LABORATORIUM KOMPUTASI
DEPARTEMEN TEKNIK SIPIL
FAKULTAS TEKNIK
UNIVERSITAS ANDALAS

NAMA :

NIM :

KELOMPOK :

ASISTEN :

PENUNTUN PRAKTIKUM

PEMROGRAMAN KOMPUTER | 2024

(TSI - 62105)

FOTRAN 77

DAFTAR ISI

MODUL I
PENGENALAN KOMPUTER

1.1 Pendahuluan

1.1.1 Tujuan

1. Memahami komponen utama komputer pribadi (*Personal Computer*).
2. Memahami tentang sistem operasi (*Operating System*).
3. Memahami tata cara pembuatan algoritma dan diagram alir (*flowchart*).

1.1.2 Komponen Komputer

Gambar 1.1. Perangkat Komputer

Secara umum, komputer pribadi mempunyai komponen yang saling berhubungan dan berinteraksi antara satu dengan lainnya. Komponen tersebut antara lain :

a. *Central Processor Unit* (CPU)

Gambar 1.2. *Processor Intel*

Kegunaan utama dari CPU adalah untuk memproses data dalam *main memory*. CPU mampu mengolah data, baik data aritmatik maupun operasi logika yang tersimpan dalam *main memory*. CPU merupakan bagian yang paling rumit dalam komputer yang terdiri atas *chip* untuk melakukan semua instruksi dalam komputer. CPU mempunyai *register* dan *chache memory* untuk menyimpan semua instruksi dan data dari dan ke *main memory*. Di samping itu, dalam CPU terdapat jaringan yang sangat rumit untuk melakukan perintah yang akan dilakukan dalam ALU (*Arithmatic and Logic Unit*).

b. *Main Memory.*

Gambar 1.3. RAM

Perangkat lunak/program (*software*) dan data-data yang diperlukan akan disimpan dalam *main memory*. Perintah-perintah dalam program diambil dan dieksekusi oleh CPU dari *main memory*. Semua data maupun pernyataan yang berkenaan dengan program disimpan dalam *main memory* ini. Secara fisik *main memory* merupakan *cell* yang diindeks berdasarkan alamat-alamat tertentu. *Cell* berisi sejumlah *bytes*. Tiap *byte* mempunyai 8 *bits*. Memory dapat dalam bentuk RAM (*Random Access Memory*) maupun dalam bentuk ROM (*Read Only Memory*). RAM hanya dapat menyimpan data untuk sementara hingga program selesai dieksekusi. Sedangkan ROM dapat menyimpan program yang sifatnya tetap atau permanen tetapi hanya bisa dibaca.

c. *Secondary Storage.*

Gambar 1.4. *Harddisk*

Secondary Storage merupakan penyimpan data secara permanen. *Harddisk* merupakan salah satu *Secondary Storage* yang populer karena kemampuannya untuk menyimpan data dalam kapasitas banyak.

d. *Input Device.*

Keyboard

Mouse

Scanner

Joystick

Gambar 1.5. *Input Device*

e. *Output Device*

Projector

Monitor

Speaker

Printer

Gambar 1.6. *Output Device*

I/O *device* merupakan jalur untuk mengirim dan menerima data dari dan ke komputer. I/O *device* akan terhubung dengan perangkat lain seperti *keyboard*, *monitor*, *proyektor*, *printer* dan perangkat lain.

1.2 Sistem Operasi (*Operating System*)

Sistem Operasi merupakan perangkat lunak yang mengatur pengoperasian, pengendalian dan mengkoordinir semua kegiatan dalam perangkat komputer. Sistem operasi merupakan perangkat lunak dasar yang harus dimiliki oleh setiap perangkat komputer pribadi. Contoh sistem operasi yang sering digunakan untuk komputer pribadi adalah *Windows*, *Macintosh* dan *LINUX*.

- Microsoft Windows atau yang lebih dikenal dengan sebutan Windows adalah keluarga sistem operasi. yang dikembangkan oleh Microsoft, dengan menggunakan antarmuka pengguna grafis. Sistem operasi Windows telah

berevolusi dari MS-DOS, sebuah sistem operasi yang berbasis modus teks dan *command-line*.

- Linux adalah sistem operasi berbasis **UNIX** yang dapat berjalan di berbagai macam perangkat keras terutama di PC berbasis Intel x86.
- Dikatakan berbasis UNIX karena Linux dikembangkan menurut standar yang dimiliki UNIX dan kemampuan yang sama dengan UNIX.
- Macintosh (MAC) adalah salah satu jenis komputer berbasis powerPC yang di produksi oleh *apple*. MAC dibuat khusus untuk komputer Macintosh dan tidak kompatibel dengan PC yang berbasis IBM.

1.3 Pemrograman Komputer

1.3.1 Bahasa Pemrograman

Pada dasarnya pemrograman komputer merupakan serangkaian perintah yang terstruktur dan sistematis kepada komputer. Karena komputer hanya dapat memahami perintah dalam bentuk bahasa mesin, maka diperlukan perangkat lunak yang berguna untuk menginterpretasikan perintah dari *high level language* ke bahasa mesin. Walaupun saat ini banyak perangkat lunak yang tersedia, seperti Java, C/C++, Pascal, COBOL, Visual Basic. Namun bahasa pemrograman Fortran lebih mudah dipahami dan dipelajari.

1.3.2 Kegunaan Bahasa Pemrograman

Pada praktikum ini hanya digunakan bahasa FORTRAN-77, yang umumnya juga digunakan untuk menyelesaikan masalah – masalah di bidang teknik sipil. Karena lebih mudah dalam pengoperasian perhitungannya. Adapun mata kuliah yang menggunakan bahasa pemrograman ini seperti Metode numerik dan teknik simulasi.

1.4 Pembuatan Program

Sebelum menyusun pemrograman, langkah-langkah berikut sangat penting untuk dilakukan :

1. Perumusan Masalah

   Perumusan masalah merupakan proses untuk mendeskripsikan masalah serta penjabaran masalah tersebut dalam bentuk yang lebih spesifik, yakni menentukan bagian-bagian yang masuk dalam kategori input, proses, dan output. Contoh sederhana berikut memperlihatkan perumusan masalah dimaksud.

   Perhitungan Kecepatan (V) ditentukan dengan persamaan V=S/T. Dari rumusan tersebut terlihat bahwa data masukan adalah Jarak (S), dan waktu (T). Penyelesaian persamaan V=S/T merupakan proses dan output adalah kecepatan (V).

2. Algoritma

   Algoritma merupakan langkah-langkah yang harus dilakukan dalam menyelesaikan masalah di atas. Langkah-langkah ini harus sistematis dan terstruktur, sehingga penyelesaian masalah diperoleh sesuai dengan yang diinginkan. Algoritma masalah dalam contoh di atas dapat ditulis sebagai berikut:

   1. Mulai
   2. Baca: Jarak (S), Waktu (T)
   3. Hitung: Kecepatan (V) = Jarak (S) / Waktu (T)
   4. Tulis: Kecepatan (V)
   5. Selesai

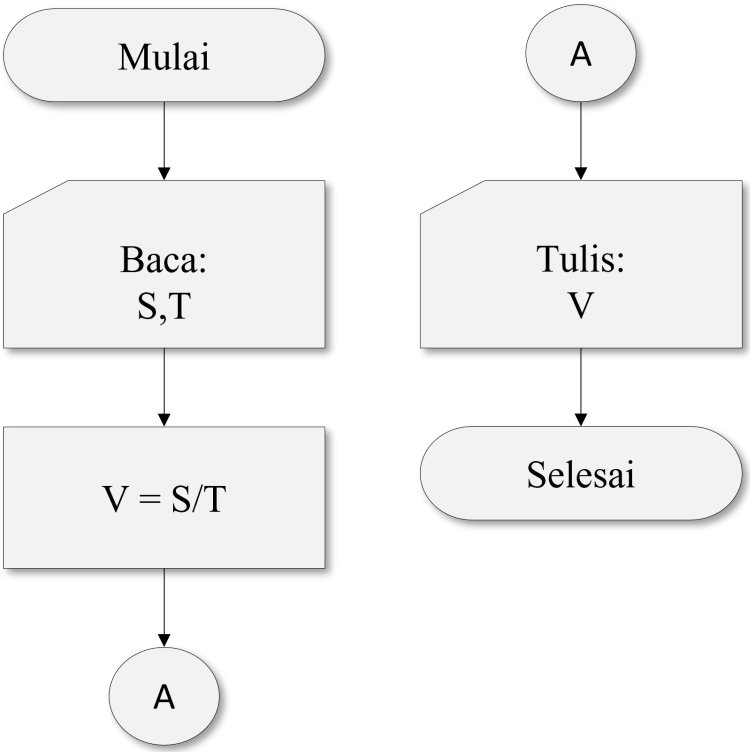
3. Diagram Alir

   Diagram alir merupakan gambaran/notasi tentang alur (pola pikir) program komputer yang akan dibuat. Diagram alir memegang peranan penting baik dalam menyusun program komputer maupun pada saat pengecekan ulang jika program komputer mempunyai kesalahan.

4. Pembuatan Diagram Alir Dengan Software

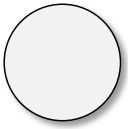
Pada praktikum ini, pembuatan *Flowchart* menggunakan software *Microsoft Visio*, yang merupakan salah satu software yang digunakan dalam pembuatan diagram alir dalam suatu pemrograman.

Gambar di bawah ini adalah contoh diagram alir dengan menggunakan *Microsoft Visio* yang menjelaskan algoritma sebelumnya.



Keterangan notasi dari diagram alir di atas:

Tabel 1.1.a Gambaran/notasi yang umum digunakan dalam diagram alir

No.	Gambar/notasi	Arti
1.		Titik terminal untuk mulai, selesai dan kembali. Jika untuk memulai tulis MULAI, untuk selesai tulis SELESAI.
2.		Input dan output. Jika digunakan untuk input, tulis BACA : dan jika untuk output, tulis : dan diikuti dengan variabel yang akan dibaca /ditulis.
3.		Menyatakan suatu proses. Tuliskan proses yang akan dilakukan komputer dalam kotak yang tersedia.
4.		Tanda penghubung. Huruf atau angka numerik dalam lingkaran menunjukkan bahwa alur program dilanjutkan ke tahapan dengan alamat nomor yang diberikan.
5.		Gambar / notasi ini untuk memulai dan mengakhiri proses perulangan.
6.		Kedua notasi ini digunakan untuk alur dengan keputusan (*if conditional*).

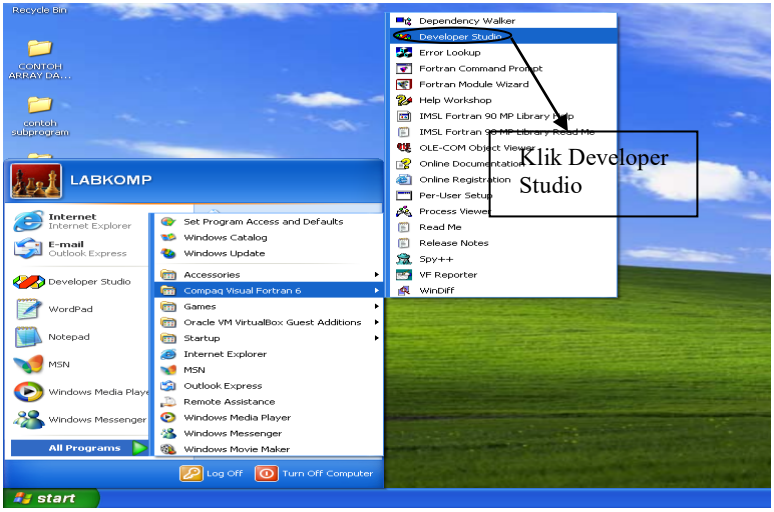
7.		Gambar / notasi ini untuk subprogram.
8.		Garis aliran. Ujung panah menunjukkan arah aliran.

1.5 Cara Membuka Compact Visual Fortran

Untuk membuka Compact Visual Fortran hampir sama dengan membuka aplikasi lain yang tersedia pada komputer.

Langkah-langkah nya sebagai berikut:

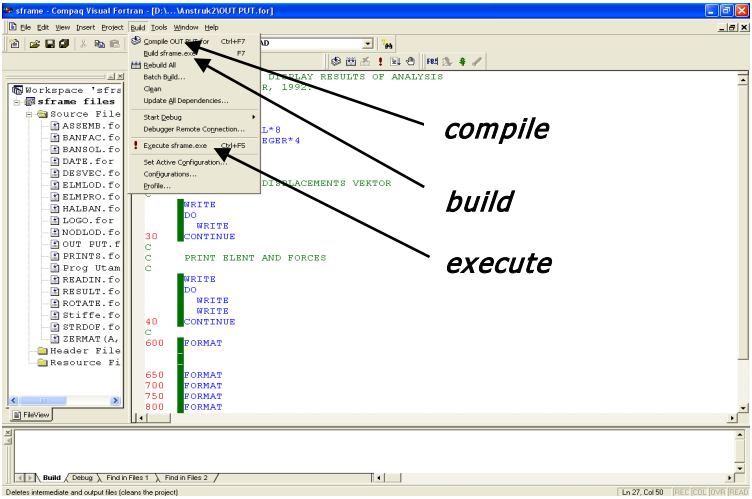
1. *Shortcut* developer studio pada jendela *windows* jika tidak ada,
   klik start pada bagian kiri bawah dan klik *developer studio*
2. Jika tidak pilih pada all programs lalu klik *Compaq Visual Fortran* 6.
   Lalu klik *developer studio*.



Gambar 2.1. Mengaktifkan *Compaq Visual Fortran*

1.6 Cara Menjalankan Program

Program yang telah disusun dalam bentuk program sumber (*source code*) harus diterjemahkan ke dalam bahasa mesin. *Interpreter* untuk menerjemahkan tersebut dikenal sebagai *compiler*. Dalam praktikum ini digunakan *compiler* Compaq Visual Fortran Edisi 6.5.0. **Gambar 2.2**. memperlihatkan cara untuk mengaktifkan *compiler Compaq Visual Fortran*. Cara membuat dan menyusun program computer dalam editor akan dijelaskan asisten dalam praktikum.



Gambar 2.2. Menu untuk *Compile*, *Build* dan *Execute*

Untuk mendapatkan output dari suatu program, dilakukan tahap-tahap sebagai berikut :

1. Tahap *Compile*

   Tahap *compile* bertujuan untuk mengubah bahasa manusia ke bahasa mesin. langkahnya sebagai berikut:

   - klik pada **[build] > [compile]**
   - atau klik  pada *toolbar*
   - atau **Ctrl + F7**

2. Tahap *Build*

   Tahap *build* bertujuan untuk membangun suatu program.

Langkahnya sebagai berikut:

- klik pada **[build] > [build** nama file**.exe]**
- atau klik  pada *toolbar*
- atau **F7**

3. Tahap *Execute*

Execute program bertujuan untuk menjalankan program yang sudah diterjemahkan ke bahasa mesin dan yang di bangun.

Langkahnya sebagai berikut:

- klik pada **[build] > [execute]**
- atau klik  pada *toolbar*
- atau **Ctrl + F5**

MODUL II
DASAR-DASAR PEMROGRAMAN

2.1 Pendahuluan

2.1.1 Tujuan

1. Memahami dasar-dasar pemrograman komputer yang meliputi tentang format dan struktur bahasa pemrograman bahasa FORTRAN-77 beserta tipe data dan variabel yang digunakan.
2. Dapat menyusun program komputer sederhana.

2.1.2 Bahasa Pemrograman Fortran

Fortran adalah bahasa tingkat tinggi pertama dan merupakan singkatan dari *FORmula TRANslator*. Dalam praktikum ini kita menggunakan bahasa FORTRAN standar 77(selanjutnya ditulis FORTRAN-77).

Bahasa pemrograman FORTRAN-77 merupakan *high level language* yang digunakan untuk membuat perintah tertentu kepada komputer. Dalam penyusunan program, struktur pemrograman dengan bahasa FORTRAN-77 mengikuti pola berikut:

1.Untuk program utama

[PROGRAM <nama program>]
<deklarasi external file dan module>
<deklarasi variabel>
<inisialisasi variabel>
.
.
<executable statement>
.
STOP
END

2.Untuk subprogram

<blok subprogram>

<deklarasi variabel>

<inisialisasi variabel>

<executable statement>

STOP

END

Walaupun dalam bahasa Fortran tidak membedakan huruf besar dan kecil, akan tetapi sangat disarankan untuk selalu menggunakan huruf besar dalam menyusun program komputer. Berikut ini akan dijabarkan tentang bagian-bagian dari struktur program tersebut.

2.2 Elemen dari Program Fortran

1. Nama Program (*Program Statement*)

Program statement, dalam bentuk nama program, diperlukan untuk mengidentifikasi program. Nama program sebaiknya dibuat berdasarkan apa yang akan dilakukan dalam program tersebut. Sangat disarankan, nama program identik dengan nama file (*source code*).

Contoh : untuk nama program

" PROGRAM MENGHITUNG VOLUME BALOK"

untuk nama file

"PROGRAM MENGHITUNG VOLUME BALOK.FOR"

2. Inti Program (*Statement*)

Statement merupakan inti program yang berupa instruksi-instruksi kepada komputer. Logika akan dituangkan ke dalam bentuk *statement* untuk diproses.

3. Komentar (*Command*)

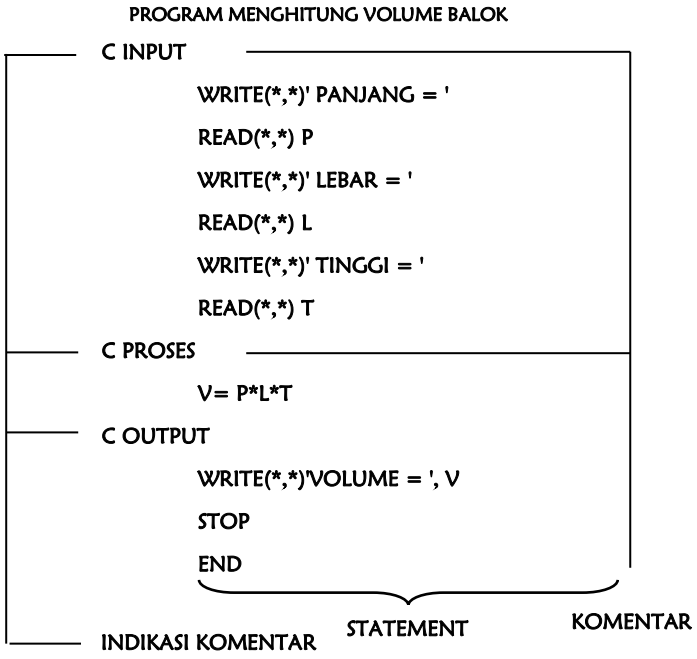
Walaupun komentar tidak akan diproses oleh *compiler*, komentar mempunyai fungsi yang penting, terutama untuk menjelaskan bagaimana program itu disusun, arti dari variabel-variabel yang digunakan. Komentar ini sangat penting,

baik bagi yang menyusun program itu sendiri maupun bagi pihak lain yang membaca program tersebut. Dalam pemrograman, *comments* dinotasikan dengan "C", "D" atau "*" (asterik), dalam pembuatan suatu program disarankan menggunakan satu notasi *comment*.

Contoh:

```
      PROGRAM MENGHITUNG VOLUME BALOK
C     INPUT
      WRITE(*,*)'PANJANG='
      READ(*,*)P
      WRITE(*,*)'LEBAR='
      READ(*,*)L
      WRITE(*,*)'TINGGI='
      READ(*,*)T
C     PROSES
      V=P*L*T
C     OUTPUT
      WRITE(*,*)'VOLUME=',V
      STOP
      END
```

Contoh *Listing Program* menggunakan Notasi *Comment*



2.3 Tipe Variabel

Dalam FORTRAN-77 terdapat 3 tipe variabel yang paling umum digunakan, yakni:

- **REAL** (mewakili angka desimal (*floating point*))
- **INTEGER** (bilangan bulat)
- **CHARACTER** (rangkaian dari karakter tertentu).

Penamaan variabel sebaiknya memiliki arti. Disarankan hanya menggunakan maksimum 6 karakter dalam setiap nama variabel. Penggunaan nama variabel yang panjang sering kali tidak sesuai untuk komputer tertentu. *Default* untuk bahasa FORTRAN-77 adalah :

- **REAL untuk semua variabel dengan awal huruf A-H,O-Z.**
- **INTEGER untuk semua variabel dengan awal huruf I-N.**

Walau semua variabel tersebut secara otomatis masih dalam kategorinya masing-masing, akan tetapi sangat direkomendasikan untuk selalu mendeklarasikannya di awal program sumber. Berikut menunjukkan deklarasi untuk variabel REAL dan INTEGER.

EKSPLISIT

Dalam *Statement* EKSPLISIT variabel yang di definisikan hanya variabel yang ingin diganti tipe variabelnya. Contoh untuk variabel integer:

```
INTEGER A

READ(*,*)A
WRITE(*,*)A

READ(*,*)AP
WRITE(*,*)AP
STOP
END
```

Contoh *Listing Program* Penggunaan Integer

Output:

```
3
           3
3
   3.000000
Press any key to continue_
```

Contoh *Output Program* Penggunaan Integer

Jadi integer A akan menjelaskan variabel A sebagai integer, namun tidak dapat menjelaskan variabel AP sebagai integer.

IMPLICIT

Statement IMPLICIT mendefinisikan tipe dari semua variabel yang huruf pertama dari nama yang disebutkan. Huruf yang sama tidak boleh didefinisikan ulang dengan *statement* IMPLICIT. Variabel atau huruf yang sudah didefinisikan dengan *statement* IMPLICIT dapat didefinisikan ulang dengan *statement* EKSPLISIT.

Contoh penggunaannya:

```
IMPLICIT INTEGER A

READ(*,*)A
WRITE(*,*)A

READ(*,*)AP
WRITE(*,*)AP

STOP
END
```

Contoh *Listing Program* penggunaan *Implicit* Integer

Output:

```
3
           3
3
           3
Press any key to continue_
```

Contoh *Output Program* untuk penggunaan Implicit Integer

Jadi dengan menggunakan Implicit Integer akan membuat semua variabel yang diawali huruf A akan di definisikan sebagai Integer. Di samping itu, secara lengkap tipe data yang digunakan dalam FOTRAN-77 diberikan sebagai berikut:

Tabel 2.1 Tipe Variabel

Tipe	Byte	Keterangan
INTEGER	4	Bilangan bulat (-2147483647 – 2147483647)
INTEGER*2	2	Bilangan bulat (-32767 – 32767)

INTEGER*4	4	Bilangan bulat (-2147483647 – 2147483647)
REAL	4	Bilangan desimal (*floating point*) Untuk bilangan positif 8.43E-37 – 3.37E38 Untuk bilangan negatif -3.37E38 – -8.43E-37
REAL*4	4	Sama dengan REAL
REAL*8	8	Bilangan desimal (*floating point*)
CHARACTER*n <var1>	-	Banyak ruang untuk semua variabel yang didefinisikan sebagai karakter
CHARACTER <var1>*n	-	Banyak ruang untuk variabel tertentu

2.4 Dasar-Dasar Pemrograman

Alur dari sebuah program ditunjukkan pada gambar di bawah.

Gambar 2.3. *Flowchart*

Secara umum, pemrograman komputer hanya terbagi atas 3 bagian, yakni *input*, proses dan *output*. Proses *Input/Output* berhubungan dengan alat (*device*) I/O standar seperti *keyboard*, konsol (monitor), file serta periferal lainnya. Bagian terbesar dari pemrograman biasanya terletak pada bagaimana proses (proses perhitungan misalnya) dilakukan. Bagian *Input/Output* dapat dikelompokkan pada pembahasan yang sama, sedangkan bagian proses ada dipisahkan menurut kegunaannya.

Dalam penulisan program sumber (*source code*), aturan/format berikut harus selalu dipatuhi sesuai tabel di bawah ini:

Tabel 2.2 Penulisan *Source Code*

1	Kolom 1	Komentar. Jika pada kolom 1 diberi huruf C atau *, maka semua karakter yang

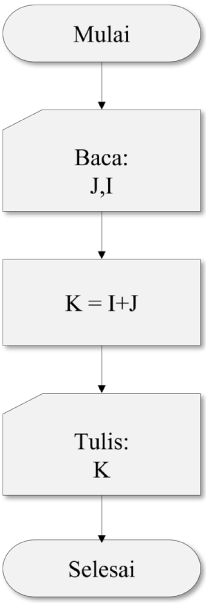
		mengikutinya dikategorikan sebagai komentar.
2	Kolom 1-5	Tempat penulisan label yang merupakan alamat perintah lanjutan. Label harus ditulis dengan bilangan bulat (integer)
3	Kolom 6	Tempat tanda sambung. Semua karakter dapat digunakan sebagai tanda sambung. Disarankan untuk tidak menggunakan karakter / dan \.
4	Kolom 7-72	*Main body program*. Tempat penulisan semua perintah yang diinginkan.
5	Kolom 73-80	Tidak digunakan.

Input/Output (I/O)

Perintah untuk *Input/Output* digunakan dalam membaca data masukan dan menuliskan data keluaran dari memori komputer dari dan ke periferal tertentu.

- Perintah untuk **Input** menggunakan pernyataan **READ**.
- Perintah untuk **Output** menggunakan pernyataan **WRITE**.

Diagram alir (*Flowchart*)



Gambar 2.4. Contoh penggunaan *Flowchart*

Contoh penggunaan *Input/Output:*

```
READ(*,*)J
READ(*,*)I
K=I+J
WRITE(*,*)K
STOP
END
```

Contoh *Listing Program* sederhana

Bila program dijalankan akan didapat hasil :

```
7 ------------------------> ........hasil dari READ(*,*)J
8 ------------------------> ........hasil dari READ(*,*)I
                  15 -----> ........hasil dari WRITE(*,*)K
Press any key to continue
```

Contoh *Output Program* sederhana

2.4.1 FORMAT

Pernyataan FORMAT digunakan untuk membentuk format (bentuk) data masukan/keluaran yang meliputi tentang tata letak, tipe dan panjang data.

Ada 3 macam format pada fortran, yaitu:

1. Format bebas

   Contoh: WRITE(*,*)'LABKOMP'

2. Format dalam

   Contoh: WRITE(*,'(A)')'LABKOMP'

3. Format luar

   Bentuk umum Format luar ditulis sebagai berikut:

<label> FORMAT (*edit list*)

Keterangan:

<label>: Menunjukkan label yang merupakan alamat dari pernyataan READ/WRITE

edit list: Merupakan *edit descriptor* yang merupakan perintah untuk mengatur tata letak, tipe dan panjang data

Contoh penggunaan :

```
7     FORMAT(A)
      WRITE(*,7)'LABKOMP'
      STOP
      END
```

Contoh *Listing Program* Penggunaan Format Dalam

2.4.2 *Edit Descriptor*

Edit descriptor merupakan perintah dalam membentuk format I/O. Dalam **Tabel 2.3.** berikut diberikan *edit descriptor* yang umum digunakan dalam FORTRAN-77.

Tabel 2.3 Edit Descriptor dalam FORTRAN-77

No	Tipe Editing	Bentuk umum	Arti notasi	
1	Character	**<n>A<w>**	**<n>** **<w>**	jumlah pengulangan jumlah digit character
2	Integer	**<n>I<w>**	**<n>** **<w>**	jumlah pengulangan jumlah digit integer

3	Real	**<n>F<w>.<d>**	**<n>** **<w>** **<d>**	jumlah pengulangan jumlah digit termasuk desimal jumlah digit desimal
4	*Apostrophe*	**'<k>'**	**<k>**	karakter
5	*Positional*	**<n>X**	**<n>**	jumlah pengulangan
6	Slash	**/**		turun satu baris
7	Backslash	****		naik satu baris

Contoh :
Penggunaan *edit descriptor* I.

```
J=12345
WRITE(*,'(I4)')J
WRITE(*,'(I5)')J
WRITE(*,'(I6)')J
STOP
END
```

Contoh *Listing Program* Penggunaan *Edit Descriptor* I

Bila program dijalankan akan didapat hasil :

```
****
12345
 12345
Press any key to continue_
```

Contoh *Output Program* Penggunaan *Edit Descriptor* I

Penjelasan :

- Hasil **** didapat karna nilai I adalah 5 digit sedangkan format output hanya untuk 4 digit (I4)
- Hasil 12345 didapat dari format I5
- Hasil 12345 didapat dari format I6.

  → Terdapat blank 1 digit karena digunakan I6

Contoh Penggunaan *edit descriptor* F:

```
B=1234.567
WRITE(*,'(F8.2)')B
WRITE(*,'(F8.3)')B
WRITE(*,'(F8.4)')B
STOP
END
```

Contoh *Listing Program Edit Descriptor* F

Bila program dijalankan akan didapat hasil :

```
 1234.57
1234.567
********
Press any key to continue_
```

Contoh *Output Program* Penggunaan *Edit Descriptor* F

Penjelasan :

F8.3 - variabel dapat menyimpan nilai sebesar **8** digit termasuk koma

- maksimal **3** angka di belakang koma

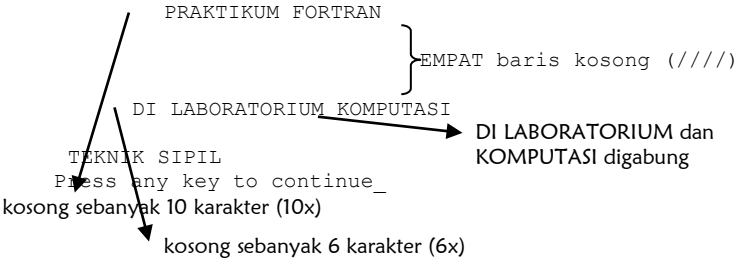
1	2	3	4	.	5	6	7

Contoh penggunaan *edit descriptor* ' ', X, /, \

```
WRITE(*,'(10X,A,////)')'PRAKTIKUM FORTRAN'
WRITE(*,'(6X,A,\)')'LABORATORIUM'
WRITE(*,'(X,A,//)')'KOMPUTASI'
WRITE(*,*)'TEKNIK SIPIL'
STOP
END
```

Contoh *Listing Program Edit Descriptor* ' ', X, /, \

Bila program dijalankan akan didapat hasil :



Contoh *Output Program* Penggunaan *Edit Descriptor* ' ', X, /, \

MODUL III
PROSES OPERATOR DAN LOGIKA

3.1 Pendahuluan

3.1.1 Tujuan

1. Memahami perintah-perintah berupa proses dalam pemrograman Komputer.
2. Memahami tentang logika dan kontrol dalam proses.
3. Dapat menyusun program komputer lanjutan.

Perhitungan Aritmatika sangat diperlukan dalam bidang teknik sipil dan untuk menyelesaikan sebuah perhitungan kita membutuhkan operator aritmatika, maka dalam modul ini akan dibahas tentang operator aritmatika pada FORTRAN-77.

3.2 Operator aritmatika

Operator aritmatika memegang peranan penting dalam pemrograman komputer dengan FORTRAN-77 karena dengan operator aritmatika inilah rumus-rumus matematika diterjemahkan ke program sumber. Operator aritmatika dan hierarkinya disusun sebagai berikut :

Tabel 3.1. Operator Aritmatika dan Hierarki

Operator	Operasi/Arti	Hierarki
**	Pangkat	1
*	Kali	2
/	Bagi	2
+	Tambah	3
-	Kurang	3
=	Sama dengan	-

Proses aritmatika dengan hierarki yang lebih tinggi akan di proses terlebih dahulu. Akan tetapi jika digunakan tanda kurung"( )", maka proses aritmatika ini yang akan diproses terlebih dahulu, walaupun hierarkinya lebih rendah.

Contoh :

3**2+3/2*(4-6) ⟶ 3^2 + (3:2) * (4-6) =6

(7+3)/4**0.5-10 ⟶ (7+3) : $\sqrt{4}$ – 10 = - 5

3.3 Logika dan Kontrol

Ada 2 bentuk logika dan 1 kontrol yang umum digunakan dalam FORTRAN-77. Logika IF dan blok IF, sedangkan kontrol yakni GOTO.

1. IF

Pernyataan IF digunakan untuk mengatur alur program komputer sesuai dengan arah yang diinginkan. Bentuk umum penyataan IF adalah :

IF(<ekspresi>) *<statement>*

Tabel 3.2. Bentuk Umum Penggunaan *IF*

<ekspresi>	Menunjukkan ekspresi, logika atau aritmatik, yang harus dibandingkan.
<statement>	Pernyataan/perintah yang harus dilakukan jika hasil perbandingan dalam **<ekspresi>** benar.

Untuk *logical expression*, membandingkan dua nilai numerik, ***<statement>*** dalam IF di atas disusun dengan menggunakan salah satu *comparison (relation)* operator berikut:

Tabel 3.3 Operator *Logical Expression*

Operator	Arti
.GT.	Lebih besar dari
.GE.	lebih besar dan sama dengan
.LT.	Lebih kecil dari
.LE.	Lebih kecil dan sama dengan

.EQ.	Sama dengan
.NE.	Tidak sama dengan

Contoh penggunaan Operator:

```
WRITE(*,*)'NILAI A = '
READ(*,*)A
WRITE(*,*)'NILAI B = '
READ(*,*)B
IF(A.GT.B)WRITE(*,*)'NILAI A TERBESAR'
IF(A.LT.B)WRITE(*,*)'NILAI A TERKECIL'
STOP
END
```

Contoh *Listing Program* Penggunaan *Logical Expression*

Bila program dijalankan akan didapat hasil :

```
 NILAI A =
3
 NILAI B =
1
NILAI A TERBESAR
Press any key to continue_
```

Contoh *Output Program* Penggunaan *Logical Expression*

Untuk *logical variable*,

<statement> menggunakan salah satu dari *logical operator* berikut. Hasil akhir dari IF logika hanya benar atau salah (.TRUE. atau .FALSE.).

Tabel 3.4. Operator *Logical Variable*

Operator	Arti
NOT	Tidak
.AND.	Dan
.OR.	Atau

Contoh penggunaan 1 :

```
      WRITE(*,*)'NILAI A = '
      READ(*,*)A
      WRITE(*,*)'NILAI B = '
      READ(*,*)B
      WRITE(*,*)'NILAI C = '
      READ(*,*)C
      IF((A.GT.B).AND.(A.GT.C))WRITE(*,*)'NILAI A
     -TERBESAR'
      IF((A.LT.B).OR.(A.LT.C))WRITE(*,*)'NILAI A
     -BUKAN YANG TERBESAR'
      STOP
      END
```

Contoh *Listing Program* Penggunaan *Logical Variable* (.AND. dan .OR.)

Bila program dijalankan akan didapat hasil :

```
 NILAI A =
6
 NILAI B =
4
 NILAI C =
2
 NILAI A TERBESAR
Press any key to continue
```

Contoh *Output Program* Penggunaan *Logical Variable*

Contoh penggunaan 2 :

```
      WRITE(*,*)'NILAI A = '
      READ(*,*)A
      WRITE(*,*)'NILAI B = '
      READ(*,*)B
      IF(NOT(A.GT.B)) WRITE(*,*)'NILAI A TERKECIL'
      STOP
      END
```

Contoh *Listing Program* Penggunaan *Logical Variable* (NOT)

Bila program dijalankan akan didapat hasil :

```
 NILAI A =
4
 NILAI B =
8

 NILAI A TERKECIL
Press any key to continue
```

Contoh *Output Program* Penggunaan *Logical Variable*

Dalam eksekusinya, operator aritmatika akan dilaksanakan pertama kali, disusul dengan *comparison (relation) operator* dan *logical operation.*

2. Blok IF

Blok IF digunakan untuk pengaturan logika program yang lebih luas dan kompleks dengan pilihan yang lebih variatif. Blok IF disusun dengan struktur dasar IF-THEN-ELSE-ENDIF. Pola blok IF tersebut diberikan dalam Tabel 3.5. berikut.

Tabel 4.5. Pola blok IF

IF-THEN-ENDIF	IF-THEN-ELSE-ENDIF	IF-THEN-ELSEIF-THEN-ELSE-ENDIF

IF(<ekspresi>)THEN
pernyataan 1
pernyataan 2
…
pernyataan n
ENDIF
	IF(<ekspresi>)THEN
	pernyataan 1
	pernyataan 2
	…
	pernyataan n
	ELSE
	pernyataan a
	pernyataan b
	…
	pernyataan z
	ENDIF
		IF(<eksp1>)THEN
		pernyataan 1
		pernyataan 2
		…
		pernyataan n
		ELSEIF(<eksp2>)THEN
		pernyataan a
		pernyataan b
		…
		pernyataan z
		ELSE
		pernyataan a1
		pernyataan b2
		…
		Pernyataan zn
		ENDIF

Contoh penggunaan:

```
WRITE(*,*)'NILAI A = '
READ(*,*)A
WRITE(*,*)'NILAI B = '
READ(*,*)B
WRITE(*,*)'NILAI C = '
READ(*,*)C
IF((A.GT.B).AND.(A.GT.C))THEN
WRITE(*,*)'NILAI A PALING BESAR'
ELSE
WRITE(*,*)'NILAI A BUKAN YANG TERBESAR'
ENDIF
STOP
END
```

Contoh *Listing Program* Penggunaan Blok *IF*

Bila program dijal kan akan didapat hasil :

```
 NILAI A =
10
 NILAI B =
5
 NILAI C =
4
NILAI A PALING BESAR
Press any key to continue_
```

Contoh *Output Program* Penggunaan Blok *IF*

Hal yang perlu diperhatikan bahwa jangan mengontrol alur program menuju blok IF (Contohnya, menggunakan *statement GOTO*).

3. GOTO

Pernyataan GOTO digunakan untuk memindahkan alur program mengikuti label yang ditunjukkan dalam perintah GOTO. Ada banyak jenis GOTO yang digunakan, tetapi dalam praktikum ini hanya digunakan *Unconditional* GOTO.

Unconditional GOTO

Pola GOTO adalah pola yang paling sederhana, dengan bentuk umum:

GOTO <label>

Pernyataan ini memerintahkan komputer untuk melanjutkan proses alamat yang diberikan label.
Contoh penggunaan:

```
      WRITE(*,*)'NILAI A = '
      READ(*,*)A
      WRITE(*,*)'NILAI B = '
      READ(*,*)B
      IF(A.GT.B)GOTO 1
      IF(A.LT.B)GOTO 2
1     WRITE(*,*)'NILAI A BESAR DARI B'
      GOTO 3
2     WRITE(*,*)'NILAI A KECIL DARI B'
3     STOP
      END
```

Contoh *Listing Program* Penggunaan *GOTO*

Jika a>b maka program akan dilanjutkan ke label 1, sedangkan jika a<b maka program akan dilanjutkan ke label 2.

Bila program dijalankan akan didapat hasil :

```
 NILAI A =
3
 NILAI B =
7
 NILAI A KECIL DARI B
Press any key to continue_
```

Contoh *Output Program* Penggunaan *GOTO*

MODUL IV
PROSES PERULANGAN *(LOOP)*

4.1 Pendahuluan

4.1.1 Tujuan

1. Memahami perintah-perintah perulangan.
2. Dapat mengaplikasikannya ke dalam program komputer.

Dalam prakteknya banyak dijumpai suatu perintah harus dikerjakan berulang-ulang hingga diperoleh hasil sesuai dengan yang diinginkan. Perintah-perintah kontrol dan logika yang dijelaskan dalam modul 3 dapat digunakan untuk mengulang perintah-perintah yang sama dalam program. Perhatikan contoh sederhana proses penjumlahan angka 1 hingga 10 di bawah ini.

```
      I=0
2     I=I+1
      WRITE(*,*)I
      IF(I.LT.10)GOTO 2

      STOP
      END
```

Contoh *Listing Program* Perulangan

Bila program dijalankan akan didapat hasil :

```
           1
           2
           3
           4
           5
           6
           7
           8
           9
           10
Press any key to continue_
```

Contoh *Output* Program Perulangan

Dari contoh di atas dapat dilihat bahwa program akan terus diproses selama masih memenuhi syarat **(I.LT.10)** dan akan berhenti setelah syarat tidak lagi terpenuhi.

4.2 DO ...CONTINUE

```
        DO <label> var=n1,n2,inc
        pernyataan 1
        pernyataan 2
        ...
        pernyataan n
<label> CONTINUE
```

Tabel 4.1 Bentuk Umum *DO...CONTINUE*

<label>	Label yang menunjukkan batasan awal dan akhir dari proses yang akan diulang.
Var	Variabel yang merupakan variabel pengontrol perulangan.
n1	Nilai awal perulangan.
n2	Nilai akhir perulangan.
Inc	*Incremental*, penambahan langkah setiap perulangan. *Default*=1

Contoh penggunaan :

```
      DO 20 I=1,10,2
      WRITE(*,*)I
20    CONTINUE

      STOP
      END
```

Contoh *Listing Program* Perulangan dengan Menggunakan *DO CONTINUE*

Bila program dijalankan maka akan didapat hasil :

```
             1
             3
             5
             7
             9
 Press any key to continue_
```

Contoh *Output* Program Perulangan dengan menggunakan *DO CONTINUE*

Pada contoh di atas, nilai awal 1 dan nilai akhir 10 sedangkan *incremental* bernilai 2 sehingga akan terjadi 2 lompatan pada nilai awal sehingga didapatkan nilai 3 dan dituliskan begitu seterusnya seperti contoh di atas. Lalu, program akan berhenti ketika mendekati nilai akhir yaitu 10.

Selain itu, perhatikan perbedaannya jika proses penjumlahan angka 1 hingga 10 dalam contoh awal diprogram dengan menggunakan penggunaan DO dalam contoh penerapan berikut.

Contoh penggunaan :

```
      DO 20 I=1,6
      WRITE(*,'(2X,A,I2)')'SEMESTER KE ',I
20    CONTINUE

      STOP
      END
```

Contoh *Listing Program* Perulangan dengan Menggunakan *DO CONTINUE*

Bila program dijalankan akan didapat hasil yang sama seperti contoh pertama tadi. Akan terjadi kata perulangan sebanyak 8 buah.

```
  SEMESTER KE  1
  SEMESTER KE  2
  SEMESTER KE  3
  SEMESTER KE  4
  SEMESTER KE  5
  SEMESTER KE  6
Press any key to continue_
```

Contoh *Output Program* Perulangan dengan Menggunakan *DO CONTINUE*

Contoh penggunaan :

```
      DO 10 I=1,5
      READ(*,*)B
   10 CONTINUE
      WRITE(*,*)B
      STOP
      END
```

Contoh *Listing Program* Perulangan dengan Menggunakan *DO CONTINUE*

Apabila program dijalankan maka akan didapat hasil :

```
 1
 3
 7
 9
 5
    5.000000
Press any key to continue_
```

Contoh *Output* Program Perulangan dengan menggunakan *DO CONTINUE*

Contoh di atas merupakan contoh untuk proses perulangan tunggal. Yaitu *output* yang keluar adalah nilai terakhir dari nilai B. Dengan pola yang sama dimungkinkan untuk menyusun perulangan. Dengan demikian akan ada lompatan dalam (*inner loop*) dan lompatan luar (*outer loop*) seperti diperlihatkan dalam contoh perkalian matriks berikut:

Contoh penggunaan :

```
      DO I=1,2
      DO J=1,2

      WRITE(*,*)I,J
      ENDDO
      ENDDO
      STOP
      END
```

Contoh *Listing Program* Perulangan dengan menggunakan *DO ENDDO*

Bila program dijalankan akan didapat hasil :

```
         1         1
         1         2
         2         1
         2         2
Press any key to continue_
```

Contoh *Output Program* Perulangan dengan menggunakan *DO ENDDO*

Pada contoh di atas, perintah *ENDDO* mempunyai tujuan yang sama dengan *CONTINUE.*

4.3 DO WHILE...END DO

Perintah *DO WHILE...END DO* memproses satu atau sekelompok *statement* secara berulang berdasarkan syarat yang diberikan.

Contoh penggunaan :

```
I=0
DO WHILE(I.LT.10)
I=I+2
WRITE(*,*)I
ENDDO

STOP
END
```

Contoh *Listing Program* Perulangan dengan menggunakan *DO WHILE...END DO.*

Bila program dijalankan akan didapatkan hasil :

```
         2
         4
         6
         8
        10
Press any key to continue_
```

Contoh *Output Program* Perulangan dengan menggunakan *DO WHILE... END DO.*

Hal yang perlu dicatat bahwa setiap *loop* harus tertutup dimulai dari *loop* yang paling dalam. Di samping itu tidak diizinkan proses perhitungan dialihkan ke dalam *loop* (Contohnya dengan perintah *GOTO*).

MODUL V
PROSES *ARRAY*

5.1 Pendahuluan

5.1.1 Tujuan

1. Memahami penggunaan *array*.
2. Dapat mengaplikasikannya ke dalam program komputer.

5.2 Array

Kontras dengan variabel tunggal, dimana hanya ada satu nilai untuk tiap variabel, variabel yang menggunakan *array* mempunyai sekelompok nilai dengan variabel yang sama. Berikut contoh sederhana penggunaan *array*.

```
DIMENSION N(4)

READ(*,*)N(1)
READ(*,*)N(2)
READ(*,*)N(3)
READ(*,*)N(4)

WRITE(*,*)N
STOP
END
```

Contoh *Listing Program* Penggunaan *Array*

Bila program dijalankan maka akan keluar *output* seperti berikut :

```
2
3
4
5
           2           3           4           5
Press any key to continue_
```

Contoh *Output Program* Penggunaan *Array*

Pada program ini, terlihat hasil dari nilai N ada 4 buah yaitu 2,3,4,5 karena nilai yang kita inputkan sebanyak 4 kali dengan 4 ruang yang tersedia ("dapat dilihat pada DIMENSION N(4) "),

Ada 2 bentuk deklarasi yang umum digunakan, yaitu:

a. Dengan *DIMENSION*

Penyataan *DIMENSION* mempunyai bentuk umum :

DIMENSION var(n1:m1,n2:m2,..,nl:ml),...

Tabel 5.1 Keterangan

Var	Variabel yang mempunyai indeks, baik real, integer maupun karakter.
n1:m1	Nilai awal dan nilai maksimum indeks ke-1.
n2:m2	Nilai awal dan nilai maksimum indeks ke-2.
nl:ml	Nilai awal dan nilai maksimum indeks ke-l.

Dalam penyataan *DIMENSION*, variabel real, integer maupun karakter dapat dituliskan dalam satu satuan *DIMENSION*. *Default* **n1**, **n2**, dan **nn** adalah 1. Jika nilai awal **n1**, **n2**, dan **nn** adalah 1, jumlah indeks dalam penyataan *DIMENSION* cukup ditulis nilai maksimumnya saja.

Contoh penggunaan :

```
DIMENSION K(3,3)

DO I=1,2
DO J=1,2
READ(*,*)K(I,J)
ENDDO
ENDDO

DO I=1,2
WRITE(*,*)(K(I,J),J=1,2)
ENDDO
STOP
END
```

Contoh *Listing Program* Penggunaan *Array* dengan Cara *DIMENSION*

Bila program dijalankan maka akan keluar *output* seperti berikut :

```
2
3
4
5
            2           3
            4           5
Press any key to continue_
```

Contoh *Output Program* Menggunakan Array dengan Cara *DIMENSION*

b. Dengan Cara Eksplisit

Bentuk lain dari deklarasi *array* adalah dengan menyatakan secara eksplisit tipe data yang digunakan dalam *array* tersebut. Bentuk umumnya adalah

<type data> var(n1:m1,n2:m2,..,nl:ml),...

Keterangan :

<type data> = Tipe data yang digunakan dalam *array*, real, integer atau karakter.

Dengan penyataan eksplisit ini, variabel real, integer dan karakter harus dipisahkan dalam tiap kelompoknya.

Contoh penggunaan :

```
INTEGER K(3,3)

DO I=1,2
DO J=1,2
READ(*,*)K(I,J)
ENDDO
ENDDO

DO I=1,2
WRITE(*,*)(K(I,J),J=1,2)
ENDDO
STOP
END
```

Contoh *Listing Program* Penggunaan *Array* dengan Cara Eksplisit

Bila program dijalankan maka akan keluar *output* seperti berikut :

```
 2
 3
 4
 5
             2            3
             4            5
 Press any key to continue_
```

Contoh *Output Program* Penggunaan *Array* dengan Cara Eksplisit

Variabel ber-*array* biasa digunakan untuk menerjemahkan variabel berindeks dalam matematika. Perhatikan kembali contoh perkalian matriks dalam modul sebelumnya.

```
      DO 10 I=1,4,1
      DO 20 J=1,4,1
      A(I,J) = 0
      DO 30 K=1,4,1
      A(I,J)=A(I,J)+B(I,K)*C(K,J)
30    CONTINUE
20    CONTINUE
10    CONTINUE
      STOP
      END
```

Dengan notasi matematika perkalian matriks ditulis sebagai berikut :

$$a_{ij} = \sum_{k=1}^{4} \left(b_{ik} . c_{kj} \right)$$

Jika dijabarkan dalam bentuk matematis dapat ditulis sebagai berikut :

$$a_{11} = b_{11}.c_{11}+b_{12}.c_{21}$$
$$a_{12} = b_{11}.c_{12}+b_{12}.c_{22}$$
$$a_{21} = b_{21}.c_{11}+b_{22}.c_{21}$$
$$a_{22} = b_{21}.c_{12}+b_{22}.c_{22}$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \text{ x } \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

$$\begin{bmatrix} b_{11}.c_{11} + b_{12}.c_{22} & b_{11}.c_{12} + b_{12}.c_{22} \\ b_{21}.c_{11} + b_{22}.c_{21} & b_{21}.c_{12} + b_{22}.c_{22} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \text{ x } \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

MODUL VI
OPEN FILE

6.1 Pendahuluan

6.1.1 Tujuan

1. Memahami tentang metode pengolahan data dan *file*.
2. Dapat mengaplikasikannya ke dalam program komputer.

Seperti dijelaskan dalam Modul 1 semua data yang akan dan diproses di CPU dibaca dari main *memory* komputer. *Main memory* hanya menyimpan data tersebut secara *temporary* (sementara) hingga *program* selesai dieksekusi. Jika data tersebut ingin disimpan secara tetap, maka semua data tersebut harus disimpan dalam bentuk *file*. Dalam Modul 2 juga telah disinggung bagaimana cara membaca dan menulis data dari media monitor maupun *file* ke *main memory*. Pada Modul 6 ini akan dijelaskan tentang perintah yang umum digunakan dalam FORTRAN-77 untuk mengatur data dan *file* dalam pemrograman komputer.

6.2 *Input* dan *Output* pada Fortran

Perintah – perintah yang digunakan dalam pengaturan input dan output :

A. OPEN

Penyataan OPEN digunakan untuk membuka *file*, baik *file* yang telah tersedia maupun *file* yang harus disediakan. Setelah *file* ini terbuka, maka *main memory* akan membaca/menulis data ke dalam *file* ini. Bentuk umumnya adalah,

OPEN(<unit spec>,FILE='<nama file>',STATUS='<status>',

ACCESS='<access>',FORM='<form>',RECL=,rec-leng)

Tabel 6.1.a Bentuk Umum *Open File*

<unit spec>	Menunjukkan nomor spesifikasi *file*. Nomor spesifikasi merupakan nilai integer kecuali 5 dan 6.
<nama file>	Nama *file* yang akan dibuka.

Tabel 6.1.b Bentuk Umum *Open File*

<status>	Status *file* yang dibuka. **STATUS='OLD'** untuk *file* yang telah tersedia, *file* untuk data masukan misalnya. **STATUS='NEW'** digunakan untuk membuat *file* baru, dan **STATUS = 'UNKNOWN',** untuk *file* baru atau lama. Secara *default* jika status tidak ditulis berarti **STATUS = 'UNKNOWN'.**
<access>	Metode pengaksesan data dari *main memory* ke *file*. **ACCESS='DIRECT'**, pengaksesan dilakukan ke lokasi yang dikehendaki. **ACCESS='SEQUENTIAL'**, pengaksesan dilakukan secara berurut dari *record* awal. Karena *default* **ACCESS='SEQUENTIAL'**, pada umumnya metode **ACCESS** ini tidak dituliskan.
<form>	Menunjukkan bentuk format penyimpanan. ***form*='FORMATTED'**, data disimpan terformat, ***form*='UNFORMATTED'** data disimpan tanpa format.
rec-leng	Panjang *record* yang ditunjukkan oleh bilangan bulat (integer). Perintah ini hanya digunakan untuk **ACCESS='DIRECT'**

Contoh :

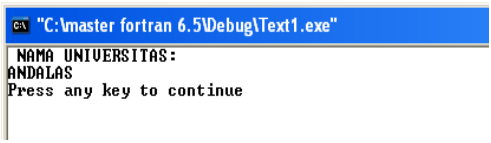
```
CHARACTER*19 C

WRITE(*,*)'NAMA UNIVERSITAS: '
READ(*,*)C
OPEN(1,FILE='OUTPUT.TXT')

WRITE(1,*)'UNIVERSITAS ',C
STOP
END
```

Angka 1 menunjukkan data dikeluarkan di ***output*.txt**

Contoh *Listing Program* Pengaturan Data dan *File*

Apabila program dieksekusi akan didapat hasil :

1. *Input* pada layar *output*.

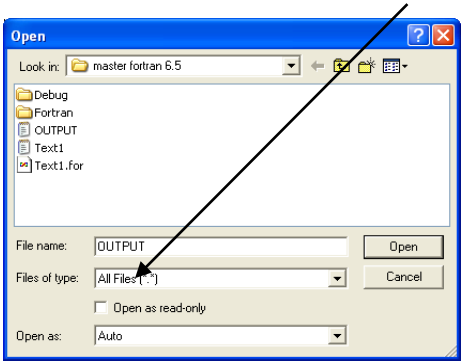


Gambar 6.1 Contoh *Output* Pada Layar *Output*

2. Buka *Output* pada *file notepad*

   Untuk melihat hasil dari perhitungan, lakukan langkah berikut :

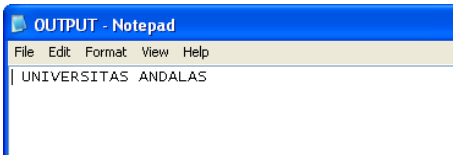
   1. Klik ***File > open*** **(Ctrl O)** atau klik 
   2. Ubah pilihan ***Files of type*** menjadi *All Files* (*.*)



Gambar 6.2 Kotak dialog *open*

   3. *Double* klik nama *file* yang ingin dibuka.

      Hasil perhitungan dalam *file* yaitu :



Gambar 6.3 Contoh *Output* Program dalam *File*

Contoh di atas merupakan contoh pengolahan data dengan *input* pada layar output dan *output* dalam *file.*

Secara umum ada 4 cara pengolahan data dengan *file* :

1. *Input* pada layar *output* dan *output* dalam *file*.
2. *Input* dalam *file* dan *output* pada layar *output.*
3. *Input* pada layar *output* dan *output* pada layar *output*.
4. *Input* dalam *file* dan *output* dalam *file.*

Contoh :

- data.doc (data akan tersimpan dalam format Word)
- simpan.xls (data akan tersimpan dalam Excel)
- output.abc (.abc adalah *file extension* buatan dan hanya bisa dibuka dalam *notepad* atau fortran)
- input.tugas (hanya bisa dibuka dalam *notepad* atau fortran)
- asadff (tanpa *file extension*, hanya bisa dibuka dalam *notepad* atau fortran)

B. CLOSE

Pernyataan CLOSE digunakan untuk menutup *file* yang tidak diperlukan. Penutupan *file* ini diperlukan untuk space dalam *main memory* komputer. Bentuk umumnya adalah.

CLOSE(<unit spec>, STATUS='<status>')

Tabel 6.2 Bentuk Umum *Close File*

<unit spec>	Menunjukkan nomor spesifikasi file. Nomor spesifikasi merupakan nilai integer kecuali 5 dan 6.
<status>	Status lanjutan dari file setelah ditutup. ***status*= 'KEEP'** digunakan agar file tetap seperti semula, ***status*= 'DELETE'** digunakan untuk menghapus file.

Contoh penggunaan :

CLOSE(14,STATUS='DELETE')

MODUL VII
SUBPROGRAM

7.1 Pendahuluan

7.1.1 Tujuan

1. Memahami perintah dan penggunaan *subprogram*.
2. Dapat mengaplikasikannya ke dalam program komputer, salah satunya dalam *Function.*

Subprogram digunakan untuk mengalihkan sebagian alur program ke dalam blok program tertentu. Efektivitas, efisiensi dan kemudahan pengaturan merupakan alasan utama dalam pengalihan alur program ke dalam *subprogram*. Dengan adanya *subprogram*, blok proses yang mungkin akan dilakukan berulang-ulang, hanya perlu disusun satu kali. Untuk selanjutnya hanya perlu dipanggil/diaktifkan.

Salah satu aplikasi dari *subprogram* adalah *function*. *Function* hanya dapat digunakan untuk menentukan nilai satu variabel, baik real, maupun integer. Pada umumnya digunakan untuk mendefinisikan fungsi tertentu dalam program.

7.2 Function

Bentuk umum dan cara pemanggilan/pengaktifan *Function* ada dua yaitu :

a) Dengan cara biasa
b) Dengan cara *subprogram*

Berikut akan dijelaskan penggunaan *Function* dengan cara biasa dan menggunakan *subprogram* :

a. Dengan cara biasa

Bentuk umum dari *Function* sebagai berikut :

<var1> (<var2>)=<pers>

Tabel 7.1 Keterangan bentuk umum *Function* dengan Cara biasa

<var1>	Variabel yang digunakan sebagai variabel fungsi dan tidak bisa digunakan dalam array
(var2)	Variabel yang digunakan di dalam persamaan.
(pers)	Persamaan dari fungsi. Contoh: x^2+5x+4

Contoh penggunaan *Function* dengan cara biasa :

```
F(X)=X**2+5*X+6
READ(*,*)X
WRITE(*,*)F(X)
STOP
END
```

Contoh *Listing Program* dengan Cara Biasa

Bila program dijalankan maka akan terlihat sebagai berikut :

```
4

    42.00000

Press any key to continue_
```

Contoh *Output Program* dengan Cara Biasa

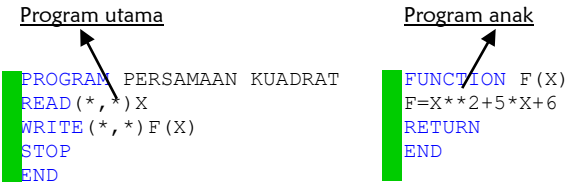
b. Dengan cara *subprogram*

Bentuk lain dari *function* adalah dengan menyatakan secara *subprogram.*

Tabel 7.2 Bentuk Umum *Function* dengan Cara *Subprogram*

Bentuk umum	Cara pemanggilan
FUNCTION <var1> (<var2>)	PROGRAM <nama program
<deklarasi variabel>	utama>
<inisialisasi variabel>	.
	.
<var1>=<pers>	WRITE(*,*)<var1>(<var2>)
<executable statement>	.
RETURN	.
END	**STOP**
	END

Contoh penggunaan Function dengan menggunakan sub program :



Contoh *Listing Program* dengan Cara *Subprogram*

Bila program dijalankan maka akan terlihat output sebagai berikut:

```
5
  56.00000
Press any key to continue_
```

Contoh *Output Program* dengan Cara *Subprogram*

Dengan menggunakan *function*, akar-akar dari suatu persamaan dapat diselesaikan. Contohnya dengan menggunakan bentuk logika IF dan perulangan seperti di bawah ini :

```
REAL B(100)
F(X)=X**2-16
J=0
DO I=-16,16
IF (F(I).EQ.0) THEN
J=J+1
B(J)=I
ENDIF
ENDDO

DO I=1,J
WRITE(*,*)B(I)
ENDDO
STOP
END
```

Contoh *Listing Program* Mencari Akar Persamaan dengan Menggunakan Logika IF

Bila program dijalankan maka akan terlihat output sebagai berikut:

```
-4.00000
 4.00000
Press any key to continue_
```

Contoh *Output* Mencari Akar Persamaan dengan Menggunakan Logika IF

7.3 *SUBROUTINE*

Subroutine merupakan *Subprogram* yang hampir mirip dengan program utama (perhatikan kembali penjelasan dalam modul 2). Dalam *Subroutine* dapat dilakukan serangkaian/sekelompok *input*, proses maupun *output*. Bentuk umum dan cara pemanggilan/pengaktifan *subroutine* diberikan sebagai berikut.

Tabel 7.3. Bentuk Umum *Subprogram*

Bentuk umum	Cara pemanggilan
SUBROUTINE <nm>(vr1,vr2..vrn) <deklarasi varibel> <inisialisasi variabel> <executable statement> RETURN END	PROGRAM <nama program utama> . . CALL <nm>(vra,vrb..vrz) . STOP END

Keterangan dari bentuk umum *Subprogram* sebagai berikut :

- <nm> : untuk nama *subprogram* (harus sama pada program utama dan *subprogram*
- Vr : variabel yang digunakan dalam program utama tidak perlu Sama dengan variabel yang digunakan pada *subprogram*, namun jumlah variabel yang digunakan harus sama

Contoh penggunaan 1 (Pemanggilan tanpa variabel) :

```
CALL GARIS
WRITE(*,*)'FORTRAN'
CALL GARIS                    } PROGRAM UTAMA
STOP
END

SUBROUTINE GARIS
WRITE(*,*)'----------'         } PROGRAM ANAK
RETURN
END
```

Contoh *Listing Program* Pemanggilan tanpa Variabel

Bila program di atas dijalankan akan menampilkan output seperti berikut :

```
 ----------
 FORTRAN
 ----------
Press any key to continue_
```

Contoh *Output Program* Pemanggilan tanpa Variabel

Garis yang terdapat di antara Fortran merupakan hasil dari *call* garis. Sedangkan *call* garis merupakan anak program dengan *subroutine* yang sama yaitu *subroutine* garis. Di dalam *subroutine* garis itulah garis tersebut dituliskan. Jadi pada program utama kita hanya memanggil anak program saja.

Contoh penggunaan 2 (Pemanggilan biasa / variabel tetap) :

```
PROGRAM NILAI                          } PROGRAM UTAMA
A=2                                    }
B=4                                    }
C=6                                    }
CALL OUTPUT (A,B,C)                    }
STOP                                   }
END                                    }

SUBROUTINE OUTPUT (A,B,C)              } PROGRAM ANAK
WRITE(*,*)'NILAI A=',A                 }
WRITE(*,*)'NILAI B=',B                 }
WRITE(*,*)'NILAI C=',C                 }
RETURN                                 }
END                                    }
```

Contoh *Listing Program* Pemanggilan biasa / variabel tetap

Bila program di atas dijalankan akan menampilkan output seperti berikut :

```
 NILAI A = 2.000000
 NILAI B = 4.000000
 NILAI C = 6.000000
Press any key to continue_
```

Output Program Pemanggilan biasa / variabel tetap

Contoh penggunaan 3 (Pemanggilan dengan variabel berbeda) :

```
PROGRAM NILAI                          } PROGRAM UTAMA
A=1
B=2
C=3
CALL NILAI (A,B,C)
WRITE(*,*)
CALL ANGKA (A,B,C)
STOP
END

SUBROUTINE NILAI (A,B,C)               } PROGRAM ANAK
WRITE(*,*)'NILAI A=',A
WRITE(*,*)'NILAI B=',B
WRITE(*,*)'NILAI C=',C
RETURN
END

SUBROUTINE ANGKA (X,Y,Z)               } PROGRAM ANAK
WRITE(*,*)'NILAI A=',X
WRITE(*,*)'NILAI B=',Y
WRITE(*,*)'NILAI C=',Z
RETURN
END
```

Contoh *Listing Program* Pemanggilan berbeda

Bila program di atas dijalankan akan menampilkan output seperti berikut :

```
 NILAI A=   1.000000
 NILAI B=   2.000000
 NILAI C=   3.000000

 NILAI A=   1.000000
 NILAI B=   2.000000
 NILAI C=   3.000000
Press any key to continue_
```

Contoh *Output Program* Pemanggilan berbeda

Contoh penggunaan 4 (Pemanggilan dengan variabel acak) :

```
PROGRAM NILAI                          } PROGRAM UTAMA
A=1
B=2
C=3
CALL NILAI (B,C,A)
STOP
END

SUBROUTINE NILAI (A,B,C)               } PROGRAM ANAK
WRITE(*,*)'NILAI A=',A
WRITE(*,*)'NILAI B=',B
WRITE(*,*)'NILAI C=',C
RETURN
END
```

Contoh *Listing Program* Pemanggilan acak

Bila program di atas dijalankan akan menampilkan output seperti berikut :

```
 NILAI A=   2.000000
 NILAI B=   3.000000
 NILAI C=   1.000000
Press any key to continue_
```

Contoh *Output Program* Pemanggilan acak

Perbedaan letak variabel pada pemanggilan di program utama dengan program anak dapat dilakukan, dengan catatan bahwa nilai variabel juga akan bertukar sesuai dengan letak variabel yang kita inginkan.