BAB II

LANDASAN TEORI

2.1 Konsep Steganografi

Steganografi didefinisikan sebagai ilmu dan seni untuk menyembunyikan pesan rahasia (*hiding message*) sedemikian rupa sehingga keberadaan (eksistensi) pesan tidak terdeteksi oleh indera manusia. Saat ini di dalam dunia digital, teknik steganografi banyak digunakan untuk menyembunyikan informasi rahasia dengan berbagai maksud. Salah satu tujuan dari steganografi adalah mengirim informasi rahasia melalui jaringan tanpa menimbulkan kecurigaan (Muhammad Hakim A., Chris Sanders, 2006).

2.1.1 Steganografi Pada Media Digital *File Image*

Pada komputer, *image* yang tampil di layar monitor merupakan kumpulan *array* yang mereprentasikan intensitas cahaya yang bervariasi pada *pixel*. *Pixel* adalah titik di layar monitor yang dapat diatur untuk menampilkan warna tertentu. *Pixel* disusun di layar monitor dalam susunan baris dan kolom, ini yang dinamakan resolusi monitor. Resolusi monitor yang sering dijumpai adalah 640x480, 800x600, 1024x768. Resolusi 640x480 akan menampilkan *pixel* sejumlah 640 baris dan 480 kolom, sehingga total pixel yang digunakan 640x480 = 307.200 *pixel*. Melalui *pixel* inilah suatu gambar dapat dimanipulasi untuk menyimpan informasi yang akan digunakan sebagai salah satu pengimplementasian Steganografi.

Sebagaimana dikemukakan oleh Andino Masaleno (2003, 3) bahwa "steganografi pada media digital *file image* digunakan untuk mengeksploitasi keterbatasan kekuatan sistem penglihatan manusia dengan cara menurunkan kualitas warna pada *file image* yang belum disisipi pesan rahasia. Sehingga dengan keterbatasan tersebut manusia sulit menemukan gradasi penurunan kualitas warna pada *file image* yang telah disisipi pesan rahasia."

2.1.2 *File Image Bitmap 24-bit*

Aplikasi steganografi yang dibuat akan menyisipkan pesan pada format *file bmp 24-bit*. Format *file bmp* merupakan format *file* standar sistem operasi *MS Windows 3.11/9x/NT* dan *IBM OS/2*. Format *file* ini mendukung resolusi warna dari *monochrom* hingga *True Color* (16,7 juta warna).

Format *file bmp 24-bit* menggunakan model warna *RGB*. Pada model warna *RGB*, warna yang ditampilkan di layar monitor disusun oleh tiga buah warna primer, yaitu *Red* (Merah), *Green* (Hijau), dan *Blue* (Biru). Pada model warna *RGB*, setiap titik pada layar monitor berisi angka yang menunjukkan intensitas yang dipilih pada suatu tabel warna *RGB*. Jadi, pada setiap titik dapat dipilih salah satu warna dari *RGB*.

Warna dari tabel *RGB* memiliki tiga buah warna kombinasi warna yaitu R, G, dan B yang menentukan proporsi warna merah, hijau, dan biru dari warna tersebut. Warna di tabel yang dapat dipilih untuk mengisi warna pada sebuah *pixel* adalah 256x256x256 = 16,7 juta warna (Chris Sanders, 2006; Rich Gray, 2003; Andino Masaleno, 2003).

2.1.3 Steganografi Menggunakan Metode *LSB*

Sebagaimana dikemukakan oleh Muhammad Hakim A. (2003: 2) bahwa "saat ini setidaknya terdapat 6 teknik steganografi", yaitu:

1. Teknik substitusi (*substitution techniques*), yaitu dengan melakukan substitusi *pixel-pixel* tertentu dari berkas wadah (*cover*) dengan *pixel-pixel* informasi yang disimpan. Contohnya adalah metode *Least Significant Bit* (*LSB*).
2. Teknik Domain Transformasi (*Domain Transformation Techniques*), yaitu dengan cara menyimpan informasi rahasia pada transformasi ruang (misalnya domain frekuensi) dari berkas wadah (*cover*). Contoh algoritma yang tergolong teknik ini adalah steganografi pada domain *DCT* (*Discrete Cosine Transform*).
3. Teknik *Spread Spectrum* (*Spread Spectrum Techniques*). Ide dari teknik ini mengadopsi teknik *Spread Spectrum* pada saluran komunikasi. Dalam teknik ini informasi rahasia yang disimpan disebarkan pada suatu frekuensi tertentu dari berkas wadah (*cover*).
4. Teknik Statistik (*Statistical Techniques*). Dengan teknik ini data di-*encoding* melalui pengubahan beberapa informasi statistik dari berkas wadah (*cover*). Berkas wadah dibagi dalam blok-blok di mana setiap blok tersebut menyimpan satu *pixel* informasi rahasia yang disembunyikan. Jika *pixel* yang ditemukan pada suatu blok untuk menyimpan data adalah *pixel* '1', maka tidak dilakukan perubahan nilai *pixel*. jika sebaliknya maka dilakukan perubahan nilai *pixel*. Meskipun secara teoritis mungkin untuk

dilakukan, pada kenyataannya teknik ini agak sulit untuk diimplementasikan.

5. Teknik Distorsi (*Distortion Techniques*). Informasi yang hendak disembunyikan disimpan berdasarkan distorsi sinyal.
6. Teknik Pembangkit Wadah (*Cover Generation Techniques*). Teknik ini menyembunyikan informasi rahasia dengan pembangkitan wadah (*cover*). Misalnya aplikasi pembangkit teks/paragraf bahasa inggris otomatis.

Metode yang digunakan untuk penyembunyian pesan rahasia pada aplikasi ini adalah dengan cara menyisipkan pesan ke dalam bit rendah (*Least Significant Bit*) pada data *pixel* yang menyusun *file image bmp 24-bit*. Metode modifikasi *Least Significant Bit* (*LSB*) ini bisa dikatakan sebagai metode yang menggunakan teknik substitusi seperti yang telah dijelaskan di atas.

Pada *file image bmp 24-bit*, setiap *pixel* pada *image* terdiri dari susunan tiga warna, yaitu merah, hijau, dan biru (*RGB*) yang masing-masing disusun oleh bilangan 8 bit (1 *byte*) dari 0 sampai 255 atau dengan format biner 00000000 sampai 11111111. Sebagai contoh *file image bmp 24-bit* dengan warna merah murni dalam format biner akan terlihat sebagai berikut:

00000000 00000000 11111111

00000000 00000000 11111111

Sedangkan untuk warna hijau murni dalam format biner akan terlihat sebagai berikut:

00000000 11111111 00000000

00000000 11111111 00000000

Kemudian untuk warna biru murni dalam format biner akan terlihat sebagai berikut:

11111111 00000000 00000000

11111111 00000000 00000000

Dari uraian di atas dapat dilihat bahwa informasi dari warna biru berada pada bit pertama sampai dengan bit delapan, dan informasi warna hijau berada pada bit sembilan sampai dengan bit 16, sedangkan informasi warna merah berada pada bit 17 sampai dengan bit 24.

Metode penyisipan *Least Significant Bit* (*LSB*) ini adalah menyisipi pesan dengan cara mengganti bit ke-8, 16, dan 24 pada representasi biner *file image* dengan representasi biner pesan rahasia yang akan disembunyikan (Andino Masaleno, 2003: 18). Dengan demikian pada setiap *pixel file image bmp 24-bit* dapat disisipkan 3 bit pesan. Sebagai contoh terdapat data *original file image* sebagai berikut:

00100111 11101001 11001000

00100111 11001000 11101001

11001000 00100111 11101001

Misalkan representasi biner huruf A adalah 01000001. Dengan menyisipkan ke dalam *pixel* di atas, maka akan dihasilkan:

00100110 11101001 11001000

00100110 11001000 11101000

11001000 00100111 11101001

Terlihat pada bit ke-8, 16, dan 24 diganti dengan representasi biner huruf A dan hanya tiga bit rendah yang berubah (cetak tebal). Untuk penglihatan mata manusia sangatlah mustahil untuk dapat membedakan warna pada *file image* yang sudah diisi pesan rahasia jika dibandingkan dengan *file image* asli sebelum disisipi dengan pesan rahasia.

2.2 Representasi Bilangan

Dalam representasi bilangan ini akan dibahas suatu sistem bilangan. Sistem bilangan adalah suatu cara untuk mewakili besaran dari suatu item fisik. Sistem bilangan menggunakan basis tertentu yang tergantung dari jumlah bilangan yang digunakan. Suatu sistem komputer mengenal beberapa sistem bilangan, seperti:

1. Sistem bilangan desimal dengan basis 10 (*deca* berarti 10), mengandung 10 macam simbol bilangan, yaitu: 0, 1, 2, 3, 4, 5, 6, 7, 8, dan 9.
2. Sistem bilangan biner dengan basis 2 (*binary* berarti 2), menggunakan 2 macam simbol bilangan, yaitu: 0 dan 1.
3. Sistem bilangan hexadesimal dengan basis 16 (*hexa* berarti 6 dan *deca* berarti 10), menggunakan 16 macam simbol bilangan, yaitu: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E dan F.

2.2.1 Konsep Bilangan Desimal, Biner, dan Hexadesimal

Perbedaan mendasar dari konsep bilangan desimal, bilangan biner, dan bilangan hexadesimal adalah berkenaan dengan basis. Bilangan desimal yaitu

suatu bilangan dengan basis 10. Bilangan biner yaitu suatu bilangan dengan basis 2. Sedangkan hexadesimal yaitu suatu bilangan dengan basis 16 (Yudi Wibisono, 2005). Untuk ketiga sistem bilangan tersebut, perhatikan penjelasan berikut ini.

Misalkan terdapat bilangan hexadesimal sebesar 9AF. Apabila bilangan hexadesimal tersebut diubah ke dalam bilangan desimal maka akan menjadi 2479. Sedangkan apabila bilangan hexadesimal tersebut diubah ke dalam bilangan biner maka akan menjadi 100110101111. Bagaimana dapat memahami perubahan-perubahan seperti yang telah dijelaskan di atas? Untuk itu, selanjutnya akan dibahas bagaimana cara megubah bilangan desimal ke bilangan biner, cara mengubah bilangan biner ke bilangan desimal, cara mengubah bilangan hexadesimal ke bilangan biner, cara mengubah bilangan hexadesimal ke bilangan desimal dan lain sebagainya.

2.2.2 Konversi Bilangan Desimal ke Biner

Sebelum mempelajari tentang sistem bilangan biner, sebaiknya mengetahui terlebih dahulu tentang sistem bilangan yang umum dipakai, yaitu bilangan desimal (bilangan dengan basis 10). Perhatikan tebel berikut ini.

Contoh bilangan	$10^2 = 100$, $10^1 = 10$, $10^0 = 1$
Jumlah simbol	Sepuluh
Simbol	0, 1, 2, 3, 4, 5, 6, 7, 8, dan 9.

Untuk menghitung suatu basis bilangan, harus dimulai dari nilai yang terkecil (yang paling kanan). Pada basis 10, maka kalikan nilai yang paling kanan

dengan 10^0 ditambah dengan nilai di kirinya yang dikalikan dengan 10^1, dan seterusnya.

Contoh:

$1234 = (1x10^3) + (2x10^2) + (3x10^1) + (4x10^0)$

$= 1000 + 200 + 30 + 4$.

Untuk konversi bilangan desimal ke bilangan biner, perhatikan contoh berikut. Misalkan akan dikonversi suatu bilangan desimal yaitu 1234 ke bilangan biner. Caranya adalah sebagai berikut:

$1234 / 2 = 617$	sisa 0
$617 / 2 = 308$	sisa 1
$308 / 2 = 154$	sisa 0
$154 / 2 = 77$	sisa 0
$77 / 2 = 38$	sisa 1
$38 / 2 = 19$	sisa 0
$19 / 2 = 9$	sisa 1
$9 / 2 = 4$	sisa 1
$4 / 2 = 2$	sisa 0
$2 / 2 = 1$	sisa 0
$1 / 2 = 0$	sisa 1

Bit biner terbesar dimulai dari bawah, sehingga $1234_{(10)} = 10011010010_{(2)}$. Dari contoh ini menunjukkan bahwa bilangan desimal (bilangan dengan basis 10) dari 1234 sama dengan bilangan biner (bilangan dengan basis 2) dari 10011010010.

2.2.3 Konversi Bilangan Biner ke Desimal

Setelah mengetahui bagaimana cara mengonversi bilangan desimal ke bilangan biner, sekarang akan dibahas cara mengonversi bilangan biner ke bilangan desimal. Untuk bilangan biner (bilangan dengan basis 2), perhatikan tabel berikut ini.

Contoh bilangan	$2^2 = 4,\ 2^1 = 2,\ 2^0 = 1$
Jumlah simbol	Dua
Simbol	0 dan 1

Untuk bilangan biner, kalikan bilangan paling kanan terus ke kiri dengan $2^0, 2^1, 2^2, 2^3$, dan seterusnya. Jadi, untuk konversi bilangan biner ke desimal, perhatikan contoh berikut. Misalkan akan dikonversi suatu bilangan biner yaitu 10101 ke bilangan desimal. Caranya adalah sebagai berikut:

$$10101_{(2)} = (1x2^4) + (0x2^3) + (1x2^2) + (0x2^1) + (1x2^0)$$

$$= 16 + 0 + 4 + 0 + 1$$

$$= 21_{(10)}$$

Jadi, $10101_{(2)} = 21_{(10)}$. Dari contoh ini menunjukkan bahwa bilangan biner (bilangan dengan basis 2) dari 10101 sama dengan bilangan desimal (bilangan dengan basis 10) dari 21.

2.2.4 Konversi Bilangan Hexadesimal ke Biner

Setelah mengetahui bagaimana cara mengonversi bilangan desimal ke bilangan biner dan juga cara mengonversi bilangan biner ke bilangan desimal, sekarang akan dibahas cara mengonversi bilangan hexadesimal ke bilangan biner. Cara mengonversi bilangan hexadesimal ke bilangan biner bisa dikatakan sebagai suatu cara pengembangan dari kedua konversi di atas, yaitu konversi bilangan desimal ke bilangan biner dan konversi bilangan biner ke bilangan desimal. Untuk itu, perhatikan tabel berikut ini.

Contoh bilangan	$0A$, $1B$, $2C$
Jumlah simbol	Enam belas
Simbol	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, dan F

Hal yang perlu diperhatikan adalah simbol huruf (A, B, C, D, E, dan F) pada bilangan hexadesimal. Simbol-simbol tersebut secara berturut-turut nilainya setara dengan 10, 11, 12, 13, 14, dan 15 pada bilangan desimal (bilangan dengan basis 10). Untuk itu, cara mengonversi bilangan hexadesimal ke bilangan biner dapat dilakukan langkah-langkah sebagai berikut:

1. Setarakan simbol-simbol pada bilangan hexadesimal dengan bilangan desimal.
2. Ubah simbol-simbol pada bilangan hexadesimal yang telah disetarakan ke dalam bilangan biner.

Perhatikan contoh berikut. Misalkan akan dikonversi suatu bilangan hexadesimal yaitu 9AF, maka dengan langkah-langkah di atas diperoleh:

1. Bilangan hexadesimal 9, setara dengan 9 pada bilangan desimal. Jadi, bilangan hexadesimal 9 bilangan binernya adalah 1001.
2. Dengan cara yang sama, maka diperoleh bilangan hexadesimal A dan bilangan hexadesimal F, secara berturut-turut bilangan binernya adalah 1010 dan 1111.
3. Untuk bilangan hexadesimal 9AF, maka bilangan binernya adalah 100110101111.

Jadi, $9AF_{(16)} = 100110101111_{(2)}$. Dari contoh ini menunjukkan bahwa bilangan hexadesimal (bilangan dengan basis 16) dari 9AF sama dengan bilangan biner (bilangan dengan basis 2) dari 100110101111.

2.2.5 Konversi Bilangan Hexadesimal ke Desimal

Apabila telah memahami cara mengonversi bilangan hexadesimal ke bilangan biner, maka untuk melakukan konversi dari bilangan hexadesimal ke desimal dapat dikerjakan dengan mudah. Pada konversi bilangan hexadesimal ke desimal, tinggal mengubah bilangan biner yang telah didapat ke dalam bilangan desimal. Prosesnya dapat dilihat seperti di bawah ini:

Hexadesimal ⟶ Biner ⟶ Desimal

Untuk lebih jelasnya, bagaimana cara mengonversi bilangan hexadesimal ke bilangan desimal, dapat dilakukan langkah-langkah sebagai berikut:

1. Setarakan simbol-simbol pada bilangan hexadesimal dengan bilangan desimal.
2. Ubah simbol-simbol pada bilangan hexadesimal yang telah disetarakan ke dalam bilangan biner.
3. Ubah bilangan biner yang telah didapat ke bilangan desimal.

Perhatikan contoh berikut. Misalkan akan dikonversi suatu bilangan hexadesimal yaitu ABC, maka dengan langkah-langkah di atas diperoleh:

1. Bilangan hexadesimal A, setara dengan 10 pada bilangan desimal. Jadi, bilangan hexadesimal A bilangan binernya adalah 1010.
2. Dengan cara yang sama, maka diperoleh bilangan hexadesimal B dan bilangan hexadesimal C, secara berturut-turut bilangan binernya adalah 1011 dan 1100.
3. Untuk bilangan hexadesimal ABC, maka bilangan binernya adalah 101010111100.
4. Untuk bilangan biner 101010111100, maka bilangan desimalnya adalah 2748.

Jadi, $ABC_{(16)} = 2748_{(10)}$. Dari contoh ini menunjukkan bahwa bilangan hexadesimal (bilangan dengan basis 16) dari ABC sama dengan bilangan desimal (bilangan dengan basis 10) dari 2748.

2.3 *Borland Delphi*

Borland Delphi merupakan program aplikasi *database* yang berbasis *object pascal* dari *Borland* (Andino Masaleno, 2003: 4). *Borland Delphi* juga memberikan fasilitas pembuatan aplikasi *visual*. *Borland Delphi* memiliki komponen-komponen *visual* maupun *nonvisual* berintegrasi yang akan menghemat penulisan program. Terutama dalam perancangan antarmuka grafis (*Graphical User Interface*), kemampuan *Borland Delphi* untuk menggunakan *Windows API* (*Application Programming Interface*) ke dalam komponen-komponen *visual* menyebabkan program *Borland Delphi* yang bekerja dalam lingkungan *Windows* menjadi lebih mudah. Kelebihan *Borland Delphi* dalam hal kompilasi program juga menjadi faktor yang mempengaruhi pemilihan bahasa pemrograman yang digunakan. Karena program dikembangkan berdasarkan bahasa *pascal* yang dikenal luas, maka untuk pengembangan program akan lebih mudah. *Borland Delphi* juga mempunyai kemampuan bekerja untuk pengolahan gambar dengan tersedianya unit *GRAPHICS.*

Warna *pixel* dalam *Borland Delphi* dijabarkan dalam 4 *byte hexadecimal*, dengan 3 *byte* terendahnya adalah nilai intensitas *RGB*, dengan susunan sebagai berikut: *$00BBGGRR*, sebagai contoh nilai dari *$00FF0000* berarti intensitas biru murni, *$0000FF00* adalah hijau murni, dan *$000000FF* adalah merah murni. *$00000000* adalah hitam dan *$00FFFFFF* adalah putih. Untuk mengambil intensitas *RGB* suatu *pixel* dapat menggunakan perintah *GetRValue*, *GetGValue*, *GetBValue*. Contoh *GetRValue* (*$000000FF*) berarti mengambil nilai intensitas merah. Sedangkan untuk menggabungkan ketiga nilai intensitas *RGB* tersebut

dapat menggunakan perintah *RGB* (*R, G, B*), contoh *RGB* (*$FF, $FF, $FF*). Pada pembuatan aplikasi steganografi ini, akan menggunakan *Borland Delphi 7*.

2.3.1 Mengenal *Borland Delphi*

Sebelum memulai *Delphi*, *install* program *Delphinya* terlebih dahulu, bisa menggunakan *Delphi* 5 atau 6. Sebenarnya *Delphi* versi berapapun tidak masalah, karena produk *Borland* ini selalu menjaga kompatibilitas antar versinya. Untuk menginstalasi *Delphi* cukup mudah, tinggal ikuti petunjuk yang diberikan pada tampilan di layar. Sesudah *Delphi* terinstalasi pada komputer, mulailah dengan mengenal dahulu apa itu *Delphi*? Kelompok bahasa pemrograman apa? *Delphi* adalah kompiler/penerjemah bahasa *Delphi* (awalnya dari *Pascal*) yang merupakan bahasa tingkat tinggi sekelas dengan *Basic*, *C*. Bahasa Pemrograman di *Delphi* disebut bahasa *procedural* artinya bahasa/sintaksnya mengikuti urutan tertentu. Ada jenis pemrograman non-prosedural seperti pemrograman untuk kecerdasan buatan seperti bahasa Prolog. *Delphi* termasuk keluarga *Visual* sekelas *Visual Basic*, *Visual C*, artinya perintah-perintah untuk membuat *Object* dapat dilakukan secara *visual*. Pemrogram tinggal memilih *Object* apa yang ingin dimasukkan ke dalam *Form*, lalu tingkah laku *Object* tersebut saat menerima *event* tinggal dibuat programnya. *Delphi* merupakan bahasa berorientasi *Object*, artinya nama *Object*, properti, dan *procedure* dikemas menjadi satu kemasan (*encapsulate*).

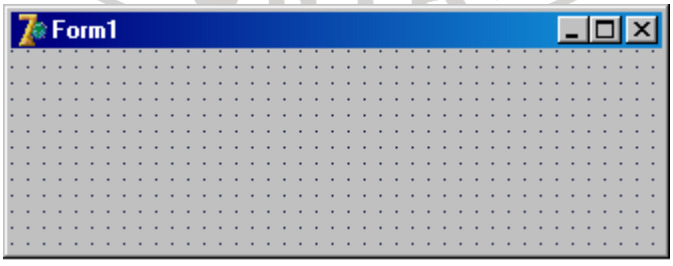
Sebelum mempelajari struktur pemrograman ada baiknya kenali dahulu tampilan *IDE* yang merupakan *editor* dan *tools* untuk membuat program *Delphi*.

Pada *IDE* akan ditampilkan *Form* baru yang merupakan aplikasi *windows* yang akan dibuat. Aplikasi berbasis *windows* sering disebut dengan jendela (*windows*). Bagaimana membuat aplikasi berbasis *windows* (berbasis grafik dan bukan berbasis teks seperti pada *DOS*)? Caranya dengan membuat sebuah *Form*. Pada pemrograman berbasis *windows* akan dihadapkan pada satu atau beberapa jendela. Jendela ini dalam *Delphi* disebut juga dengan *Form*.

Delphi adalah sebuah perangkat lunak (bahasa pemrograman) untuk membuat aplikasi komputer berbasis *windows*. *Delphi* merupakan bahasa pemrograman berbasis *Object*, artinya semua komponen yang ada merupakan *Object-object*. Ciri sebuah *Object* adalah memiliki nama, *properti*, dan *procedure*. *Delphi* disebut juga *visual programming* artinya komponen-komponen yang ada tidak hanya berupa teks (yang sebenarnya program kecil) tetapi muncul berupa gambar-gambar.

2.3.1.1 Membuat Sebuah *Form*

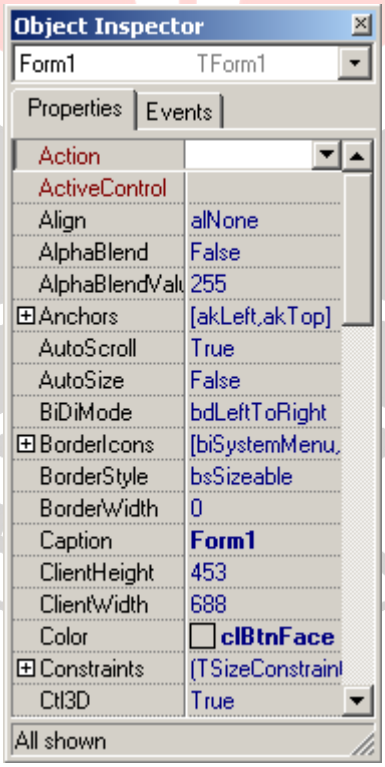
Saat pertama kali masuk ke *Delphi*, akan dihadapkan pada sebuah *Form* kosong yang akan dibuat secara otomatis. *Form* tersebut diberi nama *Form1*. *Form* ini merupakan tempat bekerja untuk membuat antarmuka pengguna.



Gambar 2.1: *Form*

2.3.1.2 Mengganti Nama *Form*

Biasakan sebelum menjalankan program, sebaiknya ganti nama *Form* dan beri judul sesuai program yang dibuat. *Delphi* akan secara otomatis memberi nama *Form1*, *Form2*, *Form3*, dan seterusnya. Nama *Form* tersebut kurang mengandung arti dan akan menyulitkan bila *Form* yang dibuat cukup banyak. Saat membuka *Delphi* pertama kali, nampak sebuah jendela *Object Inspector*. Jika tidak muncul tekan tombol *F11*. Pada *Object Inspector* ada dua buah halaman (*tab*) yaitu *Properties* dan *Events*. *Properties* digunakan untuk mengganti properti (kepemilikan) sebuah komponen. Sedangkan *Events* digunakan untuk membuat *procedure* yang diaktifkan (*trigered*) lewat sebuah *event*.



Gambar 2.2: *Object Inspector*

Semua properti diurutkan berdasarkan alpabetik dan dapat juga diurutkan berdasarkan kategori. Gantilah judul *Form* dengan *Hello* melalui properti

Caption, sedangkan nama *Form* dengan nama *frmHello* melalui properti *Name*. *Caption* digunakan untuk menyimpan keterangan yang dimunculkan pada *Form*, sedangkan *Name* digunakan sebagai Nama dari *Object* tersebut. Sebagai catatan, isi dari properti *Name* harus diawali alpabet dan tidak menggunakan spasi atau tanda baca. Sekarang sudah dapat membuat aplikasi *Form* kosong dengan tombol standar *windows*: *Minimize*, *Maximize*, dan *Close*. Ukuran *Form* dapat diubah dengan menarik pada bingkai *Form* menggunakan *mouse* (*drag* = klik tombol kiri *mouse*, tahan tombol tersebut lalu geser ke kiri/kanan atau atas/bawah). *Form* dapat dipindahkan dengan meletakkan kursor pada *Form* kemudian menggesernya (*drag*).

2.3.1.3 Menyimpan *Form*

Pada *Delphi* ada 3 buah file utama (**.dpr, *.pas dan *.dfm*).

1) **.dpr* adalah file proyek yang dibuat berisi program kecil untuk:
   - Mendefinisikan *Unit* yang ada dalam *file* proyek.
   - Menginisialisasi data.
   - Membangun *Form*.
   - Menjalankan aplikasi.

```
uses
Forms,
Unit1 in 'Unit1.pas' {Form1};
begin
Application.Initialize;
Application.CreateForm(Tform1, Form1);
Application.Run;
end.
```

2) **.pas* adalah unit-unit (*pascal code file*), bisa terdiri satu atau banyak *file*.

3) **.dfm* adalah *file* definisi *Form* (*special pseudo code file*), bisa terdiri satu atau banyak *file*.

```
object Form1: Tform1
Left = 200
Top = 108
Width = 696
Height = 480
Caption = 'Form1'
Font.Charset = DEFAULT_CHARSET
Font.Color = clWindowText
Font.Height = -11
Font.Name = 'MS Sans Serif'
Font.Style = []
PixelsPerInch = 96
TextHeight = 13
 object Button1: Tbutton
Left = 176
Top = 116
Width = 75
Height = 25
Caption = 'Button1'
TabOrder = 0
 end
end
```

Setiap *Form* (*.dfm*) harus memiliki sebuah *Unit* (*.pas*), tetapi dapat juga memiliki *Unit* tanpa sebuah *Form* (hanya *code* saja). Jika ingin melihat *code* tersebut, klik kanan *mouse*, lalu pilih *VIEW AS TEXT* atau tekan tombol *Alt-F12*. Sebaiknya tidak mengubah isi *code* tersebut, karena akan menyebabkan masalah serius. Untuk kembali ke bentuk *Form*, pilih *VIEW AS FORM* atau tekan tombol *Alt-F12* kembali.

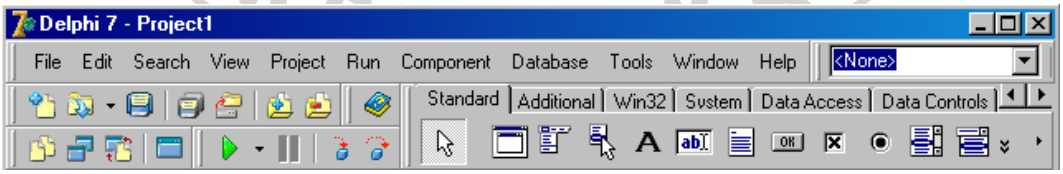
Pilih *submenu Save Project* atau *Save Project As* pada menu *File*, dan *Delphi* akan menanyakan nama *file source code* untuk *Unit* (**.pas*) dan nama *file* proyeknya (**.dpr*). Beri nama *file Form* dengan *HELLO.PAS* dan *project*

HELLO.DPR. Sesudah disimpan, jalankan program dengan menekan tombol *F9* atau pilih menu *Run*.

2.3.1.4 Menempatkan Komponen pada *Form*

Karena *Delphi* merupakan bahasa pemrograman *visual*, maka komponen-komponen akan nampak pada layar. Ada empat cara menempatkan komponen pada *Form*. Misal, memilih komponen *Button* pada *Components Palette* bagian *Standard Page*. Pilih salah satu langkah berikut:

- Klik pada kompenen tersebut, pindahkan kursor ke *Form*, sambil menekan tombol kiri *mouse* (*drag* komponen dan geser pada *Form*).
- Pilih komponen (klik komponen yang diinginkan) pada *Components Palette* kemudian klik pada *Form* di mana komponen itu akan diletakkan.
- Klik ganda pada komponen yang diinginkan, maka komponen tersebut akan ditambahkan pada *Form*.
- Dapat menggunakan *Copy* dan *Paste* bila ingin membuat komponen yang sama yang sudah ada pada *Form*. Caranya *Shift*-Klik kiri pada komponen yang ada di *Form*, lalu pilih menu *Copy* kemudian pilih menu *Paste*.



Gambar 2.3: Kumpulan Komponen

2.3.1.5 Mengatur Tataletak Komponen

Bila ingin merapihkan tataletak komponen pilih menu *View* | *Alignment Palette*, maka muncul sebuah *Toolbox Align* dengan *icon* perapihan (*alignment icons*). Dengan *toolbox* ini, dapat merapikan beberapa komponen sekaligus, caranya buat fokus beberapa komponen, lalu klik *icon* pada *toolbox* yang diinginkan. Untuk mengetahui arti *icon* tersebut gerakan *mouse* pada tombol tersebut, lalu akan muncul penjelasan singkat kegunaan *icon* tersebut atau lihat *Help* (tekan *F1*). Langkah yang penting adalah mengubah nama dan keterangan kompenen *button*.

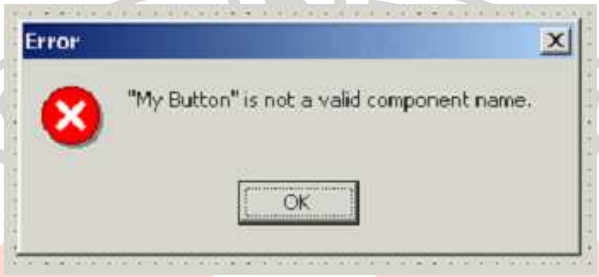
2.3.1.6 Mengubah Nilai Properti

Ubah nilai properti *Caption* menjadi 'Katakan *Hello*' dan nilai properti *Name* menjadi *btnHello*. Langkah ini mirip dengan mengubah nama dan keterangan sebuah *Form*. Setiap komponen sebaiknya diberi nama yang memiliki arti dan diawali oleh jenis komponennya. Misal nama dari *Form Hello* adalah "*frmHello*" atau nama dari *button Hello* adalah "*btnHello*". Tujuannya adalah mengelompokkan komponen-komponen sejenis, karena pada *Object Inspector* nama komponen diurutkan berdasarkan alpabet. Properti *name* adalah properti *internal* dan digunakan untuk memberi nama pada sebuah komponen. Nama ini adalah sebuah variabel yang mengacu pada komponen tersebut. Beberapa aturan penamaan komponen atau variabel atau *identifer* sebagai berikut:

- Diawali alpabet, berikutnya boleh angka, garis bawah.
- Tidak memakai spasi atau tanda-tanda baca atau operator.

- Boleh huruf kapital atau kecil, tidak ada perbedaan.
- Tidak menggunakan kata kunci (*reserve word*) yang digunakan *Delphi*.
- Biasakan nama komponen diawali kelompok komponennya, misal *btnHello, frmHello, rgrKelas.*

Berikut contoh penamaan yang keliru menggunakan spasi:



Gambar 2.4: Pesan Kesalahan

Sesudah mengubah property, *code* programnya dapat dilihat (tekan *Alt-F12*) sebagai berikut:

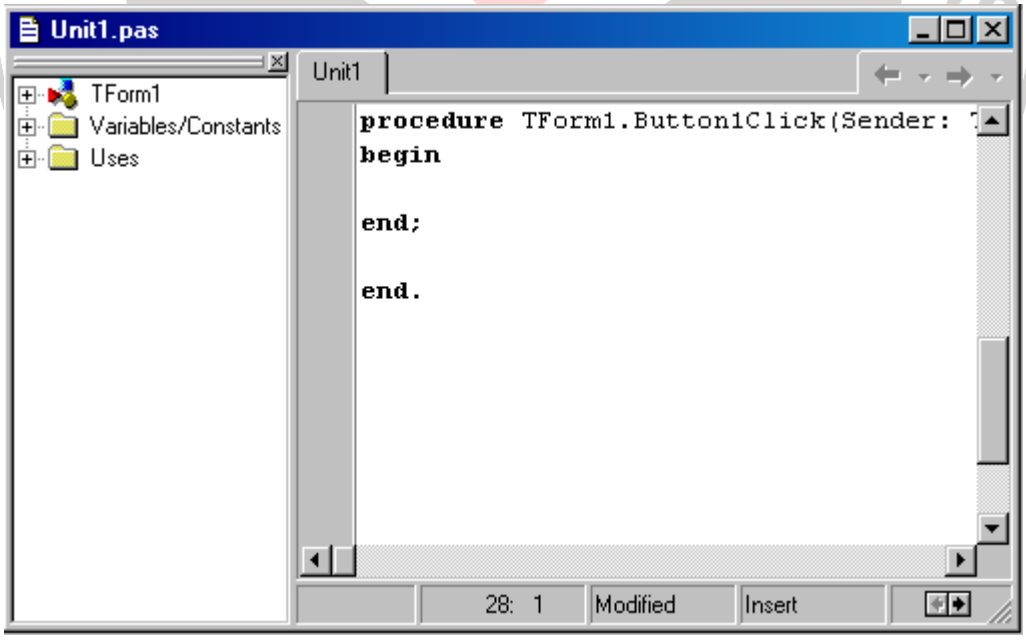
```
object Form1: Tform1
        :
        Caption = 'Hello'
        :
        object btnHello: Tbutton
                Caption = 'Katakan Hello'
        End
End
```

2.3.1.7 Membuat *Method/Procedure* lewat *Event*

Saat menekan tombol pada sebuah *Form* atau komponen, *windows* memberitahukan bahwa aplikasi mengirim pesan yang dibangkitkan oleh *event* tertentu. *Delphi* akan menanggapi dengan menerima *event* atau panggilan tersebut. Hal ini yang dinamakan penanganan *event* (*event-handler method*). *Delphi* mendefinisikan sejumlah *event* pada setiap komponennya. Daftar *event* ini bebeda

untuk setiap komponen. *Event* yang paling umum pada komponen *Button* adalah *OnClick*. Artinya jika komponen *Button* tersebut diklik, maka akan melakukan *procedure* apa? Ada beberapa teknik yang dapat dilakukan untuk menangani *event*, misal *OnClick* pada komponen *button*:

- Klik ganda pada *button* tersebut, maka sebuah *method/procedure* *btnHelloClick*.
- Pilih *button*, kemudian pilih *Object Inspector's combo box* (*called the Object Selector*), pilih *Tab Events*, dan klik ganda pada area putih disebelah kanan *event OnClick*.
- Pilih *button*, pilih *Tab Events*, dan masukkan nama *method* yang dikehendaki, misal *btnHelloClick* pada area putih di sebelah kanan *event OnClick*.



Gambar 2.5: *Procedure* yang dibangkitkan lewat *event OnClick*

Procedure atau penanganan *event* tersebut dapat dihapus pada *editor Unit*. Hapus blok *procedure* tersebut dan hapus pada bagian definisi *procedure* yang ada di atasnya. Sekarang isi *procedure* tersebut dengan perintah untuk menampilkan pesan sebagai berikut:

```
procedure TForm1.Button1Click (Sender: Tobject);
begin
  MessageDlg (‘Hello, guys’, mtInformation, [mbOK], 0);
end;
```

Perintah ini sangat sederhana, yaitu untuk menampilkan pesan. Fungsi ini mempunyai empat parameter. Untuk rincinya dapat melihat Bantuan (*F1*).

- Parameter pertama: kalimat yang akan dimunculkan (pesannya).
- Parameter kedua: tipe *message box* seperti *mtWarning*, *mtError*, *mtInformation*, atau *mtConfirmation*. Coba lihat perbedaannya!
- Parameter ketiga: kumpulan tombol yang akan digunakan seperti *mbYes*, *mbNo*, *mbOK*, *mbCancel*, atau *mbHelp*.
- Parameter keempat: untuk *help context* atau nomor halaman pada *Help*, beri angka nol jika tidak mempunyai *file help*.

Gambar 2.6: *Message Box*

2.3.1.8 Kompilasi dan Jalankan Program

Tekan tombol *Run* atau pilih menu *Run | Run, Delphi does the following*:

1. Kompilasi *Pascal source code file* yang mendefinisikan *Form-Form* yang ada *(.pas, .dfm*).
2. Kompilasi *project file* (*.dpr*).
3. Buat *executable* (*.exe*) *file*.
4. Jalankan *executable file*, biasanya pada *mode* pencarian kesalahan (*debug mode*).

2.3.2 Perintah *Sequence* / Diproses Secara Berurutan

Perintah-perintah ini akan diproses oleh kompiler secara berurutan. Contoh, terdapat 3 baris perintah. Kompiler akan memroses mulai dari baris-1, baris-2 kemudian baris-3. Contoh program untuk menampilkan pesan sebanyak 3 kali seperti berikut:

```
MessageDlg ('Hello, guys', mtInformation, [mbOK], 0);
MessageDlg ('Hello, lady', mtInformation, [mbOK], 0);
MessageDlg ('Hello, everybody', mtInformation, [mbOK], 0);
```

2.3.2.1 Menu dan Perintah pada *Delphi*

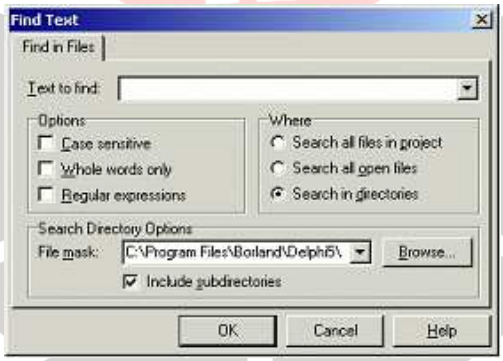
Ada empat cara untuk memberi perintah pada lingkungan *Delphi* (*Delphi environment*):

- Gunakan *Menu*.
- Gunakan *Short Cut* (misal *F9, F12*, dan sebagainya).
- Gunakan *SpeedBar* (atau *toolbar*).

- Gunakan *SpeedMenu* (lokal menu yang diaktifkan dengan tombol *mouse* kanan).

Berikut menu utama yang ada pada *Delphi* (untuk mempelajarinya gunakan *Help Delphi*):

- ***Menu File***. Menu ini berhubungan dengan *file* seperti membuat, menyimpan, dan mengakhiri sebuah pekerjaan.
- ***Menu Edit***. Menu ini berhubungan dengan penyuntingan apa yang dikerjakan seperti *Undo*, *Redo*, *Cut*, *Copy*, *Paste* atau dapat dengan tombol *Ctrl+Z*, *Ctrl+X*, *Ctrl+C*, *Ctrl+V*.
- ***Menu Search***. Menu ini berhubungan dengan pencarian dan penggantian data.



Gambar 2.7: Jendela *Find Text*

- ***Menu View***. Menu ini berhubungan dengan penampilan atau apa yang akan ditampilkan.
- ***Menu Project***. Menu ini berhubungan dengan proyek yang sedang dibuat, misal *Unit* yang akan ditambahkan ke proyek ini, *Unit* apa yang akan dihapus, dan sebagainya.

- ***Menu Run***. Menu ini berhubungan dengan menjalankan program, mencari kesalahan (*debug*), dan sebagainya.
- ***Menu Component***. Menu ini berhubungan dengan komponen, misal menambah komponen baru, menghapus komponen yang ada.
- ***Menu Database***. Menu ini berhubungan dengan *Database*, *Database Form Wizard* dan *Database Explorer*.
- ***Menu Tools***. Menu ini berhubungan dengan pengaturan, *tool-tool* pembantu *Delphi*.
- ***Menu Help***. Menu ini berhubungan dengan informasi mengenai *Delphi*, *Help*.

2.3.2.2 *Component, Property, Method,* dan *Event*

Code yang akan dilihat, serupa dengan struktur Bahasa *Pascal*. *Delphi* adalah bahasa pemrograman berbasis *Object*, artinya pendekatan pembuatan program melalui *Object-object* yang ada. Misalnya *Object Form*, *text* dan sebagainya. Setiap *Object* akan memiliki properti (atribut) dan *method* yang diaktifkan oleh *event*.

Sebuah komponen adalah sebuah *Object* pada *Palette*. Sebuah *Object* adalah sebuah komponen dalam *Component Palette* atau sesuatu yang dibuat melalui bahasa pemrograman. Jadi, sebuah *Object* adalah secara umum kelas dari kumpulan sesuatu. Komponen pasti *Object* namun tidak selalu merupakan komponen, misal *TstringList* adalah sebuah *Object* (kumpulan karakter), dan bukan sebuah komponen (Teddy Marcus Zakaria, 2003: 12).

Sebuah *property* tidak lain adalah sebuah variabel milik sebuah komponen misal *Caption*, *Text* yang dapat diubah nilai baik melalui *object Inspector* atau melalui program (Teddy Marcus Zakaria, 2003: 12). Beberapa istilah berikut yang mirip, dan sering digunakan:

- *Procedure* adalah kumpulan perintah yang melakukan suatu proses tertentu (Teddy Marcus Zakaria, 2003: 12).
- *Function* adalah sama dengan *procedure*, tetapi proses tersebut dapat mengembalikan suatu nilai misal hasilnya = 1 (Teddy Marcus Zakaria, 2003: 12).
- *Method* adalah *procedure* atau *function* yang tergabung pada sebuah komponen (Teddy Marcus Zakaria, 2003: 12).
- *Subroutine* adalah istilah umum dari semuanya misal pada bahasa *Basic* (Teddy Marcus Zakaria, 2003: 12).

Sebuah *method* adalah sebuah *function* yang tergabung dalam sebuah *Object* (Teddy Marcus Zakaria, 2003: 12). Contoh *ListBox* (dapat berarti sebuah *array of strings*) yang memiliki *Method* (*Clear*) yang membuat *Listbox* tersebut menjadi kosong. *CLEAR* adalah sebuah *Method* pada *ListBox* tersebut.

```
Begin
    ListBox1.Clear; // Mengosongkan isi ListBox
    ListBox1.Items.LoadFromFile('c:\Data1.txt');
     //properti Items (bertipe string) memiliki method untuk LoadFromFile
end;
```

Sebuah *Event* adalah sebuah aksi pengguna (*User Action*) misal *Mouse Click*, *KeyPressed*. Setiap *Events* diawali dengan kata '*On*' (Teddy Marcus Zakaria, 2003: 12). Contoh :

Nama *event*:	Nama *method*:
OnClick ..	*Button1Click*(*Sender*: *Tobjec*t)
OnKeyDown ..	*Button1KeyDown*(*Sender*: *Tobject*)
OnMouseMove ..	*Button1MouseMove*(*Sender*: *Tobject*)

2.3.2.3 Cara *Delphi* Bekerja

Saat *Components* ditambahkan pada *Form1* dan merubah nilai properti, *Delphi* akan membuat (*pseudo*) *code* (dalam *Unit1.dfm*) untuk mendefinikan apa yang dilakukan. Secara normal, tidak diharapkan mengubah *Unit1.dfm file* dan hanya bekerja pada *Form1* secara *Visual*. Ini yang dinamakan bahasa pemrograman *Visual* (*Visual Programming*).

Delphi (*IDE*) adalah sebuah *Visual Interface* antara *User* dan komputer (yang berjalan di atas *windows*) (Teddy Marcus Zakaria, 2003: 13). *Delphi* menerjemahkan *Visual Components* (*Buttons*, *Panels*,..) yang ada pada *Form* ke dalam sebuah *code-code* komputer (*pseudo in Unit1.dfm*) yang mendefinisikan bagaimana dibentuknya *Form* dan komponennya termasuk juga propertinya. Ada beberapa kejadian saat mengompilasi program, yaitu:

- *Delphi* akan memanggil *file .dpr* (*file project*).
- *Delphi* meminta program yang ada dalam proyek tersebut dan *file .dpr* memberikan sebagai berikut:

```
uses
Forms,
Unit1 in ‘Unit1.pas’ {Form1};
```

- *Delphi* meminta, apa yang dilakukan pertama kali? *.dpr file* memberikan sebagai berikut:

```
begin
    Application.Initialize; itializes stuff
    Application.CreateForm(Tform1, Form1);
    Application.Run;
 end.
```

2.3.2.4 *Forms,* Dialog *Boxes,* dan *Messages*

- **Menampilkan *Form* atau *Windows*.**

  Kita dapat bekerja dengan beberapa *Form* pada sebuah *project*. Saat berada di *Form1* untuk menampilkan *Form* yang lain misal *Form2* sebagai berikut:

  – Form2.Show;

  Membuka (*Shows*) *Form2* (tetapi *user* diijinkan untuk dapat mengklik *Form1*).

  – Form2.ShowModal;

  Membuka (*Shows*) *Form2* (tetapi *user* tidak diijinkan mengklik *Form1*).

- **Menampilkan Pesan (*Message*).**

  – ShowMessage('Ini kotak pesan’);

  Tampilan sederhana sebuah baris/teks; *User* dapat menekan tombol *OK* untuk keluar dari kotak pesan (*message windows*).

  – MessageDlg('Msg',mtConfirmation,[mbYes],0);

  Mirip *ShowMessage* tapi dapat lebih dari satu tombol pilihan.

- If MessageDlg('Please say YES or NO',mtConfirmation, ([mbYes,mbNo],0)=mrYES then
  begin
  Label1.Text:='Tekan tombol YES';
  end;

- *TmsgDlgType = mtWarning, mtError, mtInformation, mtConfirmation, mtCustom.*

- *TMsgDlgBtn = mbYes, mbNo, mbOK, mbCancel, mbAbort, mbRetry, mbIgnore, mbAll, mbHelp.*

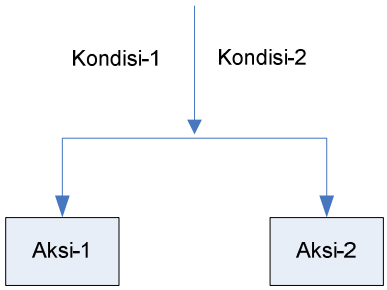
- *Return values = mrNone, mrYes, mrNo, mrOk, mrCancel, mrAbort, mrRetry, mrIgnore, mrAll.*

- **Meminta Masukan (*Input Box*)**

```
InputBox(...);
  var
    InputString: string;
  begin
    InputString:= InputBox('Masukkan Nama', 'Please Enter your Name','');
  end;
```

2.3.3 Perintah Pencabangan / Struktur Pemilihan

Pada beberapa kasus terkadang kita menginginkan komputer melakukan suatu aksi tertentu bila suatu kondisi terpenuhi. Keberadaan perintah bersyarat pada suatu program memberikan pencabangan proses seperti ditunjukkan oleh *Error*! *Not a valid link*. Bahasa *Pascal* menyediakan dua cara penyajian perintah bersyarat, yaitu *If…Then…Else* dan *Case…of*.



Gambar 2.8: Pencabangan Pada Program

Pada prinsipnya pemilihan aksi dapat dikelompokkan menjadi 3 bagian:

- Pemilihan 1 kasus. Gunakan perintah *IF-THEN*.
- Pemilihan 2 kasus. Gunakan perintah *IF-THEN-ELSE*.
- Pemilihan N kasus. Gunakan *Case-of* (Catatan: sebenarnya dapat menggunakan *if-then-else* bersarang).

2.3.3.1 *IF-THEN*: Pemilihan 1 kasus

Perintah bersyarat *If-then* digunakan hanya melakukan 1 aksi bila kondisi dipenuhi. Bentuk sintaks dari perintah bersyarat ini adalah:

```
If <kondisi_pencabangan> then
    Begin
    ... {aksi-1}
    end;
```

2.3.3.2 *IF-THEN-ELSE*: Pemilihan 2 kasus

Perintah bersyarat *If* umumnya digunakan untuk melakukan pencabangan sederhana (antara 2 atau 3 cabang) atau untuk pencabangan yang banyak, di mana kondisi yang menjadi prasyaratnya melibatkan lebih dari satu parameter. Bentuk sintaks dari perintah bersyarat ini adalah:

```
If <kondisi_pencabangan> then
        Begin
        ... {aksi-1}
        end
else
        Begin
        ... {aksi-2}
        end;
```

2.3.3.3 *IF-THEN-ELSE*: Pemilihan N kasus

Bentuk sintaks dari perintah bersyarat ini adalah:

```
If <kondisi_pencabangan1> then
    Begin
        ... {aksi-1}
    end
else
    If <kondisi_pencabangan2> then
        Begin
            ... {aksi-2}
        end
    else
        Begin
            ... {aksi-3}
        End;
```

2.3.3.4 *CASE-OF*: Pemilihan N kasus

Perintah bersyarat *Case* umumnya digunakan untuk kondisi dengan banyak pencabangan. Syarat pencabangan pada bentuk ini hanya boleh melibatkan satu buah parameter dengan tipe data bukan *Real*. Pemeriksaan kondisi di sini lebih tepat disebutkan dalam hubungan relasi sama dengan (=). Dengan demikian bila parameter bernilai tertentu maka dilakukan suatu aksi terkait, bila bernilai lain maka dilakukan aksi yang lain juga, demikian seterusnya.

```
Case <Parameter> Of
        <nilai_1>  :  <aksi_1> ;
        <nilai_2>  :  <aksi_2> ;
        ...
        <nilai_n>  :  <aksi_n> ;
        Else   <aksi_n+1> ;
End;
```

2.3.4 Struktur Pengulangan

Dalam menyelesaikan masalah, terkadang kita harus melaku suatu proses yang sama lebih dari satu kali. Untuk itu perlu dibuat suatu algoritma pengulangan. *Pascal* memberikan tiga alternatif pengulangan, yaitu dengan *For*, *While*, atau *Repeat*. Masing-masing memiliki karakteristik. Ada dua hal yang penting dalam merancang perintah pengulangan, yaitu:

- Inisialisasi awal.
- Nilai akhir pengulangan atau kondisi berhenti.

2.3.4.1 *FOR-TO-DO*

Pada pengulangan dengan *For*, inisialisasi awal dan kondisi akhir ditentukan dengan menggunakan suatu *variable* kendali yang nilainya dibatasi dalam suatu *range* tertentu. Sintaks untuk perintah ini adalah:

```
For  <variable_kendali>  :=  <nilai_awal>  to  <nilai_akhir> do
        Begin
        …   {aksi}
        End;
```

atau

```
For <variable_kendali> := <nilai_awal> downto <nilai_akhir> do
    Begin
        … {aksi}
    End ;
```

Perbedaan antara *to* dan *downto* adalah pada kondisi nilai awal dan akhir. Pada *to*, nilai awal lebih kecil dari nilai akhir, sedangkan pada *downto* nilai awal lebih besar dari nilai akhir. Perlu diingat, bahwa *variable* kendali harus dideklarasikan dengan tipe data *integer*.

2.3.4.2 *WHILE-DO*

Pada metode pengulangan ini aksi hanya akan diproses bila 'kondisi_pengulangan' terpenuhi, bentuk sintaks dari pengulangan ini adalah:

```
While <kondisi_ pengulangan> do
    Begin
    …  {aksi}
    End ;
```

Selama 'kondisi_pengulangan' bernilai *true* maka aksi akan dilakukan, dan baru akan berhenti setelah 'kondisi_pengulangan' bernilai *false*. Karena 'kondisi_pengulangan' diperiksa pada bagian awal, maka ada kemungkinan aksi tidak pernah dilakukan, yaitu bila 'kondisi_pengulangan' tidak pernah bernilai *true*.

2.3.4.3 *REPEAT – UNTIL*

Metode pengulangan ini juga melakukan pengulangan berdasarkan pemeriksaan kondisi pengulangan. Hanya saja sistem seakan-akan memaksa

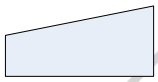
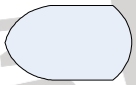
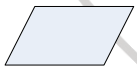
untuk melakukan pengulangan, sampai diketahui adanya ‘kondisi_berhenti’. Bentuk sintaks dari pengulangan ini adalah:

```
Repeat
    …  {aksi}
Until <kondisi_ berhenti>
```

Berlawanan dengan *While*, yang akan memroses aksi hanya bila ‘kondisi_pengulangan’ bernilai *true*, pada pengulangan *Repeat*, sistem akan memroses aksi selama ‘kondisi_berhenti’ bernilai *false*. Dengan demikian pasti aksi akan selalu diproses (minimal satu kali). Pada tipe ini, pengulangan dapat terjadi terus-menerus (tidak pernah berhenti), yaitu bila ‘kondisi_berhenti’ tidak pernah bernilai *true*.

2.4 Diagram Alir (Flowchart)

Diagram alir (*flowchart*) merupakan diagram yang menunjukkan alir di dalam program atau prosedur sistem secara logika (Sulistyo Sudarmono, 2006: 42). Diagram alir digunakan terutama untuk alat bantu komunikasi dan untuk dokumentasi. Tujuan utama dari penggunaan *flowchart* adalah untuk menggambarkan suatu tahapan penyelesaian masalah secara sederhanna, terurai, rapi, dan jelas dengan menggunakan simbol-simbol yang standar. Tahap penyelesaian masalah yang disajikan harus jelas, sederhana, efektif, dan tepat. Dalam penulisan *flowchart* dikenal dua model, yaitu *System Flowchart* dan *Program Flowchart*. Beberapa simbol pada diagram alir serta fungsinya dapat dilihat pada tabel berikut ini:

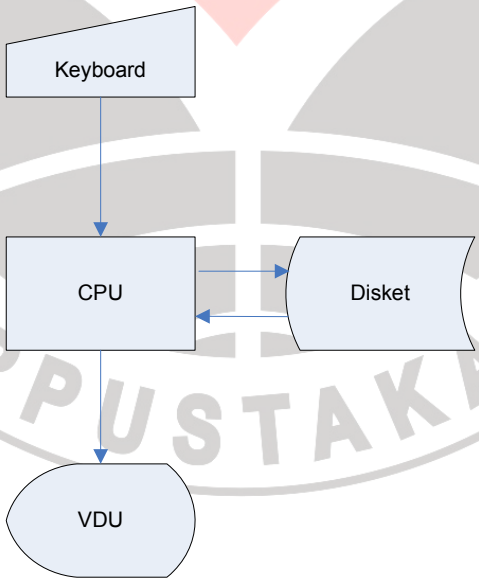
Simbol	Nama	Fungsi
	Pita Magnetik	Digunakan untuk menunjukkan input/output menggunakan pita magnetik.
	*Process*	Digunakan untuk mewakili suatu proses.
	*Keyboard*	Digunakan untuk menunjukkan input yang menggunakan *on-line keyboard* (input manual).
	Magnetic Disk	Digunakan untuk menunjukkan data tersimpan (*stored data*).
	*Punched Paper Tape*	Digunakan untuk menunjukkan input/output kertas berlubang.
	*On Line Storage/ VDU*	Digunakan untuk menunjukkan output yang ditampilkan di monitor.
	*Manual Operation*	Menunjukkan pekerjaan manual
	*Input/Output*	Digunakan untuk memasukkan suatu data dan melihat hasilnya.
	Garis Alir	Digunakan untuk menunjukkan arus dari proses.

Tabel 2.1: Beberapa simbol diagram alir

Diagram Alir Sistem (*System Flowchart*)

Diagram Alir Sistem merupakan diagram alir yang menggambarkan suatu sistem peralatan komputer yang digunakan dalam proses pengolahan data serta hubungan antar peralatan tersebut. Diagram alir sistem ini tidak digunakan untuk menggambarkan urutan langkah untuk memecahkan masalah, tetapi hanya untuk menggambarkan prosedur dalam sistem yang dibentuk.

Dalam menggambar *flowchart* biasanya digunakan simbol-simbol standar, tetapi *programmer* juga dapat membuat simbol-simbol sendiri apabila simbol-simbol yang telah tersedia di rasa masih kurang. Dalam kasus ini, *programmer* harus melengkapi gambar *flowchart* dengan kamus simbol yang digunakannya, agar *programmer* lain dapat mengetahui maksud dari simbol-simbol tersebut. Berikut ini adalah contoh penggunaan Diagram Alir Sistem.

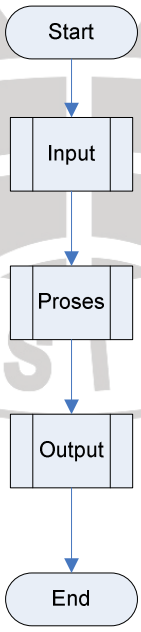


Gambar 2.9: Contoh penggunaan Diagram Alir Sistem

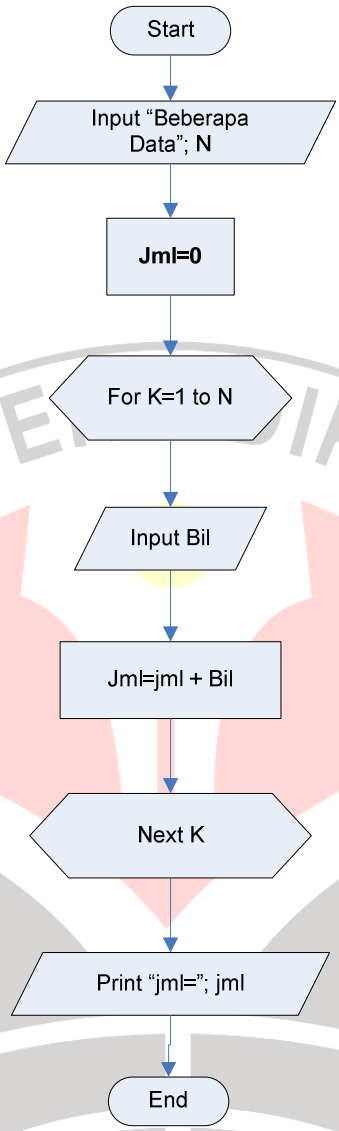
Diagram Alir Program (*Program Flowchart*)

Diagram Alir Program merupakan diagram alir yang menggambarkan urutan logika dari suatu prosedur pemecahan masalah. Dalam menggambarkan *Program Flowchart*, telah tersedia simbol-simbol standar. Tetapi seperti pada *System Flowchart*, *programmer* juga dapat menambahkan simbol-simbol tertentu, yang tentunya *programmer* harus melengkapi penggambaran *Program Flowchart* tersebut dengan kamus simbol.

Pada penggambaran *Program Flowchart* terdapat dua jenis metode, yaitu *Conceptual Flowchart* dan *Detail Flowchart*. *Conceptual Flowchart* menggambarkan tentang alur dari suatu pemecahan masalah secara global saja, sedangkan *Detail Flowchart* menggambarkan alur pemecahan masalah secara rinci. Berikut ini adalah contoh penggunaan *Conceptual Flowchart* dan *Detail Flowchart*.



Gambar 2.10: Contoh penggunaan *Conceptual Flowchart*



Gambar 2.11: Contoh penggunaan *Detail Flowchart*