



PEMULA

**BELAJAR DASAR**

# **ALGORITMA & PEMROGRAMAN**



**Edy Budiman**

**ISBN: 978-602-14706-5-7**

## GLOSARIUM

### *Program*

Program adalah kata, ekspresi, pernyataan atau kombinasi yang disusun dan dirangkai menjadi satu kesatuan prosedur yang menjadi urutan langkah untuk menyesuaikan masalah yang diimplementasikan dengan bahasa pemrograman.

### *Bahasa Pemrograman*

Bahasa pemrograman merupakan prosedur atau tata cara penulisan program dalam bahasa pemrograman, terdapat dua faktor penting yaitu sintaksis dan semantik. Sintak adalah aturan-aturan gramatikal yang mengatur tata cara penulisan kata, ekspresi dan pernyataan sedangkan semantik adalah aturan-aturan untuk menyatakan suatu arti.

### *Pemrograman*

Pemrograman merupakan proses mengimplementasikan urutan langkah-langkah untuk menyelesaikan suatu masalah dengan bahasa pemrograman.

### *Pemrograman Terstruktur*

Pemrograman Terstruktur merupakan proses mengimplementasikan urutan langkah langkah untuk menyelesaikan suatu masalah dalam bentuk program yang memiliki rancang bangun yang terstruktur dan tidak berbelit-belit sehingga mudah ditelusuri, dipahami dan dikembangkan oleh siapa saja.

### *Aplikasi ( software application)*

adalah suatu subkelas perangkat lunak komputer yang memanfaatkan kemampuan komputer langsung untuk melakukan suatu tugas yang diinginkan pengguna.

### *Argumen*

sebuah nilai yang dilewatkan ke metode ketika metode dipanggil.

### *Boolean*

tipe data yang memiliki dua buah nilai, yaitu true atau false (benar atau salah). Untuk besaran nilai tidak bisa di tetapkan.

### *Char*

Merupakan tipe data karakter. Semua data yang hanya terdiri dari 1 karakter tergolong dalam tipe ini. Misalnya data jenis kelamin yang hanya diisikan huruf L atau P. Penulisan data tipe char harus diapit oleh tanda petik tunggal. Karakter-karakter yang diperbolehkan terdefinisi dalam tabel ASCII

### *Direktori*

adalah komponen dari sistem berkas yang mengandung satu berkas atau lebih atau satu direktori lainnya atau lebih, yang disebut dengan **subdirektori**. Batasan jumlah berkas atau subdirektori yang dapat ditampung dalam sebuah direktori tergantung dari sistem berkas yang digunakan, meskipun sebagian sistem berkas tidak membatasinya (batasan tersebut disebabkan ukuran media penyimpanan di mana direktori berada).

### *Float*

digunakan untuk menandakan nilai-nilai yang mengandung presisi atau ketelitian tunggal (*single-precision*) yang menggunakan ruang penyimpanan 32-bit. Presisi tunggal biasanya lebih cepat untuk processor-processor tertentu dan memakan ruang penyimpanan setengah kali lebih sedikit dibandingkan presisi ganda (*double precision*). Permasalahan yang timbul dari pemakaian tipe *float* untuk nilai-nilai yang terlalu kecil atau justru terlalu besar, karena nilai yang dihasilkan akan menjadi tidak akurat. Contoh penggunaan variabel : float suhu.

### *Fungsi*

adalah bagian dari program yang dibuat terpisah untuk melaksanakan fungsi tertentu yang menghasilkan suatu nilai untuk dikembalikan ke program utama.

### *Integer*

adalah tipe data untuk angka numerik yang tidak menggunakan koma, untuk tipe data 32 bit. Merupakan tipe data bilangan bulat, baik yang negatif, nol, maupun bilangan positif

### *Konstanta*

Konstanta adalah variabel yang nilai datanya bersifat tetap dan tidak bisa diubah. Jadi konstanta adalah juga variabel bedanya adalah pada nilai yang disimpannya. Jika nilai datanya sepanjang program berjalan tidak berubahubah, maka sebuah variabel lebih baik diperlakukan sebagai konstanta.

### *Metadata*

adalah informasi terstruktur yang mendeskripsikan, menjelaskan, menemukan, atau setidaknya membuat menjadikan suatu informasi mudah untuk ditemukan kembali, digunakan, atau dikelola.

### *Method*

merupakan suatu operasi berupa fungsi-fungsi yang dapat dikerjakan oleh suatu object. Method didefinisikan pada class akan tetapi dipanggil melalui object. Contoh : pada object mangga : terdapat method ambilRasa , kupasKulit dan lain-lain.

### *Prosedur*

adalah kumpulan ekspresi-ekspresi algoritma yang berguna untuk menjalankan proses tertentu. Prosedur sudah banyak dikenal mulai dari bahasa mesin hingga bahasa level tinggi (Query). Dalam bahasa Java prosedur biasanya diawali dengan kata “void”. Dan kebanyakan aplikasi berjalan melalui prosedur.

### *Real*

Merupakan tipe data bilangan pecahan. Semua bilangan yang mengandung tanda desimal tergolong dalam tipe ini. Tanda desimal yang dipakai adalah tanda titik, bukan tanda koma

### *String*

adalah tipe data untuk teks yang merupakan gabungan huruf, angka, whitespace (spasi), dan berbagai karakter. Fungsi ini digunakan untuk membuat identifier String/teks. String juga sering disebut sebagai “array of char”.

### *Subrutin*

adalah bagian dari program yang dibuat terpisah untuk melaksanakan sebagian dari tugas yang harus diselesaikan oleh suatu program.

### *Windows*

adalah sistem operasi yang dikembangkan oleh Microsoft Corporation yang menggunakan antarmuka dengan berbasis GUI (*Graphical User Interface*) atau tampilan antarmuka bergrafis.

## DAFTAR ISI

<b>Halaman Sampul</b>	i
<b>Glosarium</b>	ii
<b>Daftar Isi</b>	v
<b>Kata Pengantar</b>	vii
 <b>BAB 1 PENDAHULUAN</b>	
1.1 Profil Lulusan dan Program Studi	1
1.2 Kompetensi Lulusan	2
1.3 Capaian Pembelajaran (Learning Outcome)	2
1.4 Standard Kompetensi Matakuliah	
a. Standard Kompetensi	3
b. Kompetensi Dasar	3
c. Bagan Analisis Kompetensi	3
1.5 Garis Besar Rencana Pembelajaran	5
 <b>BAB 2 KONSEP DASAR ALGORITMA DAN PEMROGRAMAN</b>	
2.1 Konsep Dasar Algoritma	8
2.2 Penyajian Algoritma	12
a. Deskriptif	12
b. Pseudocode	13
c. Flowchart	16
2.3 Struktur Pemrograman	20
2.4 Input dan Output	24
2.5 Pengertian Pemrograman	31
 <b>BAB 3 TIPE DATA, VARIABEL DAN NILAI</b>	
3.1 Tipe Data	39
3.2 Variabel	44
3.3 Konstanta	45
3.4 Pemberian Nilai	46
3.5 Menampilkan Nilai	48
3.6 Ekspresi ( <i>expression</i> )	48
Contoh dan latihan Soal	52
 <b>BAB 4 RUNTUNAN, PEMILIHAN DAN PENGULANGAN</b>	
4.1 Instruksi Runtunan ( <i>Sequential</i> )	56
4.2 Instruksi Pemilihan ( <i>Selection</i> )	60
4.3 Instruksi Pengulangan ( <i>Repetition</i> )	68
Latihan	75

<b>BAB 5</b>	<b>STUDY KASUS</b>	
5.1	Akses Data Langsung	76
5.2	Menjumlahkan Deret	77
5.3	Mengelompokkan Data	79
5.4	Memilih Operasi Berdasarkan Data Input	80
5.5	Pemilihan 2 Kasus	82
	Pemilihan lebih dari 2 Kasus	89
<b>BAB 6</b>	<b>ARRAY, PROSEDUR DAN FUNGSI</b>	
6.1	Array	106
6.2	Prosedur	118
6.3	Fungsi	124
	Latihan	131
<b>DAFTAR PUSTAKA</b>		

## KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa, sehingga buku ajar mata kuliah Algoritma dan Pemrograman ini dapat disusun dengan baik. Buku ini disusun sedemikian rupa agar dapat digunakan dengan mudah oleh mahasiswa informatika sebagai panduan dalam memahami Mata Kuliah Algoritma dan Pemrograman, meliputi konsep dasar algoritma dan pemrograman, *flowchart* dan *Pseudo code*, tipe data, variable, konstanta, pemberian nilai, runtunan(*sequential*), pemilihan (*selection*), pengulangan(*repetition*), array, prosedur(*procedure*) dan fungsi(*function*).

Terima kasih yang sebesar-besarnya kami ucapkan pada berbagai pihak yang telah membantu dan mendukung pembuatan buku bahan ajar ini. Harapan kami semoga buku ini dapat memberikan manfaat bagi para pembacanya.

Samarinda, Agustus 2015

Penulis

## BAB II

# KONSEP DASAR ALGORITMA DAN PEMROGRAMAN

### 2.1. Konsep Dasar Algoritma

#### a. Pengertian Algoritma

Pandangan mengenai komputer sebagai sebuah mesin yang “pintar” adalah pendapat yang salah, karena komputer hanyalah suatu alat yang diberi serangkaian perintah oleh manusia sehingga dapat menyelesaikan permasalahan secara cepat, akurat, bahkan berulang-ulang tanpa kenal lelah dan bosan. Sekumpulan instruksi yang merupakan penyelesaian masalah itu dinamakan program. Agar program dapat dilaksanakan oleh komputer, program tersebut harus ditulis dalam suatu bahasa yang dimengerti oleh komputer. Bahasa komputer yang digunakan dalam menulis program dinamakan bahasa pemrograman. Urutan langkah-langkah yang sistematis untuk menyelesaikan sebuah masalah dinamakan algoritma.

Algoritma berarti solusi. Ketika orang berbicara mengenai algoritma di bidang pemrograman, maka yang dimaksud adalah solusi dari suatu masalah yang harus dipecahkan dengan menggunakan komputer. Algoritma harus dibuat secara runtut agar komputer mengerti dan mampu mengeksekusinya. Analisis kasus sangat dibutuhkan dalam membuat sebuah algoritma, misalnya proses apa saja yang sekiranya dibutuhkan untuk menyelesaikan masalah yang harus diselesaikan .

Algoritma berasal dari kata *algoris* dan *ritmis* yang pertama kali diungkapkan oleh Abu Ja'far Mohammad Ibn Musa Al Khwarizmi (825M) dalam buku Al-Jabr Wa-al Muqobla. Dalam pemrograman algoritma berarti suatu metode khusus yang tepat dan terdiri dari serangkaian langkah-langkah yang terstruktur dan dituliskan secara



sistematis yang akan dikerjakan untuk menyelesaikan masalah dengan bantuan komputer.

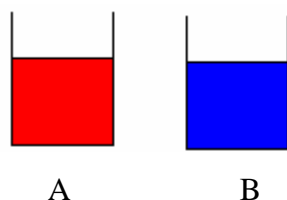
Algoritma adalah urutan logis pengambilan keputusan untuk pemecahan masalah. Kata logis merupakan kata kunci. Langkah-langkah tersebut harus logis, ini berarti nilai kebenarannya harus dapat ditentukan, benar atau salah. Langkah-langkah yang tidak benar dapat memberikan hasil yang salah. Sebagai contoh tinjau persoalan mempertukarkan isi dua buah bejana, A dan B. Bejana A berisi larutan yang berwarna merah, sedangkan bejana B berisi air berwarna biru. Kita ingin mempertukarkan isi kedua bejana itu sedemikian sehingga bejana A berisi larutan berwarna biru dan bejana B berisi larutan berwarna merah.

Contoh :

Misalkan terdapat dua buah gelas, gelas A dan gelas B. Gelas A berisi air berwarna merah dan gelas B berisi air berwarna biru, kita ingin menukarkan isi air kedua gelas tersebut, sehingga gelas A berisi air berwarna biru dan gelas B berisi air berwarna merah.

Algoritma Tukar\_Isi\_Gelas

1. Tuangkan air dari gelas A ke gelas B
2. Tuangkan air dari gelas B ke gelas A



Algoritma diatas tidak menghasilkan pertukaran yang benar, langkah-langkahnya tidak logis, karena yang terjadi bukan pertukaran tetapi percampuran antara air di gelas A dengan air di gelas B. Sehingga algoritma Tukar\_Isi\_Gelas diatas salah.

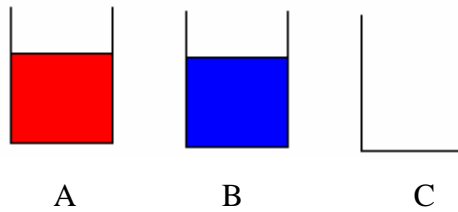
Dari permasalahan diatas algoritma yang benar adalah bahwa untuk menukarkan isi air pada gelas A dengan isi air pada gelas B maka dibutuhkan sebuah gelas bantuan yang dipakai untuk menampung salah satu air dalam gelas tersebut misalkan gelas C.

Sehingga algoritma yang benar dari permasalahan diatas adalah :

### Algoritma Tukar\_Isi\_Gelas

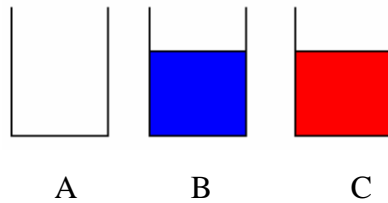
1. Tuangkan air dari gelas A ke gelas C
2. Tuangkan air dari gelas B ke gelas A
3. Tuangkan air dari gelas C ke gelas B

Keadaan awal sebelum pertukaran

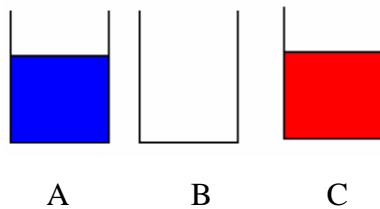


Proses pertukaran :

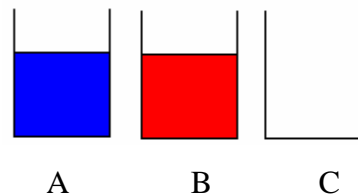
1. Tuangkan air dari gelas A ke gelas C



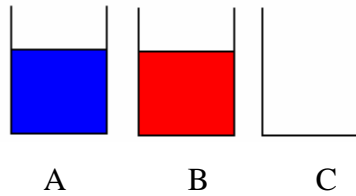
2. Tuangkan air dari gelas B ke gelas A



3. Tuangkan air dari gelas C ke gelas B



Keadaan setelah pertukaran



Sekarang algoritma Tukar\_Isi\_Gelas diatas sudah diperbaiki, sehingga isi air pada gelas A dan isi air pada gelas B dapat dipertukarkan dengan benar.

dalam kehidupan sehari-hari Algoritma bisa ditemukan, misalnya sbb:

Proses	Algoritma	Contoh
Membuat Kue	Resep Kue	Campurkan 2 butir telur kedalam adonan, kemudian kocok hingga mengembang
Membuat Pakaian	Pola pakaian	Gunting kain dari pinggir kiri bawah ke arah kanan atas sepanjang 15 cm
Praktikum Kimia	Petunjuk Praktikum	Campurkan 10 ml Asam Sulfat ke dalam 15 ml Natrium hidroksida

**Tahap pemecahan masalah** adalah Proses dari masalah hingga terbentuk suatu algoritma. **Tahap implementasi** adalah proses penerapan algoritma hingga menghasilkan solusi. **Solusi** yang dimaksud adalah suatu program yang merupakan implementasi dari algoritma yang disusun.

Ciri algoritma yang baik adalah :

- Algoritma memiliki logika perhitungan atau metode yang tepat dalam menyelesaikan masalah.
- Menghasilkan output yang tepat dan benar dalam waktu yang singkat.
- Algoritma ditulis dengan bahasa yang standar secara sistematis dan rapi sehingga tidak menimbulkan arti ganda (*ambiguous*).
- Algoritma ditulis dengan format yang mudah dipahami dan mudah diimplementasikan ke dalam bahasa pemrograman.
- Semua operasi yang dibutuhkan terdefinisi dengan jelas.
- Semua proses dalam algoritma harus berakhir setelah sejumlah langkah dilakukan.

## 2.2. Penyajian Algoritma

Algoritma adalah independen terhadap bahasa pemrograman tertentu, artinya algoritma yang telah dibuat tidak boleh hanya dapat diterapkan pada bahasa pemrograman tertentu. Penulisan algoritma tidak terikat pada suatu aturan tertentu, tetapi harus jelas maksudnya untuk tiap langkah algoritmanya. Namun pada dasarnya algoritma dibagi menjadi beberapa macam berdasarkan format penulisannya, yaitu:

### 1) Deskriptif

Algoritma bertipe deskriptif maksudnya adalah algoritma yang ditulis dalam bahasa manusia sehari-hari (misalnya bahasa Indonesia atau bahasa Inggris) dan dalam bentuk kalimat. Setiap langkah algoritmanya diterangkan dalam satu atau beberapa kalimat.

Sebagai contoh misalnya algoritma menentukan bilangan terbesar dari 3 bilangan berikut ini:

Algoritma Menentukan\_bilangan\_terbesar\_dari\_3\_bilangan :

1. Meminta input 3 bilangan dari user, misalkan bilangan a, b, dan c.
2. Apabila bilangan a lebih besar dari b maupun c, maka bilangan a merupakan bilangan terbesar
3. Jika tidak (bilangan a tidak lebih besar dari b atau c) berarti bilangan a sudah pasti bukan bilangan terbesar. Kemungkinannya tinggal bilangan b atau c. Apabila bilangan b lebih besar dari c, maka b merupakan bilangan terbesar. Sebaliknya apabila bilangan b tidak lebih besar dari c, maka bilangan c merupakan yang terbesar.
4. Selesai.

## 2) Pseudocode

**Pseudo** berarti imitasi dan **code** berarti kode yang dihubungkan dengan instruksi yang ditulis dalam bahasa komputer (kode bahasa pemrograman). Apabila diterjemahkan secara bebas, maka pseudocode berarti tiruan atau imitasi dari kode bahasa pemrograman. Pada dasarnya, pseudocode merupakan suatu bahasa yang memungkinkan programmer untuk berpikir terhadap permasalahan yang harus dipecahkan tanpa harus memikirkan *syntax* dari bahasa pemrograman yang tertentu. Tidak ada aturan penulisan *syntax* di dalam pseudocode. Jadi pseudocode digunakan untuk menggambarkan logika urutan dari program tanpa memandang bagaimana bahasa pemrogramannya. Walaupun pseudocode tidak ada aturan penulisan *syntax*, di dalam buku ini akan diberikan suatu aturan-aturan penulisan *syntax* yang cukup sederhana agar pembaca dapat lebih mudah dalam mempelajari algoritma-algoritma yang ada di dalam buku ini.

Pseudocode yang ditulis di dalam buku ini akan menyerupai (meniru) *syntax-syntax* dalam bahasa Pascal atau C . Namun dibuat sesederhana mungkin sehingga tidak akan ada kesulitan bagi pembaca untuk memahami algoritma-algoritma dalam buku ini walaupun pembaca belum pernah mempelajari bahasa Pascal atau C. Contoh algoritma menentukan bilangan terbesar dari tiga bilangan yang ditulis dalam bentuk pseudocode :

---

### Algoritma Menentukan\_terbesar\_dari\_3\_bilangan

---

**Deklarasi:**

a,b,c, terbesar : integer

**Deskripsi:**

Read(a,b,c)

If (a>b) and (a>c) then

Terbesar ← a

Else

If b>c then

Terbesar ← b

Else

---

---

```
Terbesar ← c
Endif
Endif
Write (terbesar)
```

---

### Structure English dan Pseudocode

*Structure English* merupakan alat yang cukup efisien untuk menggambarkan suatu algoritma. Basis dari *structure english* adalah bahasa inggris, tetapi juga bisa digunakan bahasa indonesia, sedangkan *pseudocode* berarti kode yang mirip dengan kode pemrograman sebenarnya. *Pseudocode* berasal dari kata *pseudo* yang berarti imitasi/mirip/menyerupai dan *code* yang berarti program. *Pseudocode* berbasis pada kode program yang sesungguhnya seperti Pascal, C, C++. *Pseudocode* lebih rinci dari *structure english* misalnya dalam menyatakan tipe data yang digunakan.

#### Contoh struktur Indonesia

- Baca data jam\_kerja
- Hitung gaji adalah jam\_kerja dikalikan tarif
- Tampilkan gaji

#### Pseudocode dengan Pascal :

- Read jam\_kerja
- Gaji := jam\_kerja \* tarif
- Write gaji

### Aturan Penulisan Teks Algoritma

Langkah-langkah penyelesaian masalah dalam teks algoritma dapat ditulis dalam notasi apapun, dengan syarat bahwa langkah-langkah tersebut mudah dipahami dan dimengerti. Tidak ada notasi yang baku dalam teks algoritma sebagaimana notasi dalam bahasa pemrograman (notasi dalam algoritma disebut dengan notasi algoritmik).

Setiap orang dapat membuat aturan penulisan dan notasi algoritmik sendiri. Berkaitan hal itu untuk memudahkan translasi notasi algoritmik ke dalam bahasa pemrograman, sebaiknya notasi algoritmik tersebut berkorespondensi dengan notasi bahasa pemrograman secara umum. Sebagai contoh :

Tulis nilai X dan Y

Dalam notasi algoritmik menjadi :

```
Write(X,Y)
```

Notasi `write` ini berarti nilai X dan Y dicetak ke piranti keluaran. Notasi `write` ini berkorespondensi dengan `write` atau `writeln` dalam bahasa pascal, `printf` dalam bahasa C, `cout` dalam bahasa C++. Jadi, translasi `write(X,Y)` dalam masing-masing bahasa tersebut adalah :

```
writeln(X,Y); { dalam bahasa pascal }  
printf("%d %d", X,Y); /* dalam bahasa C */  
cout<<X<<Y; /* dalam bahasa C++ */
```

Perhatikan bahwa setiap bahasa pemrograman mempunyai aturan sendiri dalam menggunakan perintah penulisan.

Contoh lain :

Isikan nilai X ke dalam max

Ditulis dalam notasi algoritmik menjadi :

```
max ← X
```

Notasi “←” berarti mengisi (*assign*) peubah (*variable*) `max` dengan nilai peubah X.

Translasi notasi “←” kedalam bahasa Pascal adalah “:=”, dalam bahasa C adalah “=”, dalam bahasa C++ adalah “=”. Translasi `max ← X` dalam masing-masing bahasa adalah :

```
max := X; { dalam bahasa Pascal }  
max = X; /* dalam bahasa C */  
max = X; /* dalam bahasa C++ */
```

### 3) Flowchart



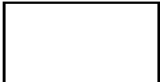
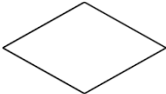

Dalam *structure English* / struktur Indonesia digambarkan tahap-tahap penyelesaian masalah dengan menggunakan kata-kata (teks). Kelemahan cara ini adalah dalam penyusunan algoritma sangat dipengaruhi oleh tata bahasa pembuatnya, sehingga kadang-kadang orang lain sulit memahaminya. Oleh sebab itu kemudian dikembangkan metode yang menggambarkan tahap-tahap pemecahan masalah dengan merepresentasikan symbol-simbol tertentu yang mudah dimengerti, mudah digunakan dan standar.

Salah satu penulisan simbol tersebut adalah dengan menggunakan *flowchart*.




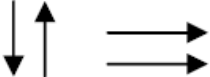
*Flowchart* terdiri dari dua macam yaitu :

#### 1) Flowchart Program

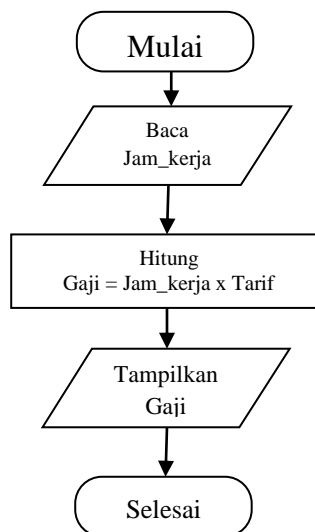
Bagan alir program adalah suatu bagan yang menggambarkan arus logika dari data yang akan diproses dalam suatu program dari awal sampai akhir. Bagan alir program merupakan alat yang berguna bagi programmer untuk mempersiapkan program yang rumit. Bagan alir terdiri dari simbol-simbol yang mewakili fungsi-fungsi langkah program dan garis alir (*flow lines*) menunjukkan urutan dari simbol yang akan dikerjakan.

	Simbol Terminal, simbol yang digunakan untuk menyatakan awal atau akhir suatu program.
	Simbol Input/Output, simbol yang digunakan untuk menunjukkan operasi masukan atau keluaran
	Simbol Proses, simbol yang digunakan untuk menggambarkan proses pengolahan data
	Simbol Keputusan, simbol yang digunakan untuk menyatakan suatu pilihan berdasarkan suatu kondisi tertentu
	Simbol persiapan (Preparation), simbol yang digunakan untuk memberikan nilai awal pada suatu variabel atau pencacah



	Simbol proses terdefinisi (predefined process symbol), simbol yang digunakan untuk proses yang detilnya dijelaskan terpisah, misal dalam bentuk subroutine
	Simbol Penghubung ke halaman lain, simbol yang digunakan untuk menghubungkan bagian diagram alir pada halaman yang berbeda
	Simbol Penghubung ke halaman yang sama, symbol yang digunakan untuk menghubungkan bagian diagram alir pada halaman yang sama
	Simbol Arah aliran, simbol yang digunakan untuk menunjukkan arah aliran proses

Contoh penggunaan *flowchart* program :



Gambar 2.1. *flowchart* program Hitung Gaji




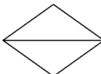







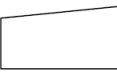
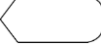
Pedoman membuat flowchart :



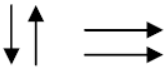
- Flowchart dibuat dari atas ke bawah dimulai dari bagian kiri suatu halaman.
- Kegiatan dalam flowchart harus ditunjukkan dengan jelas.
- Kegiatan dalam flowchart harus jelas dimana akan dimulai dan dimana akan berakhir.
- Kegiatan yang ada dalam flowchart digunakan kata yang mewakili pekerjaan.
- Kegiatan dalam flowchart harus sesuai dengan urutannya.

- f. Kegiatan yang terpotong dihubungkan dengan simbol penghubung.
- g. Simbol-simbol yang digunakan flowchart adalah simbol-simbol standar.

## 2) Flowchart system

Bagan alir sistem berbeda dengan bagan alir program. Bagan alir program sifatnya lebih terperinci tentang langkah-langkah proses di dalam program dari awal sampai akhir. Bagan alir sistem hanya menggambarkan arus data dari sistem. Simbol symbol yang digunakan pada bagan alir sistem ada yang sama dan ada yang berbeda dengan simbol-simbol yang digunakan pada bagan alir program.

	Simbol Dokumen	Simbol yang menunjukkan dokumen yang digunakan untuk input dan output baik secara manual, mekanik maupun komputerisasi.
	Simbol operasi Manual	Simbol yang menunjukkan pekerjaan yang dilakukan secara manual.
	Simbol Proses	Simbol yang menunjukkan kegiatan proses operasi program komputer.
	Simbol pengurutan	Simbol yang menunjukkan proses pengurutan dokumen di luar komputer.
	Simbol Offline Storage	Simbol yang menunjukkan file non komputer yang diarsip urut angka (numeric).
	Simbol Offline Storage	Simbol yang menunjukkan file non komputer yang diarsip urut huruf (Alphabetic).
	Simbol Offline Storage	Simbol yang menunjukkan file non komputer yang diarsip urut tanggal (Chronological).
	Simbol Magnetic tape	Simbol yang menunjukkan Input Output yang menggunakan pita magnetic.
	Simbol Magnetic Drum	Simbol yang menunjukkan Input Output yang menggunakan Drum magnetic.
	Simbol Magnetic Storage	Simbol yang menunjukkan Input Output yang menggunakan Diskette.
	Simbol Hard Disk Storage	Simbol yang menunjukkan Input Output yang menggunakan Hard Disk.
	Simbol Keyboard	Simbol yang menunjukkan Input Output yang menggunakan on line keyboard
	Simbol Display	Simbol yang menunjukkan Output yang ditampilkan dilayar terminal

	Simbol Penghubung ke halaman lain	Simbol yang digunakan untuk menghubungkan bagian diagram alir pada halaman yang berbeda
	Simbol Penghubung ke halaman yang sama,	Simbol yang digunakan untuk menghubungkan bagian diagram alir pada halaman yang sama
	Simbol Arah aliran	Simbol yang digunakan untuk menunjukkan arah aliran proses

Dalam sebuah algoritma langkah-langkah penyelesaian masalahnya dapat berupa struktur urut (*sequence*), struktur pemilihan (*selection*), dan struktur pengulangan (*repetition*).

Ketiga jenis langkah tersebut membentuk konstruksi suatu algoritma.

### 1) Struktur Urut (*sequence*)

Struktur urut adalah suatu struktur program dimana setiap baris program akan dikerjakan secara urut dari atas ke bawah sesuai dengan urutan penulisannya.

### 2) Struktur Pemilihan (*selection*) atau Penyeleksian Kondisi

Pada struktur pemilihan tidak setiap baris program akan dikerjakan. Baris program yang dikerjakan hanya yang memenuhi syarat saja. Struktur pemilihan adalah struktur program yang melakukan proses pengujian untuk mengambil suatu keputusan apakah suatu baris atau blok instruksi akan diproses atau tidak. Pengujian kondisi ini dilakukan untuk memilih salah satu dari beberapa alternatif yang tersedia.

### 3) Pengulangan

Salah satu kelebihan komputer adalah kemampuannya untuk melakukan melakukan pekerjaan yang sama berulang kali tanpa mengenal lelah. Struktur pengulangan disebut kalang (*loop*), dan bagian algoritma yang diulang (*aksi*) dinamakan badan kalang (*loop body*).

### 2.3. Struktur Algoritma

Agar algoritma dapat ditulis lebih teratur maka sebaiknya dibagi ke dalam beberapa bagian. Salah struktur yang sering dijadikan patokan adalah sebagai berikut:

#### ▪ Bagian Kepala (Header)

Kepala algoritma adalah bagian yang terdiri dari nama algoritma dan penjelasan algoritma. Aturan pemberian nama algoritma mengacu pada aturan pemberian nama pengenalan yang akan dibahas pada bagian berikutnya. Nama algoritma hendaknya singkat, namun mencerminkan isi algoritma secara keseluruhan. Bagian penjelasan algoritma berisi penjelasan mengenai hal-hal yang dilakukan oleh algoritma secara singkat. Contoh kepala algoritma dapat dilihat di bawah ini.

##### Algoritma LUAS\_SEGITIGA

```
{Menghitung luas segitiga berdasarkan panjang alas dan tinggi  
segitiga yang diinput user. Luas segitiga dapat diperoleh dari  
rumus Luas = 0,5 x alas x tinggi}
```

##### Algoritma Menentukan\_Bilangan\_Prima

```
{Menentukan apakah suatu bilangan bulat yang diinputkan oleh user  
merupakan bilangan prima atau komposit. Algoritma akan mencetak  
kata "prima" apabila bilangan tersebut adalah bilangan prima, dan  
sebaliknya akan mencetak kata "komposit" bila bilangan tersebut  
bukan bilangan prima}
```

#### ▪ Bagian Deklarasi

Bagian deklarasi berisikan semua nama pengenalan yang dipakai di dalam algoritma. Nama tersebut dapat berupa nama tetapan (konstanta), nama peubah (variabel), nama tipe, nama prosedur, dan nama fungsi. Contohnya dapat dilihat di bawah ini.

**DEKLARASI**

```

02| {nama tetapan}
03| const Npeg = 100 {jumlah pegawai}
04| const phi = 3.14 {nilai phi}
05| {nama tipe}
06| type TTitik : record {tipe koordinat bidang kartesius}
07| < x,
08| y : integer
09| >
10| {nama peubah}
11| c : char {karakter yang dibaca}
12| Q : TTitik {titik dalam koordinat kartesius}
13| ketemu : boolean {keadaan hasil pencarian}
14|
15| function IsPrima(input x:integer) □ boolean
16| {mengembalikan nilai true bila x adalah prima, atau false
   bila x adalah komposit}
17|
18| procedure Tukar(input/output a,b : integer)
19| {mempertukarkan isi variabel a dan b}

```

- **Bagian Deskripsi**

Memuat langkah-langkah penyelesaian masalah, termasuk beberapa perintah seperti baca data, tampilkan, ulangi, dan sebagainya.

Berikut ini adalah contoh sebuah algoritma yang mengikuti struktur tersebut diatas:

**Algoritma Luas\_lingkaran**

{ menghitung luas sebuah lingkaran apabila jari-jari lingkaran tersebut diberikan }

**Deklarasi**

{ Defenisi nama tetapan }

**const** N = 10;

**const** phi = 3.14;

{ defenisi nama peubah / variable }

**real** jari\_jari, luas;

**Deskripsi**

**read**(jari\_jari);

luas = phi \* jari\_jari \* jari\_jari;

**write**(luas);

Contoh berikut ini adalah algoritma untuk menghitung nilai rata sejumlah angka yang dimasukkan lewat keyboard.

**Algoritma Nilai\_Rata**

{ menghitung nilai rata sejumlah bilangan yang dimasukkan lewat keyboard }

**Deklarasi**

**integer** x, N, k, jumlah;

**real** nilai\_rata;

**Deskripsi**

{ masukkan jumlah data }

**read**(N);

k  $\leftarrow$  1;

jumlah  $\leftarrow$  0;

**while** (k  $\leq$  N) **do**

{ baca data }

**read**(x);

jumlah  $\leftarrow$  jumlah + x;

k  $\leftarrow$  k + 1;

**endwhile**

{ hitung nilai rata }

nilai\_rata  $\leftarrow$  jumlah / N;

**write**(nilai\_rata);

**b. Komentar (*Comment*) Algoritma**

Penulisan algoritma yang baik selalu disertai pemberian komentar. Komentar merupakan suatu penjelasan mengenai suatu hal yang tertulis dalam algoritma dan komentar bukanlah merupakan bagian dari langkah-langkah penyelesaian suatu algoritma.

Komentar ditulis di antara tanda baca **kurung kurawal buka** dan **kurung kurawal tutup**. “ { } “. Pada bagian kepala algoritma, penjelasan algoritma merupakan suatu komentar. Contoh lainnya dapat Anda lihat dalam contoh bagian deklarasi di atas. Pemberian komentar bukanlah suatu keharusan, namun algoritma

yang disertai komentar-komentar yang tepat akan memudahkan pembaca lainnya untuk mengerti algoritma yang dibuat.

### **c. Penamaan/Pengenal (*Identifier*)**

---

Dalam algoritma, ada beberapa hal yang harus diberi nama atau pengenal. Hal-hal tersebut meliputi: nama algoritma, nama tetapan (konstanta), nama peubah (variabel), nama tipe, nama prosedur, dan nama fungsi. Pemberian nama harus mengikuti aturan-aturan sebagai berikut:

- 1) hanya boleh terdiri dari huruf, angka, atau garis bawah
- 2) tidak boleh dimulai dengan angka
- 3) tidak membedakan huruf kapital maupun huruf kecil (*non case-sensitive*)
- 4) panjang tidak dibatasi
- 5) harus unik, artinya tidak boleh ada nama pengenal yang sama untuk hal yang berbeda
- 6) hendaknya mencerminkan kegunaannya

Pemberian nama pengenal ini biasanya mengikuti salah satu dari 2 gaya berikut:

1. pemisah antar-kata menggunakan tanda garis bawah. Contoh: Luas\_Segitiga
2. pemisah antar-kata menggunakan huruf kapital Contoh: LuasSegitiga

Beberapa contoh penamaan yang valid maupun tidak valid:

- Nama\_4\_Pegawai {valid}
- 2\_luas {salah, karena diawali angka}
- ^LuasSegitiga {salah, karena terdapat tanda ^}
- Luas\_Segitiga\_&\_Lingk {salah, karena terdapat tanda &}
- Hati2 {valid}

## 2.4. Input dan Output

Menurut Donald E. Knuth (1973), algoritma memiliki 5 ciri pokok, yaitu:

- Finiteness Algoritma harus selalu berakhir setelah melakukan sejumlah langkah berhingga.
- Definiteness Setiap langkah dari suatu algoritma harus terdefinisi secara tepat (logis).
- Input Suatu algoritma dapat tidak memiliki input, ataupun memiliki satu atau lebih dari satu input.
- Output Suatu algoritma harus memiliki paling sedikit satu output.
- Effectiveness Algoritma yang dibuat diharapkan efektif, artinya setiap langkah yang hendak dilaksanakan dalam algoritma haruslah sederhana sehingga dapat secara prinsip dilakukan dengan tepat dan dalam waktu yang masuk akal apabila dilakukan secara manual oleh manusia dengan menggunakan pensil dan kertas.

### 1) Input

Input artinya meminta data yang diperlukan dari user. Sebagai contoh, dalam menghitung luas persegi panjang, tentu diperlukan data berupa besarnya panjang dan lebar bangun persegi panjang tersebut. Dengan kata lain, algoritma menentukan luas persegi panjang mempunyai 2 input berupa panjang dan lebar persegi panjang. Algoritma di buku ini menggunakan kata kunci **read** untuk menginput data. Bentuk penulisannya adalah

```
Read(variabel1, variabel2, ..., variabeln)
```

Data yang dapat diinputkan hanyalah data berupa integer, real, char, atau string. Sedangkan **data boolean tidak dapat** diinputkan menggunakan read. Dalam algoritma, kita tidak perlu memikirkan dari peralatan mana user menginput data, apakah dari mouse, keyboard, scanner, dan lain sebagainya. Hal itu merupakan masalah pemrograman. Pembuat algoritma juga tidak perlu memikirkan masalah tampilan saat penginputan berlangsung. Contohnya adalah



```

01| Algoritma ContohPenginputan
02| {contoh penggunaan read untuk menginput sejumlah data
    dari user}
03| Deklarasi:
04| a,b,c : integer
05| d : real
06| e : char
07| f : string
08| g : boolean
09| Deskripsi
10| {menginput data bil bulat, kemudian dimasukkan ke
    variabel a}
11| Read(a)
12| {menginput 2 data integer, dimasukkan ke variabel b dan
    c}
13| read(b,c)
14| {menginput data pecahan, kemudian dimasukkan ke variabel
    d}
15| read(d)
16| {menginput data bulat dan pecahan}
17| read(a,d)
18| {menginput data bulat, pecahan, dan 1 karakter}
19| read(a,d,e)
20| {menginput data string}
21| read(f)
22| {berikut ini adalah suatu kesalahan}
23| read(g) {data boolean tidak dapat diinputkan}

```

## 2) Output

Output artinya mencetak informasi yang dihasilkan oleh algoritma. Sebagai contoh dalam algoritma menghitung luas persegi panjang, hasil akhir yang diinginkan adalah luas persegi panjang. Dengan kata lain, algoritma tersebut memiliki satu output

yaitu luas persegi panjang. Algoritma dalam buku ini menggunakan kata kunci **write** untuk mencetak suatu data. Bentuk penulisannya adalah

```
write(data1, data2, ..., datan)
```

dimana data dapat berupa suatu data konstan, nama konstanta, ekspresi, maupun suatu variabel. Dalam algoritma, kita tidak mempermasalahkan ke peralatan mana data tersebut akan dicetak, karena merupakan masalah pemrograman. Kita juga tidak mempermasalahkan dalam format bagaimana data tersebut dicetak. Contoh:

```
01| Algoritma ContohPengoutputan
02| {berisi contoh-contoh penggunaan write
    untuk mencetak data}
03| deklarasi:
04| const k = 0.5
05| a : integer
06| b : real
07| c : char
08| d : string
09| e : boolean
10| deskripsi
11| read(a) {menginput sebuah data integer}
12| write(a) {mencetak data hasil input tadi}
13| write(b) {tidak boleh, karena variabel b
    belum ada isinya}
14| c ← „y“
15| d ← „stmik“
16| write(c,d) {mencetak isi variabel c dan
    d}
17| write(„Susi kuliah di „,d) {mencetak Susi
    kuliah di stmik}
18| write(5 + 6) {mencetak bilangan 11, yaitu
```

```
hasil dari 5+6}  
19| e ← 2 = 1 + 3  
20| write(e) {mencetak FALSE karena e  
berisikan false}  
21| write(k) {mencetak 0.5 karena k berisi  
0.5}
```

**Latihan:**

- 1) Sebutkan yang termasuk bahasa pemrograman prosedural?
- 2) Memahami sintaks program dengan menggunakan bahasa pascal dan bahasa C?
- 3) Mampu membuat algoritma mencari jumlah 3 buah bilangan bulat dengan flowchart?
- 4) Mampu membuat algoritma mencari hasil kali dari 2 buah bilangan dengan flowchart?

**Penyelesaian :**

Bahasa pemrograman prosedural : Pascal, C, Cobol, Basic, Fortran.

2. Sintaks program dengan menggunakan bahasa pascal dan bahasa C adalah sebagai berikut :

**Bahasa Pascal**

```
Program nama_program;  
[deklarasi label]  
[deklarasi konstan]  
[deklarasi tipe]  
[deklarasi variable]  
[deklarasi subprogram]  
Begin  
Pernyataan;  
.....  
Pernyataan  
End.
```

**Bahasa C**

```
#include <stdio.h>  
[deklarasi subprogram]  
Main()  
{  
[deklarasi variabel]  
Pernyataan;  
.....  
Return 0;  
}
```

Algoritma mencari jumlah 3 buah bilangan bulat dengan flowchart bilangan bulat

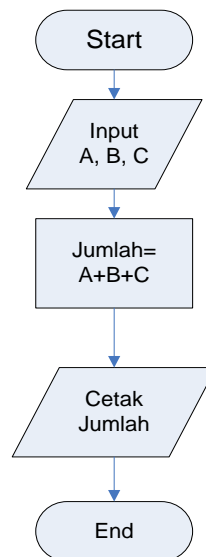
**Deklarasi**

a,b,c : integer  
jumlah : integer

**Deskripsi**

Read(a,b,c)  
Jumlah  $\leftarrow$  a + b + c  
Write(jumlah)

**Flowchartnya :**



Gambar 2.4. Flowchart Algoritma Mencari Jumlah 3 Buah Bilangan Bulat

2) Algoritma mencari hasil kali dari dua buah bilangan dengan flowchart

**Algoritma mencari hasil kali dari dua buah bilangan**

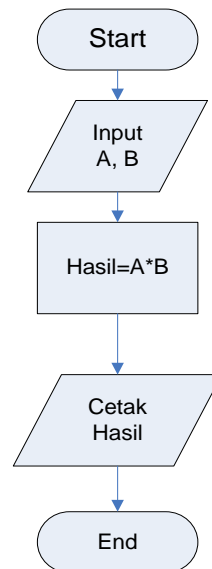
**Deklarasi**

a,b : integer  
hasil : integer

**Deskripsi**

Read(a,b)  
hasil  $\leftarrow$  a \* b  
Write(hasil)

**Flowchartnya :**



Gambar 2.5. Flowchart Algoritma Mencari Hasil Kali dari Dua Buah Bilangan

### TUGAS

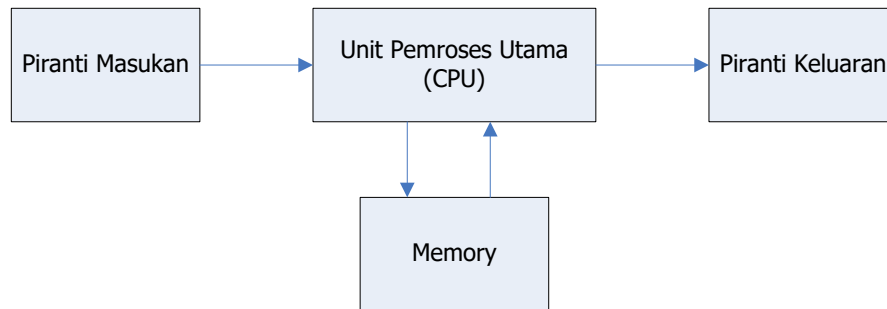
- 1) Buatlah algoritma dengan struktur indonesia dan flowchart untuk menukarkan isi dua buah nilai variabel yang diinputkan.
- 2) Buatlah algoritma dengan struktur indonesia dan flowchart untuk menentukan nilai terbesar diantara dua buah input.
- 3) Buatlah algoritma dengan struktur indonesia dan flowchart untuk menentukan nilai terbesar diantara tiga buah input.
- 4) Buatlah algoritma dengan struktur indonesia dan flowchart untuk menentukan input bilangan bulat termasuk bilangan genap atau ganjil atau nol.

## 2.5. Pengertian Pemrograman

Komputer hanyalah salah satu pemroses. Agar dapat dilaksanakan oleh komputer, algoritma harus ditulis dalam notasi bahasa pemrograman sehingga dinamakan program. Jadi program adalah perwujudan atau implementasi algoritma yang ditulis dalam bahasa pemrograman tertentu sehingga dapat dilaksanakan oleh komputer. Program ditulis dalam salah satu bahasa pemrograman, dan kegiatan membuat program disebut *pemrograman (programming)*. Orang yang menulis program disebut *pemrogram (programmer)*. Tiap-tiap langkah di dalam program disebut *pernyataan* atau *instruksi*. Jadi, program tersusun atas sederetan instruksi. Bila suatu instruksi dilaksanakan, maka operasi-operasi yang bersesuaian dengan instruksi tersebut dikerjakan oleh komputer.

### a) Mekanisme Pelaksanaan Algoritma oleh Pemroses

Secara garis besar komputer tersusun atas empat komponen utama: piranti masukan, piranti keluaran, unit pemroses utama dan memori. Unit pemroses utama (*Central Processing Unit – CPU*) adalah “otak” komputer, yang berfungsi mengerjakan operasi-operasi dasar seperti operasi perbandingan, operasi perhitungan, operasi membaca dan operasi menulis. Memori adalah komponen yang berfungsi menyimpan atau mengingat-ingat. Yang disimpan di dalam memori adalah program (berisi operasi-operasi yang akan dikerjakan oleh CPU) dan data atau informasi (sesuatu yang diolah oleh operasi-operasi). Piranti masukan atau keluaran (*I/O devices*) adalah alat yang memasukkan data atau program ke dalam memori, dan alat yang digunakan komputer untuk mengkomunikasikan hasil-hasil aktivitasnya. Contoh piranti masukan adalah : *keyboard, mouse, scanner dan disk*. Contoh alat keluaran adalah : *monitor, printer, plottter dan disk*.



Gambar 1.  
Komponen-komponen Utama Komputer

### b) Bahasa Pemrograman

Saat ini kita dapat berkomunikasi dengan komputer dengan menggunakan bahasa yang kita mengerti. Hal ini dapat kita lakukan karena para ahli telah berhasil membuat kamus yang disebut dengan bahasa pemrograman yang akan menterjemahkan bahasa yang kita buat menjadi bahasa mesin, kamus ini disebut dengan **Compiler**. Proses penterjemahan bahasa manusia ke bahasa mesin disebut dengan kompilasi.

Hal terpenting dalam menjalankan komputer adalah program. Dalam pemrograman dikenal beberapa bahasa pemrograman, seperti juga manusia mengenal bahasa-bahasa yang digunakan untuk berkomunikasi. Manusia dalam berkomunikasi menggunakan kata atau karakter sedangkan komputer dengan kode 0 dan 1.

Untuk mempermudah manusia berkomunikasi dengan komputer, maka diciptakan bahasa pemrograman. Dengan adanya bahasa pemrograman ini, bila manusia ingin berkomunikasi dengan komputer tidak harus menterjemahkan ke dalam 0 dan 1. Bila hal itu dilakukan betapa rumitnya suatu program.

Secara umum bahasa pemrograman dibagi menjadi empat kelompok :

#### ➔ Bahasa Aras Rendah (*Low Level Language*)

Merupakan bahasa yang berorientasi pada mesin. Pemrogram dengan bahasa ini harus berpikir berdasarkan logika mesin berpikir, sehingga bahasa ini kurang fleksibel dan sulit dipahami.

Contoh : Bahasa mesin, Bahasa rakitan (assembly)



➔ Bahasa Aras Menengah (*Middle Level Language*)

Merupakan bahasa pemrograman yang menggunakan aturan-aturan gramatikal dalam penulisan *ekspresi* atau pernyataan dengan standar yang mudah dipahami manusia serta memiliki instruksi-instruksi tertentu yang langsung bisa diakses oleh komputer.

Contoh : Bahasa C

➔ Bahasa Aras Tinggi (*Hight Level Language*)

Merupakan bahasa pemrograman yang menggunakan aturan-aturan gramatikal dalam penulisan *ekspresi* atau pernyataan dengan standar bahasa yang langsung dapat dipahami oleh manusia.

Contoh : Bahasa Pascal, Basic, COBOL

➔ Bahasa Berorientasi Objek (*Object Oriented Programming*)

Dengan bahasa berorientasi objek kita tidak perlu menuliskan secara detail semua pernyataan dan *ekspresi* seperti bahasa aras tinggi, melainkan cukup dengan memasukkan kriteria-kriteria yang dikehendaki saja.

Contoh : Delphi, Visual Basic, C++

Adapaun bahasa-bahasa pemrograman berdasarkan tipe tersebut antara lain :

Bahasa Pemrograman	Tipe	Dibuat
FORTTRAN	Prosedural	1950
BASIC	Prosedural	1960
LISP	Fungsional	1950
Prolog	Deklaratif	1970
Ada	Prosedural	1970
SmalTalk	Berorientasi Objek	1970
Pascal	Prosedural	1970
C	Prosedural	1970
C++	Berorientasi Objek	1980

Agar komputer memahami program yang disusun dengan bahasa pemrograman, maka dibutuhkan suatu penerjemah yaitu *Interpreter* dan *Compiler*.

a. Interpreter

*Interpreter* berasal dari kata *to interpret* yang berarti menerjemahkan atau mengartikan. *Interpreter* merupakan penerjemah bahasa pemrograman yang menerjemahkan instruksi demi instruksi pada saat eksekusi program. Pada saat penerjemahan interpreter akan memeriksa sintaksis (sintak program), semantik (arti perintah), dan kebenaran logika. Jika ditemukan kesalahan sintaksis (*syntak error*) maka interpreter akan menampilkan pesan kesalahan dan eksekusi program langsung terhenti.

b. Compiler

Berasal dari kata *to compile* yang berarti menyusun, mengumpulkan atau menghimpun. *Compiler* merupakan penerjemah bahasa pemrograman yang menerjemahkan instruksi-instruksi dalam satu kesatuan modul ke dalam bahasa mesin (*objek program*), kemudian objek program akan mengalami *linking* yang berfungsi untuk menggabungkan modul-modul tersebut dengan modul-modul lain yang berkaitan seperti data tentang karakteristik mesin, file-file pustaka atau objek program lainnya yang berkaitan dengan objek lainnya menghasilkan file *Executable* program yang akan dieksekusi oleh komputer.

Perbedaan Interpreter dan Compiler

Interpreter	Compiler
a. Menerjemahkan instruksi per instruksi	a. Menerjemahkan secara keseluruhan sekaligus
b. Bila terjadi kesalahan kompilasi, dapat langsung dibetulkan secara interaktif	b. Bila terjadi kesalahan kompilasi, <i>Source</i> program harus dibenarkan dan proses kompilasi diulang kembali
c. Tidak menghasilkan objek program	c. Menghasilkan objek program
d. Tidak menghasilkan <i>executable</i> program karena langsung dijalankan pada saat program diinterpretasi	d. Menghasilkan <i>executable</i> program,

e. Proses interpretasi terasa cepat, karena tiap-tiap instruksi langsung dikerjakan dan output langsung dilihat hasilnya	sehingga dapat dijalankan di keadaan prompt sistem
f. Source program terus dipergunakan karena tidak dihasilkan <i>executable</i> program	e. Proses kompilasi lama karena sekaligus menterjemahkan seluruh instruksi program
g. Proses pengerjaan program lebih lambat karena setiap instruksi dikerjakan harus diinterpretasikan ulang kembali	f. Source program sudah tidak dipergunakan lagi untuk mengerjakan program
h. Keamanan dari program kurang terjamin, karena yang selalu digunakan adalah <i>source</i> program	g. Proses mengerjakan program lebih cepat, karena <i>executable</i> program sudah dalam bahasa mesin
	h. Keamanan dari program lebih terjamin, karena yang dipergunakan <i>executable</i> program.

### c) Tahapan dalam Pemrograman

Langkah-langkah yang dilakukan dalam menyelesaikan masalah dalam pemrograman dengan komputer adalah:

#### - Definisikan Masalah

Berikut adalah hal-hal yang harus diketahui dalam analisis masalah supaya kita mengetahui bagaimana permasalahan tersebut:

- Kondisi awal, yaitu *input* yang tersedia.
- Kondisi akhir, yaitu *output* yang diinginkan.
- Data lain yang tersedia.
- Operator yang tersedia.
- Syarat atau kendala yang harus dipenuhi.

### **Contoh :**

Menghitung biaya percakapan telepon di wartel. Proses yang perlu diperhatikan adalah:

- a) *Input* yang tersedia adalah jam mulai bicara dan jam selesai bicara.
- b) *Output* yang diinginkan adalah biaya percakapan.
- c) Data lain yang tersedia adalah besarnya pulsa yang digunakan dan biaya per pulsa.
- d) Operator yang tersedia adalah pengurangan (-), penambahan (+), dan perkalian (\*).
- e) Syarat kendala yang harus dipenuhi adalah aturan jarak dan aturan waktu.

### **- Buat Algoritma dan Struktur Cara Penyelesaian**

Jika masalahnya kompleks, maka dibagi ke dalam modul-modul. Tahap penyusunan algoritma seringkali dimulai dari langkah yang global terlebih dahulu. Langkah global ini diperhalus sampai menjadi langkah yang lebih rinci atau detail. Cara pendekatan ini sangat bermanfaat dalam pembuatan algoritma untuk masalah yang kompleks. Penghalusan langkah dengan cara memecah langkah menjadi beberapa langkah.

Setiap langkah diuraikan lagi menjadi beberapa langkah yang lebih sederhana. Penghalusan langkah ini akan terus berlanjut sampai setiap langkah sudah cukup rinci dan tepat untuk dilaksanakan oleh pemroses.

### **- Menulis Program**

Algoritma yang telah dibuat, diterjemahkan dalam bahasa komputer menjadi sebuah program. Perlu diperhatikan bahwa pemilihan algoritma yang salah akan menyebabkan program memiliki untuk kerja yang kurang baik. Program yang baik memiliki standar penilaian:

- a) Standar teknik pemecahan masalah

#### **- Teknik *Top-Down***

Teknik pemecahan masalah yang paling umum digunakan. Prinsipnya adalah suatu masalah yang kompleks dibagi-bagi ke dalam beberapa

kelompok masalah yang lebih kecil. Dari masalah yang kecil tersebut dilakukan analisis. Jika dimungkinkan maka masalah tersebut akan dipilah lagi menjadi subbagian-subbagian dan setelah itu mulai disusun langkah-langkah penyelesaian yang lebih detail.

- **Teknik *Bottom-Up***

Prinsip teknik *bottom up* adalah pemecahan masalah yang kompleks dilakukan dengan menggabungkan prosedur-prosedur yang ada menjadi satu kesatuan program sebagai penyelesaian masalah tersebut.

b) Standar penyusunan program

- Kebenaran logika dan penulisan.
- Waktu minimum untuk penulisan program.
- Kecepatan maksimum eksekusi program.
- Ekspresi penggunaan memori.
- Kemudahan merawat dan mengembangkan program.
- *User Friendly*.
- *Portability*.
- Pemrograman modular.

c) Mencari Kesalahan

- a. Kesalahan sintaks (penulisan program).
- b. Kesalahan pelaksanaan: semantik, logika, dan ketelitian.

- **Uji dan Verifikasi Program**

Pertama kali harus diuji apakah program dapat dijalankan. Apabila program tidak dapat dijalankan maka perlu diperbaiki penulisan sintaksisnya tetapi bila program dapat dijalankan, maka harus diuji dengan menggunakan data-data yang biasa yaitu data yang diharapkan oleh sistem. Contoh data ekstrem, misalnya, program menghendaki masukan jumlah data tetapi *user* mengisikan bilangan negatif. Program sebaiknya diuji menggunakan data yang relatif banyak.

- **Dokumentasi Program**

Dokumentasi program ada dua macam yaitu dokumentasi internal dan dokumentasi eksternal. Dokumentasi internal adalah dokumentasi yang dibuat di dalam program yaitu setiap kita menuliskan baris program sebaiknya diberi komentar atau keterangan supaya mempermudah kita untuk mengingat logika yang terdapat di dalam instruksi tersebut, hal ini sangat bermanfaat ketika suatu saat program tersebut akan dikembangkan. Dokumentasi eksternal adalah dokumentasi yang dilakukan dari luar program yaitu membuat user guide atau buku petunjuk aturan atau cara menjalankan program tersebut.

- **Pemeliharaan Program**

- a. Memperbaiki kekurangan yang ditemukan kemudian.
- b. Memodifikasi, karena perubahan spesifikasi.

## Tugas

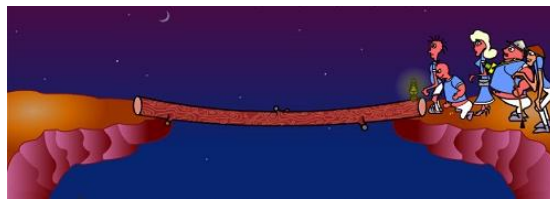
1. Membuat algoritma kegiatan/rutinitas mahasiswa
2. Tugas Diskusi : Mencari solusi penyelesaian masalah game Grestelline:
  - a. Game 1 *Wolf Sheep and Cabbage*



- b. Game 2 *Cannibals and Missionaries*



- c. Game 3 *Family Crisis*



- d. Game 4 *Elevators Logic*



- e. Game 5 *Golo In The Cave*



## BAB III

# TIPE DATA, VARIABLE, NILAI DAN INSTRUKSI UTAMA

Pada prinsipnya suatu program komputer memanipulasi data untuk menjadi informasi yang berguna. Ada tiga hal yang berkaitan dengan data, yaitu:

1. **Tipe data** : setiap data memiliki tipe data, apakah data itu merupakan angka bulat (integer), angka biasa (real), atau berupa karakter (char), dsb.
2. **Variabel** : setiap data diwakili oleh suatu variable, dan variable ini diberi nama agar bisa dibedakan terhadap variable lainnya.
3. **Nilai** : setiap data memiliki harga atau nilai, misalnya umur seseorang diwakili oleh variabel UMUR yang bertipe bilangan, dan memiliki nilai 20 tahun. Perlu diketahui bahwa dalam representasi nilai data dalam komputer setiap tipe data memiliki batasan nilai masing-masing.

### 3.1. Tipe Data

Dalam dunia pemrograman komputer selalu melibatkan data, karena pemrograman tidak bisa terlepas dari kegiatan mengolah data menjadi informasi yang diperlukan. Untuk menjamin konsistensi data dan efisiensi penggunaan memori komputer, maka data dibedakan menjadi beberapa tipe. Dalam algoritma, tipe data yang ada lebih sedikit dibanding bahasa pemrograman dikarenakan algoritma hanya menekankan pada penyelesaian masalah. Beberapa tipe data tersebut adalah:

- **Integer**

Merupakan tipe data bilangan bulat, baik yang negatif, nol, maupun bilangan positif. Dalam algoritma, semua bilangan bulat termasuk dalam tipe ini, tanpa ada batasan.



- **Real**

Merupakan tipe data bilangan pecahan. Semua bilangan yang mengandung tanda desimal tergolong dalam tipe ini. Tanda desimal yang dipakai adalah tanda titik, bukan tanda koma. Sebagai contohnya, nilai 0,5 (nol koma lima) ditulis menjadi **0.5**. Tipe ini juga mendukung penulisan dalam bentuk ilmiah, misalnya **2.64E-2** artinya  $2.64 \times 10^{-2}$ .

- **Char**

Merupakan tipe data karakter. Semua data yang hanya terdiri dari 1 karakter tergolong dalam tipe ini. Misalnya data jenis kelamin yang hanya diisi huruf L atau P. Penulisan data tipe char harus diapit oleh **tanda petik tunggal**. Karakter-karakter yang diperbolehkan terdefinisi dalam **tabel ASCII**.

- **String**

Merupakan tipe data kalimat. Semua data yang terdiri dari 1 karakter atau lebih dapat digolongkan ke dalam tipe ini. Syaratnya sama dengan tipe char, yaitu harus diapit oleh **tanda petik tunggal**.

- **Boolean**

Merupakan tipe data yang hanya mempunyai nilai **TRUE** atau **FALSE**. Penulisan **TRUE** ataupun **FALSE** tidak membedakan huruf kapital ataupun non-kapital. Hanya saja penulisannya tidak boleh disingkat menjadi huruf **T** atau huruf **F** saja.

Berikut ini merupakan tabel contoh-contoh data berbagai tipe.

Nama Tipe	Nilai yang diizinkan
Integer	8
	-74
	0
Real	7.5
	3E2
	3.2E-4
	0.0
	0.5

Char	'a' '!' '3' ' ' '?' “ “ {yaitu 1 buah karakter tanda petik tunggal}
String	'mahasiswa' 'a' '7483' '?w?' '.....' 'pergi!' 'I '1 love you 4ever'
Boolean	True tRuE FALse False

Ada dua kategori dari tipe data, yaitu: tipe dasar dan tipe bentukan.

- Tipe dasar : adalah tipe data yang selalu tersedia pada setiap bahasa pemrograman, antara lain: *bilangan bulat* (integer), *bilangan biasa* (real), *bilangan tetap* (const), *karakter* (character atau char), *logik* (logic atau boolean).
- Tipe bentukan : adalah tipe data yang dibentuk dari kombinasi tipe dasar, antara lain: *larik* (array), *rekaman* (record), *string* (string).

#### **Bilangan bulat (integer)**

- Bilangan atau angka yang tidak memiliki titik desimal atau pecahan, seperti: 10, +255, -1024, +32767.
- Tipe dituliskan sebagai : **integer** atau **int**
- Jangkauan nilai : bergantung pada implementasi perangkat keras komputer, misalnya dari -32768 s/d +32767, untuk algoritma tidak kita batasi.
- Operasi aritmetik : tambah + , kurang - , kali \* , bagi / , sisa hasil bagi %

- Operasi perbandingan : lebih kecil  $<$  , lebih kecil atau sama  $< =$  , lebih besar  $>$  , lebih besar atau sama  $> =$  , sama  $=$  , tidak sama  $><$

#### **Bilangan biasa (real)**

- Bilangan atau angka yang bisa memiliki titik desimal atau pecahan, dan ditulis sebagai: 235.45, +1023.55, -987.3456 atau dalam notasi ilmiah seperti: 1.245E+03, 7.45E-02, +2.34E-04, -5.43E+04, dsb.
- Tipe dituliskan sebagai : **real**
- Jangkauan nilai : bergantung pada implementasi perangkat keras komputer, misalnya dari -2.9E-39 s/d +1.7E+38, untuk algoritma tidak kita batasi.
- Operasi aritmatik dan perbandingan juga berlaku bagi bilangan biasa.

#### **Bilangan tetap (const)**

- Bilangan tetap (**const**) adalah tipe bilangan baik bernilai bulat maupun tidak yang nilainya tidak berubah selama algoritma dilaksanakan.
- Tipe dituliskan sebagai : **const**
- Jangkauan nilai meliputi semua bilangan yang mungkin

#### **Karakter (character)**

- Karakter adalah data tunggal yang mewakili semua huruf, simbol baca, dan juga simbol angka yang tidak dapat dioperasikan secara matematis, misalnya: 'A', 'B', ... , 'Z', 'a', 'b', ..., 'z', '?', '!', ':', ';' dst.
- Tipe dituliskan sebagai : **char**
- Jangkauan nilai meliputi semua karakter dalam kode ASCII, atau yang tertera pada setiap tombol keyboard.
- Operasi perbandingan dapat dilakukan dan dievaluasi menurut urutan kode ASCII, sehingga huruf 'A' (Hex 41) sebenarnya lebih kecil dari huruf 'a'.

#### **Logik (Logical)**

- Tipe data logik adalah tipe data yang digunakan untuk memberi nilai pada hasil perbandingan, atau kombinasi perbandingan.
- Tipe dituliskan sebagai : **boolean**
- Jangkauan nilai ada dua: true dan false

- Contoh:  $45 > 56$  hasilnya false, Amir < Husni hasilnya true
- Ada beberapa operasi untuk data jenis logik, antara lain: **and**, **or**, dan **not**

### **Array (larik)**

- Array adalah tipe data bentukan, yang merupakan wadah untuk menampung beberapa nilai data yang sejenis. Kumpulan bilangan bulat adalah array integer, kumpulan bilangan tidak bulat adalah array real.
- Cara mendefinisikan ada dua macam, yaitu:
  - Nilai\_ujian : array [1 .. 10] of integer; atau
  - int nilai\_ujian[10];
- Kedua definisi diatas menunjukkan bahwa nilai\_ujian adalah kumpulan dari 10 nilai bertipe bilangan bulat.

### **String**

- String adalah tipe data bentukan yang merupakan deretan karakter yang membentuk satu kata atau satu kalimat, yang biasanya diapit oleh dua tanda kutip.
- Sebagai contoh : nama, alamat, dan judul adalah tipe string.
- Cara mendefinisikannya adalah:
  - String Nama, Alamat; atau
  - Nama, Alamat : String;

### **Record (rekaman)**

- Record adalah tipe data bentukan yang merupakan wadah untuk menampung elemen data yang tipe-nya tidak perlu sama dengan tujuan untuk mewakili satu jenis objek.
- Sebagai contoh, mahasiswa sebagai satu jenis objek memiliki beberapa elemen data seperti: nomer\_stb, nama, umur, t4lahir, jenkel.
- Cara mendefinisikan record mahasiswa tersebut adalah sbb:

```
Type DataMhs : record
```

```
< nomer_stb : integer,  
  nama_mhs : string,  
  umur : integer,  
  t4lahir : string,  
  jenkel : char;  
>
```

### 3.2. Variabel

Variable adalah nama yang mewakili suatu elemen data seperti: jenkel untuk jenis kelamin, t4lahir untuk tempat lahir, alamat untuk alamat, dan sebagainya. Ada aturan tertentu yang wajib diikuti dalam pemberian nama variable antara lain:

- Harus dimulai dengan abjad, tidak boleh dengan angka atau simbol
- Tidak boleh ada spasi diantaranya
- Jangan menggunakan simbol-simbol yang bisa membingungkan seperti titik dua, titik koma, koma, dsb.
- Sebaiknya memiliki arti yang sesuai dengan elemen data
- Sebaiknya tidak terlalu panjang
  - Contoh variable yang benar : Nama, Alamat, Nilai\_Ujian
  - Contoh variable yang salah : 4XYZ, IP rata, Var:+xy,458;

Variabel (perubah) merupakan suatu nama yang menyiratkan lokasi memori komputer yang dapat digunakan untuk menyimpan nilai, dimana isinya dapat diubah-ubah. Variabel dapat dipandang sebagai abstraksi dari lokasi. Hasil evaluasi dari variabel adalah nilai dari variabel itu. Nilai dari suatu variabel dapat diubah dengan *assignment statement*. Sebuah *assignment statement* terdiri dari sebuah variabel di sebelah kirinya dan suatu ekspresi disebelah kanannya.

Algoritmik	Bahasa Pascal	Bahasa C
<b>Deskripsi</b>		
Jumlah $\leftarrow$ B1 + B2	Jumlah := B1 + B2	Jumlah = B1 + B2

Variabel jumlah diubah nilainya menjadi nilai dari ekspresi  $B1 + B2$  setelah dievaluasi. Dalam suatu program Pascal maupun C, setiap variabel yang akan digunakan terlebih dahulu dideklarasikan, dimana setiap variabel harus mempunyai tipe. Deklarasi variabel berguna untuk memberi informasi kepada compiler serta membantu programmer untuk berpikir secara jelas dan berencana.

Algoritmik	Bahasa Pascal	Bahasa C
<b>Deklarasi</b>	<b>Var</b>	
B1,B2,jumlah:integer	B1,B2,jumlah:integer;	Int B1,B2,jumlah;

### 3.3. Konstanta

Variabel yang mempunyai nilai yang sifatnya tidak bisa diubah, nilai ditentukan pada saat pendefinisian. Misal :

```
<nama konstanta1> = <nilai1>;
Phi = 3.14;
```

Konstanta merupakan suatu nilai yang telah ditetapkan di awal pembuatan algoritma dan nilainya tidak dapat diubah oleh proses dalam algoritma. Cara mendefinisikan konstanta adalah dengan menambahkan kata kunci **const** diawal nama konstanta dan **diletakkan di bagian deklarasi**. Contoh:

```
01| Deklarasi:
02| const phi = 3.14
03| const k = 0.5
04| const password = „SeCReT“
```

### 3.4. Pemberian Nilai

Penugasan atau **Assignment** merupakan pemberian nilai ke variabel **secara langsung**. Notasi yang digunakan adalah  $\leftarrow$ . Nilai yang dapat diberikan adalah **tetapan, peubah, ekspresi**, maupun **nilai yang dihasilkan oleh fungsi**. Syarat penugasan adalah nilai yang diberikan **harus sesuai** dengan tipe variabel. Apabila tipenya tidak sama, maka berlaku tipe yang lebih luas dapat menampung tipe yang lebih sempit. Tipe **integer** dapat ditampung oleh tipe **real**, sebaliknya tipe **real tidak dapat ditampung** oleh tipe **integer**. Begitu pula dengan **string** dan **char**, char dapat diberikan ke string, namun tidak sebaliknya.

Contohnya:

```
01| Deklarasi:
02| a,b : integer
03| c,d : real
04| nama1, nama2 : string
05| huruf : char
06| ketemu : boolean
07|
08| Deskripsi:
09| a  $\leftarrow$  3 {boleh}
10| b  $\leftarrow$  a + 4 {boleh}
11| c  $\leftarrow$  3 {boleh}
12| a  $\leftarrow$  0.5 {tidak boleh karena a tidak dapat menampung
    real}
13| a  $\leftarrow$  b + c {tidak boleh karena b+c bertipe real}
14| huruf  $\leftarrow$  „?“ {boleh}
15| nama1  $\leftarrow$  „Ani“  $\leftarrow$  {boleh}
16| nama2  $\leftarrow$  nama1 + huruf {boleh}
17| nama1  $\leftarrow$  „a“ {boleh}
```

```
18| huruf ← nama1 {tidak boleh karena nama1 bertipe
string}
19| ketemu ← true {boleh}
20| ketemu ← „true“ {tidak boleh karena „true“ bertipe
string}
```

Ada dua cara yang dapat digunakan untuk memberi nilai pada suatu variable yaitu melalui proses : *assignment* dan *pembacaan*.

a. Pemberian nilai dengan cara assignment mempunyai bentuk umum sebagai berikut:

- Variable ← nilai;
- Variable1 ← variable2;
- Variable ← ekspresi;

Contoh assignment:

- Nama ← “Ali bin AbuThalib”;
- Jarak ← 100.56;
- X ← Jarak;
- Rentang ←  $X + 50 - 3*Y$ ;

b. Pemberian nilai dengan cara pembacaan dapat dilakukan melalui instruksi dengan bentuk umum sbb:

- read(variable); atau
- read( variable1, variable2, ... );

Contoh pembacaan data:

- read>Nama);
- read(Jarak, Rentang, X);



### 3.5. Menampilkan Nilai

Agar hasil pelaksanaan algoritma dapat dikomunikasikan maka nilai variable yang diproses dalam algoritma dapat ditampilkan.

Instruksi untuk menampilkan nilai variable adalah:

**write**(variable, ... );

Contoh penampilan nilai adalah sebagai berikut:

```
write("nama anda : ", Nama);  
write("nilai ujian = ", nilai);  
write("Jumlah variable = ", X + Y + Z);
```

### 3.6. Ekspresi (Expression)

Ekspresi adalah serangkaian perhitungan nilai yang menghasilkan suatu nilai yang diinginkan. Ekspresi terdiri dari **operand** dan **operator**. Operand adalah nilai-nilai yang akan dihitung. Operator adalah lambang operasi yang dipakai dalam perhitungan. Contoh:  $6 + 3$ , angka 6 dan 3 adalah operand, sedangkan tanda + merupakan operator. Operand dapat berupa **tetapan** (konstanta), **peubah** (variabel), atau **hasil dari suatu fungsi**. Operator dapat berupa operator **unary**, yaitu operator yang hanya memerlukan 1 operand, dan operator **binary**, yaitu operator yang memerlukan 2 operand. Beberapa jenis operator yang dipakai dalam algoritma adalah:

#### a. operator aritmatika

Lambang Operator	Jenis	Tipe Operand1	Tipe Operand2	Tipe Hasil
+, -	unary	Integer Real		Integer Real
+, -, *	Binary	Integer Integer Real Real	Integer Real Integer	Integer Real Real

			Real	Real
Div	Binary	integer	integer	integer
/	Binary	Integer Integer	Integer	Real
		Real Real	Real	Real
			Integer	Real
			Real	Real
Mod	Binary	Integer	Integer	Integer

#### b. Operator Perbandingan

Lambang Operator	Jenis	Tipe Operand1	Tipe Operand2	Tipe Hasil
<	Binary	Integer	Integer	Boolean
<=	Binary	Integer	Real	Boolean
>	Binary	Real	Real	Boolean
>=	Binary	Char	Char	Boolean
=	Binary	Char	String	Boolean
<>	Binary	String	String	Boolean
	Binary	Boolean	Boolean	Boolean

Keterangan:

- perbandingan char mengacu pada urutan karakter dalam tabel ASCII.  
Contoh: „a“ > „A“ akan menghasilkan true
- perbandingan string, dibandingkan per karakter yang berkesesuaian mulai dari karakter pertama.  
Contoh: „abc“ > „aBc“ akan menghasilkan true.
- perbandingan boolean, false diidentikkan dengan angka 0, sedangkan true diidentikkan dengan angka 1. Jadi true > false akan menghasilkan true.

#### c. Operator String

Yaitu lambang + yang merupakan operasi penyambungan (*concatenation*)

Contoh: „anak“ + „ku“ akan menghasilkan „anakku“

#### d. Logika

Lambang Operator	Jenis	Tipe Operand1	Tipe Operand2	Tipe Hasil
not	Unary	Boolean		Boolean
and, or, xor	Binary	Boolean	Boolean	Boolean

a	not a
True	False
False	True

### Tingkatan-tingkatan dalam operator

Precedence	Operator	Keterangan
First (High)	NOT + -	Unary operator
Second	* / div mod	Perkalian dan pembagian
	AND	Boolean
Third	+ -	Penjumlahan
	OR XOR	Boolean
Fourth (Low)	= < >	Relasional
	>= <=	

- Transformasi data dan peubah dalam bentuk persamaan yang direlasikan oleh operator dan operand.
- Operand adalah data, tetapan, peubah, atau hasil dari suatu fungsi.
- Operator adalah simbol-simbol yang memiliki fungsi untuk menghubungkan operand sehingga terjadi transformasi. Jenis-jenis operator adalah sbb:
  - a) Operator aritmetika : operator untuk melakukan fungsi aritmetika seperti: + (menjumlah), - (mengurangkan), \* (mengalikan), / (membagi).
  - b) Operator relational : operator untuk menyatakan relasi atau perbandingan antara dua operand, seperti : > (lebih besar), < (lebih kecil), >= (lebih besar atau sama), <= (lebih kecil atau sama), == (sama), != (tidak sama) atau ><.

- c) Operator logik : operator untuk merelasikan operand secara logis, seperti
- d) && (and), || (or), dan ! (not).
- e) Operator string : operator untuk memanipulasi string, seperti :  
+ (*concatenation*), dan **substr** (substring, mencuplik).
- Berdasarkan pada jenis operator yang digunakan maka ada empat macam ekspresi, yaitu: *ekspresi aritmetika*, *ekspresi relational*, *ekspresi logik*, dan *ekspresi string*.
- **Ekspresi Aritmetika** : ekspresi yang memuat operator aritmetika, contoh:
  - $T \leftarrow 5 * (C + 32) / 9;$
  - $Y \leftarrow 5 * ((a + b) / (c + d) + m / (e * f));$
  - $Gaji \leftarrow GaPok * (1 + JumNak * 0.05 + Lembur * 1.25);$
- **Ekspresi Relational** : ekspresi yang memuat operator relational, contoh:
  - $Nilai\_A > Nilai\_B$
  - $(A + B) < (C + D)$
  - $(x + 57) != (y + 34)$
- **Ekspresi Logik** : ekspresi yang memuat operator logik, contoh:
  - $m \leftarrow (x > y) \&\& (5 + z)$
  - $n \leftarrow (!A \parallel !(B \&\& C))$
- **Ekspresi String** : ekspresi dengan operator string, contoh:
  - $Alamat \leftarrow \text{"Jl. P. Kemerdekaan"} + \text{"Km 9 Tamalanrea"}$
  - $Hasil \leftarrow \text{"Saudara :"} + Nama + \text{"adalah mahasiswa"}$
  - $Tengah \leftarrow Substr(Kalimat, 5, 10);$

### Latihan Dan Tugas

1. Susun algoritma yang menghitung pajak pertambahan nilai (ppn) 12.50% dengan meminta harga barang yang dibeli dari pengguna program.

#### Algoritma PPN

```
{ menghitung pajak pertambahan nilai 12.50% dari harga barang }
```

##### Deklarasi

```
real harga, pajak, total;
```

##### Deskripsi

```
write ("Masukkan harga barang : ");
```

```
read(harga);
```

```
pajak ← 0.125 * harga;
```

```
total = harga + pajak;
```

```
write("Harga = ", harga, " pajaknya = ", pajak);
```

```
write("Total = ", total);
```

2. Susun algoritma yang meminta data dasar mahasiswa (mis: Nama, Alamat, e\_mail, dan telepon) kemudian menampilkan-nya kembali tersusun.

#### Algoritma Data\_dasar

```
{ membaca dan menampilkan data dasar mahasiswa }
```

##### Deklarasi

```
string nama, alamat, e_mail, telepon;
```

##### Deskripsi

```
write ("Masukkan nama anda : ");
```

```
read(nama);
```

```
write("Dimana alamatnya : ");
```

```
read(alamat);
```

```
write("No telepon : ");
```

```
read(telepon);
```

```
write("Alamat e-mail : ");
```

```
read(e_mail);
```

```
write(nama);
```

```
write(alamat, telepon);
```

```
write(e_mail);
```

3. Buatlah algoritma untuk menghitung luas segitiga jika diketahui panjang alas dan tinggi segitiga. Analisis: input : alas (a) dan tinggi (t) segitiga output : luas (L) segitiga rumus :  $L = 0,5 \times a \times t$

Langkah pengerjaan:

- meminta input data alas dan tinggi segitiga dari user
- menghitung luas segitiga menggunakan rumus
- mencetak output berupa luas segitiga

**01| Algoritma Menghitung\_Luas\_Segitiga**

**02|** {Menghitung luas segitiga jika diketahui panjang alas dan tinggi segitiga. Alas dan tinggi diinput dari user. Kemudian Luas dihitung menggunakan rumus  $Luas = 0,5 \times \text{alas} \times \text{tinggi}$ . Kemudian mencetak output berupa luas segitiga}

**03| Deklarasi:**

**04| a, t, L : real**

**05| Deskripsi**

**06| read(a,t)**

**07|  $L \leftarrow 0.5 * a * t$**

**08| write(L)**

4. Buatlah algoritma untuk menghitung komisi yang diterima salesman berdasarkan jumlah penjualan yang dicapainya. Salesman tersebut mendapat komisi 10% dari hasil penjualannya. Input algoritma ini adalah nama salesman dan jumlah penjualan yang dicapainya. Sedangkan outputnya adalah nama salesman dan besar komisi yang diperolehnya. Analisis: input: nama salesman (nama) dan jumlah penjualan (j) output: nama salesman (nama) dan besar komisi (komisi) rumus:  $\text{komisi} = 10\% \times \text{jumlah penjualan}$  Langkah pengerjaan:

- menginput data nama salesman dan jumlah penjualan
- menghitung komisi menggunakan rumus
- mencetak nama dan komisi

01| Algoritma Menghitung\_Komisi\_Salesman

02|{Menghitung besarnya komisi yang diperoleh salesman berdasarkan jumlah penjualan. Besarnya komisi adalah 10% dari jumlah penjualan yang dicapainya. Kemudian algoritma akan mencetak nama salesman dan komisi sebagai outputnya.}

03| Deklarasi:

04| nama : string {nama salesman}

05| j : integer {jumlah penjualan}

06| komisi : real {komisi yang diperoleh}

07| Deskripsi

08| read(nama, j)

09| komisi  $\leftarrow 0.1 * j$

10| write(nama, komisi)

5. Buatlah algoritma untuk menghitung gaji karyawan. Diberikan nama karyawan dan besarnya gaji pokok. Gaji bersih yang diterima pegawai adalah gaji pokok ditambah besarnya tunjangan kemudian dikurangi pajak. Tunjangan karyawan dihitung 20% dari gaji pokok, sedangkan pajak adalah 15% dari gaji pokok ditambah tunjangan. Keluaran yang diharapkan adalah nama karyawan, besarnya tunjangan, pajak, dan gaji bersihnya. Analisis: input: nama karyawan (nama) dan besarnya gaji pokok (gaji\_pokok) output: nama karyawan (nama), tunjangan (tunj), pajak (pjk), dan gaji bersih (gaji\_bersih) rumus:  $tunj = 20\% \times gaji\_pokok$   
 $pjk = 15\% \times (gaji\_pokok + tunj)$   $gaji\_bersih = gaji\_pokok + tunj - pajak$

Langkah pengerjaan:

- menginput nama karyawan dan gaji pokok
- menghitung tunjangan
- menghitung pajak
- menghitung gaji bersih
- mencetak nama, tunjangan, pajak, dan gaji bersih

6. Buatlah algoritma untuk mengkonversi jam-menit-detik ke total detik. Data jam-menit-detik diinput dari user. Contoh, misalnya data jam-menit-detiknya adalah 1 jam 30 menit 40 detik, maka besarnya total detik adalah 5440 detik. Analisis: input: jam (j), menit (m), detik (d) output: total detik (total) Rumus: ingat bahwa 1 jam = 3600 detik dan 1 menit = 60 detik. maka total detik = jam x 3600 + menit x 60 + detik Langkah pengerjaan:

- menginput jam, menit, dan detik
- menghitung total detik menggunakan rumus
- mencetak total detik

7. Buatlah algoritma untuk mengkonversi total detik ke bentuk jam-menit-detik. Data total detik diinput oleh user. Contohnya, misalnya data total detiknya adalah 5440 detik, maka outputnya adalah 1 jam 30 menit 40 detik.

Analisis:

input: total detik (total)

output: jam (j), menit (m), detik (d)

rumus:

Pada dasarnya, yang hendak dikerjakan adalah mencari tahu total detik tersebut sama dengan berapa jam dan berapa sisa detiknya. Dari sisa detik tersebut, barulah dicari berapa menit dan sisa berapa detik. Contohnya: misalkan total detiknya adalah 5440 detik. Maka 5440 detik dibagi 3600 adalah 1 jam sisa 1840 detik. Kemudian, 1840 detik dibagi 60 adalah 30 menit sisa 40 detik. Jadi 5440 detik = 1 jam 30 menit 40 detik.

Dari contoh di atas dapat disimpulkan rumus-rumus berikut: jam = total div 3600 sisa = total mod 3600 menit = sisa div 60 detik = sisa mod 60 Langkah pengerjaan:

- menginput total detik
- menghitung jam dari total detik
- menghitung sisa pembagian jam dari total detik
- menghitung menit dari sisa
- menghitung detik dari sisa
- mencetak jam, menit, dan detik



## BAB IV

# RUNTUNAN, PEMILIHAN DAN PENGULANGAN

(sequence, selection & repetition)

### 4.1. Instruksi Runtunan (Sequential)

Instruksi runtunan adalah instruksi yang dikerjakan secara beruntun atau berurutan baris per-baris mulai dari baris pertama hingga baris terakhir, tanpa ada loncatan atau perulangan.

- Tiap instruksi dikerjakan sekali satu per-satu
- Urutan pelaksanaan instruksi sama dengan urutan penulisan algoritma
- Instruksi terakhir merupakan akhir dari algoritma
- Urutan penulisan instruksi bisa menjadi penting, bila diubah dapat menyebabkan hasil yang berbeda.

#### Contoh 1:

##### ▪ Algoritma Runtunan\_1

```
{ menunjukkan urutan yang berbeda memberi hasil yang berbeda }  
Deklarasi  
integer A, B;  
Deskripsi  
A ← 10;  
A ← 2 * A;  
B ← A;  
write(B);
```

Algoritma diatas menampilkan hasil : 20

##### ▪ Algoritma Runtunan\_2

```
{ menunjukkan urutan yang berbeda memberi hasil yang berbeda }
```

```

Deklarasi
integer A, B;
Deskripsi
A ← 10;
B ← A;
A ← 2 * A;

```

Dengan urutan yang diubah maka algoritma ini memberi hasil : 10

#### ▪ Algoritma Runtunan\_3;

{ algoritma untuk menghitung luas sebuah segitiga }

##### **Deklarasi**

```

real Alas, Tinggi;
real Luas;
Deskripsi
write ("Masukkan panjang alasnya : ");
read (Alas);
write ("Masukkan tingginya : ");
read (Tinggi);
Luas ← Alas * Tinggi / 2;
write ("Luas segitiga = ", Luas);

```

#### ▪ Algoritma Runtunan\_4;

{ algoritma menampilkan gaji bersih pegawai }

##### **Deklarasi**

```

string nama;
real gajipokok, tunjangan, pajak;
real gajibersih;
Deskripsi
write ("Masukkan nama pegawai : ");
read (nama);
write ("Masukkan gaji pokoknya : ");

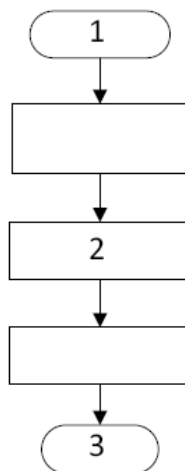
```

```

read (gajipokok);
tunjangan ← 0.25 * gajipokok;
pajak ← 0.15 * (gajipokok + tunjangan);
gajibersih ← gajipokok + tunjangan - pajak;
write ("Gaji saudara : ", nama);
write ("adalah = ", gajibersih);

```

Suatu struktur program dimana setiap baris program akan dikerjakan secara urut dari atas ke bawah sesuai dengan urutan penulisannya.



Dari flowchart diatas mula-mula pemroses akan melaksanakan instruksi baris program 1, instruksi baris program 2 akan dikerjakan jika instruksi baris program 1 telah selesai dikerjakan. Selanjutnya instruksi baris program 3 dikerjakan setelah instruksi baris program 2 selesai dikerjakan. Setelah instruksi baris program 3 selesai dilaksanakan maka algoritma berhenti.

#### Contoh 1 :

Akan dihitung luas pesegi panjang yang diketahui panjang dan lebarnya, maka algoritmanya sebagai berikut :

#### Algoritma Luas Pesegi Panjang :

Diketahui sebuah pesegi panjang yang memiliki panjang dan lebar.

Deskripsi :

1. mulai
2. Baca panjang
3. Baca lebar
4. Hitung luas = panjang \* lebar
5. Tampilkan luas
6. selesai

### Contoh 2 :

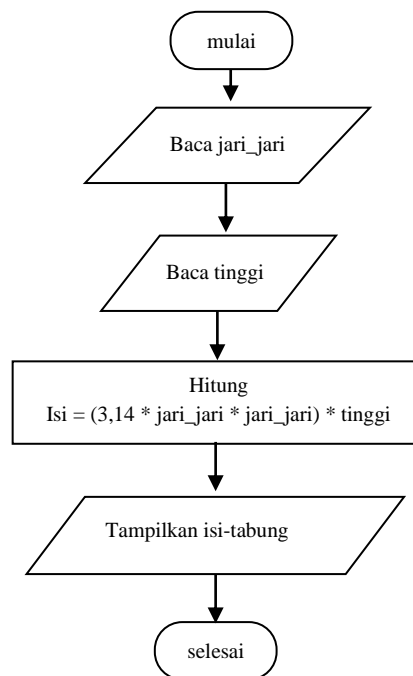
Akan dihitung isi sebuah tabung yang diketahui jari-jari lingkaran dan tinggi tabung.

#### Algoritma Isi Tabung

Diketahui sebuah tabung yang diketahui jari-jari tabung dan tinggi tabung.

Deskripsi :

1. mulai
2. Baca jari\_jari
3. Baca tinggi
4. Hitung  $luas\_lingk = 3.14 * jari\_jari * jari\_jari$
5. Hitung  $isi\_tabung = luas\_lingk * tinggi$
6. Tampilkan isi\_tabung
7. selesai



Gambar 4.2. *Flowchart* menghitung isi tabung

Dari kedua algoritma dan flowchart diatas terlihat bahwa algoritma yang kedua lebih sedikit baris intruksinya, sehingga menyebabkan pemrosesan menjadi lebih cepat selesai dengan hasil yang sama dengan algoritma pertama. Pada algorima yang kedua

jika diimplementasikan dalam program kebutuhan variabelnya juga lebih sedikit sehingga menghemat penggunaan memori.

#### 4.2. Instruksi Pemilihan (*Selection*)

Instruksi pemilihan adalah instruksi yang dipakai untuk memilih satu aksi dari beberapa kemungkinan aksi berdasarkan suatu persyaratan.

Pada pemrograman penyeleksian dilakukan pada suatu pernyataan boole, yang dapat menghasilkan nilai benar (true) atau nilai salah (false). Biasanya sebuah pernyataan pemilihan terdiri dari operand-operand yang dihubungkan dengan operator relasi dan digabungkan dengan operator logika.

Contohnya :

- $7 = 7$  (Benilai benar, sebab 7 sama dengan 7)
- $5 = 9$  (Bernilai salah, sebab 5 tidak sama dengan 9)
- $4 > 2$  (Bernilai benar, sebab 4 lebih besar dari pada 2)
- $3 \neq 8$  (Bernilai benar, sebab 3 tidak sama dengan 8)
- $X = 10$  (Dapat benilai benar atau salah, tergantung isi variabel X)
- $(X > 3) \text{ And } (Y < 12)$

(Dapat benilai benar atau salah, tergantung isi variabel X dan Y)

Struktur pemilihan dalam penulisan program diimplementasikan dengan instruksi **IF**.

Ada kalanya sebuah instruksi dikerjakan jika kondisi tertentu dipenuhi. Penulisan pemilihan secara umum :

If kondisi then

**Aksi**

Dalam bahasa indonesia, if berarti “jika” dan then artinya “maka”. Kondisi adalah persyaratan yang dapat bernilai salah atau benar. Aksi hanya dilakukan jika kondisi bernilai benar. Perhatikan kata yang digarisbawahi, if dan then merupakan

kata kunci(*keywords*) untuk struktur pemilihan ini. Dalam kehidupan sehari-hari, kita sering menuliskan pernyataan tindakan bila suatu persyaratan dipenuhi. Misalnya :

```
If Zaki memperoleh juara kelas then  
    Ayah akan membelikannya sepeda  
If jalan panenan macet then  
    Ambil alternatif jalan dipati ukur
```

Struktur pemilihan if-then hanya memberikan satu pilihan aksi jika kondisi dipenuhi (bernilai benar), dan tidak memberi pilihan aksi lain jika bernilai salah. Bentuk pemilihan yang lebih umum ialah memilih satu dari dua buah aksi bergantung pada nilai kondisinya :

```
If kondisi then  
    Aksi 1  
Else  
    Aksi 2
```

Else artinya “kalau tidak”. Bila kondisi bernilai benar, aksi 1 akan dikerjakan, tetapi kalau tidak, aksi 2 yang akan dikerjakan. Misalnya pada pernyataan berikut:

```
If hari hujan then  
    Pergilah dengan naik beca  
Else  
    Pergilah dengan naik motor
```

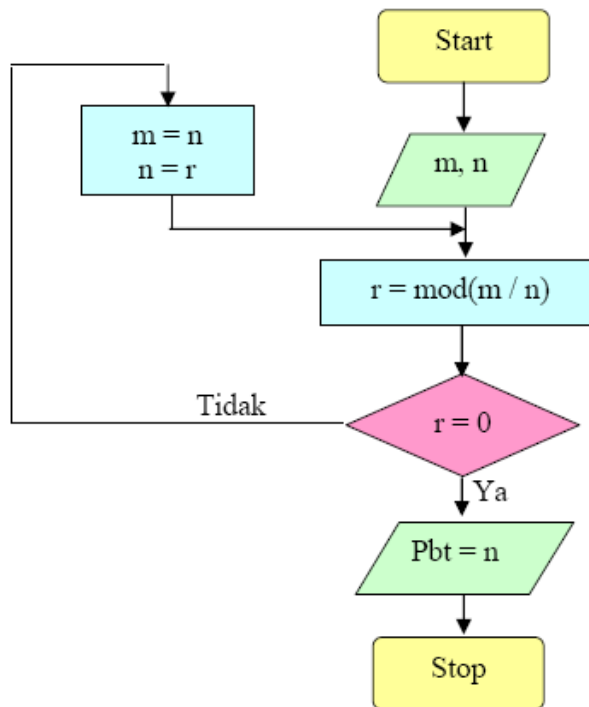
Jika kondisi “hari hujan” bernilai benar, maka aksi “pergilah dengan naik beca” dilakukan, sebaliknya aksi “pergilah dengan naik motor” akan dilakukan jika “hari hujan” tidak benar.

Contoh : *Algoritma Euclidean*

Yaitu proses untuk menentukan pembagi bersama terbesar dari dua bilangan bulat.

Ada dua bilangan bulat  $m$  dan  $n$  ( $m \geq n$ ). Carilah pembagi terbesar (pbt) dari kedua bilangan tersebut, yaitu bilangan positif terbesar yang habis dibagi  $m$  dan  $n$ .

Berikut dengan dengan Flow-Chart :



Gambar 2.3. Algoritma *Euclidean*

### Dengan Pseudo-Code

#### Deskripsi

- 1) Bagilah  $m$  dengan  $n$ , misalkan  $r$  adalah sisanya
- 2) Jika  $r = 0$ , maka  $n$  adalah jawabannya. **Stop**  
Jika  $r \neq 0$ , lakukan langkah 3
- 3) Ganti nilai  $m$  dengan nilai  $n$ , dan nilai  $n$  diganti dengan nilai  $r$ , ulangi langkah 1

Contoh dengan angka:

$m = 30; n = 12$

- Hitung  $r = \text{sis}(m/n)$   $r = \text{sis}(30/12) = 6$
- Check  $r, r \neq 0$ , lakukan langkah 3
- $m = n; n = r$   $m = 12; n = 6$
- Hitung  $r = \text{sis}(m/n)$   $r = \text{sis}(12/6) = 0$
- Check  $r; r = 0$ ; selesai  $Pbt = n = 6$

Jadi  $Pbt(30,12) = 6$

Ada dua bentuk instruksi pemilihan yang sering digunakan yaitu:

- a. Instruksi if / then / else
- b. Instruksi case
  - **Instruksi if / then / else**

**bentuk 1 kasus:**

```
if (syarat)
then aksi
endif
```

contoh :

```
if ( x > 100 )
then x ← x + 1
endif.
```

**bentuk 2 kasus:**

```
if ( syarat )
then aksi-1
else aksi-2
endif.
```

contoh :

```
if ( a > 0 )
then write ("bilangan ini positif ")
else write ("bilangan ini negatif ")
endif.

bentuk bersusun (lebih dari 1 syarat) :
if ( syarat-1 )
then aksi-1
else if ( syarat-2 )
then aksi-2
```



```
else aksi-3
endif
endif.
```

Contoh :

#### Algoritma Pemilihan\_1

```
{ contoh algoritma apakah bilangan genap atau bilangan ganjil }
Deklarasi
integer bilangan;
Deskripsi
write ("masukkan satu bilangan bulat : ");
read (bilangan);
if ( bilangan % 2 == 0 )
then write ( "bilangan genap ! ");
else write ( "bilangan ganjil ! ");
endif
```

#### Algoritma Pemilihan\_2

```
{ contoh algoritma ini menerima 2 bilangan bulat kemudian menetapkan
bilangan yang terbesar }
Deklarasi
integer A, B, C, maks;
Deskripsi
write ("masukkan bilangan 1 : ");
read ( A );
write ("masukkan bilangan 2 : ");
read ( B );
if ( A > B )
then maks ← A;
```

```
else maks ← B;  
endif.  
write ( "maksimum = ", maks);
```

### Algoritma Pemilihan\_3

{ contoh algoritma ini menerima 3 bilangan bulat kemudian menetapkan bilangan yang terbesar, memanfaatkan bentuk bersusun }

Deklarasi

integer A, B, C, maks;

Deskripsi

```
write ("masukkan bilangan 1 : ");  
read ( A );  
write ("masukkan bilangan 2 : ");  
read ( B );  
write ("masukkan bilangan 3 : ");  
read ( C );  
if ( A > B )  
then if ( A > C )  
then write ( " maksimum = ", A );  
else write ( " maksimum = ", C );  
endif  
else if ( B > C )  
then write ( " maksimum = ", B );  
else write ( " maksimum = ", C );  
endif  
endif
```

#### ▪ Instruksi case

Instruksi **case** digunakan apabila setiap aksi dilaksanakan berdasarkan satu macam nilai dari suatu variable yang mungkin memiliki lebih dari dua macam nilai.

##### Bentuk instruksi case :

```
case ( variable )  
nilai-1 : aksi-1;  
nilai-2 : aksi-2;  
nilai-3 : aksi-3;  
.....  
default : aksi-n;  
endcase.
```

##### Contoh :

Gaji karyawan pada sebuah perusahaan di-dasarkan pada jam-kerja dalam satu bulan serta posisi atau golongannya dalam perusahaan itu. Upah perjam menurut golongan adalah sebagai berikut:

Golongan	Upah/jam (Rp)
A	5000
B	6000
C	7500
D	9000

Apabila karyawan bekerja lebih dari 150 jam perminggu, maka kelebihan jam kerja tersebut dihitung sebagai lembur dengan upah/jam 25% diatas upah reguler. Buat sebuah algoritma yang menerima nama, golongan, serta jam-kerja karyawan, kemudian menampilkan gaji total-nya dalam satu bulan.

##### Algoritma Gaji\_Karyawan

{ algoritma yang menerima nama, golongan serta jam-kerja kemudian

menampilkan total gaji yang diterima karyawan }

#### **Deklarasi**

```
real gaji, total, jamkerja, lembur, upah;
```

```
string nama;
```

```
char golongan;
```

#### **Deskripsi**

```
write (" masukkan nama karyawan : ");
```

```
read ( nama );
```

```
write (" masukkan golongan-nya : ");
```

```
read ( golongan );
```

```
write (" masukkan jam kerjanya : ");
```

```
read ( jamkerja );
```

```
case ( golongan )
```

```
  ` A ` : upah ← 5000;
```

```
  ` B ` : upah ← 6000;
```

```
  ` C ` : upah ← 7500;
```

```
  ` D ` : upah ← 9000;
```

```
default : write (" golongannya salah ! ");
```

```
endcase.
```

```
if ( jamkerja > 150 )
```

```
then lembur ← ( jamkerja - 150 ) * upah * 1.25;
```

```
gaji ← 150 * upah;
```

```
else lembur ← 0;
```

```
gaji ← jamkerja * upah;
```

```
endif
```

```
total ← gaji + lembur;
```

```
write (" Gaji yang diterima sdr : ", nama, " adalah  
= Rp. ", total);
```

### 4.3. Instruksi Pengulangan (*Repetition*)

Instruksi pengulangan adalah instruksi yang dapat mengulangi pelaksanaan sederetan instruksi-instruksi lainnya berulang-kali sesuai dengan persyaratan yang ditetapkan.

Struktur instruksi pengulangan pada dasarnya terdiri atas :

- a. Kondisi perulangan : suatu kondisi yang harus dipenuhi agar perulangan dapat terjadi.
- b. Badan (*body*) perulangan : deretan instruksi yang akan diulang-ulang pelaksanaan-nya.
- c. Pencacah (*counter*) perulangan : suatu variable yang nilainya harus berubah agar perulangan dapat terjadi dan pada akhirnya membatasi jumlah perulangan yang dapat dilaksanakan.

Bentuk instruksi perulangan adalah :

- Perulangan : **while – do**
- Perulangan : **repeat – until**
- Perulangan : **for**
- Perulangan **while – do** :

Bentuk umum :

```
while (kondisi) do  
.....  
instruksi-instruksi  
.....  
endwhile.
```

**Maknanya :**

ulangi .. instruksi-instruksi .. selama kondisi masih terpenuhi.

perlu perhatian :

- ada instruksi yang berkaitan dengan kondisi sebelum while/do

- ada satu instruksi diantara instruksi-instruksi yang diulang untuk membatasi jumlah perulangan.

Contoh:

Algoritma untuk menampilkan angka 1 hingga 100

#### Algoritma Perulangan\_1

```
{ mencetak angka 1 hingga 100 }

Deklarasi
integer angka;

Deskripsi
angka ← 1;
while ( angka < 101 ) do
write ( angka );
angka ← angka + 1;
endwhile.
```

Contoh:

Mencetak syair “anak ayam yang mati” mulai dari 10 hingga habis

#### Algoritma Perulangan\_2

```
{ mencetak syair anak ayam }

Deklarasi
integer anak;

Deskripsi
anak ← 10;
while ( anak > 0 ) do
write ( "anak ayamku turun ", anak);
anak ← anak - 1;
if ( anak > 0 )
then write ( "mati satu tinggal ",
anak);
else write ( "mati satu tinggal saya
```

```
) ;  
endif.  
endwhile.
```

▪ **Perulangan Repeat – Until:**

**Bentuk Umum :**

```
Repeat  
.....  
instruksi - instruksi  
.....  
until ( kondisi ).
```

**maknanya :**

ulangi pelaksanaan instruksi-instruksi hingga kondisi terpenuhi.

perhatian :

- Apabila kondisi tidak terpenuhi maka instruksi-instruksi akan diulang
- Instruksi-instruksi akan dikerjakan sebelum kondisi diperiksa

Contoh :

menampilkan “Halo ... “ sebanyak 25 kali.

**Algoritma Perulangan\_3**

{ memakai repeat-until untuk menampilkan Halo sebanyak 25 kali }

**Deklarasi**

**integer** cacah;

**Deskripsi**

cacah  $\leftarrow$  1;

**repeat**

**write** ( “Halo ... “ );

cacah  $\leftarrow$  cacah + 1;

**until** ( cacah > 25 ).

Contoh :

gunakan repeat-until untuk menghitung jumlah angka  $1 + 2 + 3 + \dots + N$ ,  
dimana N adalah angka bulat yang dimasukkan lewat keyboard.

#### Algoritma Perulangan\_4

{ menghitung jumlah  $1 + 2 + 3 + \dots + N$ , N dimasukkan lewat keyboard }

##### Deklarasi

**integer** cacah, N, Jumlah;

##### Deskripsi

**write** ( "Masukkan nilai N : " );

**read** ( N );

cacah  $\leftarrow$  1;

Jumlah  $\leftarrow$  0;

##### repeat

Jumlah  $\leftarrow$  Jumlah + cacah;

cacah  $\leftarrow$  cacah + 1;

**until** ( cacah > N ).

**write** ( "Jumlahnya = ", Jumlah );

Contoh :

gunakan repeat-until untuk menghitung rata-rata dari N buah bilangan yang  
dimasukkan lewat keyboard.

#### Algoritma Perulangan\_5

{ menghitung rata-rata dari N buah bilangan }

##### Deklarasi

**integer** bilangan, cacah, N, Jumlah;

**real** Rata;

##### Deskripsi

**write** ( "Masukkan N : " );



```

read ( N );
cacah  $\leftarrow$  1;
Jumlah  $\leftarrow$  0;
repeat
write ("masukkan bilangan ke-", cacah);
read ( bilangan );
Jumlah  $\leftarrow$  Jumlah + bilangan;
cacah  $\leftarrow$  cacah + 1;
until ( cacah > N ).
Rata  $\leftarrow$  Jumlah / N;
write ( "rata-rata = ", Rata );

```

▪ **Perulangan for:**

Bentuk umum:

```

for ( var = awal to akhir step n)
.....
instruksi - instruksi
.....
endfor.

```

**Maknanya :**

ulangi instruksi-instruksi tersebut berdasarkan variabel perulangan mulai dari nilai awal hingga nilai akhir dengan perubahan nilai sebesar n.

perhatian :

- variabel perulangan (var) harus bertipe dasar (integer, real, atau char)
- nilai *awal* harus lebih kecil dari *akhir* bila  $n > 0$  (positif)
- nilai *awal* harus lebih besar dari *akhir* bila  $n < 0$  (negatif)
- mula-mula variabel var bernilai awal, kemudian setiap satu kali putaran maka nilai var bertambah sebesar n
- perulangan akan berhenti apabila nilai var sudah mencapai akhir

Contoh :

Menampilkan “Halo ... “ sebanyak 10 kali

#### Algoritma Perulangan\_6

{ menampilkan Halo ... memakai instruksi **for** }

##### Deklarasi

**integer** cacah;

##### Deskripsi

**for** ( cacah = 1 **to** 10 **step** 1)

**write** ( “Halo ... “);

**endfor**.

contoh :

Menghitung nilai rata dari N buah bilangan, menggunakan instruksi **for**.

#### Algoritma Perulangan\_7

{ menghitung nilai Rata dari N buah bilangan }

##### Deklarasi

**integer** cacah, N, angka, Jumlah;

**real** Rata;

##### Deskripsi

**write** ( “Masukkan berapa bilangan : “);

**read** ( N );

Jumlah  $\leftarrow$  0;

**for** ( cacah = 1 **to** N **step** 1 )

**write** ( “Masukkan bilangan ke - “, cacah);

**read** ( angka );

Jumlah  $\leftarrow$  Jumlah + angka;

**endfor**.

Rata  $\leftarrow$  Jumlah / N;

**write** ( “Rata-rata = “, Rata);

contoh :

Melakukan pencacahan mundur mulai dari 100, 99, 98, ... hingga 0

#### **Algoritma Perulangan\_8**

{ mencacah terbalik atau count down }

##### **Deklarasi**

**integer** cacah;

##### **Deskripsi**

**for** ( cacah = 100 **to** 0 **step** -1)

**write** ( cacah );

**endfor.**

**write** ( “Go !” );

**Latihan Tugas :**

1. Definisikan sebuah record untuk data pegawai yang terdiri atas elemen Nomer\_Pegawai (NIP), Nama, TgLahir, Tempat\_Lahir, Jen\_Kelamin, Agama, Status, Pangkat.
2. Tulis algoritma sederhana untuk membaca dan menampilkan kembali data pegawai yang sesuai dengan struktur record pada soal 1.
3. Buatlah algoritma untuk menghitung luas dan keliling lingkaran berdasarkan jari-jarinya.
4. Buatlah algoritma yang mengkonversi suhu dalam fahrenheit ke derajat celsius.
5. Buatlah algoritma menghitung jumlah uang yang harus dibayarkan pembeli berdasarkan beratnya buah jeruk yang dibeli. Diketahui bahwa harga barang per kg adalah 500 rupiah/100 gram. Diketahui pula pembeli berhak mendapatkan diskon sebesar 5%. Hasil keluaran yang diinginkan adalah total harga sebelum diskon, diskon, dan total harga setelah diskon.
6. Buatlah algoritma pembulatan seperti pada algoritma nomor 6. Tetapi pembulatan yang diinginkan adalah dalam ribuan. Sebagai contoh, apabila harga barang 7.423.525 rupiah, maka dibulatkan menjadi 7.424.000 rupiah.
7. Buatlah algoritma yang meminta input sebuah bilangan bulat, kemudian algoritma akan memberikan keluaran (output) berupa angka satuan, puluhan, dan ratusannya. Contoh, apabila diinputkan bilangan 1.234.567 (1 juta 234 ribu 567), maka algoritma akan menghasilkan output bilangan 7 satuan, 6 puluhan, dan 5 ratusan. Apabila diinputkan bilangan 73 (tujuh puluh tiga) maka algoritma akan menghasilkan output bilangan 3 satuan, 7 puluhan, dan 0 ratusan.

# BAB V

## STUDI KASUS

(Case Study)

Setelah mempelajari beberapa instruksi utama maka pada dasarnya modal untuk merancang algoritma sederhana sudah memadai, oleh sebab itu pada bagian ini akan disajikan beberapa contoh soal yang dapat dijadikan sebagai studi kasus bagi mahasiswa yang mempelajari algoritma.

### 5.1. Akses Data Langsung

Seorang sekretaris memerlukan satu program sederhana yang dapat membantu dia untuk mengetahui nomer telepon seseorang dengan cepat tanpa harus membuka-buka buku agendanya.

Andaikan nama-nama orang tersebut adalah sebagai berikut:

- Anton (0411) 324-678
- Bahrul (021) 434-6783
- Charles (022) 256-1234
- Daud (0411) 567-342
- Endang (0411) 344-235

#### Analisis:

1. Ketika program dijalankan maka muncul permintaan untuk memasukkan satu nama
2. Nama ini kemudian dicari misalnya dengan rentetan **if / then / else** atau dengan instruksi **case()**.
3. Bila nama tersebut ketemu maka nomer-teleponnya ditampilkan.
4. Bila nama tersebut tidak ada maka tampilkan “nama tsb tidak ada!”.

#### Algoritma Buku\_telepon

{ mencari nomer telepon seseorang }

##### Deklarasi

**string** nama, notelp;

##### Deskripsi

**write** ( "Ketik namanya : ");

**read** ( nama );

**case** (nama)

'Anton' : notelp  $\leftarrow$  '(0411) 324 - 678';

'Bahrul' : notelp  $\leftarrow$  '(021) 434 - 6783';

'Charles' : notelp  $\leftarrow$  '(022) 256-1234';

'Daud' : notelp  $\leftarrow$  '(0411) 567-342';

'Endang' : notelp  $\leftarrow$  '(0411) 344-235';

**default** : notelp  $\leftarrow$  'nama tsb tdk ada';

**endcase.**

**write** ( notelp );

#### 5.2. Menjumlahkan Deret

Buatlah sebuah algoritma untuk menghitung jumlah deret dengan N buah suku :

$$S = 1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{6} + \frac{1}{8} - \frac{1}{10} + \frac{1}{12} - \frac{1}{14} + \dots$$

##### Analisis:

1. Ketika dijalankan maka akan ada permintaan untuk memasukkan jumlah suku N
2. Bila diperhatikan maka tanda berselang seling positif dan negatif, pada posisi ganjil maka tandanya positif dan pada posisi genap tandanya negatif
3. Nilai yang dijumlahkan adalah kelipatan dari (1/2) yang dikalikan sesuai dengan posisi-nya, mula-mula 1/2 kemudian 1/(2\*2), 1/(2\*3), ....

#### Algoritma Jumlah\_Deret

{ menjumlahkan deret bersuku N }

Deklarasi

**integer** N, cacah, k;

**real** S;

Deskripsi

```
write ( "Berapa banyak suku ? ");
read ( N );
S  $\leftarrow$  1;
cacah  $\leftarrow$  1;
k  $\leftarrow$  0;
while ( cacah  $\leq$  N ) do
  cacah  $\leftarrow$  cacah + 1;
  k  $\leftarrow$  k + 2;
  if ( cacah % 2 = 0 )
    then S  $\leftarrow$  S - 1/k;
  else S  $\leftarrow$  S + 1/k;
  endif.
endwhile.
write ( "Jumlah deret = ", S );
```

#### **Algoritma Jumlah\_Deret\_V2**

{ cara lain untuk menghitung jumlah deret }

Deklarasi

integer cacah, N, k, tanda;

real S;

Deskripsi

```
write ( "Berapa banyak suku ? ");
read ( N );
S  $\leftarrow$  1;
cacah  $\leftarrow$  1;
k  $\leftarrow$  0;
tanda  $\leftarrow$  +1;
while ( cacah  $\leq$  N ) do
  k  $\leftarrow$  k + 2;
  cacah  $\leftarrow$  cacah + 1;
  tanda  $\leftarrow$  (-1) * tanda;
  S  $\leftarrow$  S + tanda * (1/k);
```

```
endwhile.  
write ( "Jumlah deret = ", S);
```

### 5.3. Mengelompokkan Data

Andaikan dari keyboard dimasukkan N buah data (bilangan bulat) kemudian akan dikelompokkan menjadi dua macam yaitu kelompok bilangan ganjil dan kelompok bilangan genap dalam bentuk jumlahan sehingga keluaran (output) berbentuk sebagai berikut:

Jumlah bilangan Ganjil = ....

Jumlah bilangan Genap = ....

#### Analisis:

1. Berapa banyak data harus diketahui terlebih dahulu  $\leftarrow N$
2. Lakukan perulangan sebanyak N kali untuk:
  - a. meminta data
  - b. memeriksa data apakah masuk ganjil atau genap
  - c. menjumlahkan data sesuai kelompoknya
3. Tampilkan hasil penjumlahan.

#### Algoritma GanjilGenap

{ mengelompokkan data dalam bentuk jumlahan bilangan ganjil dan genap }  
Deklarasi

integer cacah, N, angka, Genap, Ganjil;

Deskripsi

write ( "Berapa banyak bilangan ? " );

read ( N );

cacah  $\leftarrow$  1;

Genap  $\leftarrow$  0;

Ganjil  $\leftarrow$  0;

repeat

write ( "Masukkan bilangan ke-", cacah );

read ( angka );



```

if ( angka % 2 = 0 )
then Genap ← Genap + angka;
else Ganjil ← Ganjil + angka;
endif.
cacah ← cacah + 1;
until ( cacah > N );
write ( "Jumlah bilangan Ganjil = ", Ganjil);
write ( "Jumlah bilangan Genap = ", Genap);

```

#### 5.4. Memilih Operasi Berdasarkan Data Input

Andaikan operasi terhadap dua bilangan dapat dipilih melalui satu “menu” sebagai berikut :

Pilih Operasi yang di-inginkan :

+ Penjumlahan

- Pengurangan

/ Pembagian

\* Perkalian

Jenis operasi: \_

Masukkan angka 1: \_

Masukkan angka 2: \_

Hasil = ...

Masih mau coba (Y/T) ?

Apabila jawaban untuk mencoba ulang adalah ‘Y’ maka menu operasi diatas dimunculkan kembali dan proses yang sama berulang kembali hingga jawaban pengguna program adalah ‘T’.

#### Algoritma MenuProgram

{ memilih operasi berdasarkan pilihan pada Menu program }

**Deklarasi**

**real** angka1, angka2, hasil;

**char** pilihan, ulang;

**Deskripsi**

```
ulang ← 'Y';
while ( ulang = 'Y' || ulang = 'y' ) do
write ( "Pilih Operasi yang di-inginkan :" );
write ( "+ Penjumlahan " );
write ( " - Pengurangan ");
write ( " / Pembagian " );
write ( "* Perkalian " );
write ( " ");
write ( "Jenis operasi: ");
read ( pilihan );
write ( "Masukkan angka 1: " );
read ( angka1 );
write ( "Masukkan angka 2: " );
read ( angka2 );
case ( pilihan )
' + ' : hasil ← angka1 + angka2;
' - ' : hasil ← angka1 - angka2;
' / ' : if ( angka2 = 0 )
then write ( "hasil tak berhingga " );
else hasil ← angka1 / angka2;
endif.
'*' : hasil ← angka1 * angka2;
default : write ( "Pilihan operasi salah !");
hasil ← 0;
endcase.
if ( angka2 != 0 )
then write ( "Hasil = ", hasil );
endif.
write ( "Masih mau coba (Y/T) ? " );
read ( ulang );
endwhile.
```

### 5.5. Pemilihan 2 Kasus

Kadang terdapat 2 proses yang saling berlawanan, dan tidak boleh dikerjakan dalam satu saat. Hal inilah yang disebut pemilihan untuk 2 kasus. Proses dipilih berdasarkan syarat atau kondisi, bila memenuhi maka dikerjakan proses pertama, jika tidak memenuhi maka dikerjakan proses kedua. Sebagai contohnya, misalnya dalam menentukan apakah suatu bilangan bulat yang diinputkan user adalah bilangan genap atau bilangan ganjil. Apabila yang diinputkan adalah bilangan genap, maka dicetak keterangan yang menyatakan bilangan genap. Sebaliknya jika bukan bilangan genap, pastilah bilangan tersebut merupakan bilangan ganjil.

#### a. Algoritma Ganjil Genap

Buatlah algoritma untuk mencetak kata “GENAP” apabila bilangan bulat yang diinputkan user adalah bilangan genap, dan mencetak kata “GANJIL” apabila bilangan bulat yang diinputkan user adalah bilangan ganjil. Analisis: input: suatu bilangan bulat (n) Kita tahu bahwa suatu bilangan bulat apabila bukan genap, pastilah merupakan bilangan ganjil. Jadi, kasus ganjil dan genap merupakan sesuatu yang berlawanan. Hal ini tergolong ke dalam struktur pemilihan 2 kasus. Syarat suatu bilangan dikatakan genap adalah bilangan tersebut **habis dibagi 2**. Dengan kata lain, bilangan n dibagi dengan 2 bersisa 0. Operator untuk menghitung sisa hasil bagi adalah MOD. Jadi syarat bahwa n bilangan genap adalah  $n \bmod 2 = 0$ .

01	<b>Algoritma Menentukan_Genap_Ganjil</b>
02	{Mencetak “Genap” bila bilangan yang diinput user adalah genap, dan “Ganjil” bila ganjil}
03	Deklarasi:
04	n : integer
05	Deskripsi
06	read(n)
07	if $n \bmod 2 = 0$ then

```

08| write(„GENAP“)
09| else
10| write(„GANJIL“)
11| end if

```

Algoritma di atas dapat juga ditulis menjadi

```

01| Algoritma Menentukan_Genap_Ganjil
02| {Mencetak “Genap” bila bilangan yang diinput user adalah genap, dan
   “Ganjil” bila ganjil}
03| Deklarasi:
04| n : integer
05| genap : boolean
06| Deskripsi
07| read(n)
08| genap ← n mod 2 = 0
09| if genap = TRUE then
10| write(„Genap“)
11| else
12| write(„Ganjil“)
13| end if

```

Perhatikan baris ke-08. Kondisi atau syarat genap ganjilnya bilangan  $n$  dapat kita tampung di dalam suatu variabel bertipe boolean, dalam hal ini kita namakan *genap*. Misalnya user menginput bilangan 4. Maka  $n$  berisikan bilangan bulat 4. Diperoleh  $4 \bmod 2 = 0$  adalah bernilai TRUE. Kemudian nilai TRUE tersebut akan diisikan ke variabel *genap*. Selanjutnya dalam baris ke-09, isi variabel *genap* dibandingkan dengan nilai boolean TRUE. Karena  $\text{TRUE} = \text{TRUE}$  adalah TRUE, maka algoritma menghasilkan output “GENAP”.

Perhatikan tabel berikut:

Genap	Genap = TRUE	Not Genap	Genap = FALSE
TRUE	TRUE	FALSE	FALSE
FALSE	FALSE	TRUE	TRUE

Ternyata nilai dari ekspresi  $Genap = TRUE$  adalah sama dengan nilai dari variabel *genap*. Berdasarkan pengamatan ini, maka baris ke-09 pada algoritma di atas dapat diganti menjadi *if genap then*.

```
...
if genap then
...
```

Algoritma di atas juga dapat ditulis dari sudut pandang GANJIL, yaitu syaratnya diganti menjadi  $genap = FALSE$ . Maka

```
...
if genap = FALSE then
write('GANJIL')
else
write('GENAP') end if
```

dapat diganti menjadi

```
...
if not genap then
...
```

#### b. Algoritma Terbesar dari 2 Bilangan

Diinputkan dua buah bilangan bulat dari user. Buatlah algoritma untuk mencetak bilangan yang terbesar dari dua bilangan tersebut. Analisis: Input: 2 bilangan bulat ( $a$  dan  $b$ ) Apabila  $a > b$ , maka yang dicetak adalah  $a$ . Apabila  $a < b$ , maka yang dicetak adalah  $b$ . Tetapi apabila  $a = b$ , maka kita dapat mencetak  $a$  atau  $b$ . Oleh karena itu, kita dapat menyimpulkan bahwa kita mencetak  $a$  apabila  $a \geq b$ , dan mencetak  $b$  apabila terjadi sebaliknya.

01	Algoritma
Menentukan_Terbesar_dari_2_Bilangan_Bulat	
02	{Diinput 2 bilangan bulat. Kemudian mencetak yang terbesar}
03	Deklarasi:
04	a, b, terbesar : integer
05	Deskripsi
06	read(a,b)
07	if a >= b then
08	terbesar ← a
09	else
10	terbesar ← b
11	end if
12	write(terbesar)

### c. Algoritma Menghitung Uang Lembur

Karyawan PT “ABC” digaji berdasarkan jumlah jam kerjanya selama satu minggu. Upah per jam adalah Rp2.000,00. Bila jumlah jam kerja lebih besar dari 48 jam, maka sisanya dianggap sebagai jam lembur. Upah lembur adalah Rp3.000,00. Buatlah algoritma untuk menampilkan upah normal, uang lembur, dan total upah yang diterima karyawan.

Analisis: input: jumlah jam kerja (n) output: upah normal (upah), uang lembur (lembur), dan total upah (total) Upah per jam, upah lembur, dan batas jam lembur dapat dijadikan sebagai konstanta.

Kasus ini memberikan kita dua kemungkinan, yaitu apakah karyawan menerima lembur atau tidak menerima lembur (lembur = 0). Syarat seorang karyawan menerima lembur adalah apabila  $n > 48$ . Uang lembur yang diterima adalah selisih jam kerja dengan batas jam lembur dikalikan dengan upah lembur. Oleh karena itu, dapat kita simpulkan bahwa rumus yang dipakai adalah Apabila karyawan tidak mendapat uang lembur, maka  $\text{lembur} = 0$ ,  $\text{upah} = n \times 2000$

Apabila karyawan mendapatkan uang lembur, maka  $\text{lembur} = (n - 48) \times 3000$ , sedangkan  $\text{upah} = 48 \times 2000$ , **bukan upah =  $n \times 2000$** .

#### 01| Algoritma Upah\_Karyawan

```
02| {menentukan upah mingguan karyawan. Upah normal
    Rp2000,-/jam dan upah lembur Rp3000,-/jam.
    Apabila jam kerja karyawan lebih dari 48, maka
    sisanya dihitung lembur. Algoritma menghasilkan
    output jumlah upah normal, jumlah uang lembur,
    dan total upah yang diterima karyawan}
03| Deklarasi:
04| const upah_per_jam = 2000
05| const upah_lembur = 3000
06| const batas_lembur = 48
07| n, upah, lembur, total : integer
08| Deskripsi:
09| read(n)
10| if n > 48 then {menerima lembur}
11| upah ← batas_lembur * upah_per_jam
12| lembur ← (n - batas_lembur) * upah_lembur
13| else {tidak menerima lembur}
14| upah ← n * upah_per_jam
15| lembur ← 0 {penting}
16| end if
17| total ← upah + lembur
18| write(upah, lembur, total)
```

#### d. Algoritma Tahun Kabisat

Tahun kabisat adalah tahun yang memenuhi syarat berikut:

- (a) kelipatan 4;
- (b) bukan kelipatan 100;
- (c) kelipatan 400.

Jadi, perhatikan tabel berikut:

Tahun	Kabisat?
1996	Ya
1900	Bukan
2000	Ya

Berdasarkan keterangan di atas, buatlah algoritma untuk menentukan apakah suatu tahun merupakan tahun kabisat atau bukan. Analisis: input: tahun (n) Perhatikan syarat tahun kabisat bagian (a) dan (b). Dari 2 syarat tersebut dapat kita simpulkan bahwa syaratnya adalah kelipatan 4 **dan bukan** kelipatan 100, **atau** kelipatan 400. Dengan demikian, maka syaratnya yang lengkap adalah  $(n \bmod 4 = 0)$  and  $(n \bmod 100 \neq 0)$  or  $(n \bmod 400 = 0)$ .

Percobaan:

Tahun (n)	$n \bmod 4 = 0$	$n \bmod 100 \neq 0$	$n \bmod 400 = 0$	Hasil
1996	TRUE	TRUE	FALSE	TRUE
1900	TRUE	FALSE	FALSE	FALSE
2000	TRUE	FALSE	TRUE	TRUE

#### 01| Algoritma Menentukan\_Kabisat

```

02| { menentukan kabisat atau tidak dari suatu tahun }
03| Deklarasi:
04| n : integer
05| kabisat : boolean
06| Deskripsi
07| read(n)
08| kabisat  $\leftarrow (n \bmod 4 = 0)$  and  $(n \bmod 100 \neq 0)$  or  $(n \bmod 400 = 0)$ 
09| if kabisat then
10| write(„KABISAT“)
11| else
12| write(„BUKAN KABISAT“)
13| end if

```



#### e. Algoritma Syarat Keanggotaan Suatu Jangkauan/Range

Diketahui bahwa himpunan penyelesaian suatu pertidaksamaan adalah  $XP = x \mid -3 \leq x \leq 3$ . Diinputkan suatu bilangan  $x$ . Buatlah algoritma untuk mencetak “Penyelesaian” bila  $x \in HP$  dan mencetak “Bukan Penyelesaian” bila  $x \notin HP$ .

##### Analisis:

Persyaratan  $-3 \leq x \leq 3$  tidak memenuhi bentuk penulisan eksresi perbandingan yang benar. Maka persyaratan tersebut harus dipisah menjadi 2 bagian, yaitu  $x \geq -3$  dan  $x \leq 3$ .

Kedua persyaratan tersebut harus digabung dengan menggunakan operator Boolean **AND** (Mengapa?). Oleh karena itu, persyaratan keanggotaan  $x$  secara lengkap adalah :  $(x \geq -3) \text{ AND } (x \leq 3)$ .

01  Algoritma Menentukan_Keanggotaan_range
02  {Diinput suatu bilangan $x$ . Algoritma menghasilkan TRUE apabila $x$ adalah anggota HP = $\{x \mid -3 \leq x \leq 3\}$ dan mencetak FALSE bila bukan}
03  Deklarasi:
04  $x$ : real
05  syarat : boolean
06  Deskripsi
07  read( $x$ )
08  syarat $\leftarrow (x \geq -3) \text{ and } (x \leq 3)$
09  if syarat then
10  write(„Penyelesaian“)
11  else
12  write(„Bukan Penyelesaian“)
13  end if

## 5.6. Pemilihan Lebih dari 2 Kasus

Terkadang kita harus memilih lebih dari 2 pilihan yang ada. Misalnya menentukan apakah suatu bilangan bulat yang diinputkan user adalah bilangan positif, negatif, atau nol. Contoh lainnya adalah dalam menentukan gaji pokok karyawan berdasarkan golongan (kepangkatan).

Apabila terdapat lebih dari 2 pilihan, maka kita dapat memandang kasus tersebut sebagai kasus 2 pilihan, yaitu YA atau BUKAN. Sebagai contoh, kasus menentukan positif, nol, atau negatif; dapat dipandang sebagai kasus dengan 2 pilihan saja, yaitu apakah bilangan tersebut adalah bilangan positif atau bukan. Apabila bukan, maka bilangan tersebut hanya dimungkinkan merupakan bilangan nol atau negatif. Dalam hal ini (nol atau negatif) merupakan kasus dengan 2 pilihan. Dengan kata lain, kasus lebih dari 2 pilihan dapat dipandang menjadi kasus 2 pilihan yaitu apakah tergolong pilihan pertama atau bukan. Kemudian untuk bagian *bukan* (else), pilihan yang ada telah tereduksi (berkurang) satu. Apabila masih lebih, maka pilihan masih bisa direduksi lagi dengan cara yang sama, yaitu memandang apakah tergolong pilihan kedua atau bukan.

### a. Algoritma Menentukan Positif, Nol, atau Negatif

Buatlah algoritma untuk mencetak kata “Positif”, “Nol”, atau “Negatif” berdasarkan jenis bilangan yang diinput oleh user. Analisis: input : sebuah bilangan (x) Langkah pengerjaan:

- 1) input bilangan x
- 2) tentukan apakah x positif, jika benar maka cetak “Positif”
- 3) Jika tidak, maka tentu x mungkin nol atau negatif.
- 4) Tentukan apakah x sama dengan nol, jika benar maka cetak “Nol”
- 5) jika tidak, maka cetaklah “Negatif”

01  <b>Algoritma Menentukan_Positif_Nol_Negatif</b>
02  {Mencetak positif, nol, atau negatif sesuai dengan jenis bilangan yang diinputkan oleh user}

```

03| Deklarasi:
04| x : real
05| Deskripsi
06| read(x)
07| if x > 0 then
08| write(„Positif“)
09| else
10| if x = 0 then
11| write(„Nol“)
12| else
13| write(„Negatif“)
14| end if
15| end if

```

#### b. Algoritma Gaji Berdasarkan Golongan

Suatu perusahaan menentukan gaji pokok karyawannya menurut golongannya. Besarnya gaji pokok berdasarkan golongannya dapat dilihat pada tabel berikut.

Golongan	Gaji Pokok
A	Rp400.000,00
B	Rp500.000,00
C	Rp750.000,00
D	Rp900.000,00

Buatlah algoritma untuk menentukan gaji pokok berdasarkan golongan yang diinput user. Analisis: input: Golongan (gol) bertipe **char**. Langkah pengerjaan:

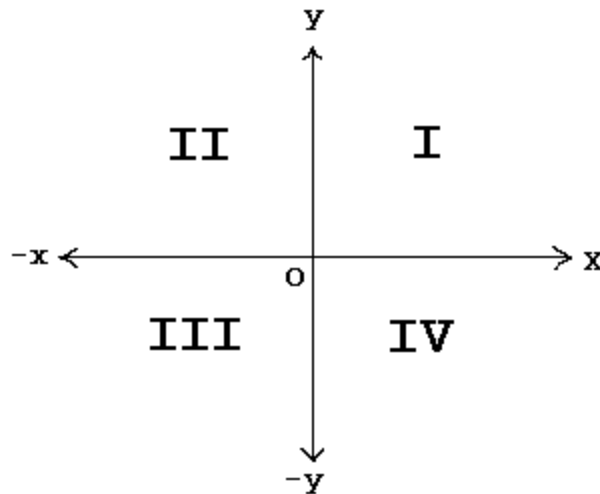
- 1) Anggap kasus terdiri dari 2 pilihan, yaitu bergolongan A atau bukan. Apabila bukan, maka kasus tereduksi menjadi 3 pilihan, yaitu bergolongan B, C, atau D.
- 2) Anggap kasus terdiri dari 2 pilihan, yaitu bergolongan B atau bukan. Apabila bukan, maka kasus tereduksi menjadi 2 pilihan, yaitu bergolongan C atau D.

### 01| Algoritma Penentuan\_Gaji\_Pokok

```
02| {Menentukan gaji pokok berdasarkan golongan..}  
03| Deklarasi:  
04| gol : char  
05| gaji : integer  
06| Deskripsi  
07| read(gol)  
08| if gol = „A“ then  
09| gaji ← 400000  
10| else  
11| if gol = „B“ then  
12| gaji ← 500000  
13| else  
14| if gol = „C“ then  
15| gaji ← 750000  
16| else  
17| gaji ← 900000  
18| end if  
19| end if  
20| end if  
21| write(gaji)
```

### c. Algoritma Penentuan Nomor Kuadran Titik Koordinat

Suatu titik dalam sistem sumbu koordinat dapat digolongkan menjadi kuadran ke-1 hingga kuadran ke-4 berdasarkan letaknya dapat diperhatikan pada gambar berikut.



Ingat, bahwa apabila titik terletak pada sumbu (x maupun y) maka titik tersebut tidak terletak pada kuadran manapun.

Buatlah algoritma untuk menentukan jenis kuadran berdasarkan suatu titik yang diinput oleh user. Analisis: Anda dapat memakai cara sebelumnya, yaitu memandangnya sebagai kasus dengan 5 pilihan, yaitu kuadran ke-1 hingga kuadran ke-4, ditambah “tidak di kuadran”. Berdasarkan gambar di atas, maka syarat untuk masing-masing kuadran adalah :

Kuadran	Syarat
I	$(x > 0) \text{ and } (y > 0)$
II	$(x < 0) \text{ and } (y > 0)$
III	$(x < 0) \text{ and } (y < 0)$
IV	$(x > 0) \text{ and } (y < 0)$

Kemudian dengan teknik reduksi, kasus dengan 5 pilihan tersebut akan tereduksi menjadi 4 pilihan, dan seterusnya.

```

01| Algoritma Penentuan_Kuadran
02| {Menentukan jenis kuadran dari suatu titik
koordinat}
03| Deklarasi:
04| x, y, kuadran : integer
05| Deskripsi
06| read(x,y)
07| if (x>0) and (y>0) then
08| kuadran ← 1
09| else
10| if (x<0) and (y>0) then
11| kuadran ← 2
12| else
13| if (x<0) and (y<0) then
14| kuadran ← 3
15| else
16| if (x>0) and (y<0) then
17| kuadran ← 4
18| else
19| kuadran ← 0
20| end if
21| end if
22| end if
23| end if
24| Write(kuadran)

```

Apabila Anda perhatikan algoritma di atas, terlalu banyak syarat yang berulang. Perhatikan bahwa syarat (x>0) terdapat pada baris ke-07 dan baris ke-16. Hal ini mengisyaratkan bahwa syarat penentuan kuadran tersebut kurang terarah dan

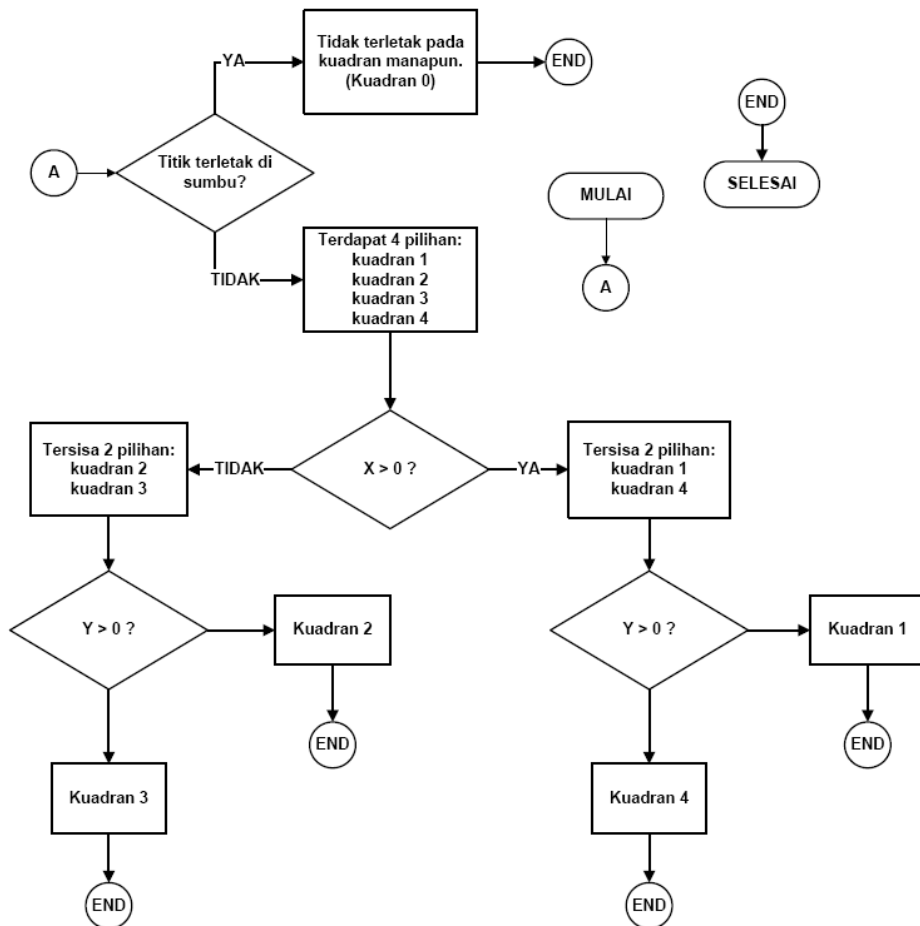
terstruktur. Maka kita dapat mengatur kembali sudut pandang kita terhadap kasus ini. Urutan pengklasifikasian pilihan yang lebih terstruktur untuk kasus ini adalah :

1. Anggap kasus terdiri dari 2 pilihan, yaitu titik terletak di salah satu kuadran, atau titik terletak di sumbu. Apabila terletak di sumbu, maka titik sudah pasti tidak terletak di salah satu kuadran. Syarat suatu titik terletak di salah satu sumbu adalah  $(x = 0)$  or  $(y = 0)$ .
2. Jika tidak terletak di sumbu, maka tinggal 4 pilihan, yaitu terletak di kuadran ke-1, kuadran ke-2, kuadran ke-3, atau kuadran ke-4.
3. Kasus dengan 4 pilihan bisa kita kelompokkan menjadi 2 pilihan saja, yaitu apakah titik terletak di sebelah kanan sumbu-y ( $x > 0$ ) atau terletak di kiri sumbu-y ( $x < 0$ ).
4. Jika terletak di sebelah kanan sumbu-y maka tinggal 2 pilihan saja, yaitu titik terletak di kuadran ke-1 atau di kuadran ke-4, bergantung pada nilai y.
5. Jika tidak (terletak di sebelah kiri sumbu-y), maka tinggal 2 pilihan pula, yaitu titik terletak di kuadran ke-2 atau di kuadran ke-3 bergantung pada nilai y pula.
6. Untuk lebih jelasnya, bisa Anda perhatikan flowchart berikut ini.

```
01| Algoritma Menentukan_Kuadran
02| {Menentukan kuadran dari suatu titik
koordinat}
03| Deklarasi:
04| x, y, kuadran : integer
05| Deskripsi
06| read(x,y)
07| if (x = 0) or (y = 0) then {terletak di salah
satu sumbu}
08| kuadran ← 0
09| else
10| if x > 0 then {di sebelah kanan sumbu-y}
11| if y > 0 then
```

```
12| kuadran ← 1
13| else
14| kuadran ← 4
15| end if
16| else {x < 0, yaitu di sebelah kiri sumbu-y}
17| if y > 0 then
18| kuadran ← 2
19| else
20| kuadran ← 3
21| end if
22| end if
23| end if
24| write(kuadran)
```





Gambar 4.1. Flowchart Kuadran

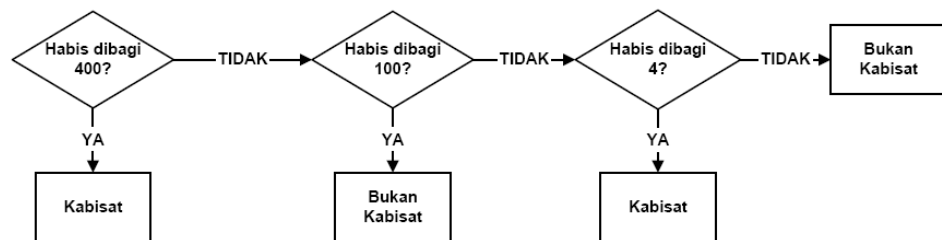
#### d. Algoritma Tahun Kabisat

Mari kita analisa kembali syarat tahun kabisat, yaitu:

- Tahun habis dibagi 400
- Tahun tidak habis dibagi 100
- Tahun habis dibagi 4

Perhatikan, bahwa syarat ke-2 lebih lemah dari syarat pertama, yaitu bahwa walaupun tahun habis dibagi 100, namun apabila tahun habis dibagi 400 maka tahun tersebut merupakan tahun kabisat. Apabila tahun habis dibagi 400 maka tahun jelas habis dibagi 4. Kesimpulan kita adalah bahwa apabila tahun habis dibagi 400, maka

secara pasti tahun tersebut merupakan kabisat. Apabila tidak habis dibagi 400, maka ada 2 kemungkinan, yaitu habis dibagi 100 atau tidak. Bila habis dibagi 100, maka secara pasti tahun tersebut **bukan** kabisat. Apabila tidak habis dibagi 100, maka ada 2 kemungkinan, yaitu habis dibagi 4 atau tidak. Bila habis dibagi 4, maka secara pasti tahun tersebut adalah tahun kabisat. Jika tidak habis dibagi 4, maka secara pasti tahun tersebut **bukan** kabisat. Untuk lebih jelasnya, perhatikanlah flowchart berikut ini.



Gambar. 5.1. Flowchart Algoritma Tahun Kabisat

#### 01| Algoritma Penentuan\_Kabisat

02| {Menentukan suatu tahun yang diinput oleh user merupakan tahun kabisat atau bukan}

03| Deklarasi:

04| n : integer {Tahun}

05| kabisat : boolean {true bila kabisat, false bila bukan}

06| Deskripsi

07| read(n)

08| if n mod 400 = 0 then

09| kabisat ← true

10| else

11| if n mod 100 = 0 then

12| kabisat ← false

13| else

14| if n mod 4 = 0 then

```

15| kabisat ← true
16| else
17| kabisat ← false
18| end if
19| end if
20| end if
21| if kabisat then
22| write(„kabisat“)
23| else
24| write(„bukan kabisat“)
25| end if

```

#### e. Algoritma Menentukan Grade Nilai Akhir

Buatlah algoritma untuk menentukan grade berdasarkan nilai akhir. Grade dapat dilihat pada tabel berikut.

NILAI AKHIR	GRADE
80 – 100	A
70 – <80	B
60 – <70	C
40 – <60	D
0 – <40	E
Selain di atas	K

Analisis:

Kita perhatikan bahwa nilai yang valid dimulai dari 0 hingga 100. Diluar itu, nilai tidak valid dan diberi grade “K”. Maka langkah pertama adalah menganggap kasus dengan 6 pilihan tersebut sebagai kasus dengan 2 pilihan, yaitu apakah nilai valid (terletak pada range 0–100) atau tidak valid. Apabila tidak valid, maka grade bernilai “K”. Jika valid, maka tersisa 5 pilihan grade.

Syarat untuk mendapatkan grade A adalah lebih dari sama dengan 80. **Tidak perlu mengecek apakah nilai kurang dari sama dengan 100** (mengapa?). Bila

tidak, maka tersisa 4 pilihan, yaitu grade B hingga grade E. Dengan teknik reduksi, maka akan tersisa 3 pilihan, dan seterusnya.

**01| Algoritma Penentuan\_Grade**

**02| {Menentukan grade berdasarkan nilai akhir. }**

03| Deklarasi:

04| nilai : real

05| grade : char

06| Deskripsi

07| read(nilai)

08| if not ((nilai>=0) and (nilai<=100)) then {nilai tidak valid}

09| grade ← „K“

10| else

11| if nilai >= 80 then

12| grade ← „A“

13| else

14| if nilai >= 70 then

15| grade ← „B“

16| else

17| if nilai >= 60 then

18| grade ← „C“

19| else

20| if nilai >= 40 then

21| grade ← „D“

22| else

23| grade ← „E“

24| end if

25| end if

26| end if

```

27| end if
28| end if
29| write(grade)

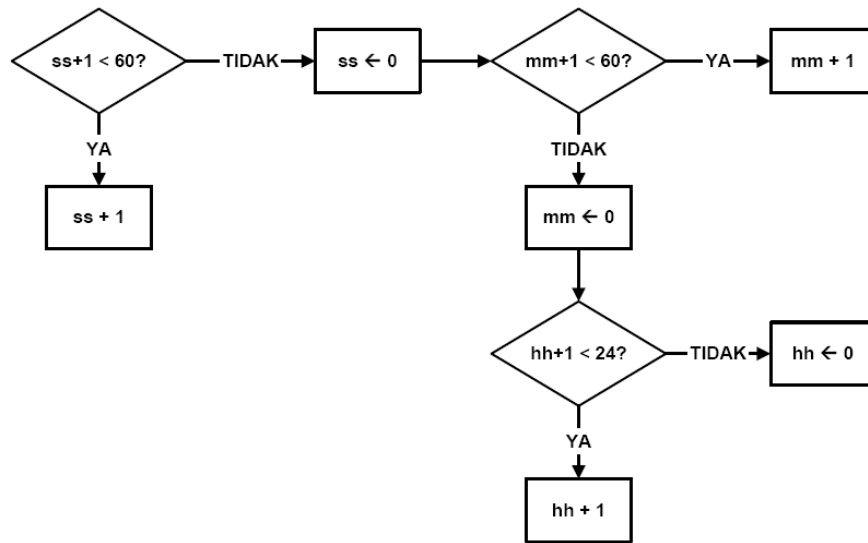
```

#### f. Algoritma Penambahan 1 Detik Pada Waktu

Diberikan suatu data waktu dalam bentuk hh:mm:ss, dengan hh adalah jam, mm adalah menit, dan ss adalah detik. Buatlah algoritma untuk menampilkan data waktu tersebut setelah ditambahkan 1 detik. Analisis: Misalkan data waktunya adalah 02:13:23, maka algoritma menghasilkan 02:13:24. Sepintas terlihat sederhana. Namun ada beberapa data waktu yang rumit bila ditambah 1 detik. Perhatikan tabel berikut

Data Waktu	Setelah penambahan 1 detik
02:13:23	02:13:24
02:13:59	02:14:00
02:59:59	03:00:00
23:59:59	00:00:00

Maka sebelum detik ditambahkan, kita perlu mengecek apakah bisa ditambahkan secara langsung seperti tabel baris ke-1, yaitu dengan syarat  $ss + 1 < 60$ . Jika tidak, maka berarti  $ss$  bernilai 0, kemudian nilai  $mm$  ditambahkan 1. Akan tetapi, sebelum menambahkan 1 ke  $mm$ , kita perlu mengecek apakah  $mm$  bisa ditambahkan secara langsung seperti tabel baris ke-2, yaitu dengan syarat  $mm + 1 < 60$ . Jika tidak, maka berarti  $mm$  bernilai 0, kemudian nilai  $hh$  ditambahkan 1. Akan tetapi, sebelum menambahkan 1 ke  $hh$ , kita perlu mengecek apakah  $hh$  bisa ditambahkan secara langsung seperti tabel baris ke-3, yaitu dengan syarat  $hh + 1 < 24$ . Jika tidak, maka berarti  $hh$  bernilai 0 seperti tabel baris ke-4.



Gambar 5.2 Algoritma untuk menampilkan data waktu setelah ditambahkan 1 detik

01| Algoritma Penambahan\_Waktu\_1\_Detik

02| {Menambah 1 detik ke data waktu dalam hh:mm:ss yang diinputkan oleh user}

03| Deklarasi:

04| hh, mm, ss: integer

05| Deskripsi

06| read(hh,mm,ss)

07| if ss + 1 < 60 then

08| ss ← ss + 1

09| else

10| ss ← 0

11| if mm + 1 < 60 then

12| mm ← mm + 1

13| else

14| mm ← 0

15| if hh + 1 < 24 then

16| hh ← hh + 1

17| else

18| hh ← 0

```

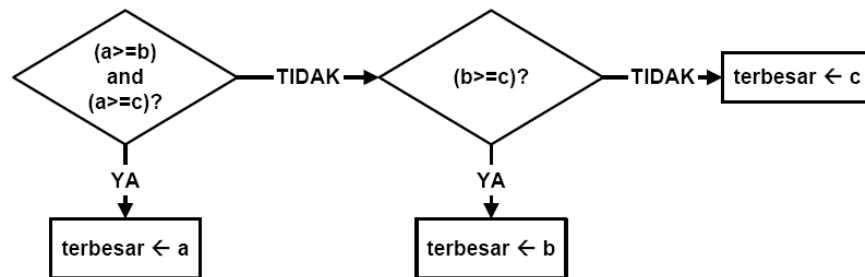
19| end if
20| end if
21| end if
22| write(hh,mm,ss)

```

#### g. Algoritma Terbesar dari 3 Bilangan (Bandingkan Satu-Satu)

Buatlah algoritma untuk menentukan yang terbesar dari 3 bilangan. Analisis: Langkah pengerjaan: Misalkan bilangan tersebut adalah bilangan  $a$ ,  $b$ , dan  $c$ .

- Bandingkan  $a$  dengan  $b$  dan  $c$ . Bila  $a$  lebih besar dari  $b$  maupun  $c$ , maka  $a$  terbesar.
- Jika tidak, maka tersisa 2 kemungkinan,  $b$  terbesar atau  $c$  terbesar.
- Bandingkan  $b$  dengan  $c$ . Bila  $b$  lebih besar dari  $c$ , maka  $b$  terbesar. Jika tidak maka  $c$  yang terbesar.



**Gambar 5.3.** Algoritma untuk menentukan yang terbesar dari 3 bilangan

```

01| Algoritma Terbesar_dari_3_Bilangan_A
02| {Menentukan terbesar dari 3 bilangan dengan menggunakan metode
    compare each to all}
03| Deklarasi:
04| a, b, c, terbesar : real
05| Deskripsi
06| read(a,b,c)
07| if (a >= b) and (a >= c) then
08| terbesar = a
09| else

```

```

10| if b >= c then
11| terbesar ← b
12| else
13| terbesar ← c
14| end if
15| end if
16| write(terbesar)

```

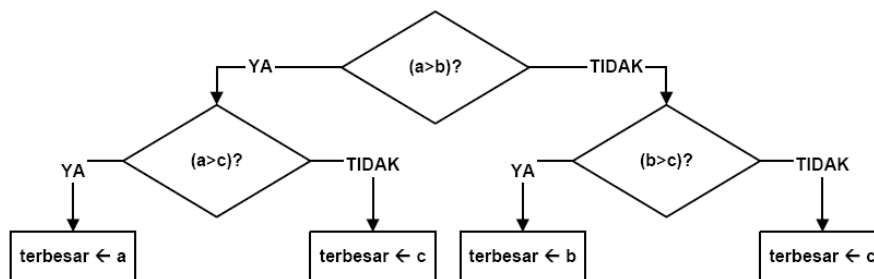
#### h. Algoritma Terbesar dari 3 Bilangan (Pohon Keputusan)

Buatlah algoritma untuk menentukan terbesar dari 3 bilangan.

Langkah Pengerjaan:

Misalnya bilangan tersebut adalah a, b, dan c.

- Apabila  $a > b$ , maka b sudah dipastikan bukan terbesar. Tersisa 2 pilihan, yaitu a yang terbesar atau c yang terbesar. Bandingkan a dan c untuk mengetahui yang mana yang terbesar.
- Jika tidak ( $a \leq b$ ), maka a sudah dipastikan bukan terbesar. Tersisa 2 pilihan, yaitu b yang terbesar atau c yang terbesar. Bandingkan b dengan c untuk mengetahui yang mana yang terbesar.



**Gambar 5.4.** Algoritma terbesar dari 3 bilangan

01| Algoritma Terbesar\_Dari\_3\_Bilangan\_B

02| { Menentukan terbesar dari 3 bilangan menggunakan metode pohon keputusan }

03| Deklarasi:

04| a, b, c, terbesar: real

05| Deskripsi



```

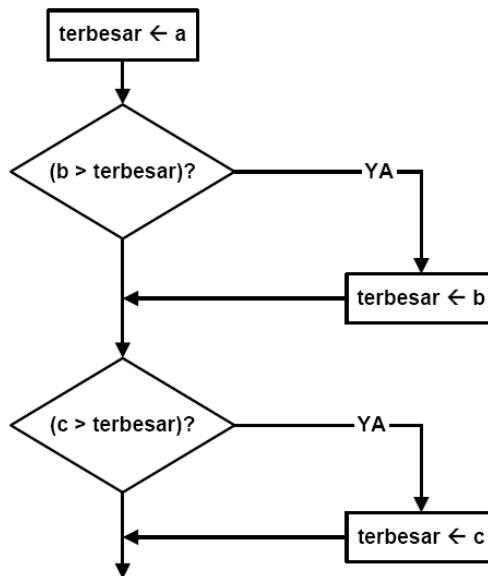
06| read(a,b,c)
07| if a>b then
08| if a>c then
09| terbesar  $\square$  a
10| else
11| terbesar  $\square$  c
12| end if
13| else
14| if b>c then
15| terbesar  $\square$  b
16| else
17| terbesar  $\square$  c
18| end if
19| end if
20| write(terbesar)

```

#### i. Algoritma Terbesar Dari 3 Bilangan (Sekuensial)

Buatlah algoritma untuk menentukan terbesar dari 3 bilangan. Analisis: Misalkan 3 bilangan tersebut adalah a, b, c.

- a. Asumsi bahwa bilangan a yang terbesar.
- b. Bandingkan asumsi dengan bilangan berikutnya, yaitu b. Jika b lebih besar, maka asumsikan terbesar adalah b. Jika tidak, maka asumsi kita bahwa a yang terbesar tidak berubah.
- c. Bandingkan asumsi dengan bilangan berikutnya, yaitu c. Jika c lebih besar, maka c adalah yang terbesar. Jika tidak, maka asumsi tidak berubah, dan asumsi itulah yang benar terbesar.



01| Algoritma Terbesar\_Dari\_3\_Bilangan\_C

02| { Menentukan terbesar dari 3 bilangan menggunakan metode sekuensial }

03| Deklarasi:

04| a, b, c, terbesar : real

05| Deskripsi

06| read(a,b,c)

07| terbesar ← a

08| if b > terbesar then

09| terbesar ← b

10| end if

11| if c > terbesar then

12| terbesar ← c

13| end if

14| write(terbesar)

## TUGAS

1. Diketahui bahwa cuaca dikatakan panas apabila temperatur lebih dari 300C dan cuaca dikatakan dingin apabila temperatur kurang dari 250C. Buatlah algoritma untuk mencetak kata “Panas”, “Dingin”, atau “Normal” sesuai dengan temperatur yang diinputkan user. **Temperatur yang diinputkan oleh user adalah dalam satuan derajat Fahrenheit.**
2. Berdasarkan teori pada soal nomor 3, buatlah algoritma untuk menghitung penyelesaian dari persamaan kuadrat apabila persamaan mempunyai penyelesaian real. ( $D \geq 0$ ). Petunjuk: untuk menghitung nilai akar digunakan **fungsi SQRT()**. Contoh:  $x \leftarrow (-b + \text{sqrt}(d)) / (2*a)$ . Ingat, hasil dari pengakaran suatu bilangan selalu bertipe real. Perintah  $\text{sqrt}(-1)$  akan menghasilkan kesalahan.
3. Buatlah algoritma untuk menentukan terbesar dari 4 bilangan dengan menggunakan 3 metode seperti pada algoritma nomor 8. , 9. , dan 10. .
4. Sebuah perusahaan memberikan kode pegawai dengan suatu bilangan yang terdiri dari 11 digit, dengan format *gddmmyyyynn*, yaitu *g* adalah nomor golongan (dari 1 hingga 4); *dd-mm-yyyy* adalah tanggal lahir dengan *dd* = tgl, *mm*=bulan, *yyyy*=tahun; dan *nn* adalah kode identitas yang menyatakan nomor urut. Misalnya 22505197803 adalah seorang pegawai bergolongan 2, lahir pada tanggal 25 Mei 1978 dan bernomor urut 3. Buatlah algoritma untuk mencetak nomor golongan, tanggal lahir, dan nomor urut pegawainya. Bulan kelahiran dicetak dalam format nama bulan, misalnya bulan ke-5 dicetak “MEI”.
5. Berdasarkan kode pegawai pada soal nomor 6 di atas, buatlah algoritma untuk menghitung gaji pegawainya. Ketentuannya adalah pajak sebesar 5% dari total gaji, besarnya gaji pokok dan tunjangan dapat dilihat pada tabel berikut.

Golongan	Gaji Pokok	Tunjangan
1	Rp400.000,00	Rp150.000,00
2	Rp600.000,00	Rp250.000,00
3	Rp900.000,00	Rp400.000,00
4	Rp1400.000,00	Rp700.000,00

## BAB VI

# ARRAY, PROSEDUR DAN FUNGSI

### 6.1. ARRAY

**Larik (Array)** adalah suatu bentuk struktur data yang menampung satu atau lebih dari satu data yang sejenis (bertipe data sama), yang diwakili oleh satu nama variabel.

- Setiap elemen atau anggota larik dapat dikenali atau diakses melalui suatu indeks.
- Larik berdimensi satu disebut *vektor*.
- Larik berdimensi dua disebut *matriks*.
- Larik berdimensi lebih dari dua disebut *tensor*.

Mendefinisikan Larik :

a. `nama_array : array [1..n] of tipe_data;`

contoh :

`A : array [ 1..10] of integer;`

b. `tipe_data nama_array [n];`

contoh :

`integer A[10];`

c. `type larik : array [1..n] of tipe_data;`

`nama_array : larik;`

contoh :

**type larik :** `array [1..10] of integer;`

`A : larik;`

**Operasi Larik :**

- membaca / mengisi larik

- mencetak / menampilkan larik
- menggeser isi larik
- menggabungkan beberapa larik
- menguraikan satu larik
- mengurutkan isi larik
- mencari elemen dalam larik

#### a. Membaca / mengisi Larik

Proses membaca atau mengisi suatu larik, dimulai dengan mendefinisikan array disertai dengan jumlah elemen yang akan disimpan, kemudian dengan memakai instruksi *perulangan* satu persatu elemen diisi dengan indeks yang berurutan mulai dari 1 hingga indeks maksimum.

Berikut ini *disajikan* dua algoritma untuk mengisi suatu larik. Algoritma yang pertama tidak menggunakan prosedur, algoritma yang kedua menggunakan prosedur.

##### Algoritma IsiLarik\_1

```
{ membaca atau mengisi larik tanpa menggunakan prosedur }
Deklarasi
const N = 10;
integer A[N];
integer indeks;
Deskripsi
for ( indeks = 1 to N step 1)
write ( "Masukkan elemen ke-", indeks)
read ( A[indeks] );
endfor.
```

##### Algoritma IsiLarik\_2

```
{ membaca atau mengisi larik dengan menggunakan prosedur }
Deklarasi
const N=100;
```

```

integer A[N];
integer K;
prosedur Baca_Larik ( input integer M, output integer A[ ]
);
Deskripsi
write ( "Masukkan Jumlah Elemen Larik ( < 100 ) : ");
read ( K );
Baca_Larik ( K, A);
prosedur Baca_Larik ( input integer M, output integer A[ ]
)
{ prosedur membaca / mengisi larik }
Deklarasi
integer indeks;
Deskripsi
for ( indeks = 1 to M step 1 )
write ( "Masukkan elemen ke-", indeks );
read ( A[indeks] );
endfor.

```

## b. Menampilkan Isi Larik

Berikut ini disajikan sebuah prosedur untuk menampilkan isi suatu larik dengan M buah elemen. Prosedur ini dapat dipanggil oleh algoritma yang memerlukan prosedur untuk menampilkan sebuah larik.

**prosedur Cetak\_Larik ( input integer M, input integer A[ ] )**

{ prosedur untuk menampilkan isi suatu larik atau array }

**Deklarasi**

**integer** indeks;

**Deskripsi**

**for** ( indeks = 1 **to** M **step** 1 )

**write** ( A[indeks] );

**endfor.**

### c. Menggeser Isi Larik

Beberapa aplikasi memerlukan pergeseran isi larik misalnya menggeser ke kiri atau menggeser ke kanan yang digambarkan sebagai berikut.

5	4	6	8	3	2	Larik Asli
---	---	---	---	---	---	------------

2	5	4	6	8	3	Setelah geser kanan
---	---	---	---	---	---	---------------------

4	6	8	3	2	5	Setelah geser kiri
---	---	---	---	---	---	--------------------

Proses Geser Kanan berarti elemen ber-indeks  $i$  digeser ke posisi ber-indeks  $i+1$ , dengan catatan elemen terakhir akan dipindahkan ke posisi pertama. Sebaliknya proses Geser Kiri berarti elemen ber-indeks  $i$  digeser ke posisi ber-indeks  $i-1$ , dengan menggeser elemen pertama ke posisi terakhir.

#### prosedur Geser\_Kanan ( in-out integer A[ ], input integer M )

{ menggeser elemen suatu larik (vektor) ke kanan,  $A[i+1] \leftarrow A[i]$  }

##### Deklarasi

**integer** indeks, temp;

##### Deskripsi

temp  $\leftarrow$  A[M];

**for** ( indeks = M-1 **to** 1 **step** -1 )

A[indeks + 1]  $\leftarrow$  A[indeks];

**endfor.**

A[1]  $\leftarrow$  temp;

#### prosedur Geser\_Kiri ( in-out integer A[ ], input integer M )

{ menggeser elemen suatu larik (vektor) ke kiri,  $A[i-1] \leftarrow A[i]$  }

##### Deklarasi

**integer** indeks, temp;

##### Deskripsi

temp  $\leftarrow$  A[1];

```

for ( indeks = 2 to M step 1 )
A[indeks-1]  $\leftarrow$  A[indeks];
endfor.
A[M]  $\leftarrow$  temp;

```

Suatu algoritma untuk menggeser elemen larik dengan memanfaatkan prosedur-prosedur yang telah digunakan diatas diberikan berikut ini.

### Algoritma Geser\_Larik

{ algoritma untuk menggeser isi larik dengan memanfaatkan prosedur prosedur larik }

#### Deklarasi

```

const Nmax = 100;
integer N, pilihan, A[Nmax];
prosedur Baca_Larik ( input integer M, input integer A[ ] );
prosedur Cetak_Larik ( input integer M, input integer A[ ] );
prosedur Geser_Kanan ( input integer A[ ], input integer M );
prosedur Geser_Kiri ( input integer A[ ], input integer M );

```

#### Deskripsi

```

write ( "Masukkan jumlah elemen larik : " );
read ( N );
Baca_Larik ( N, A );
write ( "Pilih salah satu : " );
write ( " 1. Geser Kanan " );
write ( " 2. Geser Kiri " );
read ( pilihan );
if ( pilihan = 1 )
then Geser_Kanan ( A, N );
else Geser_Kiri ( A, N );
endif.
Cetak_Larik ( N, A );

```



#### d. Menggabung (merge) Larik

Beberapa Larik dapat digabungkan menjadi satu Larik yang lebih besar.

Misalkan Larik A dengan 10 elemen akan digabungkan dengan Larik B dengan 15 elemen, tentu saja gabungannya berupa larik C harus memiliki elemen yang sama atau lebih besar dari 25.

Berikut ini adalah sebuah prosedur yang menggabungkan Larik A, N buah elemen dengan Larik B, M buah elemen, menjadi Larik C dengan L elemen dimana  $L = N + M$ .

**prosedur Gabung\_Larik ( input integer A[ ], input integer N, input integer B[ ], input integer M, output integer C[ ], output integer L )**

{ menggabungkan dua larik menjadi larik yang lebih besar }

**Deklarasi**

**integer** indeks;

**Deskripsi**

$L \leftarrow N + M$ ;

{ salin isi A ke dalam C }

**for** ( indeks = 1 **to** N **step** 1 )

C[indeks]  $\leftarrow$  A[indeks];

**endfor.**

{ salin isi B ke dalam C }

**for** ( indeks = N+1 **to** L **step** 1 )

C[indeks]  $\leftarrow$  B[indeks - N];

**endfor.**

#### e. Memisah (*split*) Larik

Sebuah Larik dapat dipisahkan menjadi beberapa Larik yang lebih kecil. Misalkan satu Larik C dengan 25 elemen dapat dipisahkan menjadi Larik A dengan 10 elemen dan Larik B dengan 15 elemen.

Berikut ini adalah sebuah prosedur yang memisahkan larik C dengan L elemen, menjadi larik A dengan N elemen dan larik B dengan M elemen.

**prosedur Pisah\_Larik (output integer A[ ], input integer N, output integer B[ ], input integer M, input integer C[ ], input integer L)**

{ memisahkan sebuah larik menjadi dua larik yang lebih kecil }

**Deklarasi**

**integer** indeks;

**Deskripsi**

{ salin isi C ke dalam A }

**for** ( indeks = 1 **to** N **step** 1 )

A[indeks]  $\leftarrow$  C[indeks];

**endfor.**

{ salin sisanya ke B }

**for** ( indeks = N+1 **to** L **step** 1 )

B[indeks - N]  $\leftarrow$  C[indeks];

**endfor.**

#### **f. Mengurutkan (Sort) Isi Larik**

Beberapa aplikasi memerlukan data yang ber-urut baik dari kecil ke besar (*ascending*) maupun dari besar ke kecil (*descending*). Pada bagian ini akan digunakan teknik sort yang sederhana yang disebut metoda gelembung (bubble sort), pada bagian lain nanti akan dibahas berbagai teknik sort yang lebih rumit.

Prinsip dari “bubble sort” adalah sebagai berikut:

- andaikan ada 5 elemen dalam larik [ 10, 8, 3, 5, 4 ]
- mula-mula ambil indeks 1 sebagai patokan A[1]=10
- bandingkan isi A[1] dengan A[2], A[3], A[4] dan A[5]
- bila A[1] > A[2] maka tukar tempat sehingga A[1] = 8 □ [ 8, 10, 3, 5, 4 ]
- bila A[1] > A[3] maka tukar tempat sehingga A[1] = 3 □ [ 3, 10, 8, 5, 4 ]
- bila A[1] > A[4] maka tukar tempat, tidak terjadi
- bila A[1] > A[5] maka tukar tempat, tidak terjadi
- Sekarang ambil indeks 2 sebagai patokan A[2] = 10
- bandingkan A[2] dengan A[3], A[4] dan A[5]

- hasilnya adalah [ 3, 4, 10, 8, 5]
- Sekarang ambil indeks 3 sebagai patokan  $A[3] = 10$
- bandingkan  $A[3]$  dengan  $A[4]$  dan  $A[5]$
- hasilnya adalah [3, 4, 5, 10, 8]
- Sekarang ambil indeks 4 sebagai patokan  $A[4] = 10$
- bandingkan  $A[4]$  dengan  $A[5]$
- hasilnya adalah [ 3, 4, 5, 8, 10 ]
- dengan demikian proses pengurutan selesai

Pada proses pengurutan yang dijelaskan diatas secara algoritma memerlukan dua buah indeks, indeks pertama sebagai patokan yang bergerak dari 1 hingga  $N-1$  ( 1 sampai 4 pada contoh diatas), dan indeks kedua sebagai pembanding yang selalu bergerak dari posisi indeks patokan + 1 hingga posisi terakhir  $N$  ( mulai dari 2 sampai 5 pada contoh diatas).

**prosedur Sort\_Larik ( in-out integer A[ ], input integer N )**

{ mengurutkan isi larik secara ascending dengan metoda bubble sort }

**Deklarasi**

**integer** idx1, idx2, temp;

**Deskripsi**

```

for ( idx1 = 1 to N-1 step 1 )
for ( idx2 = idx1 + 1 to N step 1 )
{ bila  $A[idx1] > A[idx2]$  tukar }
if (  $A[idx1] > A[idx2]$  )
then temp  $\leftarrow A[idx1]$ ;
 $A[idx1] \leftarrow A[idx2]$ ;
 $A[idx2] \leftarrow$  temp;
endif.
endfor.
endfor.

```

### g. Mencari (*Search*) elemen Larik

Mencari suatu elemen dalam larik merupakan suatu proses dasar yang sangat penting, aplikasi lanjut dari proses ini banyak ditemukan pada sistem basis data untuk menemukan suatu rekaman data, atau pada pemroses teks (*wordprocessing*) dalam mencari kata (*find*) atau mengganti kata (*replace*).

Hasil pencarian yang diharapkan antara lain:

- indikator 'Y' bila ditemukan atau 'N' bila tidak ditemukan
- posisi dalam larik dimana elemen tersebut ditemukan

Berikut ini disajikan sebuah prosedur untuk mencari satu elemen didalam larik dengan jumlah elemen sebanyak N buah.

**procedur Cari\_elemen ( input integer A[ ], input integer N, input integer x  
output char indikator, output integer posisi )**

{ suatu prosedur untuk mencari elemen x didalam larik A, dengan indikator dan posisi sebagai hasilnya }

**Deklarasi**

**integer** indeks;

**Deskripsi**

indikator  $\leftarrow$  'N';

posisi  $\leftarrow$  0;

indeks  $\leftarrow$  1;

**while** ( indeks < N+1 && indikator = 'N' ) **do**

**if** ( A[indeks] = x )

**then** posisi  $\leftarrow$  indeks;

indikator  $\leftarrow$  'Y';

**endif.**

indeks  $\leftarrow$  indeks + 1;

**endwhile.**

## h. Matriks / Larik Dua Dimensi

Salah satu struktur data larik yang juga banyak digunakan dalam berbagai aplikasi adalah *matriks* atau larik 2D (dua dimensi), satu untuk menunjukkan *baris* dan yang lainnya menunjukkan *kolom*. Susunan angka berikut ini menunjukkan matriks 4 x 5 (4 baris dan 5 kolom).

10	12	7	9	16
8	15	10	11	25
13	8	34	23	7
45	27	6	5	17

Mengisi suatu matriks berdimensi 4 x 5 dilakukan baris demi baris, mulai dari baris 1 dengan mengisi kolom 1 sampai dengan kolom 5, kemudian pindah ke baris 2 dan mengisi kolom 1 sampai dengan kolom 5, dan seterusnya.

### Algoritma Isi\_Matriks\_4x5

{ algoritma mengisi suatu matriks 4 x 5 }

#### Deklarasi

**const** baris=4, kolom=5;

**integer** brs, kol;

**integer** A[baris][kolom];

#### Deskripsi

**for** ( brs=1 to baris **step** 1 )

**for** ( kol=1 to kolom **step** 1 )

**write** ( "elemen baris-", brs, "kolom-", kol );

**read** ( A[brs][kol] );

**endfor**.

**endfor**.

Menampilkan isi dari matriks 4x5 diatas adalah sebagai berikut.

### Algoritma Tampilkan\_Isi\_Matriks

{ algoritma menampilkan isi matriks 4 x 5 }

**Deklarasi**

```
const baris=4, kolom=5;  
integer brs, kol;  
integer A[baris][kolom];
```

**Deskripsi**

```
for ( brs=1 to baris step 1 )  
for ( kol=1 to kolom step 1 )  
write ( A[brs][kol] );  
endfor.  
write ( );  
endfor.
```

Prosedur untuk mengisi dan menampilkan elemen-elemen matriks berdimensi N x M disajikan berikut ini.

**prosedur Isi\_Matriks (input integer N, input integer M, output integer A[ ] )**

{ prosedur untuk mengisi matriks berdimensi N x M }

**Deklarasi**

```
integer brs, kol;
```

**Deskripsi**

```
for ( brs=1 to N step 1 )  
for ( kol=1 to M step 1 )  
write ( "elemen baris-", brs, "kolom-", kol );  
read ( A[brs][kol] );  
endfor.  
endfor.
```

**prosedur Tampil\_Matriks ( input integer N, input integer M, input integer A[ ] )**

{ prosedur untuk menampilkan isi matriks berdimensi N x M }

**Deklarasi**

```
integer brs, kol;
```

**Deskripsi**

```
for ( brs=1 to N step 1 )  
for ( kol=1 to M step 1 )  
write ( A[brs][kol] );  
endfor.  
endfor.
```

### Latihan

1. Buatlah algoritma mencari nilai kuadrat bilangan antar 1-10, dengan menggunakan array
2. Terdapat sebuah array satu dimensi yang dibuat dengan int A[200], sudah ada isinya berupa nilai ujian mahasiswa. Susun algoritma untuk :
  - a. Mencetak nilai yang terbesar, dan mencetak jumlah mahasiswa yang mendapat nilai terbesar tersebut.
  - b. Mencetak nilai terbesar, dan mencetak berada di lokasi mana saja nilai yang terbesar tersebut berada dalam array
3. Terdapat array satu dimensi yang dibuat dengan int A[10], buatlah algoritma untuk mengisi array tersebut dengan menggunakan pointer sehingga isinya menjadi seperti gambar berikut :

1		2		3		4		5	
---	--	---	--	---	--	---	--	---	--

3. Seorang dosen ingin data-data nilai mahasiswanya dihitung dan dibuatkan programnya. Gunakan struktur untuk membuat program yang mempunyai ketentuan sebagai berikut :
  - a. Nama mahasiswa, nilai tugas, nilai UTS dan nilai UAS di input.
  - b. Proses yang dilakukan untuk mendapatkan nilai murni dari masing-masing nilai adalah :
    - Nilai murni tugas = nilai tugas x 30%
    - Nilai murni UTS = nilai UTS x 30%
    - Nilai UAS = nilai UAS x 40%
    - Nilai akhir = Nilai murni tugas + Nilai murni UTS + Nilai murni UAS
  - c. Ketentuan untuk grade nilai :
    - Nilai akhir  $\geq 80$  mendapat grade A
    - Nilai akhir  $\geq 70$  mendapat grade B
    - Nilai akhir  $\geq 59$  mendapat grade C
    - Nilai akhir  $\geq 50$  mendapat grade D
    - Nilai akhir  $< 50$  mendapat grade E

## 6.2. PROSEDUR

**Prosedur** adalah bagian dari suatu program yang disusun secara terpisah untuk melakukan suatu tugas khusus / fungsi tertentu. Pada dasarnya ada dua macam prosedur yaitu: **Subrutin ( Subprogram )** dan **Fungsi**.

**Subrutin (Subprogram)** adalah bagian dari program yang dibuat terpisah untuk melaksanakan sebagian dari tugas yang harus diselesaikan oleh suatu program.

Manfaat dari pembuatan prosedur :

- *modularisasi* : suatu program yang besar dan kompleks dapat dibagi ke dalam beberapa prosedur sehingga setiap prosedur merupakan bagian yang mudah dikerjakan, dengan demikian maka program besar tersebut menjadi mungkin diselesaikan.
- *simplifikasi* : dalam suatu program sering diperlukan suatu tugas yang berulang kali harus dikerjakan dengan nilai-nilai variabel yang berbeda, agar tidak merepotkan maka tugas ini cukup ditulis sekali saja dalam bentuk prosedur yang kemudian dipanggil berulang kali sesuai dengan kebutuhan.

### a. Bentuk Umum Prosedur:

- **Prosedur** nama\_prosedur  
{ spesifikasi dari prosedur, keadaan awal sebelum prosedur dilaksanakan dan juga keadaan akhir setelah prosedur dilaksanakan }
- **Deklarasi**  
{ deklarasi variabel-variabel prosedur }
- **Deskripsi**  
{ deskripsi dari tugas-tugas prosedur }
- **Contoh:**  
Andaikan sebuah program menyediakan fasilitas untuk menghitung luas, keliling, dan diagonal dari sebuah persegi panjang dengan kemungkinan pemilihan melalui suatu menu.



Contoh soal diatas dapat dibagi ke dalam enam prosedur yaitu: prosedur menampilkan menu, prosedur membaca dimensi persegi panjang, menghitung luas, menghitung keliling, menghitung diagonal, dan menampilkan hasil.

#### **Algoritma Empat\_Persegi\_Panjang**

{ contoh pemakaian prosedur untuk menghitung luas, keliling, dan diagonal empat persegi panjang }

##### **Deklarasi**

```
integer pilihan;  
real panjang, lebar, hasil;  
prosedur menu;  
prosedur baca_dimensi;  
prosedur hitung_luas;  
prosedur hitung_keliling;  
prosedur hitung_diagonal;  
prosedur tampil_hasil;
```

##### **Deskripsi**

```
pilihan ← 0;  
repeat  
menu;  
write ( "Masukkan pilihan anda : " );  
read ( pilihan );  
if ( pilihan < 4 )  
then baca_dimensi;  
endif.  
case ( pilihan )  
1 : hitung_luas;  
2 : hitung_keliling;  
3 : hitung_diagonal;  
4 : write ( "Selesai ... sampai jumpa " );  
default : write ( "Pilihan salah, Ulangi ! " );  
endcase.
```

```

if ( pilihan < 4 )
then tampil_hasil;
endif.
until ( pilihan = 4 ).

```

#### prosedur menu

{ menampilkan menu program }

**Deklarasi.**

**Deskripsi**

```

write ( "Menu Program Empat Persegi Panjang " );
write ( " 1. Menghitung Luas " );
write ( " 2. Menghitung Keliling " );
write ( " 3. Menghitung Diagonal " );
write ( " 4. Keluar dari Program" );

```

**prosedur baca\_dimensi**

{ membaca dimensi persegi panjang }

**Deklarasi.**

**Deskripsi**

```

write ( "Masukkan Panjang : " );
read ( panjang );
write ( "Masukkan Lebar : " );
read ( lebar );

```

#### prosedur hitung\_luas

{ menghitung luas empat persegi panjang }

**Deklarasi**

**real** luas;

**Deskripsi**

```

luas ← panjang * lebar;
hasil ← luas;

```

**prosedur hitung\_keliling**

{ menghitung keliling empat persegi panjang }

**Deklarasi**

```
real keliling;
```

**Deskripsi**

```
keliling  $\leftarrow$  2 * ( panjang + lebar );
```

```
hasil  $\leftarrow$  keliling;
```

**prosedur hitung\_diagonal**

```
{ menghitung diagonal empat persegi panjang }
```

**Deklarasi**

```
real diagonal;
```

**Deskripsi**

```
diagonal  $\leftarrow$  sqrt ( panjang ^2 + lebar^2 );
```

```
hasil  $\leftarrow$  diagonal;
```

**prosedur tampil\_hasil**

```
{ menampilkan hasil dari program ini }
```

**Deklarasi.****Deskripsi**

```
write ( "hasil = ", hasil );
```

**b. Variabel Lokal dan Variabel Global**

Penggunaan Prosedur pada suatu program menyebabkan munculnya dua kategori variabel, yaitu variabel lokal dan variabel global. **Variabel Lokal** adalah variabel yang hanya dikenal dan berlaku dalam suatu prosedur saja. **Variabel Global** adalah variabel yang berlaku di semua bagian program dan di semua prosedur.

Semua variabel yang *didefinisikan pada deklarasi suatu prosedur* adalah *variabel lokal*, dan variabel-variabel yang *didefinisikan pada deklarasi algoritma utama* adalah *variabel global*.

Program yang menggunakan banyak variabel global terasa memudahkan karena variabel variabel tidak perlu didefinisikan lagi dalam prosedur namun terlalu

banyak variable global dapat menyebabkan program sulit di-debug (cari kesalahan) dan memerlukan memory yang lebih besar.

### c. Parameter

Ketika suatu prosedur dipanggil maka pada hakekatnya bisa dilakukan pertukaran data antara program utama dan prosedur. Pertukaran ini dilakukan melalui parameter.

**Parameter Aktual** adalah parameter yang disertakan pada saat prosedur dipanggil untuk dilaksanakan, sering disebut sebagai *argumen*.

**Parameter Formal** adalah parameter yang dituliskan pada definisi suatu prosedur/fungsi. Ada tiga jenis parameter formal, yaitu:

- 1) **parameter masukan (input)** : parameter yang menerima nilai dari parameter aktual.
- 2) **parameter keluaran (output)** : parameter yang menyerahkan nilai ke parameter aktual.
- 3) **parameter masukan dan keluaran (input-output)** : parameter yang menerima nilai dari parameter aktual untuk diproses dalam prosedur kemudian diserahkan kembali ke parameter aktual setelah selesai.

Contoh algoritma berikut ini menunjukkan pemakaian parameter masukan dan parameter keluaran untuk prosedur menghitung luas segitiga.

#### Algoritma Luas\_segitiga

{menghitung luas segitiga dengan menggunakan prosedur yang memanfaatkan parameter input dan parameter output }

##### Deklarasi

**real** alas, tinggi, luas;

**prosedur** Hit\_Luas\_segi\_3 ( **input** real a, t; **output** real ls; );

##### Deskripsi

**write** ( "Masukkan alas segitiga : " );

**read** ( alas );

**write** ( "Masukkan tinggi-nya : " );

```

read ( tinggi );
Hit_Luas_segi_3 ( alas, tinggi, luas );
write ( "Luas segitiga = ", luas );
prosedur Hit_Luas_segi_3 ( input real a, t; output real
ls; )
{ prosedur menghitung luas segi_3, menerima a (alas) dan
t (tinggi),
mengembalikan ls (luas) }
Deklarasi { }
Deskripsi
ls  $\leftarrow$  a * t / 2.0;

```

Contoh algoritma berikut ini menunjukkan pemakaian parameter input/output yang digunakan untuk prosedur yang melakukan pertukaran nilai variabel.

#### Algoritma Tukar\_nilai

```

{ menukar nilai dua variabel yang dilakukan oleh suatu prosedur dengan
parameter
input/output }
Deklarasi
integer A, B;
prosedur Tukar ( in-out integer a, b );
Deskripsi
write ( "Masukkan nilai A : " );
read ( A );
write ( "Masukkan nilai B : " );
read ( B );
Tukar ( A, B );
write ( "Setelah ditukar : " );
write ( " A = ", A, " B = ", B );
prosedur Tukar ( in-out integer a, b )
{ prosedur yang melaksanakan pertukaran nilai }
Deklarasi
integer temp;
Deskripsi
temp  $\leftarrow$  a;
a  $\leftarrow$  b;
b  $\leftarrow$  temp;

```

### 6.3. FUNGSI

Fungsi pada hakekatnya serupa dengan prosedur tetapi harus mengembalikan nilai. Prosedur hanya bisa mengembalikan nilai melalui parameter input/output.

Bentuk Umum:

▪ **Fungsi** nama\_fungsi ( parameter formal )  $\leftarrow$  tipe\_hasil

{ spesifikasi fungsi }

▪ **Deklarasi**

{ variabel lokal }

▪ **Deskripsi**

{ langkah / proses yang dilakukan oleh fungsi }

.....

▪ **return** hasil.

Contoh berikut ini adalah contoh yang melaksanakan fungsi matematis

**f(x) =  $x^2 + 8x + 10$**

**Fungsi F ( input real x )  $\leftarrow$  real**

{ menghitung nilai fungsi  $f(x) = x^2 + 8x + 10$  }

**Deklarasi**

**real** y;

**Deskripsi**

y  $\leftarrow$  x \* x + 8 \* x + 10;

**return** y;

Contoh berikut ini adalah pemakaian fungsi untuk mengganti bulan dalam angka (mis. 3) menjadi nama bulan (mis. Maret).

**Algoritma Tanggal\_Lahir**

{ algoritma ini memanggil fungsi untuk menampilkan nama bulan }

**Deklarasi**

**integer** tanggal, bulan, tahun;

**string** nama\_bulan;

```

fungsi Nama_bulan ( input integer bln ) ← string;
Deskripsi
write ( "tanggal : " ); read ( tanggal );
write ( "bulan : " ); read ( bulan );
write ( "tahun : " ); read ( tahun );
nama_bulan ← Nama_bulan ( bulan );
write ( tanggal, ' - ', nama_bulan, ' - ', tahun );
fungsi Nama_bulan ( input integer bln ) ← string
{ mengembalikan nama bulan berdasarkan angka bulan }

```

#### **Deklarasi**

```

string nama_bln;

```

#### **Deskripsi**

```

case ( bln )
1 : nama_bln ← "Januari";
2 : nama_bln ← "Februari";
3 : nama_bln ← "Maret";
...
11 : nama_bln ← "Nopember";
12 : nama_bln ← "Desember";
endcase.
return nama_bln;

```

Contoh berikut ini menunjukkan pemakaian fungsi untuk menampilkan angka bulat (maksimum 4 digit) dalam bentuk kalimat, misal:

- input : 2436
- output : dua ribu empat ratus tiga puluh enam

#### **Algoritma Angka\_dalam\_kalimat**

```

{menterjemahkan angka (max. 4 digit) kedalam kalimat }

```

#### **Deklarasi**

```

integer angka, sisa, d1, d2, d3, d4;
string angka1, angka2, angka3, angka4;
fungsi digit ( input integer d ) ← string;

```

**Deskripsi**

```
write ( "Masukkan sebuah angka (maks 4 digit) " );  
read ( angka );  
{ memisahkan digit angka dalam urutan d4 d3 d3 d1 }  
d4 ← angka \ 1000;  
sisasisa ← angka % 1000;  
d3 ← sisasisa \ 100;  
sisasisa ← sisasisa % 100;  
d2 ← sisasisa \ 10;  
sisasisa ← sisasisa % 10;  
d1 ← sisasisa;  
{ menterjemahkan digit }  
if ( d4 > 1 )  
then angka4 ← digit ( d4 ) + "ribu";  
else if ( d4 = 1 )  
then angka4 ← "seribu";  
else angka4 ← ' ';  
endif.  
endif.  
  
if ( d3 > 1 )  
then angka3 ← digit ( d3 ) + "ratus";  
else if ( d3 = 1 )  
then angka3 ← "seratus";  
else angka3 ← ' ';  
endif.  
endif.  
  
if ( d2 > 1 )  
then angka2 ← digit ( d2 ) + "puluh";  
if ( d1 = 0 )
```



```

then angka1  \ \;
else angka1  digit ( d1 ) ;
endif.
else if ( d2 = 1 )
then if ( d1 = 0 )
then angka2  "sepuluh";
angka1  \ \;
else if ( d1 = 1 )
then angka2  "sebelas";
angka1  \ \;
else angka2  digit ( d1 ) + "belas";
angka1  \ \;
endif.
endif.
else angka2  \ \;
if ( d1 = 0 )
then angka1  \ \;
else angka1  digit ( d1 );
endif.
endif.
endif.
write ( angka, \ = \ , angka4 + angka3 + angka2 + angka1
);
fungsi digit ( input integer d )  string
{ menterjemahkan digit ke dalam satu kata }
Deklarasi
string kata;
Deskripsi
case ( d )
1 : kata  "satu";
2 : kata  "dua";
3 : kata  "tiga";
4 : kata  "empat";

```

```

5 : kata    "lima";
6 : kata    "enam";
7 : kata    "tujuh";
8 : kata    "delapan";
9 : kata    "sembilan";
endcase.
return kata;

```

### **Perbedaan Prosedur dan Fungsi Pada Algoritma dan Pemrograman :**

Setiap bahasa pemrograman selalu menyediakan fungsi-fungsi yang sudah didefinisikan oleh bahasa pemrograman tersebut (*built-in function*). Namun ada kalanya kita memerlukan suatu prosedur tertentu yang kita gunakan berulang kali dan tidak tersedia dalam *built-in function*.

Prosedur adalah suatu program terpisah dalam blok sendiri yang berfungsi sebagai subprogram (program bagian). Diawali dengan kata cadangan "Procedure" didalam bagian deklarasi prosedur. Procedure biasanya bersifat suatu aktifitas seperti menghitung luas, menghitung faktorial, mencari nilai maksimum/minimum.

Prosedur banyak digunakan pada program yang terstruktur karena :

- 1) Merupakan penerapan konsep program modular, yaitu memecah-mecah program yang rumit menjadi program-program bagian yang lebih sederhana dalam bentuk prosedur-prosedur.
- 2) Untuk hal-hal yang sering dilakukan berulang-ulang, cukup dituliskan sekali saja dalam prosedur dan dapat dipanggil atau dipergunakan sewaktu-waktu bila diperlukan.
- 3) Membuat kode program lebih mudah dibaca.
- 4) Dapat digunakan untuk menyembunyikan detail program

```

Notasi Algoritmik
program tukar
kamus : A,R : integer {}
procedure tukar (in/out : A,R : integer)
algoritma : input (A,R)
Tukar (A,R)

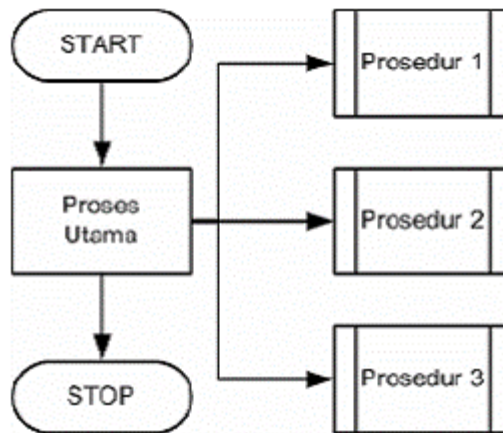
```

```

Output (A,R)
procedure tukar2(in/out : A,R : integer)
kamus lokal : B : integer
algoritma : B ← A
A ← R
R ← A

```

Pada flowchart, untuk menuliskan prosedur digunakan notasi Predefined Process. Secara skematis, penggunaan prosedur dapat dilihat pada gambar 6.6.



Gambar 6.1. Skema penggunaan prosedur

Gambar 6.1. menunjukkan ada proses utama yang terjadi, dan ada prosedur yang sebenarnya merupakan bagian dari proses utama ini. Ketika proses utama membutuhkan suatu tugas tertentu, maka proses utama akan memanggil prosedur tertentu menyelesaikan tugas tersebut.

Fungsi sama seperti halnya dengan prosedur, namun tetap ada perbedaannya yaitu fungsi mempunyai output dengan tipe variabel yang kita tentukan. Dan cara pemanggilan variabel ada 2 macam dalam pascal. Yaitu :

- Meng-outputkan nilai dari fungsi tersebut Contoh : Writeln (namafungsi(parameter));
- Dengan assignment Variabel1 := namafungsi(parameter);

Berbeda dengan procedure yang bisa tidak menggunakan parameter, fungsi harus menggunakan parameter dalam penggunaannya.

algoritmik program faktorial

```

kamus : a : integer
function faktor (a : integer) → integer
    algoritma
    input (a)
    output (faktor(a))
function faktor (a:integer) →integer
kamus :
i, hasil : integer
algoritma :
hasil ← 1
i traversal [a..1]
hasil ← hasil * i
faktor ← hasil
FUNCTION identifier (daftar parameter) : type ;

```

Blok fungsi juga diawali dengan kata cadangan Begin dan di akhiri dengan kata cadangan End dan titik koma.

**Perbedaan fungsi dengan prosedur adalah :**

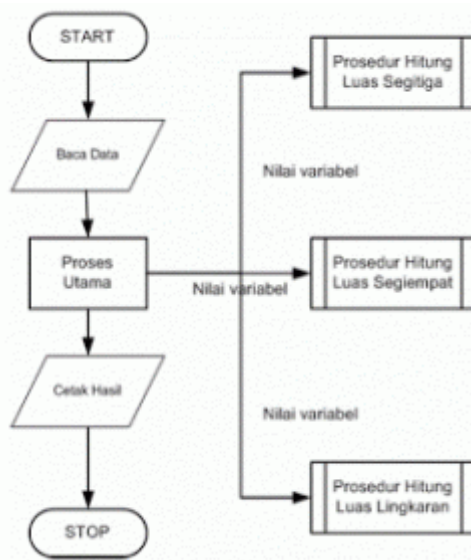
- Pada fungsi, nilai yang dikirimkan balik terdapat pada nama fungsinya ( kalau pada prosedur pada parameter yang dikirimkan secara acuan).
- Karena nilai balik berada di nama fungsi tersebut, maka fungsi tersebut dapat langsung digunakan untuk dicetak hasilnya. Atau nilai fungsi tersebut dapat juga langsung dipindahkan ke pengenalan variable yang lainnya.
- Pada prosedur, nama prosedur tidak dapat digunakan langsung, yang dapat langsung digunakan adalah parameternya yang mengandung nilai balik.

### Latihan

Buatlah algoritma menghitung luas bangun datar dan bangun ruang, dan rumus-rumus perhitungan matematik lainnya.

### Penyelesaian:

Untuk membuat algoritma ini kita dapat memandang proses perhitungan luas segitiga, luas segiempat, dan luas lingkaran sebagai bagian program yang berdiri sendiri. Kita dapat membuat prosedur untuk masing-masing proses. Dan kita akan memanggil prosedur tersebut dari proses utama (gambar 5.2).



Urutan proses pada gambar 5.2 adalah sebagai berikut.

- 1) Pembacaan data
- 2) Pada proses utama akan terjadi pengecekan pada data yang dibaca,
- 3) Apabila data yang dibaca adalah untuk segitiga, maka proses utama akan memanggil prosedur hitung luas segitiga dengan membawa nilai variable yang diperlukan oleh prosedur luas hitung segitga.
- 4) Proses perhitungan luas segitiga hanya dilakukan pada prosedur tersebut.
- 5) Setelah proses perhitungan, maka hasil perhitungan akan dibawa kembali ke proses utama untuk dicetak hasilnya. Urutan proses yang sama juga terjadi jika data yang dibaca adalah untuk segi-empat atau lingkaran.

## DAFTAR PUSTAKA

- Aho, Hopcroft, Ullman, "*Data Structures and Algorithms*", Prentice Hall, 1987.
- Knuth, D.E., "*The Art of Computer Programming*", Vol. 1 : "*Fundamentals Algorithms*", Addison Wisley, 1968.
- Sedgewick R., "*Algorithms*", Addison Wisley, 1984.
- Wirth, N., "*Algorithms & Data Stuctures*", Prentice Hall, 1986.
- Munir, R dan Lidya, L. 2001. *Algoritma dan Pemrograman Dalam Bahasa Pascal dan C*. Bandung: Informatika.
- Kadir, A dan Heriyanto. 2005. *Algoritma Pemrograman Menggunakan C++*. Yogyakarta: Penerbit Andi.
- Liem, Inggriani. Modul Kuliah Algoritma dan Pemrograman I . Bandung : ITB
- Pranata, A. 2005. *Algoritma dan Pemrograman*. Yogyakarta: Penerbit Graha Ilmu.
- P.J. Deitel, H.M. Deitel, "*C How to Program*", *Pearson International Edition Fifth Edition*, 2007.
- Stephen Prata, "*C Primer Plus*", *Sams Publishing Fifth Edition*, 2005.
- Fathul Wahid, "*Dasar-Dasar Algoritma & Pemrograman*", Penerbit Andi, Yogyakarta, 2004.
- Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran. "*Computer Algorithms / C++*, *Computer Science Press*. 1998.
- Thomas H Cormen, Charles E Leiserson, Ronald L. "*Introduction to Algorithms*", 2nd Edition. The MIT Press. New York. 1990.
- Robert Setiadi. "*Algoritma Itu Mudah*", PT Prima Info sarana Media, Kelompok Gramedia. Jakarta. 2008