka[x] := x;
    memo1.Lines.Add(IntToStr(angka[x]));
  End;
End;
```

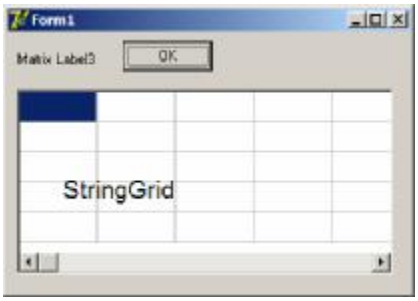
Penjelasan kode program :

- Variabel angka di diatas dijadikan variabel array mulai dari 1 s/d 20. Jadi sekarang terdapat 20 buah variabel dengan nama angka (angka[1], angka[2] ... angka[20])
- Untuk menyingkat kode program maka digunakan perulangan untuk mengisi angka[1] s/d angka [20] dengan nilai dari variabel x
- Setelah setiap kali variabel terisi, maka variabel tersebut akan ditampilkan di Memo1. Karena perbedaan type data maka harus dilakukan konversi.

Latihan 2

Agar lebih memahami tentang array multidimensi, buatlah program dengan tampilan sebagai berikut :

Gambar 5.2: Form Latihan 2



Komponen	Property	Nilai
Label1	Caption	'Matrix'
Button1	Caption	'OK'
StringGrid	FixedCols	0
StringGrid	FixedRows	0

Tujuan dari program ini adalah menampilkan variabel array multi dimensi di String Grid sekaligus menggunakan index array sebagai koordinat.

Jika sudah merancang tampilan seperti diatas maka ketikkan kode program berikut ini kedalam Button1Click :

```
Var
  matrix : array [1..3,1..3] of integer;
  X,Y : integer;
Begin
  FOR X := 1 TO 3 DO
  Begin
    FOR Y := 1 TO 3 DO
    Begin
      Label3.Caption := IntToStr(X) + ', ' + IntToStr(Y);
      matrix[X,Y] := strtoint(inputbox('INPUT NILAI','MASUKKAN NILAI','0'));
    End;
  End;

  FOR X := 1 TO 3 DO
  Begin
    FOR Y := 1 TO 3 DO
    Begin
      StringGrid1.Cells[x,y] := IntToStr(matrix[x,y]);
    End;
  End;
End;
```

Penjelasan kode program :

- Variabel matrix merupakan variabel array 2 dimensi artinya mempunyai 2 nomor index.
- Program pertama memasukkan nilai kedalam matrix. Karena matrix memiliki 2 nomor index maka harus menggunakan 2 perulangan.
- Setelah semua nilai dimasukkan kedalam variabel, maka sekarang tampilkan nilai – nilai tersebut pada String Grid dimana index matrix digunakan sebagai koordinat kolom & baris sehingga dapat merepresentasikan bentuk penyimpanan variabel array multi dimensi.

Tugas

Tugas 4.1

Buatlah sebuah program untuk menjumlahkan 2 matrix (matrix 2x2)

Modul 6

6 Record

Persiapan

- Baca buku pendukung / referensi mengenai Delphi
- Pelajari tentang Record pada Delphi
- Bawa file program pertemuan sebelumnya
- Alat tulis untuk mengerjakan laporan praktikum yang pada sampul depan tertulis Nama, NRP dan Kelas.

Pekerjaan

- Mengenal Record pada Delphi

Hasil

- Praktikan diharapkan dapat menggunakan Record untuk memudahkan dalam memasukkan data

Teori

RECORD

Pada dasarnya record hampir mirip dengan array. Yang membedakan hanyalah jika pada array hanya bisa memiliki 1 type data, tetapi pada record masing – masing field dapat memiliki type data yang berbeda – beda. Selain itu pada record sudah tidak menggunakan nomor index seperti pada array, melainkan sudah menggunakan nama yang disebut dengan field. Maka untuk menggunakannya selain menyebutkan nama recordnya juga harus menyebutkan nama field nya.

Aturan penulisan record adalah sebagai berikut :

```
Type
  Nama_rec = record
    Field1 : type_data1;
    Field2 : type_data2;
  End;
Var
  Variabel : nama_rec;
Begin
  Variabel.Field1 := ... ;
End;
```

Karena masing – masing field memiliki type data yang berbeda, maka data yang masuk juga bisa bermacam – macam sesuai dengan kebutuhan.

WITH ... DO

Untuk mempercepat kerja *programmer*, maka Delphi menyediakan fungsi WITH ... DO. Dengan menggunakan fungsi ini maka *programmer* tidak perlu lagi menuliskan nama variabel record yang cukup banyak. Jika diimplementasikan pada kode program diatas maka akan jadi seperti berikut :

```
Type
  Nama_rec = record
    Field1 : type_data1;
    Field2 : type_data2;
  End;
Var
  Variabel : nama_rec;
Begin
  WITH variabel DO
  Begin
    Field1 := ... ;
    Field2 := ... ;
  End;
End;
```

RECORD + ARRAY

Untuk memudahkan programmer, record juga dapat digabungkan dengan array sehingga mendapatkan keuntungan dari kedua bentuk variabel tersebut. Dari record programmer dapat mengambil keuntungan karena type data pada masing – masing field tidak sama. Sedangakan dari array, programmer dapat mengambil keuntungan dari penggunaan nomor index yang tentunya juga dapat dikombinasikan dengan perintah perulangan.

```
Type
  Mhs : record
    Nama : string [20];
    UTS : integer;
    UAS : integer;
  End;
Var
  Mahasiswa : array [1..10] of Mhs;
Begin
  Mahasiswa[1].Nama := 'Wonokairun';
  Mahasiswa[1].UTS := 60;
  Mahasiswa[1].UAS := 80;
  Mahasiswa[2].Nama := 'Mat Pithi';
  Mahasiswa[2].UTS := 75;
  Mahasiswa[2].UAS := 65;
  Dst …
```

Latihan

Latihan 6.1

Agar dapat memahami penggunaan record, buatlah program dengan tampilan seperti berikut ini :

Gambar 6.1: Form Latihan 1

Komponen	Property	Nilai
Edit1	Name	EdtNAMA
Edit2	Name	EdtUTS
Edit3	Name	EdtUAS
Button1	Name	BtnPROSES
Button1	Caption	PROSES

Program ini bertujuan untuk menampilkan nama mahasiswa beserta nilai UTS dan UAS nya yang nantinya akan diolah menjadi Nilai Ahir dengan kriteria Nilai Ahir = 40% UTS + 60% UAS. Untuk itu ketikkan kode program berikut pada BtnPROSESClick :

```
type
  mhs = record
    nama : string;
    uts, uas : integer;
  end;
var
  mahasiswa : mhs;
  nilai_ahir : real;
begin

  mahasiswa.nama := EdtNAMA.Text;
  mahasiswa.uts := StrToInt(EdtUTS.Text);
  mahasiswa.uas := StrToInt(EdtUAS.Text);
  nilai_ahir := (mahasiswa.uts * 0.4) + (mahasiswa.uas * 0.6);

  Memo1.Lines.Add('NAMA = ' + mahasiswa.nama);
  memo1.Lines.Add('UTS = ' + IntToStr(mahasiswa.uts));
  memo1.Lines.Add('UAS = ' + IntToStr(mahasiswa.uas));
  memo1.Lines.Add('NILAI = ' + FloatToStr(nilai_ahir));
end;
```

Latihan 6.2

Setelah memahami cara kerja record, maka sekarang ubahkan kode program diatas dengan menambahkan array.

```
type
  mhs = record
    nama : string;
    uts, uas : integer;
  end;
var
```

```
  mahasiswa : array [1..3] of mhs;
  nilai_ahir : real;
  x : integer;
begin
  FOR x := 1 TO 3 DO
  Begin
    mahasiswa[x].nama := InputBox('RECORD','MASUKKAN NAMA MAHASISWA','');
    mahasiswa[x].uts := StrToInt(InputBox('RECORD','NILAI UTS',''));
    mahasiswa[x].uas := StrToInt(InputBox('RECORD','NILAI UAS',''));
    nilai_ahir := (mahasiswa[x].uts * 0.4) + (mahasiswa[x].uas * 0.6);
  End;

  FOR x := 1 TO 3 DO
  Begin
    Memo1.Lines.Add('NAMA = ' + mahasiswa[x].nama);
    memo1.Lines.Add('UTS = ' + IntToStr(mahasiswa[x].uts));
    memo1.Lines.Add('UAS = ' + IntToStr(mahasiswa[x].uas));
    memo1.Lines.Add('NILAI = ' + FloatToStr(nilai_ahir));
    memo1.Lines.add('============================');
  End;
End;
```

Tugas

Tugas 6.1

Buatlah sebuah program kasir untuk menghitung total belanja dari 5 buah barang yang dibeli. Jika total belanja lebih dari Rp. 50.000 maka berikan diskon sebesar 10% dari total belanja. Gunakan record & array.

Modul 7

7

Procedure, Function& Unit

Persiapan

- Baca buku pendukung / referensi mengenai Delphi
- Pelajari tentang Procedure pada Delphi
- Pelajari tentang Function pada Delphi
- Bawa file program pertemuan sebelumnya
- Alat tulis untuk mengerjakan laporan praktikum yang pada sampul depan tertulis Nama, NRP dan Kelas.

Pekerjaan

- Mengenal Procedure pada Delphi
- Mengenal function pada Delphi

Hasil

- Praktikan diharapkan dapat menggunakan mengunakan procedure / function sesuai dengan kebutuhan program.

Teori

Ketika membuat sebuah program yang kompleks dengan kode program yang cukup banyak, keberadaan PROCEDURE / FUNCTION sangat membantu. Dengan memecah kode program kedalam PROCEDURE / FUNCTION maka proses pembuatan program akan jauh lebih mudah, karena kode program yang sering digunakan hanya cukup dituliskan sekali saja untuk kemudian dapat digunakan berkali – kali. Disamping itu penggunaan PROCEDURE / FUNCTION akan mempermudah programmer jika nantinya terjadi perbaikan program karena perbaikan hanya dilakukan pada tempat – tempat tertentu.

PROCEDURE (tanpa parameter)

Pada pertemuan sebelumya, telah disinggung bahwa ketika didefiniskan sebuah *event*, maka Delphi secara otomatis akan membuatkan sebuah PROCEDURE. Hal ini bertujuan untuk mengelompokkan kode program yang akan dikerjakan ketika *event* tersebut terjadi. Karena hal itulah satu komponen dapat memiliki beberapa *event* dan sebaliknya satu *event* dapat dimiliki oleh beberapa komponen. Ciri khas dari PROCEDURE adalah tidak memberikan nilai balik atau dapat dikatakan dengan perintah satu arah.

Format untuk membuat sebuah PROCEDURE adalah :

```
PROCEDURE nama_prosedur;
Begin
  ...
  ...
End;
```

Ketika dibutuhkan, maka programmer tinggal memanggil nama_prosedur maka Delphi akan menjalankan kode program yang ada di dalam PROCEDURE tersebut.

PROCEDURE (dengan parameter)

Parameter disini adalah variabel global yang berguna untuk menampung data dari kode program yang lain. Format penulisan variabel pada procedure atau yang disebut juga dengan parameter tidak berbeda dengan variabel global pada umumnya. Selain parameter, di dalam PROCEDURE juga dapat didefinisikan variabel lokal.

```
PROCEDURE nama_perosedur (variabel_global : type_data);
Var
  Variabel_lokal : type_data;
Begin
  …
  …
End;
```

FUNCTION (tanpa parameter)

FUNCTION hampir sama PROCEDURE hanya saja FUNCTION memberikan nilai kembalian / hasil sedangkan PROCEDURE tidak. Perbedaan utamanya terletak pada **nama_function** yang juga berfungsi sebagai variabel. Format penulisan FUNCTION tanpa parameter adalah sebagai berikut :

```
FUNCTION nama_function : type_data;
Begin
  …
  Nama_function := hasil;
End;
```

Contoh kasus membuat sebuah fungsi perkalian maka kode programnya akan tampak seperti berikut ini :

```
FUNCTION kali : integer;
Begin
  Kali := 2 * 3;
End;
```

FUNCTION (dengan parameter)

FUNCTION dengan menggunakan parameter tidak berbeda dengan PROCEDURE yang menggunakan paramater. Parameter adalah variabel penampung data sebagai inputan untuk proses yang ada didalam FUNCTION tersebut.

Contoh kasus membuat fungsi untuk menghitung diskon akan tampak seperti berikut ini :

```
function diskon (total : real) : real;
begin
  diskon := total * 0.1;
end;
```

Sedangkan untuk memanggil dari event *Button1Click* kode program seperti berikut :

```
procedure TForm1.Button1Click(Sender: TObject);
var
  diskonnya, total : real;
begin
  total := StrToFloat(edit1.Text);
  diskonnya := diskon(total);
  ShowMessage(FloatToStr(diskonnya));
end;
```

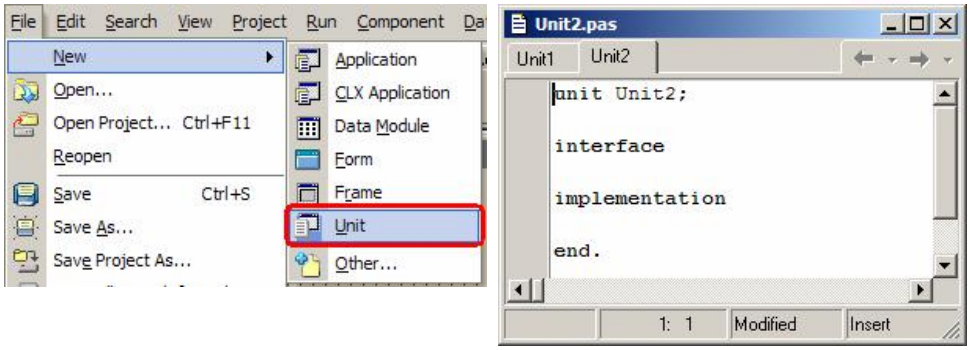
UNIT

Pada dasarnya UNIT adalah sebuah file yang berisikan PROCEDURE / FUNCTION yang dapat digunakan oleh program secara eksternal (*add on*). Secara *default* Delphi akan membuat unit untuk menyimpan semua kode program yang ada di *Code Editor* (Unit1.pas). Unit tersebut akan terhubung dengan file yang menyimpan informasi Form (Unit1.dfm).

Agar PROCEDURE / FUNCTION yang dibuat dapat digunakan pada project yang berbeda tanpa harus mengetikkan ulang, maka letakkan PROCEDURE / FUNCTION tersebut kedalam sebuah file unit.

Cara yang paling mudah untuk membuat sebuah unit adalah dengan melalui menu File **à** New **à** Unit.

Gambar 7.1: New Unit



Interface

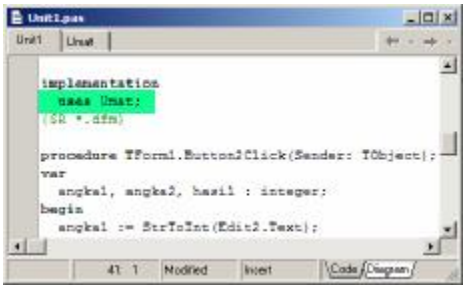
Bagian ini berisikan *header* dari PROCEDURE / FUNCTION yang akan dituliskan dibawahnya. Bagian inilah yang menghubungkan PROCEDURE / FUNCTION yang ada dengan project yang menggunakannya (bersifat global) sehingga PROCEDURE / FUNCTION yang ada dalam unit ini dapat diakses oleh project yang menggunakannya.

Implementation

Pada bagian ini programmer mendefiniskan PROCEDURE / FUNCTION secara lengkap. Jadi jika pada bagian interface hanya mendeklarasikan *header* nya saja, pada bagian implementation ini berisikan PROCEDURE / FUNCTION secara utuh.

Untuk dapat menggunakan unit ini, programmer harus menghubungkan terlebih dahulu project yang akan menggunakan dengan unit yang akan digunakan. Caranya adalah dengan mengetikkan nama unit yang akan digunakan pada project yang menggunakannya (pada bagian **`implementation`**).

Gambar 7.2: Using Unit



Latihan

Latihan 1

Untuk lebih memahami tentang PROCEDURE (tanpa parameter), tambahkan kode program berikut kedalam project anda :

```
PROCEDURE pesan;
Begin
  ShowMessage('TEST PROCEDURE');
End;
```

Ingat karena Delphi mengerjakan perintah secara sequential dari atas ke bawah maka disarankan untuk menuliskan PROCEDURE yang dibuat setelah bagian Implementation atau sebelum PROCEDURE milik Delphi. Kemudian coba panggil PROCEDURE diatas dari Button1Click dengan cara mengetikkan nama PROCEDURE-nya (pesan).

Latihan 2

Setelah memahami cara pembuatan PROCEDURE diatas, maka sekarang buatlah sebuah program untuk menghitung diskon. Utuk itu buatlah tampilan program agar seperti berikut ini :

Gambar 7.3: Latihan 2

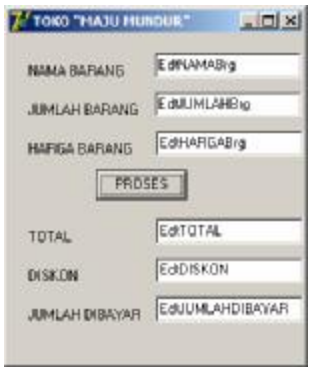
Jika tampilan program telah seperti gambar diatas, maka sekarang ketikkan kode program berikut ini :

```
PROCEDURE kuadrat (nilai : integer);
var
  hasil : integer;
Begin
  hasil := nilai * nilai;
  ShowMessage(IntToStr(hasil));
End;

procedure TForm1.Button1Click(Sender: TObject);
var
  angka : integer;
begin
  angka := StrToInt(edit1.Text);
  kuadrat(angka);
end;
```

Latihan 3

Untuk dapat memahami tentang FUNCTION, buatlah program dengan tampilan seperti gambar dibawah ini :

Gambar 7.4: Latihan 3



KOMPONEN	PROPERTY	NILAI
Edit1	Name	EdtNAMABrg
Edit2	Name	EdtJUMLAHBrg
Edit3	Name	EdtHARGABrg
Button1	Name	BtnPROSES
Button1	Caption	PROSES
Edit4	Name	EdtTOTAL
Edit5	Name	EdtDiskon
Edit6	Name	EdtJUMLAHDIBAYAR

Tujuan dari program ini adalah menghitung total belanja. Jika total belanja > 100.000 akan mendapatkan diskon sebesar 10% dari total belanja.

Setelah tampilan terselesaikan sesuai dengan gambar diatas, maka tuliskan kode program berikut ini :

```
FUNCTION total (jml : integer; harga : real) : real;
Begin
  total := jml * harga;
End;

FUNCTION diskon (total : real) : real;
Begin
  IF total > 100000 THEN
    diskon := total * 0.1;
End;

FUNCTION jumlah (ttl, dsk : real) : real;
Begin
  jumlah := ttl - dsk;
End;

procedure TForm1.BtnPROSESClick(Sender: TObject);
var
  harga, total_belanja, diskon_belanja, jumlah_dibayar : real;
  jumlah_barang : integer;
begin
  jumlah_barang := StrToInt(EdtJUMLAHBrg.Text);
  harga := StrToFloat(EdtHARGABrg.Text);
  total_belanja := total(jumlah_barang,harga);
  diskon_belanja := diskon(total_belanja);
  jumlah_dibayar := jumlah(total_belanja,diskon_belanja);

  EdtTOTAL.Text := FloatToStr(total_belanja);
  EdtDISKON.Text := FloatToStr(diskon_belanja);
  EdtJUMLAHDIBAYAR.Text := FloatToStr(jumlah_dibayar);
end;
```

Jalankan program diatas kemudian amati hasilnya. Setelah paham dengan cara penggunaan FUNCTION, maka sekarang pindahkan FUNCTION – FUNCTION tersebut kedalam Unit tersendiri (FUNCTIONnya saja). Maka dalam unit baru tersebut akan memiliki kode program seperti berikut ini :

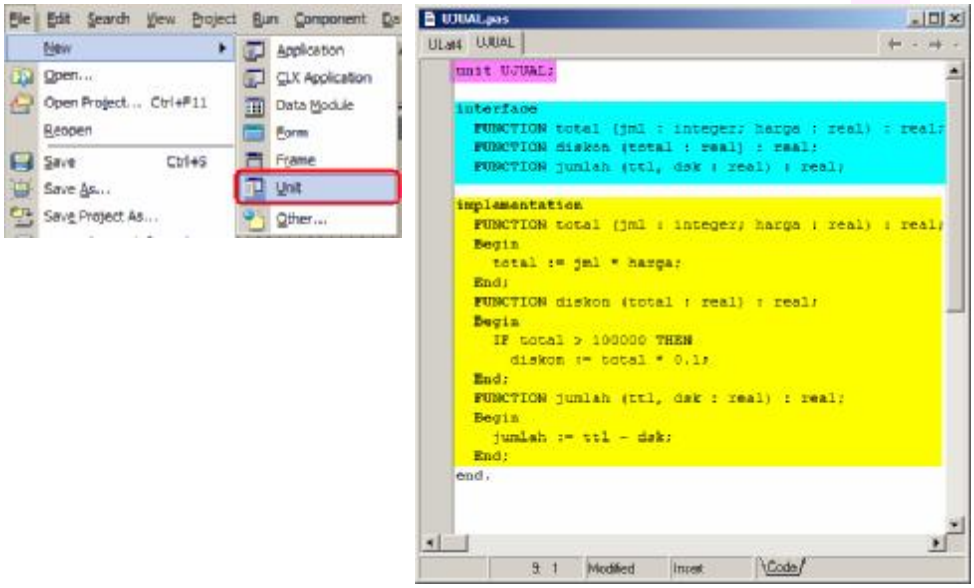
```
unit UJUAL;

interface
  FUNCTION total (jml : integer; harga : real) : real;
  FUNCTION diskon (total : real) : real;
  FUNCTION jumlah (ttl, dsk : real) : real;
```

```
implementation
  FUNCTION total (jml : integer; harga : real) : real;
  Begin
    total := jml * harga;
  End;
  FUNCTION diskon (total : real) : real;
  Begin
    IF total > 100000 THEN
      diskon := total * 0.1;
  End;
  FUNCTION jumlah (ttl, dsk : real) : real;
  Begin
    jumlah := ttl - dsk;
  End;
end.
```

Jangan lupa untuk memberikan referensi unit yang digunakan di project yang menggunakan

```
implementation
uses ujual;
```

Gambar 7.5: Unit UJUAL



Tugas

Tugas 7.1

Buatlah sebuah program kalkulator sederhana yang dapat melakukan operasi dasar matematika (penjumlahan, pengurangan, perkalian & pembagian). Tetapi rumus – rumus untuk operasi tersebut diletakkan di dalam Unit tersendiri (gunakan FUNCTION).

Modul 8

8 Exception

Persiapan

- Baca buku pendukung / referensi mengenai Delphi
- Pelajari tentang Exception pada Delphi
- Bawa file program pertemuan sebelumnya
- Alat tulis untuk mengerjakan laporan praktikum yang pada sampul depan tertulis Nama, NRP dan Kelas.

Pekerjaan

- Mengenal Exception pada Delphi

Hasil

- Praktikan diharapkan menegrti dan dapat menggunakan mengunakan *exception* sebagai *exception handler*.

Teori

Salah satu kelebihan yang dimiliki oleh Delphi adalah kemam-puannya untuk menangani *exception*. *Exception* adalah sebuah istilah pemrograman yang mengacu pada perkecualian yang diakibatkan kesalahan pada waktu menjalankan program dan tidak dapat diketahui atau didefinisikan pada waktu kompilasi.

Exception sendiri dapat diatasi dengan suatu cara yang disebut dengan *resource protection*, yaitu suatu metode untuk melindungi sumber daya program. Lebih jauh lagi cara tersebut dinamakan dengan *exception handling* (penanganan pengecualian).

Ada dua macam *resource protection* dalam Delphi, yaitu dengan blok *try-except* dan blok *tr-.finally*.

Try-Except

Blok *try-except* digunakan untuk menjalankan suatu blok per-nyataan dan mencegah Delphi agar tidak menampilkan pe-san kesalahannya. Sebagai gantinya, pemrogram dapat menam-pilkan pesan kesalahannya sendiri.

Bila tidak terjadi *exception*, maka semua baris pada bagian *try* akan dijalankan, namun bagian *except* tidak akan dijalankan. Bila terjadi *exception*, maka baris setelah

baris di mana terjadi *exception* tadi (pada bagian *try*) tidak akan dijalankan, namun proses eksekusi program dilanjutkan ke bagian *except*.

Perhatikan blok program berikut ini:

```
try
  X := Y / Z;
except
  on EZeroDivide do
    MessageDlg('Terjadi pembagian dengan nol.', mtError, [mbOK], 0);
end;
```

Pada contoh program tersebut, bagian *Try* akan mencoba dieksekusi. Di sini ada kemungkinan terjadi *exception* bila Z bernilai nol, karena akan mengakibatkan pembagian dengan nol. Bila nilai Z tidak sama dengan nol karenanya exception, maka bagian except tidak akan dijalankan. Bila nilai Z bernilai nol, maka bagian except akan dijalankan.

Try..finally

Blok try-finally digunakan untuk menjalankan suatu blok pernyataan dan selalu menjalankan bagian finally apapun yang terjadi (terjadi exception ataupun tidak). Hal ini biasanya berguna untuk memberikan finalisasi (pemberian nilai akhir, atau yang harus selalu dikerjakan terakhir kali).

Blok ini sebenarnya tidak menangani exception, hanya mena-ngani alur program agar apapun yang terjadi, bagian finally sela-lu dikerjakan.

Perhatikan blok program berikut ini:

```
try
  A := B / C;
  D := B + C;
finally
  C := 5;
end;
```

Pernyataan A := B / C; akan dieksekusi. Bila terjadi exception (bila C bernilai nol) maka D := B + C; tidak akan dikerjakan. Sebaliknya bila tidak terjadi *exception* (bila C tidak bernilai nol) maka pernyataan D := B + C; akan dikerjakan. Namun, terjadi atau tidak exception tersebut, pernyataan C := 5; tetap akan di-eksekusi.

Untuk menangani *exception* sepenuhnya yang digabung dengan *Try – Finally* dapat dilakukan dengan meletakkan blok *Try – Finally* di dalam bagian *Try* dari *Try – Except* .

Sebagai contoh:

```
try
  try
    A := B / C;
    D := B + C;
  finally
    C := 5;
  end;
except
  on EZeroDivide do
    MessageDlg('Terjadi pembagian dengan nol.', mtError, [mbOK], 0);
end;
```

Pada contoh di atas, apapun yang terjadi, baris C := 5; tetap akan dijalankan. Tetapi bila terjadi exception, bagian except akan dijalankan. Bagian except akan dijalankan setelah menjalankan bagian finally.

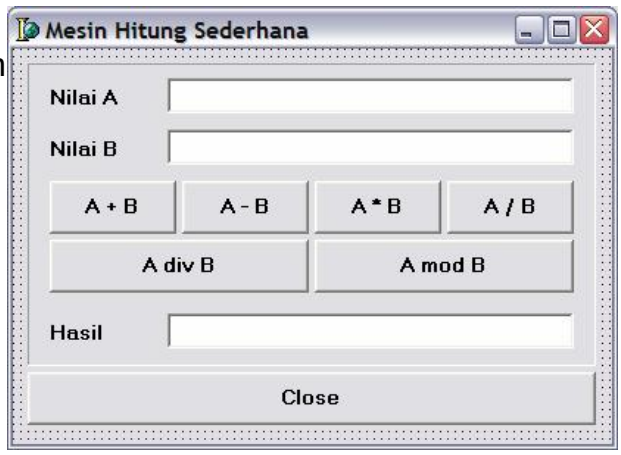
Perlu diketahui, agar kedua penanganan exception tadi be-kerja semestinya, maka perlu menonaktifkan pilihan "Stop on Delphi Exceptions" dengan cara:

- Masuklah ke menu Tools|Debugger Options.
- Pilihlah tab Language Exceptions.
- Pastikan pilihan combo box "Stop on Delphi Exceptions" dinonaktifkan.

Latihan

Latihan 8.1

Agar lebih jelas, akan dicoba membuat sebuah aplikasi yang menerapkan penggunaan keduanya. Langkah pertama, rancanglah sebuah form seperti pada gambar 8.1.



Gambar 8.1: Tampilan form Kalkulator.

Ubahlah nama dari komponen TEdit yang pertama menjadi EditNilaiA, TEdit yang kedua menjadi EditNilaiB, dan TEdit yang ketiga menjadi EditHasil. Kemudian nama komponen TButton mulai dari yang pertama (yang teratas, mulai dari kiri ke kanan) diubah masing-masing menjadi ButtonPlus, ButtonMinus, ButtonMultiply, dan ButtonDivide. Sedangkan

kedua TButton berikutnya (yang berada di bawahnya) diubah namanya masing-masing menjadi ButtonDiv dan ButtonMod. Terakhir, ubahlah nama komponen TButton terbawah menjadi ButtonClose. Kemudian ubahlah properti ReadOnly pada EditHasil men-jadi True.

Buatlah event-handler untuk masing-masing event OnClick pada keenam TButton pertama masing – masing sesuai dengan *Caption*-nya, yaitu untuk menjumlah, mengurangi, mengalikan, membagi, melakukan pembagian bulat, dan mengetahui sisa hasil bagi bilangan bulat. Masing-masing akan memiliki event-handler seperti di bawah ini:

```
procedure TForm1.ButtonPlusClick(Sender: TObject);
var
  NilaiA, NilaiB, Hasil: Real;
begin
  try
    NilaiA := StrToFloat(EditNilaiA.Text);
    NilaiB := StrToFloat(EditNilaiB.Text);
    Hasil := NilaiA + NilaiB;
    EditHasil.Text := FloatToStr(Hasil);
  except
    on EConvertError do
      EditHasil.Text := 'A atau B salah!';
    on EOverflow do
      EditHasil.Text := 'Overflow!';
    on EUnderflow do
      EditHasil.Text := 'Underflow!';
  end;
end;

procedure TForm1.ButtonMinusClick(Sender: TObject);
var
  NilaiA, NilaiB, Hasil: Real;
begin
  try
    NilaiA := StrToFloat(EditNilaiA.Text);
    NilaiB := StrToFloat(EditNilaiB.Text);
    Hasil := NilaiA - NilaiB;
    EditHasil.Text := FloatToStr(Hasil);
  except
    on EConvertError do
      EditHasil.Text := 'A atau B salah!';
    on EOverflow do
      EditHasil.Text := 'Overflow!';
    on EUnderflow do
      EditHasil.Text := 'Underflow!';
  end;
end;

procedure TForm1.ButtonMultiplyClick(Sender: TObject);
var
```

```
  NilaiA, NilaiB, Hasil: Real;
begin
  try
    NilaiA := StrToFloat(EditNilaiA.Text);
    NilaiB := StrToFloat(EditNilaiB.Text);
    Hasil := NilaiA * NilaiB;
    EditHasil.Text := FloatToStr(Hasil);
  except
    on EConvertError do
      EditHasil.Text := 'A atau B salah!';
    on EOverflow do
      EditHasil.Text := 'Overflow!';
    on EUnderflow do
      EditHasil.Text := 'Underflow!';
  end;
end;

procedure TForm1.ButtonDivideClick(Sender: TObject);
var
  NilaiA, NilaiB, Hasil: Real;
begin
  try
    NilaiA := StrToFloat(EditNilaiA.Text);
    NilaiB := StrToFloat(EditNilaiB.Text);
    Hasil := NilaiA / NilaiB;
    EditHasil.Text := FloatToStr(Hasil);
  except
    on EConvertError do
      EditHasil.Text := 'A atau B salah!';
    on EOverflow do
      EditHasil.Text := 'Overflow!';
    on EUnderflow do
      EditHasil.Text := 'Underflow!';
    on EZeroDivide do
      EditHasil.Text := 'Pembagian dengan nol!';
  end;
end;

procedure TForm1.ButtonDivClick(Sender: TObject);
var
  NilaiA, NilaiB, Hasil: Integer;
begin
  try
    NilaiA := StrToInt(EditNilaiA.Text);
    NilaiB := StrToInt(EditNilaiB.Text);
    Hasil := NilaiA div NilaiB;
    EditHasil.Text := IntToStr(Hasil);
  except
    on EConvertError do
      EditHasil.Text := 'A atau B salah!';
    on EOverflow do
```

```
      EditHasil.Text := 'Overflow!';
    on EUnderflow do
      EditHasil.Text := 'Underflow!';
    on EDivByZero do
      EditHasil.Text := 'Pembagian dengan nol!';
  end;
end;

procedure TForm1.ButtonModClick(Sender: TObject);
var
  NilaiA, NilaiB, Hasil: Integer;
begin
  try
    NilaiA := StrToInt(EditNilaiA.Text);
    NilaiB := StrToInt(EditNilaiB.Text);
    Hasil := NilaiA mod NilaiB;
    EditHasil.Text := IntToStr(Hasil);
  except
    on EConvertError do
      EditHasil.Text := 'A atau B salah!';
    on EOverflow do
      EditHasil.Text := 'Overflow!';
    on EUnderflow do
      EditHasil.Text := 'Underflow!';
    on EDivByZero do
      EditHasil.Text := 'Pembagian dengan nol!';
  end;
end;
```

Perhatikan, masing – masing bila terjadi kesalahan maka kesalahan tersebut tidak ditampilkan pada dialog pesan kesalahan (message dialog), tetapi langsung ditampilkan pada EditHasil.

Pada kode program tersebut terdapat perbedaan antara exception EDivByZero dan EZeroDivide, yaitu EDivByZero terjadi bila ada pembagian dengan nol untuk bilangan bulat, sedangkan EZeroDivide terjadi bila ada pembagian dengan nol untuk bilangan nyata.

Sekarang untuk masing-masing EditNilaiA dan EditNilaiB, buatlah event – handler pada event OnChange (dengan nama EditNilaiChange) sedemikian sehingga bila EditNilaiA ataupun EditNilaiB diubah, maka akan mengosongkan EditHasil.

```
procedure TForm1.EditNilaiChange(Sender: TObject);
begin
  EditHasil.Clear;
end;
```

Terakhir, buatlah agar jika ButtonClose diklik, aplikasi akan ditutup.

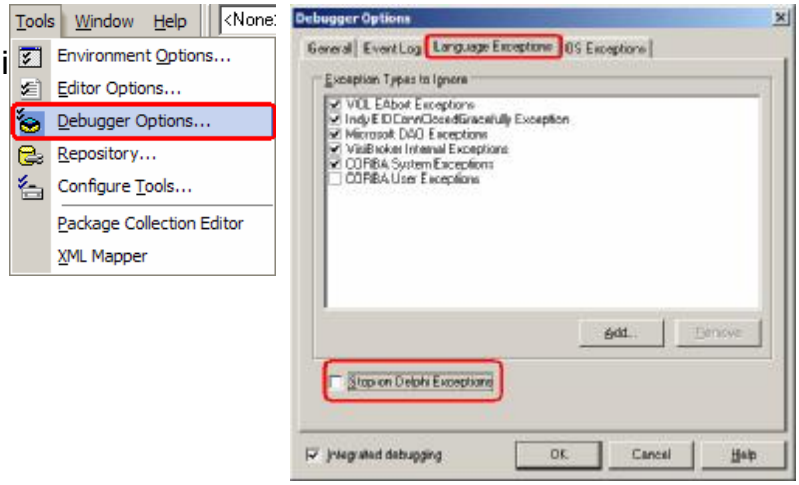
```
procedure TForm1.ButtonCloseClick(Sender: TObject);
begin
```

```
  Close;
end;
```

Simpan unitnya dengan nama MainUnit, dan project-nya dengan nama Kalkulator. Jangan lupa untuk menyediakan folder tersendiri untuk menyimpannya.

Kemudian cobalah jalankan aplikasi tersebut. Jangan lupa untuk menonaktifkan pilihan "Stop on Delphi Exceptions".

Gambar 8.2: Stop on Delphi Exceptions



Tugas

Tugas 8.1

Rekayasalah program Kalkulator diatas dengan ketentuan:

- Tambahkan satu komponen TPanel bernama PanelOperasi yang menampilkan jenis operasi yang dikerjakan pada nilai A terhadap nilai B (sesuai dengan Caption dari TButton yang diklik). Letakkan TPanel tersebut di atas EditHasil seperti pada gambar 3.2.
- Tambahkan kode program pada event-handler yang ada untuk menampilkan jenis operasi pada PanelOperasi.
- Bila terjadi kesalahan dan ditampilkan pesan kesalahan pada EditHasil, warna Font pada EditHasil menjadi merah.
- Bila tidak terjadi kesalahan dan hasilnya ditampilkan pada EditHasil, warna Font pada EditHasil menjadi biru.
- Bila isi dari EditNilaiA atau EditNilaiB diubah, selain mengosongkan EditHasil juga mengosongkan Caption pada PanelOperasi.
- Buatlah kode program yang ada menjadi seringkas mungkin.

Gambar 8.3: Tampilan form rekayasa Kalkulator.

Kemudian jalankan program tersebut. Hasil dari eksekusi aplikasi tersebut salah satunya seperti pada gambar 3.3.

Gambar 8.4: Salah satu kemungkinan hasil eksekusi Kalkulator.