8 Bahasa komputer dan Pembuatan Program

Setelah membaca bab ini anda diharapkan memahami hal-hal sebagai berikut :

- Bahasa pemrograman secara umum
- Karakteristik bahasa pemrograman
- Tingkat-tingkat bahasa pemrograman
- Bahasa tingkat tinggi
- Pembuatan program
- Pemrograman terstruktur
- Pemrograman berorientasi ke objek

Pendahuluan

Komputer merupakan mesin yang dapat diperintah. Komputer hanya memahami hidup (1) dan mati (0) karena itu bahasa komputer yang pertama kali digunakan adalah bahasa mesin yang tersusun dari bilangan 0 dan 1.

Sulitnya menyusun perintah dalam bahasa mesin, menuntut dikembangkannya bahasa yang lebih sederhana, upaya ini dimulai dengan munculnya bahasa *Assembly* tetapi bahasa *Assembly* masih tetap dirasakan sulit penggunaannya, karena itu terus dikembangkan bahasa dengan dengan tingkatan yang lebih tinggi dan disebut bahasa tingkat tinggi. Bahasa tersebut adalah *Fortran, Cobol, Basic, Pascal* dan lain-lain. Saat ini banyak juga digunakan bahasa tingkat sangat tinggi seperti *Visual Foxpro,Visual Basic, Delphi* dan bahasa bahasa yang tergabung dalam Visual Studio Net seperti Visual Basic.Net,C#.Net,J#.Net.dan C++.Net. yang dapat digunakan untuk pembuatan program berbasis web,windows dan console.

Bagian ini akan menjelaskan perkembangan bahasa-bahasa tersebut bersama dengan bahasa yang lebih tinggi lainnya, sehingga pembaca memiliki pemahaman yang lebih baik tentang bahasa komputer dan pembuatan program.

8.1 Bahasa Pemrograman Secara Umum

Bahasa pemrograman merupakan sekumpulan simbol atau kode yang digunakan untuk menjalankan komputer berdasarkan aturan *syntax* atau *grammar* yang dimilikinya

Bahasa adalah alat komunikasi yang digunakan secara lisan atau tulisan. Bahasa pemrograman merupakan sekumpulan simbol atau kode yang digunakan untuk menjalankan komputer berdasarkan aturan *syntax* atau *grammar* yang dimilikinya. Bahasa pemrograman mempunyai perbendaharaan kata yang sangat terbatas dan fungsi yang sangat spesifik.

8.2 Karakteristik Bahasa Pemrograman

Karakteristik bahasa pemrograman:
- Perintah untuk *input* dan *output*
- Perintah untuk perhitungan
- Perintah mengalihkan kendali
- Perintah untuk memindahkan, menyimpan & mencari *data*

Meskipun berbagai bahasa pemrograman didisain untuk mengatasi berbagai masalah tetapi semuanya mempunyai fungsi dasar pengoperasian komputer yang sama. Secara umum perintah-perintah dalam bahasa pemrograman dapat dikelompokan sebagai berikut:

- **Perintah untuk *input* dan *output (I/O)*,** perintah ini berisi suruhan kepada komputer untuk mengambil *data* dan menyajikan informasi. Penggunaan perintah ini secara mendetail biasanya telah disiapkan dalam buku pedoman penggunaannya.

- **Perintah untuk perhitungan,** merupakan perintah kepada komputer untuk melakukan operasi matematik secara umum seperti penambahan, pengurangan, perkalian dan pembagian.

- **Perintah mengalihkan kendali**, dengan perintah ini dapat dilakukan pengalihan urutan pengolahan dari urutan pengolahan yang normal ke pengolahan tertentu.

- **Perintah untuk memindahan, menyimpan & mencari *data***, Perintah ini digunakan untuk memindahkan, menyimpan dan mencari *data* di memori utama.

Semua bahasa pemrograman umumnya dapat melaksanakan berbagai jenis perintah, sedangkan yang membedakannya adalah ragam bahasa yang digunakan

8.3 Tingkat-Tingkat Bahasa Pemrograman

Meskipun ada berbagai macam bahasa komputer yang berbeda, namun sampai saat ini semuanya dikelompokan kedalam 4 kelompok generasi dan tiga tingkatan. Bahasa yang tingkatannya paling rendah adalah bahasa mesin. Bahasa mesin didasarkan kepada penggunaan angka 1 dan 0 (Bilangan biner). Pengelompokan bahasa tersebut adalah:

Tabel 8.1 Pengelompokan bahasa pemrograman

Generasi	Nama bahasa	Contoh	Tingkatan
Ke empat	Bahasa natural	*Microsoft Sound*	*Sangat Tinggi*
	Bahasa berorientasi ke visual	*Vis.Studio Net,Vis. Foxpro*	
	Bahasa generasi ke 4 (4GL)	*Foxpro,Access*	*Tinggi*
Ke tiga	Bahasa berorientasi ke objek	*C++, Eiffel, Simula*	
	Bahsa berorientasi ke prosedur	*Fortran,Cobol,Basic*	
Ke dua	Bahasa *assembly*	Bahasa *assesmbly*	Rendah
Ke satu	Bahasa mesin	Bahasa mesin 0 dan 1	

8.3.1. Bahasa Tingkat Rendah (Bahasa Mesin)

Bahasa mesin adalah bahasa yang langsung dipahami oleh komputer yang terdiri dari angka 1 dan 0

Bahasa yang langsung bisa dipahami oleh komputer adalah bahasa mesin. Bahasa ini menggunakan bilangan biner (0 dan 1) untuk menunjukkan perintah, alamat memori dan *data* yang diperlukan untuk memecahkan masalah yang spesifik. Semua bahasa mesin menggunakan bilangan biner akan tetapi susunannya seringkali berbeda diantara berbagai jenis komputer. Perintah dalam bahasa mesin terdiri dari dua bagian dimana bagian pertama berisi kode biner yang menunjukan perintah yang harus dilakukan (setiap komputer memiliki kode perintah yang berbeda yang disebut sebagai *opcode*) sedangkan bagian kedua berisi bilangan biner yang menunjukkan bagaimana perintah tersebut harus dijalankan (*operand*). Dalam kondisi tertentu mungkin saja tidak ada *operand* tapi pada kondisi yang lain mungkin terdapat beberapa *operand*. Ketika ada *operand* biasanya berisi *data* itu sendiri atau alamat lokasi di memori dimana *data* tersebut disimpan.

Sebagai contoh perintah dalam bahasa mesin untuk menambahkan nilai 8 untuk jumlah yang ada di *accumulator* (untuk mengingat lagi tentang *accumulator* lihat bab 3) terlihat dalam mesin IBM atau sejenisnya adalah sebagai berikut :

00000 01	00001000
ADD	8

Pada awal penggunaan komputer untuk pengolahan *data*, setiap perintah, alamat memori, dan *data* harus disusun dalam bentuk bahasa mesin di atas. Hal ini selain luar biasa sulitnya, juga membutuhkan waktu, dan sangat mudah terjadi kesalahan.

8.3.2. Bahasa Tingkat Rendah (Bahasa Assembly)

Bahasa *assembly* menggunakan *mnemonic* (simbol) tingkatannya lebih tinggi dari bahasa mesin tapi tetap merupakan bahasa tingkat rendah.

Bahasa ini tingkatannya lebih tinggi dari bahasa mesin tetapi masih tergolong kedalam bahasa tingkat rendah. Bahasa ini berisi beberapa singkatan yang mudah untuk diingat yang disebut sebagai *mnemonic* untuk menggantikan bilangan 0 dan1 dalam bahasa mesin. Sebagai contoh bahasa *assembly* menggunakan ADD, MUL, dan STO untuk penambahan, perkalian dan penyimpanan menggantikan bilangan biner 0 dan 1.

Dengan sistem pengalamatan secara simbolik, programer dapat memanggil *data* TOTAL yang tersimpan pada alamat memori tertentu tanpa harus lagi menggunakan kode 00001001. Sebagai

gantinya programmer menggunakan *variable*, suatu nama yang mengandung arti dan berfungsi untuk menunjukan alamat memori tertentu. Dengan menggunakan variabel-variabel programer dapat menggunakan bilangan *desimal, octal* dan *hexadesimal* untuk menunjukan jenis *data* dan tidak lagi harus menggunakan bilangan biner. Bagi programer saat itu bahasa *assembly* lebih mudah untuk digunakan.

Tidak seperti bahasa mesin yang bisa langsung dijalankan, bahasa *assembly* harus diterjemahkan kedalam bahasa mesin sebelum dapat dijalankan. Program khusus bernama *assembler* akan menterjemahkan bahasa *assembly* untuk jenis komputer tertentu kedalam bahasa mesin. Program yang ditulis oleh programer dalam bahasa *assembly* tersebut disebut sebagai *source code*. *Source code* ini setelah diterjemahkan kedalam bahasa mesin namanya menjadi *object code.*

Gambar 8.1 Contoh bahasa assembly untuk IBM PC

```
; Program ini fungsinya untuk menjumlahkan dua buah angka, dikurangi dengan angka yang ketiga dan
; menempatkan hasilnya di memori
;
DATA  SEGMENT                          ; mendefinisikan data yang digunakan
;
NUMBER1 Dw      20                     ; Menyimpan angka pertama (20) ke dalam memori
NUMBER2 Dw      35                     ; Menyimpan angka kedua (35) ke dalam memori
NUMBER3 Dw      12                     ; Menyimpan angka ketiga (12) ke dalam memori
RESULT   Dw     ?                      ; Memesan tempat di memori untuk menyimpan hasil
;
DATA   END
;
CODE  SEGMENT                          ; Perintah untuk melakukan pemrosesan
        ASSUME CS: CODE, DS: DATA      ; Perintah kepada komputer untuk mencari lokasi code
                                       ; dan data yang didefinisikan
;
        MOV   AX, NUMBER1              ; Memindahkan angka pertama kedalam penampungan
        MOV   BX, NUMBER2              ; Memindahkan angka kedua kedalam Register BX
        ADD   AX, BX                   ; Menggabungkan angka kedua dngan nilai dalam penampungan
        MOV   BX, NUMBER3              ; Memindahkan angka ketiga kedalam Register BX
        SUB   AX, BX                   ; Mengurangi hasil penjumlahan pada AX dengan angka ketiga
        MOV   RESULT,AX                ; Memindahkan nilai akhir dari penampungan kedalam memori
;
CODE    ENDS
          END
```

8.3.3. Bahasa Tingkat Tinggi (Generasi ketiga)

Bahasa tingkat tinggi perintah-perintahnya menggunakan simbol yang mendekati bahasa manusia.

Sampai pertengahan tahun 1990-an, bahasa yang paling banyak digunakan adalah bahasa tingkat tinggi, bahasa ini perintah-perintahnya menggunakan simbol-simbol yang mendekati bahasa yang dapat dipahami oleh manusia secara umum. Bahasa tingkat tinggi dikembangkan dengan tujuan untuk membantu programer agar lebih menfokuskan perhatiannya kepada pekerjaan yang harus diselesaikan dan mengurangi beban perhatiannya ke masalah mesin yang sangat spesifik dan detail. Bahasa tingkat tinggi lebih abstrak daripada bahasa tingkat rendah. Bahasa ini mempermudah programer untuk melakukan pengolahan *data* yang kompleks karena menggunakan lebih sedikit perintah.

8.3.4. Compillers dan Interpreters

Seperti halnya program yang dibuat dengan bahasa *asssembly*, program yang dibuat dengan bahasa tingkat tinggi pun harus diterjemahkan ke dalam bahasa mesin sebelum dapat dijalankan oleh komputer. Tidak seperti bahasa *assembly*, bahasa tingkat tinggi memiliki keuntungan yaitu tidak tergantung kepada mesin yang digunakan *(portability)* artinya program yang ditulis dengan bahasa tingkat tinggi dapat dijalankan pada berbagai jenis komputer yang memiliki program untuk menterjemahkannya. Kebanyakan bahasa tingkat tinggi dapat dijalankan setelah diterjemahkan kedalam bahasa mesin dengan program khusus yang disebut sebagai *compiller*. *Compiller* menterjemahkan kode asal (*source code*) bahasa tingkat tinggi ke bahasa mesin (*object code*).

Compiller menterjemahkan kode asal kedalam bahasa mesin secara langsung.

Interpreter menterjemahkan kode asal program kedalam bahasa mesin kode per kode.

Source code bahasa tingkat tinggi dapat ditulis diberbagai jenis komputer seperti IBM dan yang lainnya dan berbagai jenis prosesor seperti prosesor buatan Intel, AMD, IBM dan Motorola.

Interpreter memiliki fungsi yang sama dengan *compiller* bedanya *interpreter* menterjemahkan perintah per perintah dan digunakan saat pembuatan program untuk menguji penggunaan perintah-perintah tersebut. *Compiller* digunakan untuk menterjemahkan secara langsung satu atau banyak *file* karena itu digunakan pada saat pembuatan program telah selesai atau untuk menguji program.

8.3.5. Keuntungan Bahasa Tingkat Tinggi

Bahasa tingkat tinggi memberikan beberapa keuntungan dibandingkan dengan bahasa *assembly* dan bahasa mesin yaitu mudah dipelajari, ditulis, dikoreksi dan diperbaiki. Lebih jauh lagi dapat dikatakan bahwa tingkat abstraksi bahasa ini sangat tinggi sehingga banyak program-program besar yang kompleks dapat ditulis dengan lebih mudah. Dengan menggunakan bahasa tingkat tinggi selain kesalahan pembuatan program akan lebih mudah dideteksi dan dikoreksi, prinsip struktur pemrograman bahasa tingkat tinggi juga mudah untuk dipelajari dan dipahami. Akhirnya, tidak saja membuat pemrograman lebih mudah dilakukan tapi juga program yang dihasilkan menjadi lebih berguna.

Keuntungan bahasa tingkat tinggi mudah dipelajari, ditulis, dikoreksi dan diperbaiki.

8.4 Jenis Bahasa Tingkat Tinggi

Bahasa tingkat tinggi dibagi menjadi 3 (tiga) kelompok yaitu, bahasa tingkat tinggi berorientasi prosedur, bahasa tingkat tinggi berorientasi ke objek dan bahasa tingkat tinggi generasi keempat (4 GL)

Bahasa tingkat tinggi berorientasi prosedur sudah jarang dipakai saat ini, bahasa-bahasa tingkat tinggi tersebut adalah *Fortran, Cobol, Basic, Pascal, Modula-2, C, Ada* dan *RPG*. Semua bahasa pemrograman tersebut diatas sangat penting keberadaannya karena bahasa tersebut beberapa tahun yang lalu banyak

sekali digunakan dan tersebar luas diseluruh dunia. Secara ringkas bahasa-bahasa tersebut dapat diuraikan sebagai berikut:

8.4.1. Bahasa Tingkat Tinggi Berorientasi Prosedur

Fortran

Fortran merupakan bahasa tingkat tinggi yang pertama kali digunakan dan berorientasi pada matematik.

Fortran atau *formula translator*, dikembangkan oleh IBM pada tahun 1957. Didesain oleh para ahli terkemuka dibidang komputer dan matematik. *Fortran* merupakan bahasa tingkat tinggi komersil yang pertama kali digunakan, *Fortran* memiliki tingkat penggunaan yang luas dari kelompok ilmuwan

Penampilan

Fortran adalah bahasa pemrograman yang berorientasi ke matematik sehingga mampu untuk melakukan perhitungan yang komplek. *Fortran* mampu mengolah *data* dengan sangat cepat dan akurat. *Fortran* merupakan salah satu bahasa yang mempunyai standar tinggi dan mudah untuk digunakan sehingga menjadi sangat terkenal diantara para ilmuwan.

Format program

Fortran dikembangkan ketika kartu berlobang masih merupakan alat utama untuk memasukan *data* dalam skala menengah. Akibatnya, program *fortran* memiliki format tetap yaitu 80 karakter per baris. Program *fortran* berisi serangkaian perintah yang diawali dengan penentuan posisi baris. Setiap program diakhiri dengan perintah *END*. Perintah untuk menjalankan sub perintah lain dapat dilakukan dengan terlebih dahulu memberi nomor baris tertentu sehingga dapat dijadikan sebagai acuan. Program *Fortran* yang besar selalu dibagi dalam sub-sub program yang lebih kecil yang dikenal sebagai *subroutin* dan *function.*

- ***Subroutine*** adalah serangkaian pernyataan atau modul program untuk menjalankan perintah-perintah tertentu. *Subroutin* dapat ditempatkan bersamaan dengan program utama atau dapat ditempatkan dalam program tersendiri yang penting dapat dikenali oleh program utama.

- ***Function*** menyerupai *subroutine* tapi menghasilkan satu nilai yang dipergunakan oleh program utama.

Jenis Data

Jenis *data* yang dapat diproses oleh *Fortran* adalah jenis *data* matematik, misalnya *Integer*, *real*, dan c*omplex*, dan juga *data-data* yang memiliki nilai logis (*logical value*) yang berisi *True* atau *False*. *Fortran* juga memiliki kemampuan untuk mengolah *data* dalam bentuk *double-precison* (nilai angka dalam digit dua kali le-

bih banyak dari nilai angka yang biasa digunakan serta lebih akurat)

Versi *Fortran* yang lebih modern memasukan juga kedalam ciri *data*nya jenis karakter yang digunakan. *Fortran* juga memungkinkan digunakannya program *array. Array* adalah daftar *data* dalam bentuk matrik yang akan di identifikasi dalam satu nama.

Perintah pengendali

Pada awalnya *Fortran* hanya memiliki tiga perintah utama yaitu *GOTO, IF*, dan *DO*. *Fortran* versi yang lebih baru misalnya *fortran 77, 80* dan *WATFIV-s* memiliki struktur pengendalian yang lebih luas dengan adanya perintah *WHILE-ENDWHILE* dan *LOOP-ENDLOOP*.

Gambar 8.2 Contoh program FORTRAN

```
C PROGRAM NILAI RATA-RATA KELULUSAN
C
C Program ini menghitung nilai kelulusan rata-rata seorang murid, dengan
C memberikan daftar nama lengkap, SKS dan Nilai
C
      INTEGER HOURS, POINTS, TPOINTS, THOURS
      REAL GPA
      CHARACTER CNAME*20, GRADE*1
C
      TPOINTS = 0
      THOURS = 0
C
      WRITE (*, 10)
10    FORMAT (' ', 'Silahkan anda isi nama lengkap, jumlah SKS, dan Nilai: ')
      WRITE (*, 20)
20    FORMAT ('  ', 'Nama lengkap: '\) ;
      READ (*, 25) CNAME
25    FORMAT (A20)
      WRITE (*, 30)
30    FORMAT ('  ', 'Jumlah SKS:  '\) ;
      READ (*, 35) HOURS
35    FORMAT (I1)
      WRITE (*, 40)
40    FORMAT ('  ', 'Nilai: '\)
      READ (*, 45) GRADE
45    FORMAT (A1)
C     Dalam penggunaan program secara rutin, data tersebut harus di cek, untuk validasi
C
C     IF dan GOTO digunakan untuk pembuatan equivalent dari putaran WHILE
50    IF (CNAME .EQ. '                    ') GOTO 60
C     Dengan kata lain, bahwa WHILE CNAME tidak kosong, juga seperti yang berikut ini:
      IF ((GRADE .EQ.  'A') .OR. (GRADE .EQ. 'a')) THEN
            POINT = HOURS * 5
      ELSEIF ((GRADE .EQ.  'B') .OR. (GRADE .EQ. 'b')) THEN
            POINT = HOURS * 4
      ELSEIF ((GRADE .EQ.  'C') .OR. (GRADE .EQ. 'c')) THEN
            POINT = HOURS * 3
      ELSEIF ((GRADE .EQ.  'D') .OR. (GRADE .EQ. 'd')) THEN
            POINT = HOURS * 2
      ELSE
            POINT = HOURS
      ENDIF
```

Lanjutan gambar 8.2 contoh program FORTRAN

```
            TPOINT = TPOINT + POINTS
            THOURS = THOURS + HOURS
            WRITE (*, 20)
            READ (*, 25) CNAME
            WRITE (*, 30)
            READ (*, 35) HOURS
            WRITE (*, 40)
            READ (*, 45) GRADE
C           Kembali, data ini juga harus dicek untuk validasi
            GOTO 50
60          CONTINUE
C
            IF (THOURS .GT. 0) THEN
                  GPA = FLOAT (TPOINT) / THOURS
            ELSE
                  GPA = 0
            ENDIF
C
            WRITE (*, 70) TPOINT, THOURS, GPA
70          FORMAT ('0', 'Total nilai = ', I3/,
            +           '  ', 'Jumlah SKS  = ', I3/,
            +           '  ', 'Nilai rata-rata    = ', F4.2)
C
            STOP
            END
```

COBOL

COBOL merupakan bahasa yang berorientasi ke bisnis

COBOL merupakan singkatan dari *Common Business Oriented Language*. *COBOL* pertamakali diperkenalkan pada tahun 1960 dan dikembangkan oleh suatu komite yang terdiri dari pemerintah dan dunia industri yang diwakili oleh departemen pertahanan Amerika Serikat. Komite tersebut diberi nama *Conference on Data Systems Languages (CODASYL)*. *COBOL* merupakan bahasa tingkat tinggi pertama yang cocok untuk mengolah *data* bisnis. Pertama kali digunakan oleh pemerintah Amerika Serikat, *COBOL* akhirnya digunakan di seluruh dunia sebagai bahasa tingkat tinggi untuk aplikasi bisnis.

Penampilan

COBOL sangat mudah menangani *input* dan *output data alphanumerik* dalam jumlah besar dengan perhitungan yang tidak begitu komplek. Kekuatan *COBOL* yang lain adalah menggunakan *syntax* seperti bahasa inggris dan tidak tergantung kepada mesin yang digunakan. Ini berarti bahwa *COBOL* dapat dijalankan di berbagai mesin yang bebeda dengan sedikit modifikasi.

Format program

Semua program *COBOL* memiliki format yang unik. Hal mendasar dalam program *COBOL* adalah *Sentence* yang diakhiri dengan titik. *Sentence* membentuk *paragraphs, paragraph* membuat *sections* dan *section* membuat *divisions*. Setiap program *COBOL* harus memiliki empat devisi, yaitu: *IDENTIFICATION DEVISION* na-

ma dari program dan pembuatnya, tanggal penulisan,dan dokumentasi lain. *ENVIRONMENT DEVISION* menunjukkan komputer yang menjalankan program dan daftar nama *input* dan *output file* yang harus digunakan oleh program.*DATA DEVISION* merupakan daftar dan gambaran format yang digunakan untuk *variable, record* dan *file* yang digunakan oleh program. *PROCEDURE DEVISION* berisi perintah untuk melakukan pengolahan *data*.

Jenis data

Aturan utama dari *COBOL* adalah *data* harus disimpan dalam bentuk karakter kecuali *data* tersebut ditentukan dalam bentuk numerik. Kondisi ini mencerminkan bahwa *COBOL* berorientasi kepada pengolahan *data* dan menyederhanakan penanganan item yang umum seperti nama, alamat, nomor ID dan lain-lain. *COBOL* memungkinkan penggunaan sebuah *variable* sampai dengan struktur *record* yang komplek dalam sebuah tabel.

Perintah pengendali

COBOL memiliki tiga perintah pengendali yaitu *GOTO* perintah untuk pindah tanpa kondisi, *IF-ELSE* perintah untuk pindah dengan kondisi, dan *PERFORM* merupakan perintah untuk mengulang membaca *(looping)* sampai kondisi tertentu dipenuhi .

Gambar 8.3 Contoh program COBOL

```
IDENTIFICATION DIVISION
* Divisi ini berisi rincian dokumentasi umum

PROGRAM-ID.          NILAI-RATA2-KELULUSAN.
AITHOR.              AZHAR SUSANTO.
DATE-WRITTEN.        1 JANUARI, 2001.
REMARKS              PROGRAM INI MENGHITUNG NILAI KELULUSAN RATA-RATA
                     SEORANG MURID, DENGAN  MEMBERIKAN DAFTAR NAMA
                     LENGKAP, JUMLAH SKS DAN KETERANGAN KELULUSAN.

ENVIRONMENT DIVISION
* Divisi ini berisi rincian spesifikasi komputer

CONFIGURATION SECTION.
SOURCE-COMPUTER.     IBM.
OBJECT-COMPUTER      IBM.

DATA DIVISION.
* Divisi ini menerangkan tentang format data.

WORKING-STORAGE SECTION.
01      NAMA-LENGKAP                PIC X (20).
01      SKS                         PIC 9.
01      HURUF-MUTU                  PIC X
01      NILAI                       PIC 999.
01      NILAI-TOTAL                 PIC 999.
01      JUMLAH-SKS                  PIC 999.
01      NILAI-RATA-RATA             PIC 9V99.

PROCEDURE DIVISION.
* Divisi ini berisi perintah-perintah pemrosesan sebenarnya.
```

Lanjutan gambar 8.3 contoh program COBOL

```
MAIN-PROGRAM
        MOVE 0 TO NILAI-TOTAL
        MOVE 0 TO JUMLAH-SKS
        DISPLAY 'Silahkan anda isi nama lengkap, jumlah SKS, dan huruf mutu : '
        DISPLAY 'Nama lengkap: '
        ACCEPT NAMA-LENGKAP
        DISPLAY 'Jumlah SKS : '
        ACCEPT SKS
        DISPLAY 'Huruf mutu: '
        ACCEPT HURUF-MUTU
*  Dalam penggunaan program secara rutin, data tersebut harus dicek untuk validasi
        PERFORM PROCESS-INPUT
                UNTIL NAMA-LENGKAP = '                    '.

        IF JUMLAH-SKS > 0
            COMPUTE NILAI-RATA-RATA = NILAI-TOTAL / JUMLAH-SKS
        ELSE
            MOVE 0. 00 TO NILAI-RATA-RATA

        DISPLAY 'Nilai total kelulusan          = '     NILAI-TOTAL.
        DISPLAY 'Jumlah SKS                     = '     JUMLAH-SKS.
        DISPLAY 'Nilai rata-rata kelulusan      = '     NILAI-RATA-RATA.
        STOP RUN.

PROCESS-INPUT.
        IF HURUF-MUTU = 'A' OR HURUF-MUTU = 'a'
          COMPUTE  NILAI  = SKS * 5
        ELSE
                IF HURUF-MUTU = 'B' OR HURUF-MUTU = 'b'
                  COMPUTE  NILAI  = SKS * 4
                ELSE
                        IF HURUF-MUTU = 'C' OR HURUF-MUTU = 'c'
                            COMPUTE  NILAI = SKS * 3
                        ELSE
                                IF HURUF-MUTU = 'D' OR HURUF-MUTU = 'd'
                                  COMPUTE  NILAI = SKS * 2
                                ELSE
                                  COMPUTE  NILAI = SKS.
        ADD NILAI TO NILAI-TOTAL
        ADD SKS TO JUMLAH-SKS
        DISPLAY 'Nama lengkap:  '
        ACCEPT NAMA-LENGKAP.
        DISPLAY 'Jumlah SKS:  '
        ACCEPT SKS.
        DISPLAY 'Huruf mutu:  '
        ACCEPT HURUF-MUTU
* Kembali, data ini juga harus dicek untuk validasi
```

BASIC

BASIC merupakan bahasa interaktif yang sederhana tetapi paling banyak digunakan.

BASIC merupakan singkatan dari *Beginner's All-purpose Symbolic Instruction Code.* Bahasa ini dikembangkan oleh DR.John Kemeny dan Dr. Thomas Kurtz di Dartmount College pada tahun 1965. Awalnya dirancang hanya sebagai bahasa interaktif yang sederhana dan dapat digunakan pada komputer dengan sistem *time sharing*. Akan tetapi *BASIC* menjadi bahasa tingkat tinggi yang paling banyak digunakan di komputer pribadi pada periode tersebut.

Penampilan

BASIC tidak banyak menggunakan tampilan yang menyolok, tetapi lebih cenderung menunjukan kesederhanaan, disamping itu *BASIC* lebih mudah dalam mempergunakan dan mempelajarinya. Setelah beberapa jam belajar dan berlatih anda sudah dapat menulis dan menjalankan fungsi program *BASIC*. Jika dibandingkan dengan kedua bahasa sebelumnya yaitu *FORTRAN* dan *COBOL*, *BASIC* lebih agak mirip dengan *FORTRAN*. Sebagai bahasa interaktif, *BASIC* lebih sering diartikan sebagai penerjemah yang cukup ringkas. *BASIC* dapat juga digunakan untuk menyelesaikan tugas perhitungan dan mengolah *data Alphanumeric*. *BASIC* dipergunakan secara luas oleh para pemakai diberbagai jenis komputer kecil.

Format Program

Format program *BASIC* sifatnya terbuka dan sangat sederhana. Sebuah program yang terdiri dari rangkaian perintah-perintah yang diakhiri dengan pernyataan *END*. Perintah-perintah tersebut ditulis dalam bentuk format bebas, sehingga satu atau lebih perintah dapat ditulis dalam satu baris. Beberapa versi *BASIC* lainnya memerlukan nomor pada seluruh baris. Program *BASIC* dapat terbagi lagi kedalam beberapa subprogram, dan kemampuan ini bukan suatu hal yang baru disebagaian besar bahasa tingkat tinggi lainnya.

Jenis Data

Seluruh *syntax BASIC* berpegang pada bilangan-bilangan *riel* dan *string* atau gabungan dari *character-character*. Selain dari itu semua *syntax BASIC* didukung *array* yang berisi bilangan-bilangan *riel* dan *character*. Kumpulan kecil jenis-jenis *data* itu sengaja dibentuk untuk menegaskan bahwa *BASIC* secara keseluruhan sangat sederhana. Aplikasi *software house* pembuat *BASIC (Microsoft)* menambah perintah-perintah baru dari aslinya seperti menambah *integer* dan *double-precision number*.

Gambar 8.4 Contoh program BASIC

```
010 REM PROGRAM NILAI RATA-RATA KELULUSAN
020 REM
030 REM Program ini adalah menghitung nilai rata-rata kelulusan para murid, dan
040 REM memberikan daftar nama lengkap, jumlah SKS, dan Nilai.
050 REM
060 PRINT  “Silahkan anda masukan nama lengkap, jam semester, dan surat kelulusan:  “
070 INPUT; “Nama lengkap:  “, NAMA.LENGKAP$
080 INPUT; “      Jumlah SKS: “ SKS%
090 INPUT; “      Nilai: “ NILAI$
095 REM Dalam penggunaan program secara rutin, data ini harus dicek untuk validasi
100 REM
110 WHILE NAMA.LENGKAP$ <>  ‘ ‘
120         IF (NILAI$ = “A”) OR (NILAI$ = “a”) THEN
                JUMLAH.NILAI% = SKS% * 5 : GOTO 170
```

Lanjutan Gambar 8.4 Contoh program BASIC

```
130         IF (NILAI$ = "B") OR (NILAI$ = "b") THEN
               JUMLAH.NILAI % = SKS% * 4 : GOTO 170
140        IF (NILAI$ = "C") OR (NILAI$ = "c") THEN
              JUMLAH.NILAI% = SKS% * 3 : GOTO 170
150         IF (NILAI$ = "D") OR (NILAI$ = "d") THEN
               JUMLAH.NILAI% = SKS% * 2 : GOTO 170
160         IF (NILAI$ = "F") OR (NILAI$ = "f") THEN
               JUMLAH.NILAI% = SKS%
170         NILAI.TOTAL%  = NILAI.TOTAL% + JUMLAH.NILAI%
180         JUMLAH.SKS%    = JUMLAH.SKS% + SKS%
190         INPUT; "Nama lengkap:  ", NAMA.LENGKAP$
200         INPUT; "      Jumlah SKS: " SKS%
210         INPUT; "      Nilai: " NILAI$
220         WEND
230  IF JUMLAH.SKS% > 0 THEN NILAI.RATA.RATA = NILAI.TOTAL%/JUMLAH.SKS%
240 REM
250 PRINT
260 PRINT "Total nilai kelulusan       = "; NILAI.TOTAL%
260 PRINT "Total SKS                   = "; JUMLAH.SKS%
260 PRINT "Nilai rata-rata kelulusan  = "; NILAI.RATA.RATA
```

Perintah pengendali

Perintah pengendali *BASIC* mirip dengan apa yang telah dijelaskan pada *FORTRAN* diantaranya adalah perintah *GOTO*, *IF-THEN* dan putaran *FOR-NEXT*. Sebagian besar *BASIC* versi modern juga memiliki putaran *WHILE-WEND* dan benar-benar harus memperhitungkan struktur bahasa pemrogramannya. Mulai diperkenalkan pada akhir 1984, *BASIC* menyimpan seluruh kekuatan bahasa aslinya tetapi lebih cepat, lebih kuat, lebih mudah dibawa dan banyak menyajikan tampilan-tampilan baru yang memungkinkan untuk menyusun suatu program yang dapat memanipulasi *data* grafik, matematik dan teks.

Pascal

Pascal merupakan bahasa yang juga pelajaran bagaimana menulis program dengan baik.

Pascal diciptakan oleh Niklaus Wirth seorang ahli komputer dari Swiss, diberi nama *Pascal* merupakan penghormatan terhadap ahli matematika dan filosof asal Prancis abad ke 17, Blaise Pascal. Diperkenalkan pada tahun 1971, *Pascal* pada awalnya dimaksudkan untuk mengajarkan konsep penyusunan program dengan mempergunakan bahasa tingkat tinggi. *Pascal* untuk ukuran saat itu sangat sederhana, mewah, dan merupakan perwujudan dari prinsip penyusunan program yang baik sehingga cukup terkenal dikalangan para pelajar dan ilmuwan dibidang komputer, programer dan para pengguna *micro computer* lainnya.

Penampilan

Diantara bahasa-bahasa pemrograman komputer yang tersebar luas, saat itu *Pascal* merupakan salah satu bahasa pemrograman yang terbaik untuk penyusunan program, penggunaannya sangat

sederhana, dan mudah dipelajari. *Pascal* selain telah mendorong kebiasaan menyusun program yang baik, *Pascal* juga sangat serbaguna dan bahasanya sangat membantu para pengguna terhindar dari *error* saat pembuatan program. Sebagai sebuah produk saat itu *Pascal* menjadi terkenal dan menjadi bahasa pemrograman alternatif dari *BASIC* pada komputer pribadi.

Kelemahan utama dari *Pascal* adalah terbatasnya fasilitas *input/output* dan terbatasnya kemampuan dalam memanipulasi *data*. Hal ini telah membuat *Pascal* agak kurang menarik sehingga kurang banyak yang menggunakannya dalam aplikasi pengolahan *data* tradisional.

Format Program

Format program yang digunakan bahasa *Pascal* sangat baik. Nama program *data*ng pada bagian awal, diikuti oleh deskripsi *data* kemudian berbagai sub program yang digunakan dan terakhir kode mewakili program utama. Organisasi internal dari setiap subprogram *Pascal* memiliki dua jenis sub program yakni *Procedure* dan *Function* yang tercermin dari format program ini. Unit pokok dari program Pascal adalah perintah-perintahnya yang dapat dimasukan kedalam format bebas akat tetapi harus dipisahkan oleh sebuah titik koma (;).

Jenis Data

Pascal memberikan tempat berlatih sederhana dan sangat membantu dalam penyusunan jenis *data* bagi para pengguna, disamping mendukung programer. Didalam *Pascal* terpasang berbagai jenis *data* sederhana termasuk *integer*, bilangan *riel*, nilai *Boolean* (nilai *true* atau *false*), dan *character*, yang semuanya adalah jenis-jenis yang dapat disusun dalam bentuk *arrays, sets, records,* dan *files*.

Gambar 8.5 Contoh program Pascal

```
(* PROGRAM NILAI RATA-RATA KELULUSAN *)

(* Program ini menghitung nilai rata-rata setiap murid, dengan memberikan daftar nama *)
(* lengkap, jumlah SKS, dan Nilai total                                              *)

PROGRAM  Nilai_ratarata_kelulusan (INPUT, OUTPUT);

VAR
      Nama_lengkap          : STRING [20] ; (* Ini adalah bagian pernyataan tentang*)
      SKS                   : INTEGER ;     (* jenis data dari setiap variabel yang *)
      Nilai                 : CHAR ;        (* digunakan dalam program *)
      Jumlah_Nilai          : INTEGER ;
      Nilai_total           : INTEGER ;
      Jumlah_SKS            : INTEGER ;
      Nilai_rata_rata       : REAL;

BEGIN   (* nilai_ratarata_kelulusan *)

      Nilai_total      : = 0;      (* penginisialisasian untuk menampungan nilai 0 *)
      Jumlah_SKS   : = 0;
```

Lanjutan Gambar 8.5 Contoh program Pascal

```
      WRITELN (' Silahkan isi nama lengkap, SKS, dan Nilai: ' ) ;
      WRITE  ('Nama lengkap: ');                    READ (nama_lengkap) ;
      WRITE  ('            SKS:  ');                READ (SKS) ;
      WRITE  ('              Nilai:  ');            READLN (Nilai) ;
      (* dalam pengunaan program secara rutin, datanya harus dicek untuk validasi *)
WHILE nama_lengkap <> '' DO
      BEGIN   (* WHILE *)
            CASE Nilai OF
                  'A', 'a' : Jumlah_nilai: = SKS * 5;
                  'B', 'b' : Jumlah_nilai: = SKS * 4;
                  'C', 'c' : Jumlah_nilai: = SKS * 3;
                  'D', 'd' : Jumlah_nilai: = SKS * 2;
                  'F', 'f'  : Jumlah_nilai: = SKS;
            END;    (* CASE *)

         Nilai_Total: = nilai_total + Jumlah_nilai;
         Jumlah_SKS : = Jumlah_SKS + SKS;

         WRITE ('Nama lengkap : ')                  READ (nama_lengkap);
         WRITE ('Jumlah SKS: ')                     READ (SKS);
         WRITE ('Nilai : ')                         READ (Nilai);
         (* Kembali, data ini harus dicek untuk validasi *)
END;    (* WHILE*)

IF THEN
      Nilai_rata_rata : = nilai_total / Jumlah_SKS
ELSE
      Nilai_rata_rata: = 0;
      WRITELN
      WRITELN ('Nilai total kelulusan       = ', nilai_total);
      WRITELN ('SKS                         = ', SKS);
      WRITELN ('Nilai rata-rata kelulusan = ', nilai_rata_rata);
END      (* nilai_rata_rata *)
```

Perintah Pengendali

Pascal dibekali kemewahan dengan struktur pengendalian yang baik dalam penyusunan program. Sebagai pegangan, pengendalian untuk pemindahan bersyarat pada *Pascal* menggunakan perintah *IF-THEN-ELSE*, perintah *CASE* merupakan perintah yang banyak sekali pilihannya. *Pascal* memiliki tiga macam struktur *looping* (putaran/pembacaan) yang terdiri dari perintah *WHILE*, perintah *REPEATE-UNTIL*, dan perintah *FOR*. Meskipun *Pascal* mempunyai perintah *GOTO*, tetapi kelihatannya jarang sekali digunakan dalam pemrograman.

C

C merupakan bahasa yang dapat digunakan untuk membangun sistem operasi dan menggantikan bahasa *assembly*

C mulanya dirancang di laboratoruim Bell sebagai bagian dari sistem operasi UNIX. Pada tahun 1972 Dennis Ritchie merancang dan mengimplementasikan C pada mini komputer DEC PDP-11. Dia berkeinginan agar C menjadi pengganti dari bahasa *Assembly* sebagai alat untuk membangun sistem operasi. C sangat serbaguna dan mudah dijalankan serta memungkinkan para pengguna untuk mengerjakan tugas-tugas normal yang biasa dilakukan dengan bahasa *Assembly*. Teknik penyusunan programnya

mirip dan memiliki kecocokan dengan *Pascal*. C cukup terkenal dikalangan para pengguna komputer pribadi, juga biasa digunakan pada sistem komputer ukuran besar.

Penampilan

C berbeda dengan kebanyakan bahasa-bahasa tingkat tinggi lainnya, disamping itu C merupakan bagian tak terpisahkan dari lingkungan kerja komputer yaitu *hardware* dan *software*. C menganjurkan kepada programmer untuk memakai ilmu pengetahuan mereka tentang fungsi komputer. C memberikan kekuatan yang besar kepada programer, oleh karena itu C sering dilukiskan sebagai sosok yang kuat *(Robust)*.

C menjadi bahasa pilihan bagi para programer profesional yang sedang mengembangkan sistem operasi, dan *software* aplikasi seperti *word processor, spreadsheet,* dan *database managers.*

Format Program

Program C terdiri dari sebuah modul utama yang harus selalu diberi nama *Main,* dan sebuah nomor subprogram atau *functions* (fungsi-fungsi). Modul utama dan setiap fungsi harus dimulai dengan penjelasan dari seluruh item-item *data* yang digunakan. Setiap perintah harus diakhiri dengan titik koma (;). Fungsi-fungsi *(function)* ditempatkan setelah modul utama pada setiap urutan.

Jenis-Jenis Data

Jenis-jenis *data* yang fundamental atau pokok dari C yaitu *character, integer,* bilangan *riel,* bilangan *riel double-precision*, dan *pointer. Pointer* adalah nilai yang menunjukan alamat dari sebuah item *data*. C juga mendukug *arrays* dan konsep khusus dengan nama *structure,* yang memungkinkan untuk menyatukan *(Grouping)* jenis-jenis item *data* yang tidak sama.

Gambar 8.6 Contoh program C

```
/* PROGRAM NILAI RATA-RATA KELULUSAN */
/* Program ini menghitung nilai rata-rata setiap murid, dengan memberikan daftar nama */
/* lengkap, jumlah SKS, dan Nilai total                                                */

#include <stdio. H>
main ()
{
      char  nama_lengkap [20] ;
      char  nilai ;
      int     sks ;
      int     jumlah_nilai ;
      int     nilai_total = 0 ;
      int     jumlah_sks = 0 ;
      float  nilai_rata_rata = 0.0 ;

      printf  (“Masukan nama lengkap, SKS dan Nilai: “) ;
      printf  (“\nNama lengkap: “) ;
```

Lanjutan gambar 8.6

```
        scanf ("%{^\n}", nama_lengkap) ;
        if (nama_lengkap[0] !='\0') {
            printf  ("SKS: ") ;
            scanf  ("%d", &sks) ;
            printf  ("Nilai: ") ;
            scanf  ("%1s", &nilai) ;
            nilai = toupper (nilai) ;
        }
        /*dalam penggunaan program secara rutin, datanya harus dicek untuk validasi*/
        while (nama_lengkap [0] ! = '\0') {
            switch (nilai) {
                case 'A' : jumlah_nilai = sks * 5; break;
                case 'B' : jumlah_nilai = sks * 4; break;
                case 'C' : jumlah_nilai = sks * 3; break;
                case 'D' : jumlah_nilai = sks * 2; break;
                case 'F' : jumlah_nilai = sks;
            }
            nilai_total += jumlah_nilai ;
            jumlah_sks += sks ;
            getchar () ;
            printf ("\nNama lengkap: ") ;
            scanf ("%[^\n]", nama_lengkap) ;
            if (nama_lengkap [0] != '\0) {
                printf  ("SKS: ") ;
                scanf  ("%d", &sks) ;
                printf  ("Nilai: ") ;
                scanf  ("%1s", &nilai) ;
                nilai = toupper (nilai) ;
            }
            /*data-data tersebut harus dicek ulang untuk validasi.*/
        }
        if (jumlah_sks > 0)
            nilai_rata_rata = (float) nilai_total / (float) jumlah_sks ;
        printf ("\nNilai total          = %d", nilai_total) ;
        printf ("\nJumlah SKS           = %d", jumlah_sks) ;
        printf ("\nNilai rata rata      = %4.2f", nilai_rata_rata) ;
}
```

Perintah Pengendali

Perintah pengendali C mirip dengan perintah pengendali yang ada pada *Pascal*, C memiliki perintah *IF-ELSE* dan memiliki banyak pilihan perintah *Switch* sebagai syarat pengendali untuk pindah. C juga memiliki perintah *GOTO* tetapi tidak dianjurkan dalam penggunaannya. Susunan *loop* (putaran) yang dimiliki C terdiri dari perintah *WHILE, DO-WHILE* (mirip dengan *REPEATE-UNTIL* pada *Pascal*) dan *FOR*. Seluruh kata kunci dalam C, sebagai mana yang telah disimpulkan, harus diisi dengan huruf kecil.

Bahasa Tingkat Tinggi Prosedural Lainnya

Beberapa bahasa tingkat tinggi lainnya yang tidak akan dibahas secara mendetil, namun cukup penting untuk diketahui, diantaranya adalah:

Modula-2

Modula-2 bahasa yang mirip dengan *Pascal* tetapi lebih dimaksudkan untuk profesional.

Modula-2 adalah singkatan dari *Modular Language-2*, dibuat pada tahun 1978 oleh Niklaus Wirth yang telah sukses membuat kreasi sebelumnya yaitu *Pascal*. Meskipun memiliki banyak persamaan dengan *Pascal*, Wirth merancang *Modula-2* untuk para pengembang *software* profesional. *Modula-2* memiliki tampilan-tampilan yang tidak akan ditemukan didalam *Pascal*, ini membuat *Modula-2* lebih hebat, serbaguna dan cukup pantas untuk membangun sebuah sistem *software* yang besar, seperti sistem operasi, bahasa pemrograman untuk *compiller*, dan paket aplikasi. Pada tahun 1980 *Modula-2* mulai diperdagangkan, banyak sekolah dan organisasi-organisasi lainnya mulai menggunakan *Modula-2* sebagai alternatif dari *Pascal, C*, dan *Ada*.

Tampilan

Meskipun mirip dengan *Pascal, Modula-2* memilki beberapa perbedaan yang sangat penting. Didalam *Modula-2* program terdiri dari satu hingga lebih modul. Tidak seperti pada *Pascal, Modula-2* dapat digunakan untuk program *Coroutine* dan *Interrupt handler.*

Ada

Ada diambil dari nama Augusta Ada Byron, (Programer pertama). *Ada* diperkenalkan tahun 1980 yang disponsori oleh *US Department of Defense* (konsumen *software* terbesar di dunia). *Ada* sangat kuat dengan bahasa yang luas. Banyak organisasi saat itu mempergunakan Ada pada komputernya.

Penampilan

Ada khusus didesain sebagai bahasa komputer yang universal. Ia merupakan struktur bahasa, yang berpedoman pada prinsip-prinsip modul dan perancangan program yang top. Pada kenyataannya, banyak konsep-konsep *Ada* didapati pada *Pascal* dan mirip dengan *Modula-2*. Seperti pada kedua bahasa tersebut, *Ada* membutuhkan seluruh item-item *data* untuk dijadikan pernyataan yang jelas, ini sangat membantu para programer yang sedang membangun sebuah program dari gangguan-gangguan kecil. *Ada* menyerupai *Modula-2* yang dapat mengerjakan program *coroutines* dan *interrupt handler. Ada* juga didesain sebagai bahasa yang dapat mengerjakan pemrosesan *data* seperti *COBOL*. *Ada* mempunyai lebih dari 2 keistimewaan yang terkenal, yaitu kekuatanya pada penyelesaian berbagai macam tugas dan kebersihannya dari struktur programnya.

RPG III

RPG (Report Program Generator) diperkenalkan oleh IBM tahun 1964, *RPG* merupakan sebuah format baru dalam bahasa pemrograman, pada tahun 1970 muncul *RPG II* dan pada tahun

1979 *RPG III* mulai tersedia. Tidak seperti bahasa tingkat tinggi lainnya, *RPG III* bukan merupakan bahasa umum. Fungsi utama pembuatannya adalah untuk membuat laporan-laporan bisnis. Pada mulanya *RPG III* dibuat untuk digunakan pada komputer pribadi dengan memakai *Punched card* (kartu berlubang). *RPG III* dikembangkan menjadi modern, interaktif, dan digunakan sebagai alat pemroses *data* menjadi informasi.

Penampilan

Hal yang sangat menarik dari RPG III adalah kemudahan yang menjadi pegangan yang jelas bagi para pengguna untuk menyelesaikan tugas-tugas bisnis seperti laporan piutang, laporan utang, meng*update file* dan mengakses *data*base. Dalam RPG III untuk mengisi kode khusus form-form ditampilkan lewat layar komputer. Merinci spesipikasi form memerlukan user yang menggambarkan *file* yang digunakan, format dalam setiap pemasukan, melakukan perhitungan, dan merancang laporan *output*. Para pemakai tidak perlu khawatir tentang prosedur-prosedur logika yang harus diikuti dalam pembuatan *report* (laporan) karena RPG III sangat mudah untuk dipelajari dan digunakan juga cukup terkenal dikalangan bisnis yang menggunakan sistem komputer mini.

Selain 3 bahasa diatas, bahasa-bahasa lainnya adalah:

- ***ALGOL (Algorithmic Language)*** - Dikembangkan pada tahun 1958 untuk memecahkan masalah-masalah secara umum.
- ***APL (A Programming Language)*** - dikembangkan di IBM oleh Kenneth Iverson pada tahun 1962.
- ***APT*** - adalah program untuk otomatisasi yang cukup pendek, dikembangkan di *MIT (Massachusetts Institute of technology)*. Pada awal 1950-an.
- ***Forth*** - dikembangkan oleh Charles Moore di tahun 1969. *Forth* digunakan untuk masalah perbintangan.
- ***LISP*** - John McCarthy membuat bahasa *LISP* yang merupakan bahasa *artificial intelligence* pertama.
- ***PL/1 (Programing Language version 1)*** - IBM mengembangkan bahasa tingkat tinggi *PL/1* yang merupakan gabungan dari *FORTRAN, COBOL* dan *ALGOL*.
- ***PROLOG (Programming Logic)*** - Bahasa ini berguna untuk operasi *artificial intelligence* dan merupakan saingan dari *LISP*.
- ***Smalltalk*** - dikembangkan di Xerox's Palo Alto Research Center (PARC). Smalltalk-72 adalah versi yang pertama, diikuti oleh *Smalltalk-76* dan *Smalltalk-80*. *Smaltalk* memperkenalkan teknik pemrograman berorientasi objek.

- ***SNOBOL4*** - Dikembangkan di Bell Labaratories sekitar tahun 1960-an, bahasa ini sering digunakan untuk pengembangan *text editor*, *word processor* dan *interpreter* bahasa tingkat tinggi.

8.4.2. Bahasa Berorientasi ke Objek

Pada bahasa prosedural penekanan terletak kepada apa yang dilakukan (tindakan). Pada bahasa berorientasi ke objek (OOL) penekanan dilakukan terhadap objek dari tindakan. Struktur pemrograman bertingkat yang berorientasi ke objek *(Object oriented programing-OOP)* mudah untuk dirancang dan dimengerti. Hal lainnya yang perlu diperhatikan adalah arah pemrograman yang semakin banyak menggunakan gambar, video dan suara. Pemrograman berorientasi ke objek menangani masalah ini jauh lebih baik dibandingkan pemrograman berorientasi prosedur.

Pemrograman berorientasi ke objek pada dasarnya hanyalah cara berfikir dan tidak tergantung kepada bahasa tertentu. Keberadaan bahasa yang berorientasi ke objek (OOP) pada dasarnya adalah karena bahasa tersebut mendukung pemrograman yang berorientasi ke objek lebih baik dari bahasa lainnya dan bukan bahasa yang lain tidak bisa digunakan untuk pemrograman berorientasi ke objek. Beberapa bahasa yang berorientasi ke objek adalah *Simula, Smaltalk, C++, Objective-C, Eiffel,* dan *Interlisp*.

Gambar 8.7 Contoh program bahasa C++

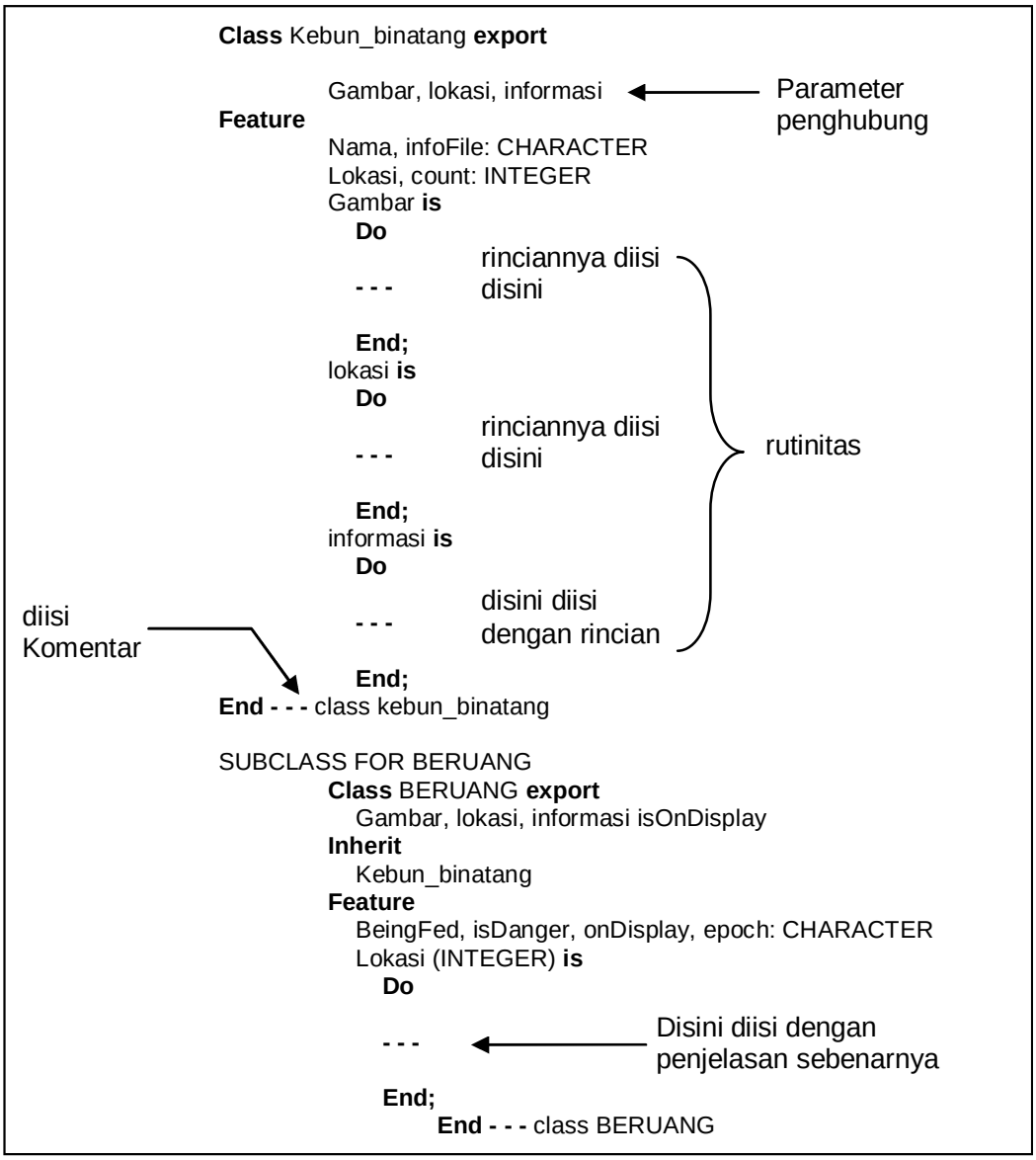
```
Class ZooAnimal {
Public:
        ZooAnimal (char*, char*, short);
        Virtual~ZooAnimal();
        Void draw();
        Virtual void locate();
        Void inform();
Protected:
        Chart*name;
        Chart*infoFile;
        Short location;
        Short count
};
```

Dilihat dari perkembangannya, Perkembangan bahasa-bahasa yang berorientasi ke objek tersebut adalah sebagai berikut:

Tabel 8.2 Perkembangan Bahasa berorientasi ke objek

Tahun	Bahasa	Pengembang
1967	Simula	Dahl dan Nygaard
Awal 1980-an	Smalltalk	Kay,Goldberg,Ingalls
Akhir 1980-an	C++ Objective-C Eiffel	Stroustrup COX Meyer

Gambar 8.8 Contoh program Eiffel



8.4.3. Bahasa Generasi keempat

Ada banyak nama untuk bahasa generasi ke empat ini diantaranya adalah *application generator, application development systems, program generator* dan lain-lain. Tujuan utama dari munculnya bahasa generasi ke empat adalah untuk meningkatkan efisiensi dan efektivitas proses pengembangan sistem atau dengan kata lain bahasa ini digunakan untuk meningkatkan kecepatan pengembangan dan pemeliharaan sistem informasi.

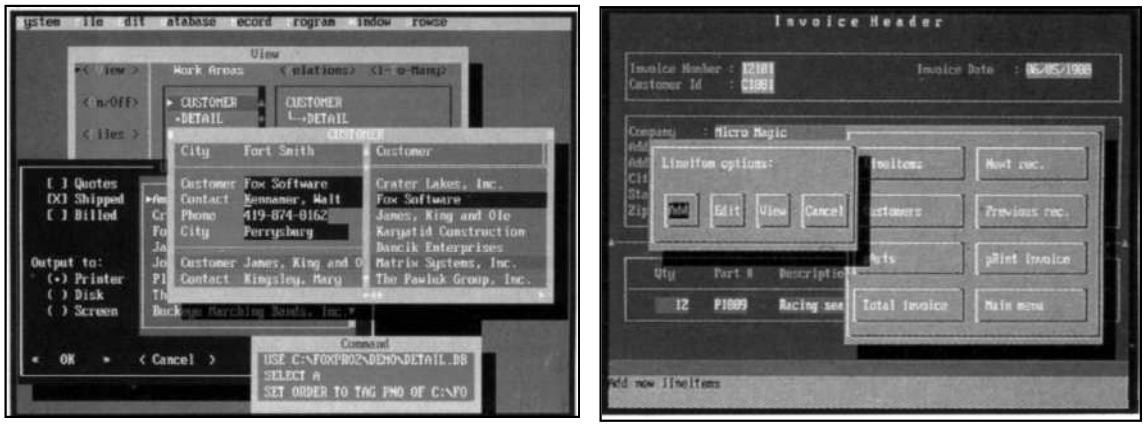
Beberapa fasilitas bahasa generasi ke empat :

- **Sistem manajemen *database*** (DBMS) merupakan jantungnya 4 GL yang menyimpan *record-record data* yang terformat juga dan yang tidak serta data grafik.

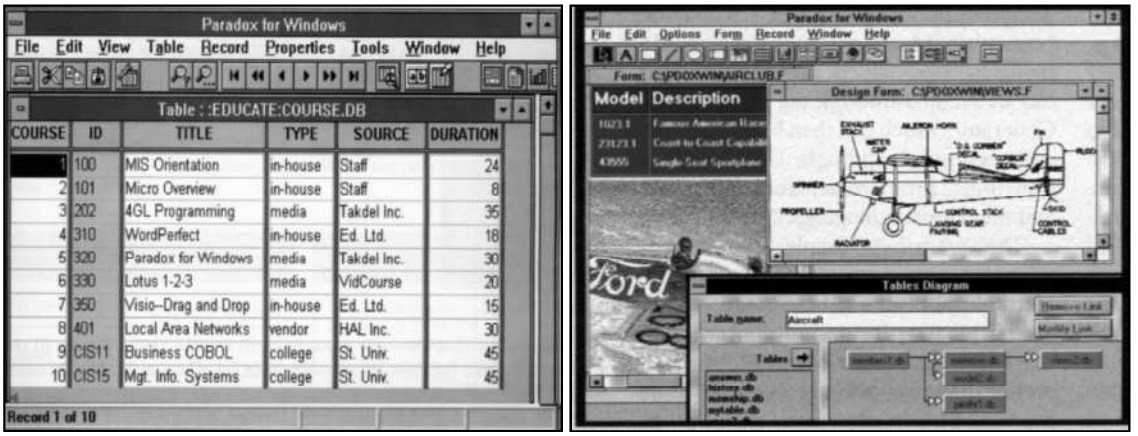
- ***Data dictionary*** sama pentingnya dengan DBMS yang berfungsi untuk mendefinisikan dan mengontrol *data*.
- **Bahasa *non procedural*** yang berisi perintah yang dijalankan berdasarkan logika programer bukan komputer.
- **Fasilitas pencarian data yang interaktif** dan dapat bekerja dengan jenis perintah yang agak natural.
- ***Report generator*** merupakan fasilitas untuk pembuatan *report* yang didalamnya juga meliputi fasilitas pencarian *data*.
- ***Word processor / text editor*** merupakan bagian *4GL* untuk menulis perintah.
- **Grafik** merupakan kemampuan yang dimiliki oleh kebanyakan 4 GL.

Beberapa bahasa generasi keempat yang banyak digunakan oleh mayarakat adalah: *Foxpro, Alphafour, Paradox, Clipper.*

Gambar 8.9 Foxpro 2 (kiri) dan Alpha four (kanan)



Gambar 8.10 Paradox for windows



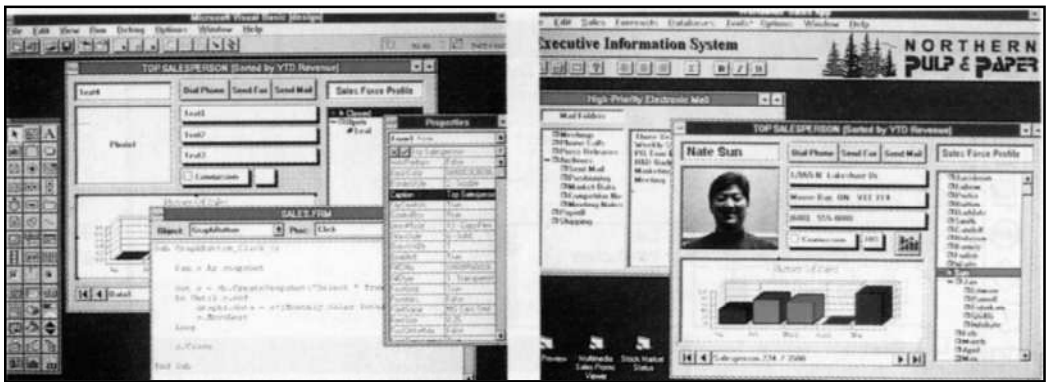
8.5 Bahasa Tingkat Sangat Tinggi

Pembuatan program aplikasi berbasis grafis saat ini jauh lebih kompleks daripada pemrograman berbasis teks. Pemrograman berbasis objek telah membantu menenangkan programer untuk sementara waktu, akan tetapi dengan munculnya sistem operasi berbasis grafis maka pemrograman menjadi sangat kompleks. Atas dasar kondisi inilah muncul bahasa pemrograman visual yang merupakan pengembangan lebih lanjut dari pemrograman berbasis obyek.

8.5.1 Bahasa pemograman secara visual

Bahasa pemrograman visual menggantikan penggunaan teks untuk membuat perintah dengan simbol-simbol gambar yang disebut *Icon*. Bahasa pemrograman visual saat ini dapat digunakan untuk membangun sistem informasi berbasis windows, console (dos dalam windows) atau web Contoh dari bahasa berorientasi ke *visual* atau gambar ini adalah *Visual Basic*, *Visual Foxpro*, *Visual, Delfi, C++,*bahasa bahasa yang tergabung dalam Visual Studio Net (VB,C#,J#,C++) dan lain-lain.

Gambar 8.11 Visual Basic

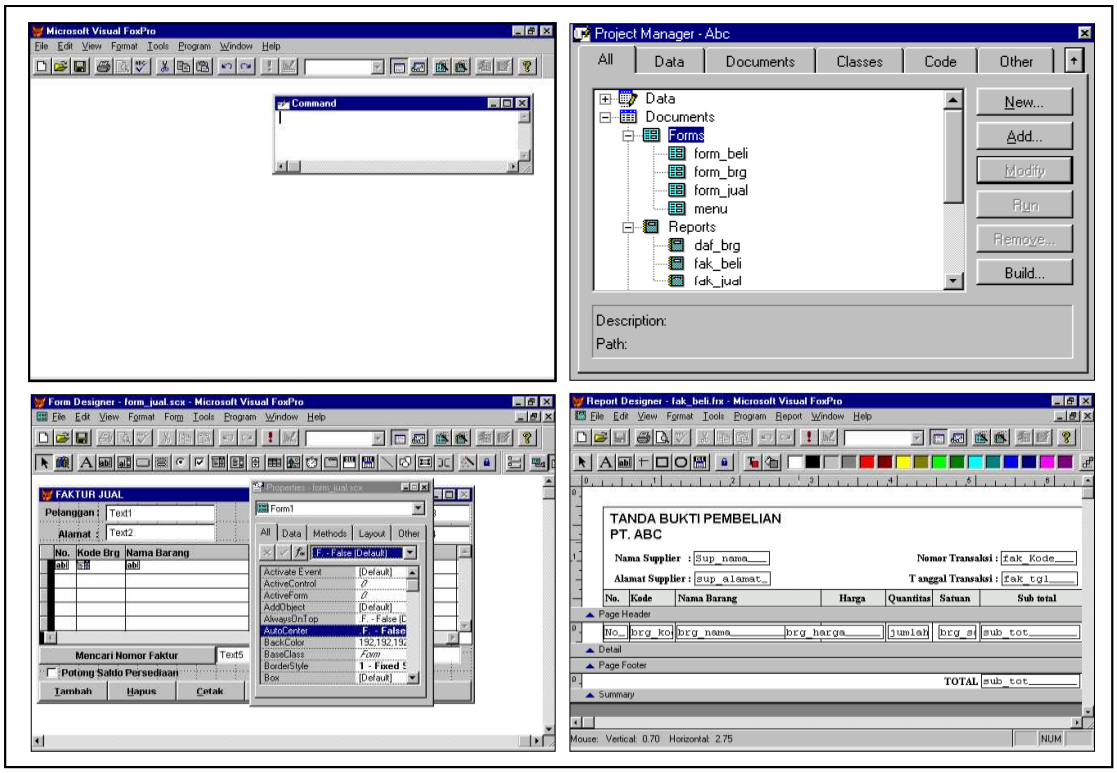


Bahasa pemrograman *visual* sangat interaktif . Dengan menggunakan bahasa ini pemrograman menjadi sangat mudah. Walaupun bahasa pemograman ini masih terus dikembangkan akan tetapi kemampuannya sampai saat telah banyak membantu para programer profesional untuk dapat membuat program jauh lebih bagus dan lebuh cepat. Beberapa hal yang menonjol dari bahasa berorientasi ke visual ini adalah :

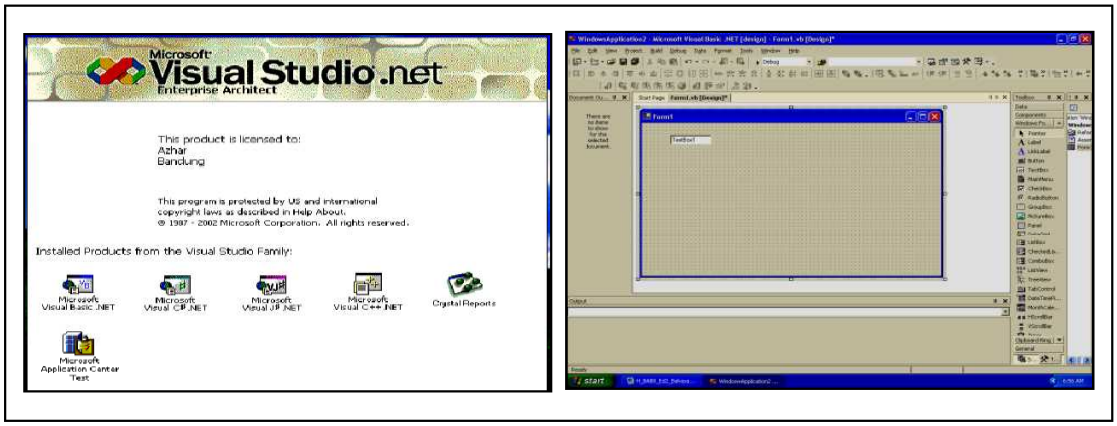
- **Fasilitas pembuatan menu** bahasa visual memiliki fasilitas (*toolbar*) yang memberikan kemudahan kepada pemakai untuk membuat menu *pulldown* dengan mudah. Bahasa *visual* yang baik akan memberikan lebih dari satu level *pulldown menu* sehingga sangat memudahkan dalam pembuatan program
- **Fasilitas pembuatan *form*** - *form* ini biasanya digunakan untuk memasukan data input dalam suatau sistem informasi.

- **Objek penghubung dengan pemakai -** merupakan bagian yang sangat sentral, *software* yang bagus akan memiliki fasilitas ini.
- **Penanganan grafis** - merupakan kemampuan dalam menangani *file* gambar seperti mengimport gambar bitmap dan format lain.
- **Penanganan teks -** sangat diperlukan untuk memberi keterangan. Teks ini biasanya disimpan pada *textbook*
- **Pemrograman dan perbaikan program** - bahasa ini memiliki kemampuan dalam menangani masalah-masalah pemrograman yang komplek.

Gambar 8.12 Visual Foxpro



Gambar 8.13. Visual studio.net 2003 dan Visual Basic.Net



8.5.2 Bahasa Natural

Bahasa natural mengacu kepada *software* yang memungkinkan komputer untuk menerima, mengintepretasikan dan menjalankan perintah dalam bahasa manusia misalkan bahasa inggris. Dengan bahasa natural programer maupun operator tidak perlu banyak lagi belajar komputer. Programer hanya menyebutkan kalimat tertentu atau menulis kalimat tertentu dan komputer akan menjalankan perintah tersebut. Bahasa natural ini masih belum berkembang tapi perkembangan ke arah itu sudah mulai banyak dirintis oleh para pengembang *software* seperti Micorsoft yang meluncurkan *software* bernama *Microsoft studio*. Dengan *software* ini operator dapat memerintahkan komputer untuk melakukan sesuatu yang sangat terbatas. Perkembangan selanjutnya terjadi pada *Windows Xp* yang memungkinkan pemakai komputer mendikte atau menulis dalam tulisan biasa dan komputer akan menginterpretasikan diktean atau tulisan tersebut.

8.6 Pembuatan Program

Proses pembuatan program adalah aktivitas yang diperlukan untuk menciptakan suatu program komputer yang sukses. Proses pembuatan program dapat dibagi kedalam 7 langkah:

1. Menentukan permasalahan.
2. Menentukan kebutuhan *software*.
3. Mendesain program.
4. Penyusunan kode program (pemrograman)
5. Pengetesan program.
6. Menginstall/memasang program pada komputer dan memelihara *(maintaining)* program.
7. Mendokumentasikan program.

8.6.1. Menentukan Permasalahan

Tugas pertama seorang programer adalah menentukan permasalahan dan bagaimana untuk memecahkannya. Hal ini dapat dilakukan dengan cara membicarakannya dengan orang yang membutuhkan program tersebut sehingga mempelajari apa yang mereka butuhkan dan bagaimana keinginan mereka tentang program tersebut. Seorang programer kadang-kadang hanya menerima gambaran secara umum apa yang dibutuhkan, dalam kasus ini masukan dari orang yang membutuhkan sangat membantu menentukan bagaimana program itu seharusnya bekerja. Sangatlah penting untuk melihat projek secara keseluruhan sehingga permasalahan dapat didefinisikan dengan jelas dan dimengerti, karena setiap tahap dalam proses pengembangan program saling berkaitan.

8.6.2. Menentukan Kebutuhan Software

Kebutuhan *software* menentukan apa yang seharusnya dilakukan oleh suatu program tanpa memberikan detail-detail bagaimana program seharusnya berjalan.

Output Yang Diinginkan *(Desired Output)*

Karena *output* adalah alasan keseluruhan untuk mengembangkan sebuah program, menspesifikasikan *output* apa yang diperlukan adalah penting seperti halnya mengidentifikasikan masalah. Ciri-ciri suatu *output* yang harus diperhatikan adalah sebagai berikut:

- ***Content* (isi) -** informasi apa yang harus ditampung oleh *output?* Sedetail apa informasi harus dirinci? Jawaban atas perta nyaan tersebut tergantung kepada permasalah yang dapat diidentifikasi .
- ***Format* -** bagaimana informasi disajikan? Apa alat *output* yang akan digunakan dan bagaimana informasi tersebut disusun. Sebagai contoh, bagaimana *output* tersebut ditampilkan, bagaimana penempatan dan warnanya adalah faktor-faktor yang perlu dipertimbangkan.
- ***Timing* -** kapan informasi tersebut ditampilkan? Apakah informasi tersebut ditampilkan secepatnya atau secara periodik?

Input Yang Dibutuhkan *(Required Input)*

Hal-hal yang harus diperhatikan oleh programer tentang *input*:

- ***Data* yang dibutuhkan** - *data* apa yang diperlukan untuk mendapatkan *output* yang diinginkan? Daftar *data* yang harus di*input* adalah bagian yang penting dari pengembangan *software* yang dibutuhkan.
- **Ketersediaan dan akses terhadap *data*** - dari mana *data* yang dibutuhkan diperoleh? Apakah *data* tersebut akan diperoleh secara langsung dari alat *on line input* atau dari media *secondary storage* seperti *magnetic disk* dan *optical disk?* Dengan kata lain apakah pemakai dapat memasuki *data* secara cepat atau *data* tersebut akan dibaca dari *file* yang dibentuk sebelumnya?
- ***Format Data*** - bagaimana susunan fisik dari *data* yang di*input*?, jika suatu program menerima *data* dalam format yang tidak sesuai dengan rancangan penggunaannya maka program tersebut akan gagal beroperasi.

Pemrosesan Yang Diperlukan

Hal yang perlu diperhatikan dalam memproses *input* menjadi *output* adalah sebagai berikut:

- **Memproses *data*** - bagaimana bentuk proses yang diperlukan untuk memproses *data* ?. Kebutuhan *software* seharus-

nya menspesifikasikan secara tepat apa yang harus dilakukan dengan *data* yang di *input*.

- ***User interface*** - bagaimana pemakai akan berinteraksi dengan progam? *User interface* berarti bagaimana seseorang berkomunikasi dengan program tersebut.
- **Peralatan yang diperlukan** - komputer dan peralatan apa yang tepat untuk menjalankan program? Kebutuhan *software* menentukan *hardware* apa yang dibutuhkan agar program dapat berjalan dengan baik.

8.6.3. Mendesain Program

Setelah programer menentukan program apa yang harus dibuat sebelum menginstruksikan komputer untuk menjalankan program, seorang programer harus membuat tahap-tahapan untuk merubah *input* menjadi *output*. Serangkaian langkah itu disebut *algoritma*. Menyusun *algoritma* adalah langkah pertama untuk meme cahkan masalah atau melakukan suatu tugas. Setelah seluruh alogaritma yang terbaik untuk memecahkan masalah dipilih atau dikembangkan, programer harus memperlihatkan secara jelas dalam kata-kata, diagram-diagram, grafik-grafik atau tabel-tabel seperti *Pseudocode, Flowcharts, Decision Table,* dan *Structure Charts*. Dalam pemrograman visual, perancangan program didasarkan kepada *object,class,method dan event* yang lebih dikenal sebagai *object oriented programming* yang akan dijelaskan kemudian.

Pseudocode

Pseudocode sifatnya tidak resmi, belum tersusun, merupakan pernyataan dari algoritma, menggunakan kata-kata, frase dan simbol matematika. *Pseudocode* merupakan bantuan yang rancangannya sangat populer untuk menyederhanakan program yang rumit. Beberapa aturannya antara lain:

- Penggunaan huruf besar seperti *READ, COMPUTE* dan *DISPLAY.*
- Menggunakan tanda untuk menunjukkan susunan umum program.
- Menunjukkan suatu putusan diantara 2 pilihan seperti
  IF (kondisi yang sebenarnya) *THEN*
  Satu atau lebih perintah
  ELSE
  Satu atau lebih perintah
  END IF
- Menunjukkan pengulangan rangkaian perintah atau *loop*
  DO WHILE
  Satu atau lebih perintah
  END DO

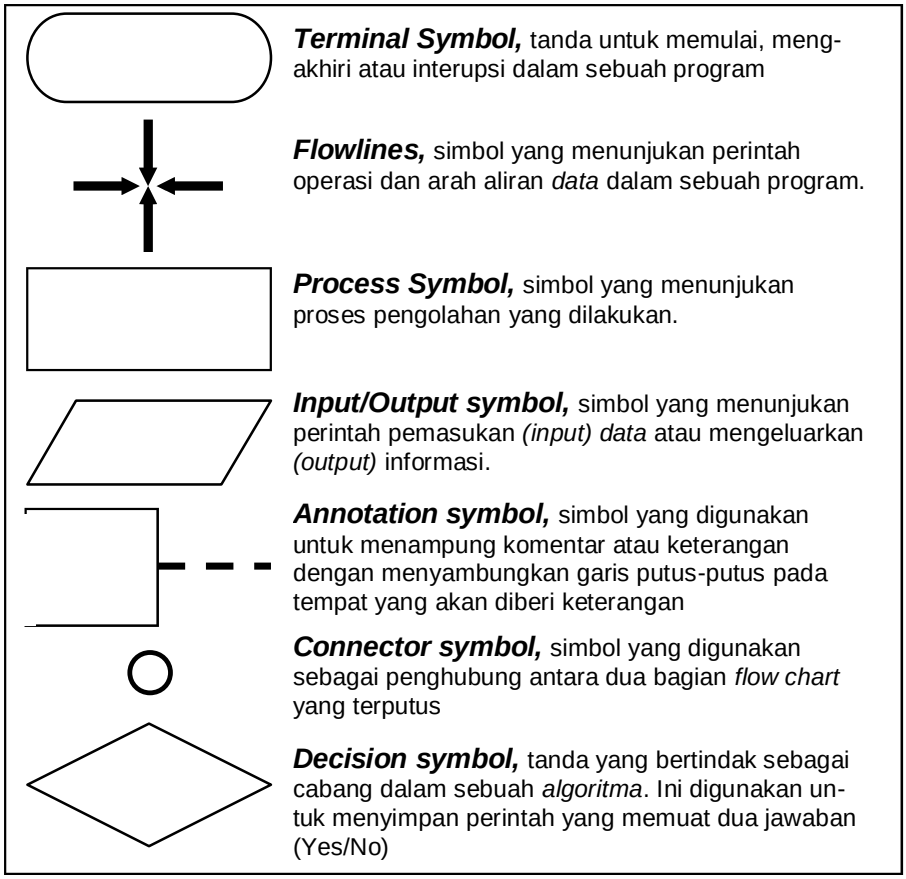
Program diatas merupakan rangkaian perintah yang dirangkum dalam *BEGIN* dan *END*.

Flowcharts

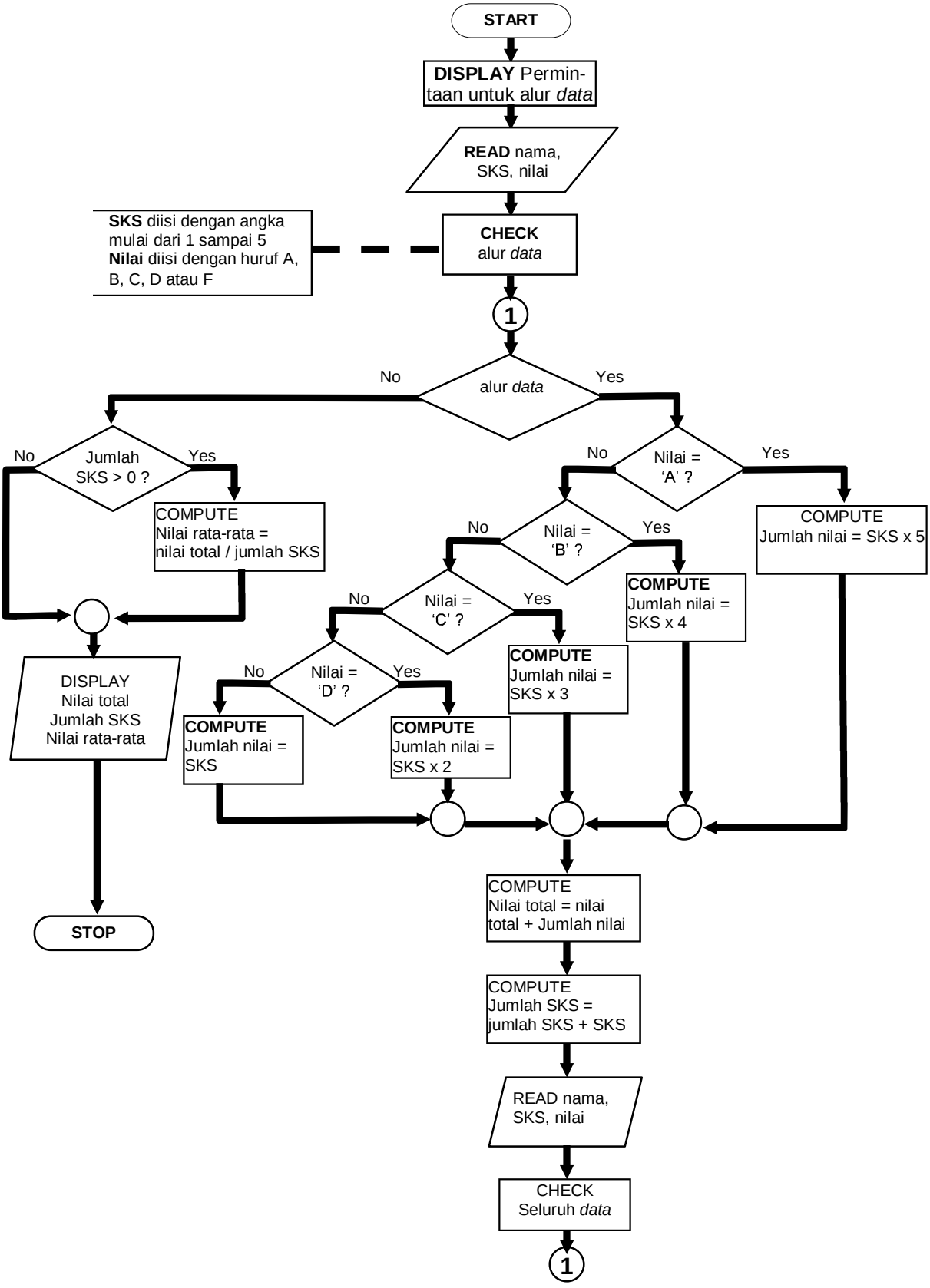
Flowcharts adalah grafik dari algoritma, *flowcharts* menunjukan langkah yang dilakukan. Tidak seperti *pseudocode*, simbol *flowchart* pedoman pemakaiannya sudah dibakukan oleh ANSI. Beberapa pedoman *flowchart* antara lain:

- Selalu menggunakan sombol ANSI,
- *Flowchart* harus dibaca dari atas ke bawah dan dari kiri ke kanan, sebanyak mungkin.
- Garis jangan bertabrakan (gunakan konektor bila terjadi) dan gunakan tanda panah untuk menunjukkan arah.
- Buatlah pesan dan tanda didalam simbol *flowchart*.
- Berusahalah agar rapi, jelas, sederhana jika perlu rincikan *flowchart* yang besar kedalam dua *flowchart* yang lebih kecil dengan menggunakan tanda penghubung. Gunakan *flowcharting template* untuk meniru *outline* simbol yang seragam dan *print* dengan jelas untuk memastikan agar mudah dibaca.

Gambar 8.13 Simbol-simbol Flowchart standar



Gambar 8.14 Flowchart untuk program nilai rata-rata



Program untuk *Flowchart* diatas tampak seperti pada gambar di bawah ini.

Gambar 8.15 Contoh Program

```
BEGIN (Program Nilai-rata-rata)
DISPLAY "Silahkan isi nama lengkap, SKS, dan nilai"
READ nama lengkap, SKS, nilai
CHECK seluruh data untuk validasi
    DO WHILE (Seluruh data harus dimasukan dan divalidasi)
        IF (nilai = 'A') THEN
                COMPUTE jumlah nilai = SKS*5
        ELSE IF (nilai = 'B') THEN
                COMPUTE jumlah nilai = SKS*4
        ELSE IF (nilai = 'C') THEN
                COMPUTE jumlah nilai = SKS*3
        ELSE IF (nilai = 'D') THEN
                COMPUTE jumlah nilai = SKS*2
        ELSE
                COMPUTE jumlah nilai = SKS
        END IF
        COMPUTE nilai total = nilai total + jumlah nilai
        COMPUTE Jumlah SKS = Jumlah SKS + SKS
        READ nama lengkap, SKS, nilai
        CHECK data lengkap untuk validasi
    END DO
        IF (Jumlah SKS > 0) THEN
                COMPUTE nilai rata-rata = nilai total / jumlah SKS
        END IF
        DISPLAY "Nilai total = " nilai total
        DISPLAY "Jumlah SKS= " jumlah SKS
        DISPLAY "Nilai rata-rata = " nilai rata-rata
END
```

Decision Tables

Decision tables adalah program yang digunakan untuk menentukan kejadian atau keadaan umum yang kompleks. *Decision table* memberikan penjelasan kepada programer, jalan singkat menyelesaikan masalah dan mendeteksi setiap kemungkinan oleh karena itu *decision table* mengurangi error (kesalahan) dalam penyelesaian program.

Structure Chart

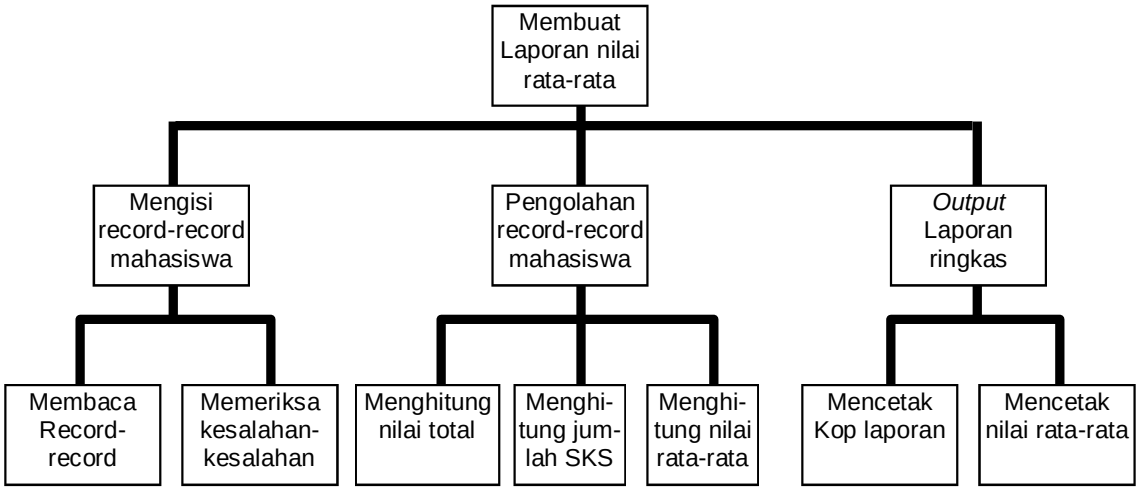
Structure chart adalah program bantuan yang membantu programer mengatur program kedalam beberapa bagian. *Structure chart* menunjukkan bagaimana modul program saling berhubungan tetapi tidak secara terperinci menjelaskan cara kerjanya. Seperti *decision table*, *structure chart* biasanya digunakan bersama dengan program lainnya.

Hipo Charts

HIPO Charts pada mulanya dikembangkan oleh IBM sebagai alat untuk membuktikan kebenaran. Untuk beberapa program, *HIPO Chart* lebih mudah dibaca dibandingkan dengan *flowcharts*. Da-

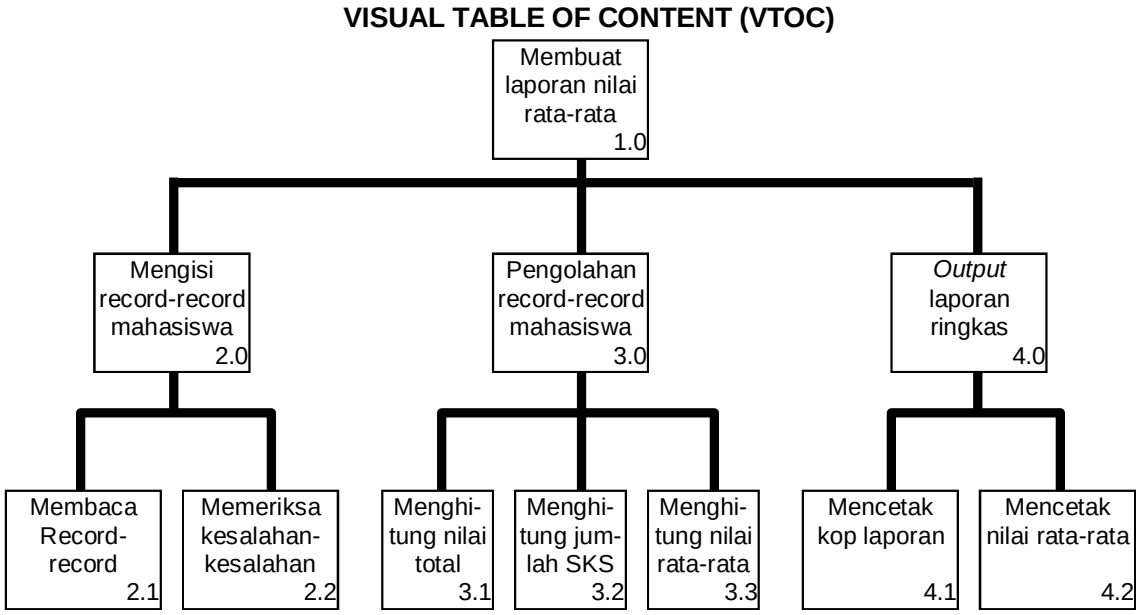
lam *HIPO Charts* perlu membuat 3 modul diagram, *VTOC (Visual table of content), Overview diagram,* dan *detail diagram.*

Gambar 8.16 Contoh struktur HIPO Chart



VTOC (Visual table of content) sangat mirip dengan susunan chart seperti pada gambar di atas hanya saja pada setiap modul diberi nomor yang menggambarkan posisi modul dalam sebuah hirarki tersebut. *VTOC* dilengkapi dengan daftar legenda dari nomor-nomor yang tercantum pada setiap modul beserta penjelasannya. *VTOC* menggambarkan susunan program secara keseluruhan dan memberikan kesan yang jelas mengenai hal yang bersangkutan.

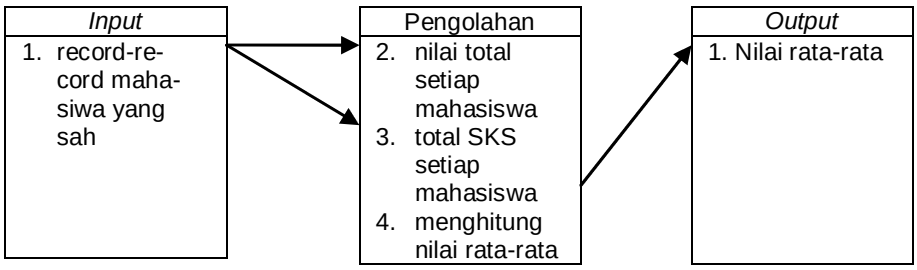
Gambar 8.17 HIPO chart visual table of content (VTOC)



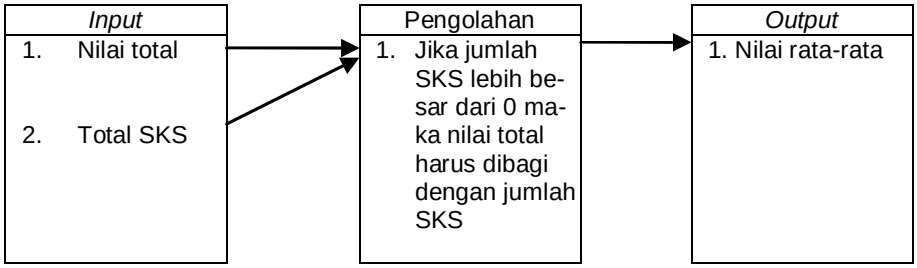
Tabel 8.2 Legenda VTOC

REF. NO	PENJELASAN
1.0	Membuat laporan ringkas nilai rata-rata
2.0	Membaca record-record nilai seluruh mahasiswa
3.0	Menghitung nilai rata-rata yang diperlukan
4.0	Memformat dan mencetak laporan ringkas
2.1	Membaca record seorang mahasiswa
2.2	Memeriksa nilai *data* sesungguhnya
3.1	Menghitung nilai total setiap mahasiswa
3.2	Menghitung jumlah SKS setiap mahasiswa
3.3	Menghitung nilai rata-rata setiap mahasiswa
4.1	Mencetak kop laporan ringkas
4.2	Mencetak nama beserta nilai rata-rata setiap mahasiswa

Gambar 8.18 Ikhtisar diagram untuk modul 3.0



Gambar 8.19 Diagram rincian modul 3.3



8.6.4. Pengkodean Program

Pengkodean yaitu proses penterjemahan *"algorithm"* secara detail ke dalam standar bahasa program. Dalam tahap proses pengembangan, membutuhkan aturan pokok yang tepat untuk format dan sintaksis (perbendaharaan kata, tata bahasa dan tanda baca).

Tahap awal dalam program pengkodean adalah memilih/ memutuskan bahasa apa yang akan dipakai. Berbagai macam bahasa yang telah dikembangkan untuk memenuhi kebutuhan. Teknik yang dipergunakan oleh seorang programer agar *Source code*nya dapat mudah dimengerti yaitu:

1. **Nama Variabel** - dalam sebuah program komputer, variabel adalah nama tempat penyimpanan yang memuat satu atau lebih *data*.

2. **Indentas** - kita lihat kembali *"the pseudocode",* perhatikan bagaimana identitas digunakan untuk membantu menunjukkan struktur yang bergaris bawah dalam suatu program.
3. **Spasi kosong** - salah satu sebab tulisan berantakan sehingga sulit untuk dibaca, termasuk *source code* program, adalah karena tidak menggunakan spasi kosong. Karena itu, saat ini banyak bahasa pemrograman memberikan fasilitas untuk itu.
4. **Keterangan** - sebenarnya semua bahasa program modern memberi kemudahan kepada kita untuk mengisi keterangan pada *source kode*. Keteragan yaitu suatu catatan sederhana yang isinya penjelaskan perintah dalam *source code*, keterangan akan diabaikan oleh komputer saat program dijalankan.

8.6.5. Pengetesan program

Setelah program dikodekan dan dimasukan, barulah timbul masalah "Apakah program berfungsi dengan baik?" Untuk itu perlu dilakukan pemeriksaan agar program yang kita masukkan sesuai dengan apa yang kita inginkan. Untuk memeriksa apakah suatu program berjalan dengan baik, ada tiga cara untuk melakukan pemeriksaan, yaitu:

Desk Cheking

Sering disebut juga pemeriksaan dengan tangan artinya proses pemeriksaan suatu program dilakukan dengan memeriksa satu per satu *source code*, apakah ada kesalahan sintaksis dan logika.

Translation

Perubahan *source code,* ke dalam instruksi bagian yang dibutuhkan komputer. Dimana proses ini dilakukan sepenuhnya oleh komputer selama penterjemahan, program ini akan memeriksa dan mencari kesalahan sintaksis. Dalam pemeriksaan ini kesalahan logika tetap tidak bisa terdeteksi.

Debugging

Proses dalam mendeteksi dan memperbaiki kesalahan logika dengan menggunakan *interpreter*. Hasil yang diperoleh dibandingkan dengan hasil yang telah dihitung dengan tangan atau kalkulator.

8.6.6. Pemasangan dan Perawatan Program

Setelah program selesai diperiksa, program tersebut kemudian dipasang atau diinstal untuk digunakan dalam operasi kegiatan sehari-hari. Dalam pemasangan program harus dipastikan terlebih dahulu bahwa program yang dipasang akan bekerja dengan baik pada komputer dengan spesifikasi tertentu.

8.6.7. Dokumentasi Program

Dokumentasi adalah suatu penjelasan yang sangat detail tentang *"algorithm"* suatu program, desain program, metode pengkodean, pemeriksaan program dan penggunaan program. Dokumentasi sangat diperlukan oleh para pengguna program yang sangat mempercayakan segala sesuatunya kepada program.

Selama ini tidak ada standar yang baku mengenai hal-hal apa saja yang harus dicantumkan dalam dokumentasi program. Umumnya dokumentasi program mencakup:

- Penjelasan tentang apa yang harus dikerjakan oleh program.
- Penjelasan tentang pemecahan masalah.
- Penjelasan tentang desain program, termasuk panduan yang digunakan.
- Penjelasan tentang proses pemeriksaan program.
- Penjelasan tentang semua proses pemeriksaan, modifikasi dan pembaharuan yang dibuat sejak program dioperasikan.
- Buku panduan manual atau instruksi yang memberi informasi kepada pengguna mengenai bagaimana menggunakan program.

8.7 Pemrograman Terstruktur

Pemrograman yang terstruktur merupakan susunan teknik pengembangan *software* termasuk didalamnya beberapa konsep penting, meskipun ada pengaruh yang kuat dari pemrosesan data, pemrograman yang terstruktur digunakan untuk mendefinisikan program secara ringkas. Secara khusus, teknik pemrograman yang terstruktur akan menghasilkan beberapa hasil berikut ini:

- **Produktivitas yang lebih besar** - lebih banyak kode yang ditulis oleh programer dalam waktu yang sangat singkat.
- **Ekonomis -** biaya pembuatan *software* dapat dikurangi.
- **Meningkatkan waktu *"debugging"*** - kesalahan yang terjadi dapat segera langsung diketahui penyebabnya dan dapat diperbaiki dengan cepat.
- **Kode program *(source code)* yang lebih jelas** - program yang terstruktur dapat mudah dimengerti, diperbaharui dan dipelihara oleh semua orang.
- **Umur perangkat lunak yang lebih lama** - program yang dapat mudah dimodifikasi, diperbaharui dan dipelihara dan dapat berumur lebih lama.

8.7.1. Control Structures

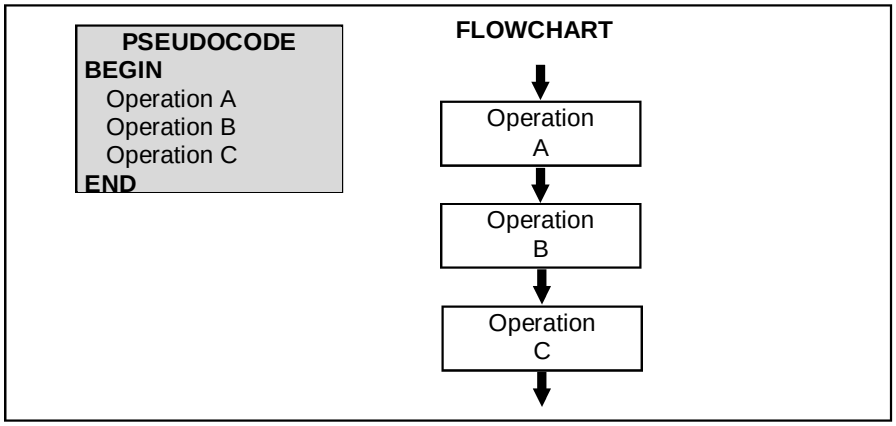
Control Structure adalah sebuah pola alur logika dalam program komputer. Dengan kata lain, sebuah rancangan kerja yang dinyatakan dalam order yang pengerjaannya akan tampil pada layar.

Ada tiga *Control Structure* program yaitu *Sequence* (rangkaian), *selection* (pilihan) dan *Iteration.*

Sequency Control Structure

Rangkaian sederhana dari sebuah operasi. Struktur ini diwakili oleh *pseudocode* sebagai rangkaian perintah-perintah didalam tanda kurung besar ([]) diantara BEGIN dan END. Perhatikan gambar di bawah ini.

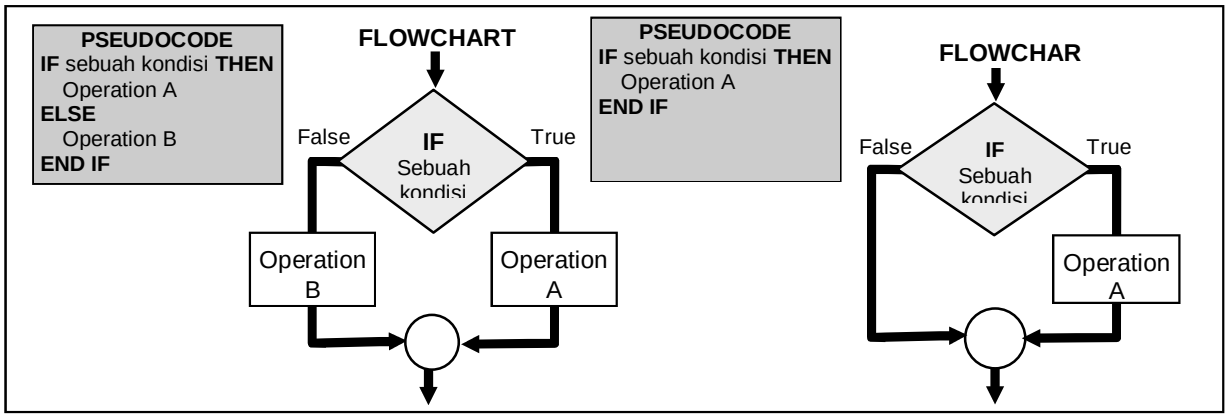
Gambar 8.20 Sequence control structure



Selection Control Structure

Menawarkan dua buah pilihan struktur *pseudocode* antara pilihan pertama yaitu dengan menggunakan *IF-THEN* atau cara yang kedua dengan menggunakan *IF-THEN-ELSE*. Lihat gambar di bawah ini.

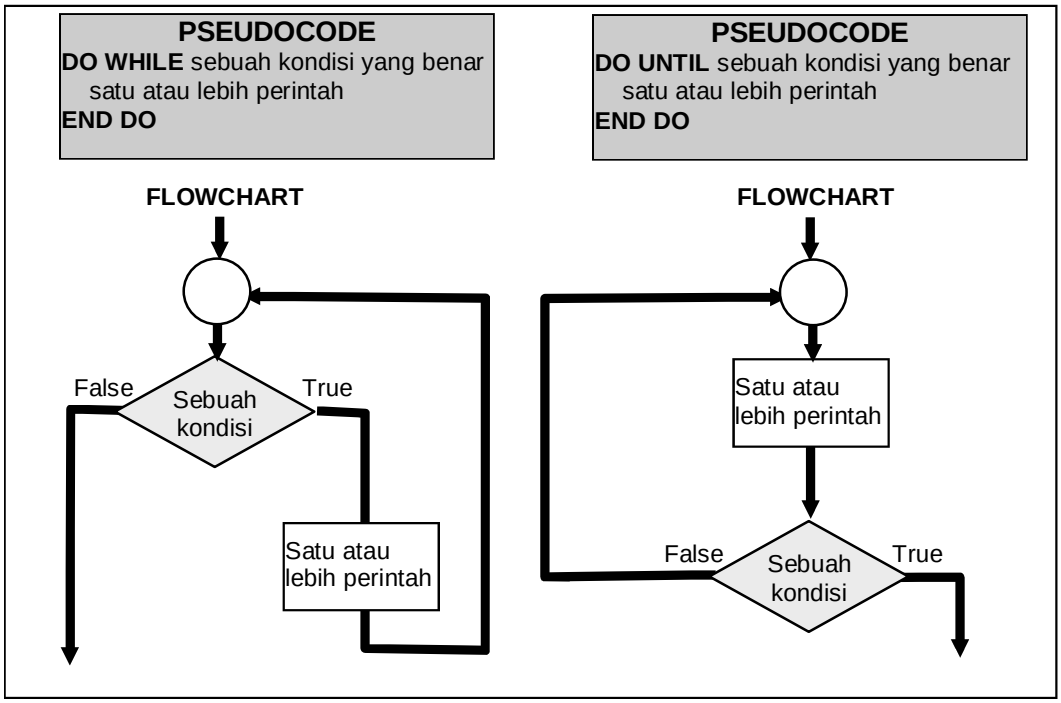
Gambar 8.21 Selection control structure



Iteration Control Structure

Iteration Control Structure merupakan loop sederhana, rangkaian cara kerja yang dilakukan secara berulang-ulang hingga berhubungan secara terstruktur. *Iteration* memiliki dua versi dasar program/perintah yaitu *DO WHILE* dan *DO UNTIL* seperti tampak pada gambar berikut ini.

Gambar 8.22 Iteration control structure



Dalam susunan perintah *DO WHILE*, sebuah syarat harus diperiksa terlebih dahulu sebelum operasi dijalankan, sebaliknya dalam susunan perintah *DO UNTIL* syarat diperiksa setelah operasi dijalankan. Dengan kata lain tes dilakukan pada puncak *(top)* program untuk perintah *DO WHILE,* dan tes dilakukan pada dasar *(bottom)* program untuk perintah *DO UNTIL*. Pada perintah *DO WHILE* operasi tidak akan dilakukan jika syarat *(condition)* bernilai *False* (salah). Berbeda dengan *DO UNTIL* operasi akan selalu dilaksanakan setidaknya satu kali karena pemeriksaan sebuah syarat/kondisi tergantung kepada operasi tersebut.

Dalam penyusunan program sederhana, semestinya harus menggunakan tiga struktur dasar di atas yakni *Sequence, Selection* dan *Iteration* ketiga struktur tersebut sering disebut sebagai *Structured Psedecode.* Demikian pula halnya dengan *flowchart* yang berisi *control structure* seperti yang terlihat pada gambar-gambar di atas biasa dipanggil sebagai *Structured Flowchart.* Dengan hanya memiliki satu jalan masuk dan satu jalan keluar, ini akan memudahkan pembuatan struktur program yang kompleks juga memudahkan dalam hal pengkodean, *debuging*, perawatan dan mudah untuk dimengerti.

8.7.2. Menghindari perintah GO TO

Banyak bahasa-bahasa pemrograman memasukan perintah *GO TO* didalamnya yang meyebabkan tampilan berpindah-pindah dari bagian satu ke bagian lainnya. Sebuah motivasi untuk membangun teknik penyusunan program yang tinggi baru terpenuhi pada awal tahun 1960-an. Ketidakdisiplinan dalam penggunaan perintah *GO TO* sering menyebabkan komplikasi yang berlebihan, sukar untuk dimengerti, sukar diperbaiki dan kemungkinan besar banyak sekali kesalahan-kesalahan dalam sebuah program. Pendek kata perintah *GO TO* dapat merubah alur logika sebuah pengolahan *data* dengan alur yang tidak terstruktur.

8.7.3. Pola Top Down

Pembuatan bentuk struktur *Top-Down* ini meliputi tiga tahap:

1. Tentukan *output* yang akan dibuat, *input* yang diinginkan dan tugas pemrosesan yang paling utama harus dibuat untuk merubah *input* menjadi *output*.

2. Pecah setiap tugas pemrosesan yang paling utama tersebut kedalam bentuk modul yang lebih kecil sehingga membentuk suatu hirarki.

3. Gunakan algoritma pada setiap modul, mulailah pada ujung pertama (modul utama) dan kemudian menurun.

8.7.4. Pembuatan Modul

Sebuah modul yang telah kita ketahui sebelumnya, adalah merupakan rangkaian perintah-perintah program yang saling berhubungan sehingga membentuk satu unit modul . Keinginan merubah sebuah program yang besar menjadi modul-modul atau menjadi sub program yang lebih kecil telah dipelajari selama bertahun-tahun. Konsep pembuatan modul program ini sangat dianjurkan dengan memperhatikan bagaimana modul-modul tersebut dibentuk dan keterkaitan diantara modul tersebut. Agar berkualitas, sebuah modul bukan hanya merupakan kumpulan perintah yang saling berhubungan tetapi juga harus merupakan rincian program yang lengkap. Sebuah modul harus memiliki satu jalan masuk *(entry)* dan satu jalan keluar *(exit)* saja, dengan begitu tidak ada batasan jumlah perintah modul yang akan dibuat, setiap modul seharusnya tidak terlalu besar, sekitar 50 baris (atau satu halaman) *source code* saja.

Coupling/Perangkai

Sebuah istilah yang menggambarkan hubungan antar modul disebut sebagai *Coupling* yang mengukur seberap erat hubungan yang terjadi diantara dua buah modul. Pada dasarnya dua buah

modul dapat di pasang jika salah satu modul mengotrol yang lainnya dengan berbagai macam cara.

Highly coupled modules menunjukan modul-modul yang memiliki hubungan sangat kuat, dalam kenyataannya salah satu tergantung pada yang lainnya, untuk memahami suatu modul kita juga harus memahami jenis modul lainnya. *Loosly coupled module* adalah modul-modul yang memiliki ketergantungan yang sangat kecil terhadap modul lainnya. Modul-modul dinyatakan *decoupled* jika modul-modul tersebut betul-betul berdiri sendiri tidak memiliki hubungan dengan yang lain.

Cohesion/Kohesi

Cohesion adalah istilah dalam pemrograman yang digunakan untuk menggambarkan eratnya ikatan diantara perintah-perintah yang ada pada suatu modul. Tepatnya, perintah-perintah yang pada modul tersebut memiliki hubungan yang sangat erat untuk sama-sama melakukan suatu fungsi tertentu. Modul yang memiliki hubungan yang sangat tinggi hanya berisi perintah-perintah yang diperlukan untuk sebuah operasi atau tugas tertentu. *Cohesion* dapat diartikan sebagai perekat yang membentuk sebuah modul.

8.7.5. Pemrograman Egoless (Secara terbuka)

Tahun 1971 Gerald Weinberg meluncukan sebuah buku yang berjudul *The Psycology of Computer Programming.* Melalui buku ini dia menemukan hubungan antara sikap programer dengan kualitas program yang mereka buat, programer sering berpikir bahwa pekerjaannya agak asosial. Banyak programer bekerja menyendiri untuk mendapatkan pemecahan masalah. Hasilnya, banyak programer yang terlalu membanggakan diri dengan hasil pekerjaannya, biayanya tinggi dan programnya susah dipahami orang lain.

Weinberg mengusulkan model *Egoless,* menurutnya model ini pemrograman merupakan aktivitas sosial terbuka bagi orang lain yang membantu memberi masukan. Jelasnya model pemrograman ini akan merupakan suatu kegiatan keseharian yang dilaku kan dengan kolega atau teman sejawat yang bekerja sama untuk menjaga *software* dari kemungkinan *error*.

8.7.6. Review Secara Terstruktur

Salah satu cara untuk menerapkan pemrograman terbuka (Egoless programming) yaitu dengan membuat jadwal *review* secara terstruktur terhadap rancangan dan pengkodean suatu program (Structured Walktroughs).

Tujuan dasar *review* secara terstruktur ini adalah untuk mendeteksi kemungkinan adanya error (kesalahan) dan bukan untuk memperbaikinya. Review ini pada dasarnya dilakukan oleh team programer dimana setiap anggota memliki tanggung jawab masing masing.

Hal yang perlu ditekankan disini adalah *review* dilakukan terhadap programnya bukan terhadap orang yang membuat program sehingga akan terhindar dari perasaan sakit hati.

Beberapa manajer pengolahan *data* tidak begitu yakin terhadap perlunya review terhadap program yang telah dibuat mungkin review ini tidak cocok untuk semua jenis pemrograman tapi review ini telah cukup berhasil pada pemrograman terstruktur.

8.7.7. Pimpinan Tim Programer

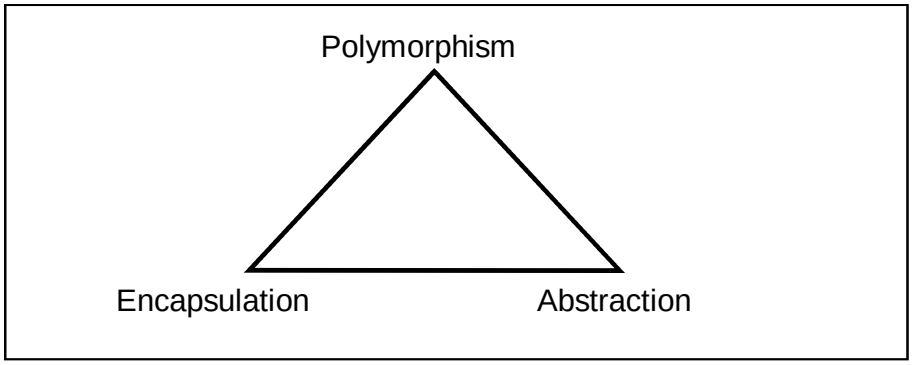
Koordoninasi terhadap projek pemrograman yang besar mungkin lebih diperlukan dari pada pemikiran tentang programer terbuka atau review yang terjadwal. Semakin banyak programer yang mengerjakan suatu program maka akan semakin kurang konstribusi terhadap program-program yang dibuat. Apa yang diperlukan oleh suatu team adalah konsep integritas. Cara yang terbaik untuk mencapai konsep intagritas tadi adalah dengan menggunakan seorang kepala programer yang bertugas dalam menyiapkan kebutuhan *software* dan perancangan seluruh struktur program untuk memimpin suatu team programer sehingga program yang dibuat konsisten dengan tujuan pemrograman.

Team programer terdiri dari terdiri dari beberapa spesialis dengan peran yang telah ditentukan misalkan sebagai penyusun dokumentasi, pemegang administrasi, programer dan penguji program sehingga projek pembuatan program bisa berjalan secara efektif dan efisien.

8.8 Pemrograman Berorientasi ke Objek

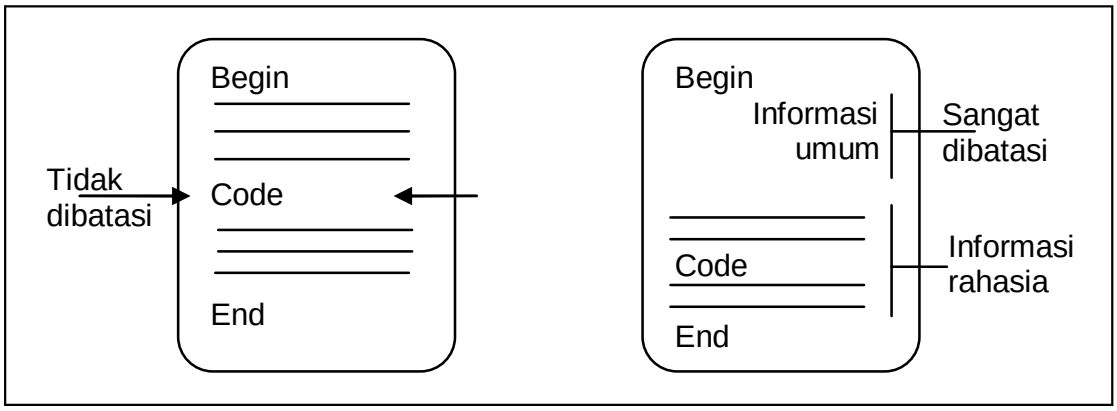
Seperti dijelaskan di atas pemrograman berbasis objek pada dasarnya merupakan cara berfikir programer pada saat membuat program. Ada tiga landasan berfikir didalam pemrograman berbasis objek yaitu : *Encapsulation* dan *information hiding*, *Clasification* dan *Abstract data types, Polymorphysm* dengan cara *Inheritance.*

Gambar 8.24 Object Oriented Triangle



- ***Encapsulation* dan *Information hiding*** - secara teknis hal ini sebenarnya tidak baru, hanya istilahnya saja yang baru. Istilah lainnya yang identik adalah pembuatan modul. Jadi program dipecah menjadi beberapa modul *(capsul)* sehingga tidak tampak *(hiding)*.

Gambar 8.23 Information hiding

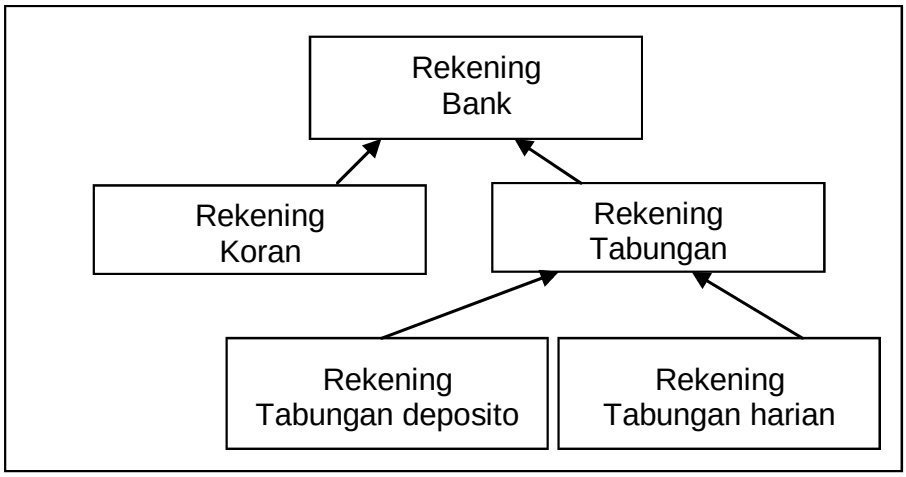


- ***Clasification* dan *Abstract data types*** - pada dasarnya menentukan jenis *data* apa yang diinginkan sehingga memberikan kemampuan operasi secara menyeluruh untuk setiap jenisnya.

  Misalkan *Integer, Character, Real* dan lain-lain

- ***Polymorphysm* dengan cara *Inheritance*** - menentukan secara eksplisit *class* apa yang diinginkan dengan menggunakan *inheritance* (turunan).

Gambar 8.25



Gambar diatas menunjukan rekening bank sebagai suatu *class.* Rekening bank terdiri dari dua sub *class* yang merupakan turunannya yaitu rekening koran dan rekening tabungan, demikian juga rekening tabungan memiliki *sub class* turunannya yaitu rekening tabungan deposito dan rekening tabungan harian.

Rangkuman

Komputer tidak dapat bekerja tanpa diprogram. Bahasa komputer pertama yang digunakan adalah bahasa mesin. Bahasa mesin merupakan bilangan biner yang terdiri dari angka 0 dan 1. Adanya bahasa *assembly* yang berisi simbol-simbol sedikit lebih menyederhanakan masalah dibandingkan bahasa mesin tapi bahasa *assembly* tetap saja dianggap sulit. Karena itu, bahasa *assembly* sebagai bahasa generasi kedua tetap dikategorikan sebagai bahasa tingkat rendah. Munculnya bahasa generasi ketiga seperti FORTRAN, COBOL, BASIC, PASCAL dan lain-lain lebih menyederhanakan lagi proses pemrograman karena bahasa tersebut menyerupai bahasa yang dimengerti oleh manusia. Perkembangan bahasa pemrograman terus berlangsung sampai sekarang sejalan dengan berkembangnya teknik pemrograman, hal ini ditunjukan dengan adanya bahasa bahasa generasi baru yang tingkatannya lebih tinggi seperti *Visual foxpro,Visual Basic,Bahasa bahasa yang tergabung dalam Visual Studio.net* dan lain-lain yang ditunjang dengan teknik pemrograman seperti pemrograman terstruktur dan pemrograman berdasarkan objek.

Soal

1. Sebutkan berbagai macam bahasa pemrograman generasi ke dua,tiga dan empat.
2. Sebutkan dua jenis bahasa tingkat rendah.
3. Mengapa PASCAL dianggap bahasa yang paling baik dalam pemrograman ?
4. Sebutkan karakteristik bahasa pemrograman secara umum.
5. Sebutkan proses-proses pembuatan program.

Tugas

1. Coba jelaskan mengapa bahasa tingkat tinggi sangat membantu sekali programer dalam membuat program.
2. Coba jelaskan apa perbedaan *compiller* dan *interpreter.*
3. Coba jelaskan mana yang harus ditentukan terlebih dahulu *output* program (informasi) atau *input* program.
4. Mengapa dalam membuat program, program harus dibuat berdasarkan kebutuhan pemakai.
5. Apa keuntungan dari pemrograman terstruktur.