

Algoritma dan Pemrograman



giving and caring the world

**POLITEKNIK TELKOM
BANDUNG
2009**

Penyusun

Dahliar Ananda
Ahmad Suryan
Paramita Mayadewi
Lutce Rasiana
Hendra Kusmayadi

Editor

Ade Hendraputra

Dilarang menerbitkan kembali, menyebarkan atau menyimpan baik sebagian maupun seluruh isi buku dalam bentuk dan dengan cara apapun tanpa izin tertulis dari Politeknik Telkom.

Hak cipta dilindungi undang-undang @ Politeknik Telkom 2009

No part of this document may be copied, reproduced, printed, distributed, modified, removed and amended in any form by

any means without prior written authorization of Telkom Polytechnic.

Kata Pengantar

Assalamu'alaikum Wr. Wb

Segala puji bagi Allah SWT karena dengan karunia-Nya *courseware* ini dapat diselesaikan.

Atas nama Politeknik Telkom, kami sangat menghargai dan ingin menyampaikan terima kasih kepada penulis, penerjemah dan penyunting yang telah memberikan tenaga, pikiran, dan waktu sehingga *courseware* ini dapat tersusun.

Tak ada gading yang tak retak, di dunia ini tidak ada yang sempurna, oleh karena itu kami harapkan para pengguna buku ini dapat memberikan masukan perbaikan demi pengembangan selanjutnya.

Semoga *courseware* ini dapat memberikan manfaat dan membantu seluruh Sivitas Akademika Politeknik Telkom dalam memahami dan mengikuti materi perkuliahan di Politeknik Telkom.

Amin.

Wassalamu'alaikum Wr. Wb.

Bandung, Agustus 2009

Christanto Triwibisono

Wakil Direktur I

Bidang Akademik & Pengembangan

Daftar Isi

Kata Pengantar	iii
Daftar Isi	v
1 Pendahuluan	1
<i>Komputer Elektronik</i>	<i>2</i>
<i>Komponen Komputer</i>	<i>2</i>
<i>Algoritma</i>	<i>3</i>
<i>Program</i>	<i>4</i>
<i>Bahasa Pemrograman</i>	<i>5</i>
<i>Klasifikasi Menurut Generasi</i>	<i>5</i>
<i>Klasifikasi Menurut Tingkatan</i>	<i>6</i>
<i>Flowchart</i>	<i>9</i>
<i>Pseudocode</i>	<i>12</i>
2 Flowchart dan Pseudocode	14
<i>Flowchart</i>	<i>15</i>
<i>Pengambilan Keputusan</i>	<i>15</i>
<i>Pengulangan Proses</i>	<i>18</i>
<i>Pseudocode</i>	<i>21</i>
<i>Struktur algoritma</i>	<i>21</i>
3 Tipe Data, Operator dan Runtunan	26
3.1 <i>Tipe Data Dasar</i>	<i>27</i>
3.2 <i>Variabel</i>	<i>30</i>
3.3 <i>Konstanta</i>	<i>35</i>
3.4 <i>Operator</i>	<i>35</i>
3.5 <i>Urutan Operasi</i>	<i>40</i>
3.6 <i>Runtunan</i>	<i>42</i>
4 Pemilihan	51
4.1 <i>Bentuk Umum IF dan Variasinya</i>	<i>53</i>
4.2 <i>Terapan bentuk-bentuk IF</i>	<i>59</i>
4.3 <i>Bentuk Umum CASE dan variasinya</i>	<i>65</i>
4.4 <i>Terapan bentuk-bentuk CASE</i>	<i>67</i>
4.5 <i>Konversi Struktur IF dan CASE ke Bahasa C</i>	<i>69</i>
5 Pengulangan	81
5.1 <i>Konsep Pengulangan</i>	<i>82</i>
5.2 <i>Sintaks WHILE</i>	<i>83</i>

5.3	Sintaks DO...WHILE	91
5.4	Sintaks FOR	100
5.5	Sintaks Pengulangan Bersarang	107
5.6	Sintaks BREAK dan CONTINUE	112
6	Array dan Tipe Data Bentukan	126
6.1	Array	127
6.1.1	Array Satu Dimensi	127
6.1.2	Array Dua Dimensi	131
6.1.3	Array Multi-Dimensi	135
6.2	Tipe Data Bentukan	136
6.3	Kombinasi Tipe Bentukan dan Array	139
6.3.1	Tipe Data Bentukan di dalam array	139
6.3.2	Array di dalam Tipe Data Bentukan.....	143
6.3.3	Array dari Tipe Bentukan yang Mengandung Array.....	147
7	Pemrograman Modular.....	156
7.1	Definisi Pemrograman Modular	157
7.2	Variabel Lokal dan Variabel Global	159
7.2.1	Variabel Lokal	159
7.2.2	Variabel Global	159
7.3	Fungsi	159
7.4	Prosedur	167
7.5	Fungsi dan Prosedur yang telah terdefinisi	171
7.6	Fungsi Rekursif	172
7.7	Unit	174
8	Mesin Karakter	183
8.	Pencarian.....	198
9	Pengurutan (Sorting).....	221
9.1	Pengertian Sort	222
9.2	Bubble Sort	223
9.3	Selection Sort	227
9.3.1	Maximum Selection Sort Ascending	227
9.3.2	Maximum Selection Sort Descending	231
	Output yang dihasilkan:.....	234
9.3.3	Minimum Selection Sort Ascending	234
	Output yang dihasilkan:.....	237
9.3.4	Minimum Selection Sort Descending	237
9.4	Insertion Sort	240

Daftar Pustaka vi

1 Pendahuluan



Overview

Komputer sudah menjadi alat bantu kehidupan manusia sehari-hari. Tanpa bantuan manusia, komputer hanya akan menjadi seonggok mesin yang tidak bisa melakukan apa-apa. Program menjadi “roh” yang dapat membuat komputer dapat bekerja dan memberi bantuan kepada manusia. Dalam membuat program harus melalui beberapa tahapan, salah satunya adalah tahap desain. Supaya perancangan program dapat dikomunikasikan dengan orang lain maka, perancangan program harus menggunakan notasi yang standar dan mudah untuk dibaca dan dipahami.



Tujuan

1. Memahami bagaimana komputer menangani data elektronik
2. Memahami komponen yang terlibat dalam memproduksi

informasi

3. Memahami perbedaan bahasa pemrograman di setiap tingkatan

Komputer Elektronik

Komputer di era modern seperti sekarang ini, sudah menjadi kebutuhan untuk mendukung aktivitas yang dilakukan oleh manusia. Bentuk fisik dari komputer pun juga beragam, kompak dan semakin praktis.

Seluruh perangkat elektronik pada umumnya terdapat sebuah komputer kecil yang berfungsi sebagai 'otak' atau pusat pengendali perangkat tersebut.

Perangkat komputer modern dapat bekerja apabila terdapat energi listrik, demikian pula dengan data yang diolah. Dengan ditemukannya energi listrik, seluruh data dalam bentuk apapun sangat dimungkinkan untuk direpresentasikan ke dalam bentuk elektronik.



Gambar 1. Komputer elektronik

Komponen Komputer

Di dalam sebuah komputer elektronik terdapat beberapa komponen/perangkat yang berfungsi untuk mengolah data. Secara umum, komponen komputer terbagi menjadi 3 (tiga) bagian, yaitu:



Alat input berfungsi sebagai media untuk memasukkan data ke dalam komputer. Contoh alat input adalah: keyboard, mouse, microphone, dll.

Alat pemroses di dalam komputer berfungsi untuk melakukan pengolahan data menjadi informasi. Contoh alat pemroses adalah: prosesor.

Alat output berfungsi sebagai media untuk menyampaikan informasi hasil pengolahan, bisa dalam bentuk tampilan menggunakan monitor ataupun dalam bentuk cetakan menggunakan printer.

Sesungguhnya, komputer itu hanyalah mesin elektronik yang tersusun atas komponen-komponen di atas. Namun dengan adanya energi listrik dan perangkat lunak, barulah komponen komputer dapat aktif dan kemudian digunakan untuk bekerja.

Algoritma

Kata 'algoritma' diturunkan dari nama belakang seorang tokoh matematikawan Persia bernama Muhammad ibn Musa al-Khuwarizmi (lahir tahun 730an, meninggal antara tahun 835 dan 850). Al-Khuwarizmi berasal dari propinsi Khorasan di negara yang saat ini bernama Uzbekistan. Uni Soviet menghormati jasa-jasa Al-Khuwarizmi dengan membuat gambar dirinya sebagai perangko.

Algoritma merupakan metode umum yang digunakan untuk menyelesaikan kasus-kasus tertentu [1]. Dalam menuliskan algoritma, dapat digunakan bahasa natural atau menggunakan notasi matematika, sehingga masih belum dapat dijalankan pada komputer.

Dalam kehidupan sehari-hari, kita sudah melakukan penyusunan algoritma untuk menyelesaikan permasalahan atau tantangan yang dihadapi. Sebagai contoh, pada saat diminta

untuk membuat telur dadar. Sebelum membuat algoritmanya, kita perlu mendefinisikan masukan (input) dan luaran (output) terlebih dahulu, dimana input berupa telur mentah, dan output berupa telur dadar yang sudah matang.

Susunan algoritmanya sebagai berikut:

1. Nyalakan api kompor
2. Tuangkan minyak ke dalam wajan
3. Pecahkan telur ayam ke dalam mangkok
4. Tambahkan garam secukupnya
5. Aduk campuran telur dan garam
6. Tuang adonan telur ke dalam wajan
7. Masak telur hingga matang

Algoritma akan lebih baik jika ditulis secara sistematis menggunakan beberapa skema, dalam buku ini akan dibahas mengenai skema *Flowchart* dan *Pseudocode*.

Program

Program adalah formulasi sebuah algoritma dalam bentuk bahasa pemrograman[1], sehingga siap untuk dijalankan pada mesin komputer. Membuat program seperti memberitahukan apa yang harus dilakukan kepada orang lain. Sebagai contoh, pada saat kita memberitahukan algoritma membuat telur dadar kepada orang lain, kita sudah melakukan pemrograman.

Pemrograman membuat telur dadar kepada orang lain akan lebih mudah karena orang tersebut sudah mengetahui apa itu telur dadar. Pada langkah yang ke-3 diminta untuk memecahkan telur, bagaimana cara orang tersebut memecahkan telur tentunya sudah diketahui dan kita tidak perlu menjelaskan terlalu detil.

Lain halnya jika kita harus menyuruh komputer untuk melakukan apa yang kita inginkan. Komputer sebenarnya hanyalah sebuah mesin bodoh yang tidak memiliki emosi dan kemampuan bersosialisasi. Oleh karena itu, untuk membuatnya menjadi mudah, diperlukan penyusunan algoritma yang benar.

Mendesain algoritma yang benar dan menterjemahkannya ke dalam bahasa pemrograman bukanlah

hal yang mudah karena bahasa pemrograman memiliki tata penulisan sendiri.

Bahasa Pemrograman

Bahasa pemrograman adalah bahasa buatan yang digunakan untuk mengendalikan perilaku dari sebuah mesin, biasanya berupa mesin komputer[2], sehingga dapat digunakan untuk memberitahu komputer tentang apa yang harus dilakukan[3].

Struktur bahasa ini memiliki kemiripan dengan bahasa natural manusia, karena juga tersusun dari elemen-elemen dasar seperti: kata benda dan kata kerja serta mengikuti aturan untuk menyusunnya menjadi kalimat.

Klasifikasi Menurut Generasi

1. *First Generation Language (1GL)*
Bahasa pemrograman ini berupa kode-kode mesin yang hanya bisa dipahami oleh mikroprosesor.
2. *Second Generation Language (2GL)*
Bahasa pada generasi ini adalah *assembly language*, dimana bahasa ini masih menggunakan kode-kode yang disebut dengan *mnemonic*. Bahasa *assembly* disebut sebagai generasi kedua karena bahasa ini bukan bahasa asli mikroprosesor, meskipun begitu programmer tetap harus mengetahui keunikan dari masing-masing mikroprosesor (register dan jenis instruksi).
3. Generasi ketiga
Bahasa pemrograman generasi ketiga sengaja didesain supaya mudah dipahami oleh manusia. Pada generasi ini mulai dikenalkan istilah variabel, tipe data, ekspresi aljabar dan sudah mendukung pemrograman terstruktur.
Contoh bahasa: FORTRAN, COBOL, ALGOL, BASIC, C, C++, Pascal, Java.
4. Generasi keempat

Pada generasi ini, bahasa pemrograman didesain untuk mengurangi *effort* dan mempercepat proses pembuatan program. Pada 3GL, pembuatan program membutuhkan waktu yang lama dan mudah sekali didapati error. Pada 4GL, telah menggunakan metodologi dimana sebuah perintah dapat menghasilkan beberapa instruksi 3GL yang kompleks dengan sedikit error[4].

Contoh bahasa:

- a. Pemrograman umum : DataFlex, WinDev, PowerBuilder
 - b. Basis data : SQL, Progress 4GL
 - c. Manipulasi data, analisis dan pelaporan : ABAP, Matlab, PL/SQL.
5. Generasi kelima

Bahasa pemrograman generasi kelima disebut sebagai *constraint-programming* atau *declarative-programming*. Program tidak dituliskan dalam bentuk algoritma melainkan dituliskan batasan atau fakta dari sebuah lingkup masalah, sehingga program akan menghasilkan luaran dalam bentuk solusi[5].

Bahasa pemrograman ini digunakan untuk membangun sistem kecerdasan buatan dan belum digunakan secara meluas di dunia industri. Contoh bahasa: Prolog, LISP, Mercury.

Klasifikasi Menurut Tingkatan

1. Low-level programming language

Tingkat bahasa pemrograman ini disebut "rendah" (*low level*) bukan karena posisinya berada di bawah, melainkan karena kurangnya abstraksi (penggambaran kode instruksi) antara bahasa natural dengan bahasa mesin. Oleh karena itu, bahasa di tingkat ini sering disebut sebagai 'bahasa mesin'. Bahasa pemrograman yang masuk kategori ini adalah bahasa mesin itu sendiri (1GL) dan bahasa *assembly* (2GL).

2. High-level programming language (HLL)
Bahasa pemrograman di tingkat ini memiliki abstraksi yang lebih banyak dan terdapat kemiripan dengan bahasa natural (bahasa Inggris), lebih mudah untuk digunakan dan mudah untuk dipindahkan antar platform.
3. Very high-level programming language (VHLL)
Bahasa ini memiliki abstraksi yang lebih tinggi dibandingkan HLL, dan digunakan untuk menunjang produktifitas programmer profesional. Biasanya VHLL digunakan hanya untuk tujuan yang spesifik, misalnya untuk keperluan bisnis: mengolah data, membuat laporan, dsb.

Paradigma Pemrograman

Paradigma pemrograman merupakan sebuah cara pandang seorang programmer dalam menyelesaikan sebuah masalah dan memformulasikannya kedalam sebuah bahasa pemrograman. Terdapat beberapa paradigma pemrograman, antara lain:

Paradigma Imperatif

Inti dari paradigma ini adalah menjalankan sebuah urutan perintah, jalankan satu perintah kemudian jalankan perintah yang selanjutnya. Sebuah program imperatif tersusun dari sekumpulan urutan perintah yang akan dijalankan oleh komputer. Pemrograman prosedural merupakan salah satu contoh dari paradigma ini, dan seringkali dianggap sebagai sebuah paradigma yang sama.

- Ide dasarnya adalah dari model komputer Von Neumann.
- Eksekusi langkah-langkah komputasi diatur oleh sebuah struktur kontrol.

- Berdasarkan urutan-urutan atau sekuensial.
- Program adalah suatu rangkaian prosedur untuk memanipulasi data. Prosedur merupakan kumpulan instruksi yang dikerjakan secara berurutan.
- Contoh bahasa pemrograman: Fortran, Algol, Pascal, Basic, C

Paradigma Fungsional

Pemrograman Fungsional adalah sebuah paradigma yang menjadikan fungsi matematika sebagai penentu dalam eksekusi komputasi. Fungsi tersebut merupakan dasar utama dari program yang akan dijalankan. Paradigma ini lebih banyak digunakan di kalangan akademis daripada produk komersial, terutama yang murni fungsional.

- Ide dasar dari matematika dan teori fungsi.
- Beberapa contoh bahasa fungsional adalah APL, Erlang, Haskell, Lisp, ML, Oz dan Scheme.

Paradigma Logika

Umumnya digunakan pada domain yang berhubungan dengan ekstraksi pengetahuan yang berbasis kepada fakta dan relasi. Dalam paradigma ini, logika digunakan secara murni untuk representasi bahasa deklaratif yang kebenarannya ditentukan oleh programmer, sedangkan pembukti-teorema atau model pembangkit digunakan sebagai pemecah masalah.

- Berasal dari pembuktian otomatis didalam intelegensia buatan.
- Berdasar kepada aksioma, aturan dan query.
- Eksekusi program menjadi proses pencarian secara sistematis dalam sekumpulan fakta, dengan menggunakan sekumpulan aturan.
- Beberapa contoh bahasa pemrograman: ALF, Fril, Gödel, Mercury, Oz, Ciao, Visual Prolog, XSB, and λ Prolog

Paradigma Berorientasi Obyek

Pemrograman berorientasi obyek muncul untuk mengatasi masalah kompleksitas dari sebuah perangkat lunak sehingga kualitas dari perangkat lunak tersebut dapat dikelola dengan lebih mudah. Caranya adalah dengan memperkuat *modularity* dan *reusability* didalam perangkat lunak tersebut. Pemrograman berorientasi obyek menggunakan obyek dan interaksi antar obyek dalam penyusunan sebuah perangkat lunak. Paradigma ini semakin banyak digunakan karena lebih mudah dalam menggambarkan kondisi yang ada pada dunia nyata.

Ide dari interaksi antar obyek yang ada pada dunia nyata.

Antar obyek saling berinteraksi dengan saling mengirimkan pesan (*message*).


Terdapat beberapa karakteristik utama, yaitu: Abstraksi, Enkapsulasi, Pewarisan dan Polimorfisme.



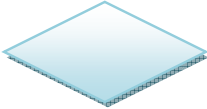





Flowchart

Dalam membuat algoritma, diperlukan suatu mekanisme atau alat bantu untuk menuangkan hasil pemikiran mengenai langkah-langkah penyelesaian masalah yang sistematis dan terurut. Pada dasarnya untuk bisa menyusun solusi diperlukan kemampuan *problem-solving* yang baik. Oleh karena itu, sebagai sarana untuk melatih kemampuan tersebut terdapat sebuah *tool* (alat) yang dapat digunakan, yakni *flowchart*.

Secara formal, *flowchart* didefinisikan sebagai skema penggambaran dari algoritma atau proses[8]. Tabel berikut menampilkan simbol-simbol yang digunakan dalam menyusun *flowchart*.

Tabel 1.1 Simbol-simbol dalam *flowchart*

	<p>Terminator</p> <p>Sebagai simbol 'START' atau 'END' untuk memulai atau mengakhiri flowchart.</p>
---	--

	Input/Output Digunakan untuk menuliskan proses menerima data atau mengeluarkan data
	Proses Digunakan untuk menuliskan proses yang diperlukan, misalnya operasi aritmatika
	Conditional / Decision Digunakan untuk menyatakan proses yang membutuhkan keputusan
	Preparation Digunakan untuk memberikan nilai awal
	Arrow Sebagai penunjuk arah dan alur proses
	Connector (On-page) Digunakan untuk menyatukan beberapa arrow
	Connector (Off-page) Digunakan untuk menghubungkan flowchart yang harus digambarkan pada halaman yang berbeda. Biasanya pada simbol ini diberi nomor sebagai penanda, misalnya angka 1.
	Display Digunakan untuk menampilkan data ke <i>monitor</i>

Berikut ini adalah flowchart untuk menggambarkan kegiatan membuat telur dadar:

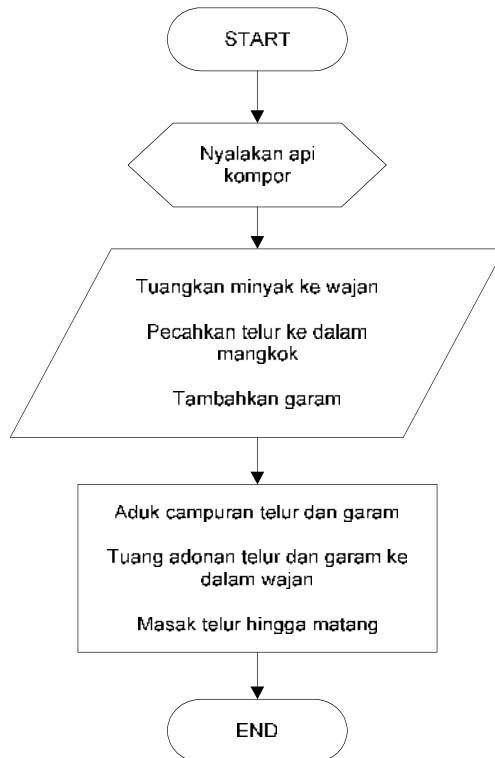


Diagram 1.1 Flowchart membuat telur dadar

Dengan menggunakan flowchart, tahapan-tahapan penting dalam algoritma dapat ditunjukkan dengan diagram di atas. Aliran proses ditunjukkan dengan arah panah atau disebut dengan 'flowlines'.

Keuntungan menggunakan *flowchart* adalah penggunaan diagram untuk menggambarkan tahapan proses, sehingga lebih mudah dilihat dan dipahami. Namun demikian, flowchart juga memiliki kelemahan, yakni jika digunakan untuk menggambarkan proses atau algoritma untuk skala kasus yang besar, maka akan dibutuhkan banyak kertas[7].

Pseudocode

Skema lain yang dapat digunakan untuk menyusun algoritma adalah *pseudocode*. *Pseudocode* adalah bentuk informal untuk mendeskripsikan algoritma yang mengikuti struktur bahasa pemrograman tertentu.

Tujuan dari penggunaan *pseudocode* adalah supaya :

- 1. lebih mudah dibaca oleh manusia
- 2. lebih mudah untuk dipahami
- 3. lebih mudah dalam menuangkan ide/hasil pemikiran

Pseudocode sering digunakan dalam buku-buku tentang ilmu komputer ataupun publikasi ilmiah untuk menjelaskan urutan proses atau metode tertentu. Seorang programmer yang ingin menerapkan algoritma tertentu, terutama yang kompleks atau algoritma baru, biasanya akan memulainya dengan membuat deskripsi dalam bentuk *pseudocode*. Setelah *pseudocode* tersebut jadi, maka langkah selanjutnya hanya tinggal menterjemahkannya ke bahasa pemrograman tertentu. *Pseudocode* ini biasanya disusun dalam bentuk yang terstruktur dengan pendekatan sekuensial (berurutan) dari atas ke bawah.

Algoritma yang menjelaskan tentang proses membuat telur dadar, sebenarnya sudah menerapkan penggunaan *pseudocode*. Sesungguhnya tidak ada suatu standar untuk menyusun algoritma menggunakan *pseudocode*.

Oleh karena *pseudocode* lebih cocok digunakan untuk menyusun algoritma dengan kasus yang besar dan kompleks, maka sangat dianjurkan kepada programmer pemula untuk mulai menggunakan *pseudocode* dalam menyelesaikan masalah. Berikut adalah contoh *pseudocode* yang dibandingkan dengan bahasa pemrograman C++.

Tabel 1.2 Notasi *pseudocode* dan bahasa C++

Pseudocode	C++
if sales > 1000 then	int sales;

```
bonus ← sales * 25%
salary ← 2000000 + bonus
endif
output(salary)

sales=1001;
if (sales > 1000)
{
    bonus = sales * 0.25;
    salary = 2000 + bonus;
}
cout << "Salary : "<<salary;
```

2 Flowchart dan Pseudocode



Overview

Algoritma dapat dituliskan ke dalam berbagai bentuk, namun struktur yang rapi dan mengikuti aturan tertentu akan membuat algoritma lebih mudah untuk dibaca dan dipahami. Selanjutnya, algoritma yang telah tersusun rapi akan diimplementasikan ke bahasa pemrograman.



Tujuan

1. Mengetahui bentuk pengambilan keputusan menggunakan flowchart
2. Mengetahui operasi boolean
3. Mengetahui bentuk pengulangan menggunakan flowchart
4. Memahami tujuan penggunaan pseudocode dalam menyusun algoritma

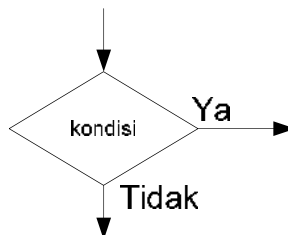
Flowchart

Seperti telah dijelaskan pada bab sebelumnya bahwa flowchart digunakan untuk menggambarkan algoritma atau proses. Flowchart disusun menggunakan simbol-simbol, maka dapat memberikan gambaran yang efektif dan jelas tentang prosedur logika.

Dalam hal melakukan koreksi atau analisis dari suatu permasalahan, flowchart dapat dengan mudah untuk dilihat dan dikomunikasikan. Hal ini dikarenakan flowchart disusun atas simbol-simbol yang mengikuti suatu standar tertentu.

Pengambilan Keputusan

Pengambilan keputusan perlu dilakukan apabila harus menentukan satu pilihan dari (minimal) dua pilihan yang ada. Dalam hal mengambil keputusan, perlu diketahui kondisi yang sedang dihadapi. Kondisi ini bisa berupa pernyataan boolean atau proses perbandingan. Dalam flowchart, simbol yang digunakan untuk pengambilan keputusan adalah berbentuk belah ketupat.



Gambar 2.1 Simbol pengambilan keputusan

Simbol pengambilan keputusan hanya memiliki satu buah input dan dua buah output yang digunakan untuk memfasilitasi hasil

dari pengujian kondisi, yaitu "Ya" atau "Tidak", "True" atau "False".

Dalam melakukan pengujian kondisi, terdapat beberapa notasi yang dapat digunakan, misalnya menggunakan notasi relasional berikut :

Tabel 2.1 Notasi relasional

>	Lebih besar dari
<	Kurang dari
≥	Lebih besar atau sama dengan
≤	Kurang dari atau sama dengan
<>	Tidak sama dengan

Dalam proses pengambilan keputusan, kadang kala terdapat beberapa syarat sekaligus. Untuk menangani hal ini dapat digunakan ekspresi aljabar boolean. Aljabar boolean merupakan kalkulus logika yang digunakan untuk menentukan nilai kebenaran dari suatu ekspresi logika [10]. Teknik aljabar ini dikembangkan oleh George Boole pada tahun 1930an, sebagai penghargaan atas penemuannya maka aljabar ini diberi nama sesuai dengan nama belakang beliau.

Dalam aljabar boolean terdapat tiga buah operasi dasar, yaitu : AND, OR, NOT ketiga-tiganya dapat digunakan secara independen atau dapat digunakan sekaligus. Keluaran (output) dari aljabar ini adalah nilai benar (TRUE) atau salah (FALSE).

Berikut ini adalah tabel yang menunjukkan ketiga hasil operasi aljabar boolean :

Tabel X AND Y

X	Y	X AND Y
---	---	---------

T	T	T
T	F	F
F	T	F
F	F	F

Tabel X OR Y

X	Y	X OR Y
T	T	T
T	F	T
F	T	T
F	F	F

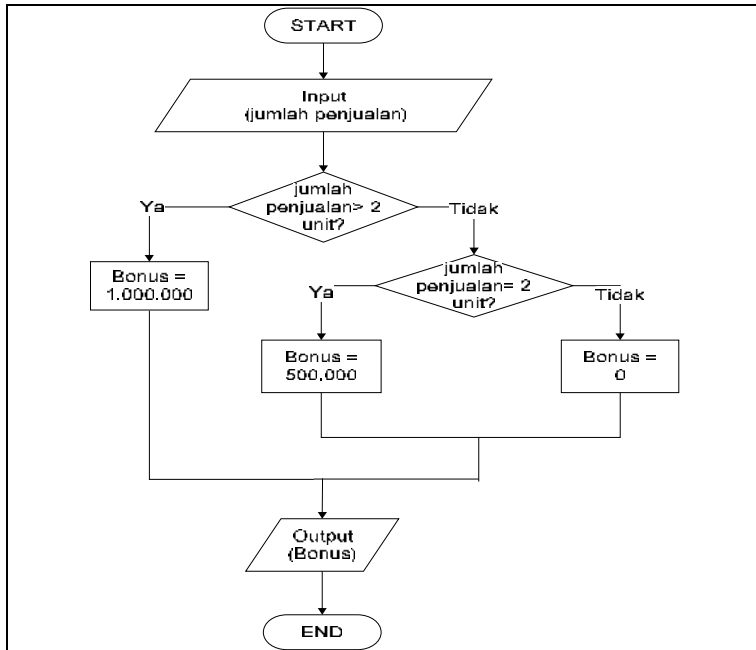
Tabel NOT X

X	NOT X
T	F
F	T

Contoh 2.1

Pemimpin sebuah perusahaan otomotif perlu menentukan besarnya bonus yang akan diberikan kepada para pegawainya yang bekerja sebagai account executive. Jika terdapat pegawai yang dalam bulan ini telah menjual mobil lebih dari dua unit, maka akan mendapatkan bonus sebesar Rp 1.000.000,- kemudian pegawai yang bisa menjual mobil tepat dua buah maka, akan mendapatkan bonus Rp 500.000,- namun jika pegawai yang dalam bulan ini penjualannya kurang dari dua unit maka, pegawai tersebut tidak mendapatkan bonus.

Jika kita gambarkan persoalan di atas menggunakan flowchart maka, akan menjadi seperti berikut :



Gambar 2.2 Flowchart penghitungan bonus

Pengulangan Proses

Dalam beberapa kasus, seringkali terdapat proses yang harus dilakukan secara berulang-ulang, sebagai contoh yang paling sederhana adalah proses berjalan kaki. Untuk bisa mencapai tujuan, kita harus melangkahkan kaki secara berulang-ulang supaya dapat menempuh jarak tertentu dan akhirnya sampai tujuan.

Pada kasus yang berhubungan dengan pengolahan informasi menggunakan komputer, terdapat proses-proses yang harus dilakukan secara berulang, mulai dari input data, proses dan output. Program yang baik adalah program yang bisa mengoptimalkan kinerja komputer, dengan cara menggunakan kembali program atau sekumpulan program dengan proses tertentu. Atau dengan kata lain terdapat bagian program yang

dapat dipanggil/digunakan secara berulang-ulang. Hal ini akan mempermudah pekerjaan programmer dalam menghasilkan solusi.

Contoh 2.2

Seorang staff IT diminta untuk menampilkan data dari sebuah tabel dimana di dalamnya terdapat seratus baris data. Jika staff tersebut harus menampilkan satu per satu, tentunya akan membutuhkan banyak kode program dan program akan menjadi tidak efektif. Bagaimana cara menyelesaikan persoalan staff IT tersebut?

Solusi:

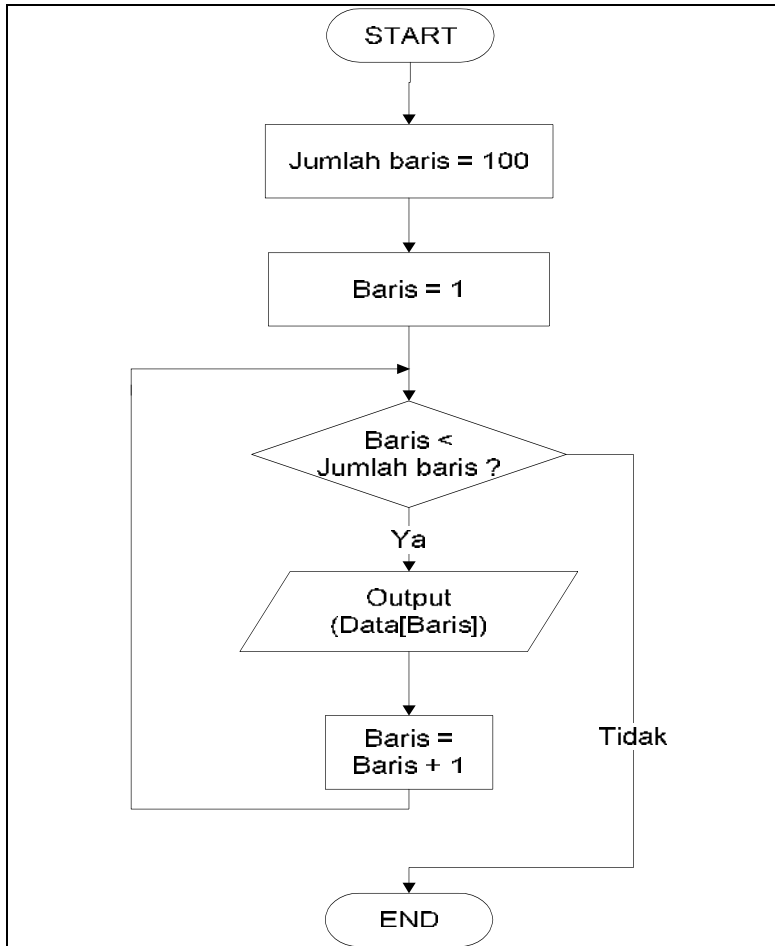
Dalam kasus ini yang diminta adalah bagaimana menampilkan data sebanyak 100 baris tanpa harus menggunakan proses output sebanyak 100 kali. Metode yang digunakan adalah pengulangan.

Dalam proses pengulangan terdapat 3 (tiga) hal penting, yaitu:

1. Inisialisasi (penentuan kondisi/nilai awal)
2. Proses
3. Kondisi berhenti

Untuk kasus menampilkan data, dapat ditentukan bahwa jumlah baris yang akan dibaca adalah 100. Baris akan dibaca mulai dari baris pertama (baris = 1). Proses yang dilakukan adalah membaca dan menampilkan isinya ke layar (output). Pembacaan akan berhenti jika baris yang dibaca sudah mencapai baris ke-100.

Jika digambarkan menggunakan flowchart maka, akan tampak sebagai berikut:



Gambar 2.3 Flowchart untuk menampilkan 100 baris data

Dari gambar dapat dilihat bahwa proses output data hanya muncul satu kali, namun karena proses output dilakukan secara berulang-ulang maka, pembacaan terhadap 10 baris data dapat dilakukan.

Pseudocode

Pada bab 1 telah dijelaskan sebelumnya mengenai keuntungan dalam menuangkan logika dan algoritma menggunakan *pseudocode*. Dalam menyelesaikan kasus yang besar dan kompleks, misalnya membuat aplikasi untuk menangani proses bisnis sebuah perusahaan maka, yang paling cocok digunakan dalam menuliskan algoritma adalah *pseudocode*.

Sesungguhnya tidak ada aturan baku dalam penulisan *pseudocode*, namun karena banyaknya bahasa pemrograman yang beredar saat ini maka, aturan penulisan *pseudocode* diarahkan untuk menyerupai aturan penulisan bahasa pemrograman tertentu. Dalam buku ini akan digunakan aturan penulisan *pseudocode* yang mendekati bahasa pemrograman Pascal.

Struktur algoritma

Struktur algoritma yang digunakan mengacu pada struktur pemrograman bahasa Pascal yang terdiri dari 3 (tiga) bagian, yaitu :



Gambar 2.4 Struktur program

Pada bagian **Judul**, digunakan sebagai tempat untuk mencantumkan nama atau judul program. Terdapat aturan penulisan judul, yakni:

1. Tidak diawali dengan angka atau karakter selain alphabet
2. Tidak terdapat karakter spasi atau karakter selain alphabet kecuali karakter underscore '_' (sebagai pengganti karakter spasi).

Contoh:

Algoritma berhitung;	Benar
Algoritma konversi bilangan;	Salah
Algoritma perhitungan_pajak;	Benar
Algoritma 2bilangan;	Salah
Algoritma *kecil;	Salah

Pada bagian **deklarasi**, digunakan sebagai tempat untuk mencantumkan variabel, konstanta, dan *record*. Mengingat cara eksekusi kode program dilakukan berurut dari atas ke bawah maka, deklarasi diletakkan di awal program setelah bagian judul. Hal-hal yang dideklarasikan pada bagian ini digunakan sebagai 'reservasi' alokasi *memory* untuk penyimpanan data dan akan digunakan selama program bekerja.

Pada bahasa pemrograman Pascal, bagian deklarasi juga berfungsi untuk mendeklarasikan nama function dan procedure.

Contoh:

```
Algoritma Coba;  
Kamus data  
  x : integer;  
  s : string;  
...
```

Pada bagian **badan program**, digunakan untuk meletakkan semua algoritma atau kode-kode program. Bagian

ini diawali dengan 'BEGIN' dan diakhiri dengan 'END'. Semua algoritma atau kode program wajib dituliskan diantara kedua penanda tersebut.

Contoh:

```
Algoritma Hello
Kamus data
  s : string
BEGIN
  s ← "Halo!"
  output(s)
END.
```

Tanda awal
algoritma

Tanda akhir
algoritma

Input dan Output

Dalam mengawali suatu proses tertentu, minimal membutuhkan suatu masukan berupa data (*input*), karena data inilah yang nantinya akan diproses dan akan menjadi keluaran (*output*).

Contoh :

Menerima masukan data dari user (pengguna)

```
Algoritma Masukkan_data
Kamus data

BEGIN
  input(x) /*x adalah variabel penampung nilai*/
END.
```

Memasukkan nilai tertentu pada variabel

```
Algoritma Masukkan_nilai
Kamus data

BEGIN
  x ← 5 /*panah ke kiri arah masuknya nilai*/
END.
```

Menampilkan isi variabel ke layar monitor

```
Algoritma Tampilan
```

```
Kamus data
```

```
BEGIN
```

```
    output(x) /*x adalah variabel yang berisi nilai*/
```

```
END.
```



Rangkuman

1. Flowchart digunakan untuk menggambarkan algoritma atau proses
2. Dalam melakukan pengujian kondisi digunakan notasi relasional
3. Simbol dalam flowchart yang digunakan untuk pengambilan keputusan adalah belah ketupat
4. Aljabar boolean merupakan kalkulus logika yang digunakan untuk menentukan nilai kebenaran dari suatu ekspresi logika
5. Program yang baik adalah program yang bisa mengoptimalkan kinerja komputer, dengan cara

menggunakan kembali program atau sekumpulan program dengan proses tertentu

6. Pseudocode cocok digunakan untuk menuliskan algoritma dengan kasus yang kompleks dan berskala besar

3 Tipe Data, Operator dan Runtunan



Overview

Tipe data, operator, dan runtunan merupakan suatu kesatuan konsep yang paling mendasar didalam pemrograman komputer, karena tipe-tipe data dasar dan operator dapat membentuk berbagai macam ekspresi yang akan digunakan dalam program.

Sedangkan runtunan merupakan konsep dasar yang dapat memberikan gambaran tentang cara kerja sebuah program dalam komputer atau dengan kata lain adalah urutan peng-eksekusian perintah pada satu argumen.



Tujuan

1. Mengetahui dan membedakan tipe-tipe data dasar
2. Mengetahui penggunaan tipe-tipe data dasar dalam program
3. Mengetahui operator dan penggunaannya dalam program
4. Mengetahui konsep runtunan dalam program

3.1 Tipe Data Dasar

Tipe data adalah himpunan nilai yang dapat dimiliki oleh sebuah data. Tipe data menentukan apakah sebuah nilai dapat dimiliki sebuah data atau tidak, serta operasi apa yang dapat dilakukan pada data tersebut. Contoh tipe data dalam dunia nyata adalah bilangan bulat. Jika sebuah data, misalnya umur, harus berupa bilangan bulat maka dapat dipastikan bahwa 25, 13, 7 dapat menjadi nilai umur, sedangkan 7.5, 19.655 bukan merupakan contoh dari nilai umur.

Contoh bilangan bulat ini dapat kita lihat dalam kasus sehari-hari –khususnya dalam hal pencacahan (ingat kembali bilangan cacah : 1,2,3,4,..... yang merupakan himpunan bagian dari himpunan bilangan bulat). Misalnya jumlah siswa dalam kelas ada 20. 20 adalah bilangan bulat. Tidak akan ditemukan pernyataan : jumlah siswa dalam kelas ada 20.5. Contoh yang lain adalah jumlah mobil yang diparkir di tempat parkir. Kita akan menggunakan bilangan bulat dalam kasus ini. Tidak pernah akan kita gunakan angka 50,33 atau 3π atau $40/7$ sebagai jumlah dari mobil yang sedang parkir.

Selain itu, misalnya data nama seseorang yaitu 'Bambang Pamungkas' yang merupakan sebuah deretan huruf dan lain sebagainya. Dalam sebuah program, setiap variabel dan konstanta memiliki tipe data yang harus dideklarasikan di awal program. Deklarasi tipe data tersebut bertujuan untuk menentukan besarnya tempat dalam memori yang akan digunakan untuk menyimpan data pada tersebut saat program dijalankan.

Tipe data dasar adalah tipe data yang dapat langsung digunakan. Secara umum terdapat 2 tipe data dasar, yaitu **numerik** dan **kategorik**. Tipe data numerik terdiri atas angka/kumpulan angka serta dapat mengalami operasi perhitungan,

sedangkan tipe data kategorik dapat berupa angka maupun huruf namun tidak dapat mengalami operasi perhitungan.

Berikut merupakan contoh beberapa tipe data dasar :

- **Integer/ bilangan bulat**

Integer adalah tipe data dasar berupa bilangan yang tidak mengandung pecahan desimal. Tipe data ini juga memiliki urutan, sehingga dapat dibandingkan satu dengan lainnya.

Contoh integer: 2 5 -10 135 2008

Secara teoritis, tipe data integer tidak memiliki batasan, yaitu dari minus tak hingga hingga plus tak hingga. Namun dalam pemrograman yang menggunakan bahasa pemrograman C++, secara umum dikenal beberapa macam tipe data integer, yaitu:

Tabel 1. Tipe data integer

Tipe	Ukuran	Nilai
Short	8 bit	-128 .. 127
Int	16 bit	-32768 .. 32767
Long	32 bit	-2147483648 .. 2147483647

- **Real/ bilangan riil**

Real adalah tipe data dasar berupa bilangan yang memiliki pecahan desimal. Dalam pemrograman, nilai dengan tipe data ini harus ditulis dengan sebuah titik sebagai pemisah bilangan utuh dan bilangan pecahannya. Tipe data ini digunakan untuk perhitungan yang melibatkan bilangan pecahan, seperti perhitungan kosinus, akar persamaan, dan sebagainya. Tipe data ini juga memiliki urutan, sehingga dapat dibandingkan satu dengan lainnya.

Contoh real: .5 0.17 -3.465 92.0 4.3000+E9

Secara teoritis, tipe data real juga tidak memiliki batasan, yaitu dari minus tak hingga hingga plus tak hingga. Namun dalam pemrograman, secara umum dikenal beberapa macam tipe data real, yaitu:

Tabel 2. Tipe data real

Tipe	Ukuran	Nilai
float	32 bit	2.9×10^{-39} .. 1.7×10^{38}
Double	48 bit	5.0×10^{-324} .. 1.7×10^{308}

Nilai pada tabel diatas berbeda dengan nilai yang ada pada tabel tipe data integer, pada tabel diatas nilai untuk tipe data merupakan tingkat ketelitian untuk masing-masing tipe data, bukan berdasarkan rentang nilai.

- **Char/ Karakter**

Char adalah tipe data dasar yang terdiri atas satu buah angka, huruf, tanda baca atau karakter khusus. Untuk menyimpan sebuah karakter, diperlukan 1 byte atau 8 bit tempat didalam memori. Dalam sebuah program, penulisan tipe data char diawali dan diakhiri dengan tanda kutip ganda. Selain itu, terdapat sebuah karakter kosong yang disebut dengan *null* atau *nil* dan dituliskan sebagai `""`.

Contoh char: `"5"` `"A"` `"?"` `"+"` `"$"`

Perhatikan bahwa `5` adalah integer sedangkan `"5"` adalah char.

- **String**

String adalah tipe data dasar yang berupa kumpulan karakter dengan panjang tertentu. Meskipun berupa kumpulan karakter, karena tipe data string sering digunakan dalam pemrograman, string dianggap sebagai tipe data dasar. Untuk penyimpanan string didalam memori, dibutuhkan 1 byte untuk tiap karakternya. Serupa dengan penulisan karakter, penulisan sebuah string juga harus diawali dan diakhiri dengan tanda petik ganda. String juga mengenal null yang dituliskan dengan `""`.

Contoh string:

- `"BANDUNG"`
- `"Politeknik Telkom Bandung"`
- `"ABC3456"`

- "Lucu"
- "30202001"
- "z"

Perhatikan bahwa sebuah karakter tunggal ("z") juga merupakan string.

- **Boolean/ bilangan logika**

Sebuah data boolean memiliki tepat dua buah kemungkinan nilai, direpresentasikan sebagai Benar dan Salah, atau True dan False, atau dapat juga dilambangkan dengan 1 dan 0. Tipe data ini dapat digunakan untuk pemilihan dengan kondisi-kondisi tertentu, dimana program harus memilih aksi apa yang akan dijalankan dengan parameter tertentu.

Tipe data ini paling sering digunakan untuk range yang memiliki dua buah nilai: lulus - tidak lulus, member - bukan member,

3.2 Variabel

Variabel atau peubah adalah obyek yang nilainya dapat berubah-ubah dalam sebuah program. Pada saat sebuah variabel dideklarasikan, program 'memesan' tempat dengan ukuran tertentu (sesuai tipe datanya) pada memori untuk menyimpan nilai dari variabel tersebut. Pemrogram dapat memberikan nama pada sebuah variabel untuk mempermudah pemanggilan variabel tersebut di dalam program. Pada saat mendeklarasikan sebuah variabel, pemrogram harus menyebutkan nama variabel dan tipe data dari variabel tersebut.

Dalam bentuk flowchart, deklarasi variabel digambarkan sebagai sebuah proses. Misalnya sebagai berikut:

<pre>x : integer nama : string TB : real</pre>
--

Gambar 1. Contoh deklarasi variabel dalam flowchart

Contoh deklarasi variabel dalam pseduecode :

```
1. KAMUS DATA {awal deklarasi variabel}
2.     x : integer
3.     nama: string
4.     TB : real
5.     jenisKelamin : char
6.     status : boolean
```

Sebelum kita menuliskan beberapa program dalam bahasa C++, ada baiknya kita mengenal terlebih dahulu struktur dan format penulisan program dalam bahasa C++.

```
1. // Contoh Program C++
2. #include <stdio.h>
3. /* Program Utama */
4. main() {
5.     printf("Selamat Datang");
6.     return 0;
7. }
```

Pada contoh program diatas, pada baris pertama dituliskan diawalannya tanda *doubleslash* (`//`). Maksudnya adalah sebagai komentar, artinya baris tersebut tidak akan dieksekusi oleh program. Kita dapat menuliskan apapun setelah tanda tersebut dan berlaku hanya satu baris. Sedangkan untuk penulisan komentar lebih dari satu baris digunakan tanda `/* .. */` dimana komentar dituliskan diantara tanda `/*` dan `*/` seperti tampak pada baris ke 3 dan 4. Biasanya tanda tersebut digunakan oleh programmer untuk memberi penanda atau keterangan pada tiap baris program seperti pada baris 5.

Pada baris kedua terdapat code `#include <stdio.h>`, yang diawali dengan tanda crash (`#`). Ini dapat kita sebut dengan *preprocessor directive*. *preprocessor directive* merupakan perintah-perintah untuk memberitahukan kepada compiler untuk melakukan berbagai macam definisi seperti

menggunakan (*include*) file library misalnya `stdio.h`, karena didalam file tersebut mengandung beberapa fungsi yang akan digunakan didalam program.

Sedangkan pada baris ke 5 – 8 merupakan isi dari program. Pada baris ke 5 terdapat instruksi `main()` dimana pada baris tersebut merupakan fungsi utama atau program utama. Maksudnya adalah pada baris tersebut merupakan penanda awal dari eksekusi sebuah program. Untuk awal instruksi ditandai dengan kurung kurawal. Seperti pada program diatas, pada baris ke-5 (tanda `{`) merupakan awal dari program utama dan berakhir pada baris ke-8.

Pada baris dke-6 (`printf("Selamat Datang")`) merupakan instruksi untuk mencetak tulisan "Selamat Datang" kelayar. Sedangkan pada baris ke-7 (`return 0`) merupakan nilai kembali dari fungsi utama yaitu nilainya adalah 0. Perlu diperhatikan bahwa setiap instruksi pada perogram harus diakhiri dengan tanda *semicolon* (`;`).

Untuk menuliskan variabel, kita dapat menuliskannya pada bagian isi program. Contoh penulisan variabelnya adalah :

```
1. #include <stdio.h>
2.
3. main () {
4.     int x;
5.     string nama;
6.     float BB;
7.     char jKelamin;
8.     bool status;
9.     ...
10. }
```

Secara teori, pemrogram dapat memberikan nama apapun pada sebuah variabel karena penamaan variabel bertujuan untuk memudahkan pemanggilan kembali. Namun, ada beberapa panduan yang biasa diacu pemrogram dalam penamaan variabel, antara lain:

- Huruf pertama pada nama variabel menunjukkan tipe data dari variabel.

Contoh: diawali dengan 'c' untuk variabel char, 'i' untuk integer, 's' untuk string, dan seterusnya. Panduan penamaan ini disebut dengan *Charles Simyoni Hungarian Notation*.

- Nama variabel harus cukup jelas menunjukkan tujuan penggunaan variabel tersebut.
Contoh: sNama adalah variabel string untuk menyimpan nama, cJenisKelamin adalah variabel char untuk menyimpan jenis kelamin, bStatus adalah variabel boolean untuk menyimpan status.
- Nama variabel tidak boleh mengandung spasi kosong atau karakter khusus ! @ # \$ % ^ & * () { } [] ' " ; : < > , . / ? | dan \. Beberapa pemrogram menggunakan '_' untuk memisahkan kata di nama variabel.
Contoh: cJenis_kelamin, sNama_orang_tua, iNilai_akhir
- Cara lain untuk memisahkan kata dalam nama variabel adalah dengan memberikan huruf besar di awal tiap kata.
Contoh: cJenisKelamin, sNamaOrangTua, iNilaiAkhir

Setelah sebuah variabel dideklarasikan, variabel dapat menyimpan nilai. Pengisian nilai ke dalam sebuah variabel dalam sebuah program dapat dilakukan dengan 2 cara, yaitu:

- **Secara langsung**

Contoh:

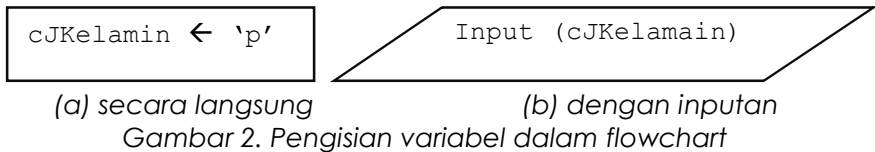
```
- cJenisKelamin = 'P'  
- sNamaOrangTua = 'Jeremy Thomas'  
- iNilaiAkhir = 99
```

- **Dengan inputan**

Contoh:

```
- Input (cJenisKelamin)  
- Input (sNamaOrangTua)  
- Input (iNilaiAkhir)
```

Penggunaan kedua cara pengisian nilai variabel ini akan diperjelas pada bab-bab selanjutnya. Dalam flowchart, pengisian nilai ke dalam variabel ditunjukkan pada gambar 2.



Contoh program untuk memberikan nilai pada sebuah variabel :

```
1. #include <stdio.h>
2. main() {
3.     int lA,lB;
4.     String NamaA, NamaB;
5.     // Pengisian secara Langsung
6.     lA = 20;
7.     NamaA = "Joko Handono";
8.     // Pengisian dengan Inputan
9.     scanf("%i",&lB);
10.    scanf("%s",&NamaB);
11.    // Menampilkan Kelayar
12.    printf("Nilai lA : %i",lA);
13.    printf("Nilai lB : %i",lB);
14.    printf("Nilai NamaA : %s",NamaA);
15.    printf("Nilai NamaB : %s",NamaB);
16. }
```

Pada contoh program diatas, kita melihat ada tanda "%i" dan "%s". Fungsi tanda tersebut adalah untuk menkonversi nilai inputan menjadi tipe yang sesuai dengan yang diterima atau mengubah nilai dari tipe data dasar menjadi tipe karakter untuk ditampilkan di layar. Karena pada dasarnya, dalam pemrograman bahasa C++ nilai input atau nilai yang dapat ditampilkan berupa karakter. Sedangkan didalam program, nilai tersebut harus sesuai dengan tipe data yang dideklarasikan. Sebagai contoh pada baris ke-10, variabel "lB" tipe datanya adalah integer. Untuk mengubah tipe masukan menjadi integer, maka digunakan "%i". Biasanya, string tersebut diawali dengan

huruf pertama tipe datanya, misalnya `float -> %f`, `String -> %s` dan seterusnya. Khusus untuk inputan, nama variabelnya harus diawali dengan string "&" seperti tampak pada baris ke 10 dan 11.

3.3 Konstanta

Pada variabel, nilai yang disimpan dapat berubah-ubah selama program dijalankan. Sedangkan pada konstanta, nilai yang disimpan tetap dan tidak dapat diubah sejak dideklarasikan hingga program berakhir..

Setelah sebuah konstanta dideklarasikan, konstanta dapat digunakan dalam program dan nilainya selalu tetap. Deklarasi konstanta dalam flowchart digambarkan sebagai sebuah proses. Misalnya:

```
iMaks = 100
fPi = 3.14
sSapa = 'Hello'
```

Gambar 3. Deklarasi konstanta dalam flowchart

Cara penulisan konstanta didalam program, di tulis dengan diawali dengan tanda crash (#) kemudian diikuti dengan `define`, selanjutnya nama konstantanya dan selanjutnya nilainya dan ditulis diluar program utama setelah pendeklarasian librari namespace. Contoh penulisannya adalah sebagai berikut :

```
1. #include <stdio.h>
2. #define iMaxs 100
3. #define fPi 3.14159
4. #define sSapa 'Hello'
5. #define newLine '\n'
6. main() {
7.     ...
8. }
```

3.4 Operator

Operator adalah pengendali operasi yang akan dilakukan pada beberapa operan sehingga membentuk sebuah ekspresi.

Secara umum, dalam sebuah ekspresi terdapat sebuah operator yang diapit dua operan. Contohnya pada ekspresi:

$x + y$

x dan y adalah operan, sedangkan

'+' adalah operatornya

Terdapat tiga macam operator yang biasa digunakan dalam pemrograman, yaitu:

- **Operator aritmatik**

Operator ini membentuk perhitungan aritmatik. Kedua operan dari operasi aritmatik ini dapat berupa nilai integer atau real.

Operator yang termasuk tipe ini adalah:

Tabel 3. Operator aritmatik

Lambang	Deskripsi	Contoh
+	Penjumlahan	$x = y + z$
-	Pengurangan	$x = y - z$
*	Perkalian	$x = y * z$
/	Pembagian	$x = y / z$
%	Modulo (sisanya)	$x = y \% z$

Output dari operasi aritmatik akan memiliki tipe data yang sama dengan tipe data kedua operannya. Misalnya, jika sebuah bilangan integer dijumlahkan dengan bilangan integer lainnya maka outputnya adalah bilangan integer juga. Selain itu perlu diperhatikan pula bahwa sebuah operator aritmatik tidak dapat diterapkan pada dua bilangan dengan tipe data yang berbeda.

Contoh program dengan operasi aritmatik:

```
1 //Program Aritmatik
2 /* IS:Tersedia dua buah bilangan integer
   FS:Hasil Modulo dua buah bilangan */
3 #include <stdio.h>
4
5 main () {
6     // Deklarasi Variabel
7     int iTambah;
```

```
8      int iAngka1, iAngka2;
9      printf("Masukan Bilangan Pertama : ");
10     scanf("%i", iAngka1);
11     printf("Masukan Bilangan Kedua   : ");
12     scanf("%i", iAngka2);
13     // Penjumlahan
14     iTambah = iAngka1 + iAngka2;
15     printf("Hasil Penjumlahan %i + %i = %i",
16           iAngka1, iAngka2, iTambah);
17     return 0;
18 }
```

Program di atas akan mengembalikan nilai hasil penjumlahan sesuai dengan inputan. Misalnya pada inputan pertama kita masukan 10 dan yang kedua kita masukan 23 maka hasilnya adalah 33. outputnya adalah:

```
Masukan Bilangan Pertama : 10
Masukan Bilangan Kedua   : 23
Hasil Penjumlahan 10 + 23 = 33
```

• **Operator Assignment**

Dalam pemrograman bahasa C++, Operator ini digunakan memasukan nilai kedalam sebuah variabel, tanpa menghilangkan atau mengosongkan nilai variabel sebelumnya. Contoh penggunaan operator ini adalah sebagai berikut :

Tabel 4. Operator relasional

Lambang	Deskripsi	Contoh
+=	Menambahkan	x += 1
-=	Mengurangkan	x -= 1
*=	Mengalikan	x *= 2
/=	Membagi	x /= 2
%=	Mem-mod	x %= 2

• **Increase and decrease**

Penulisan ini dilambangkan dengan ++ (Increade) dan -- (decrease). Operator ini berfungsi untuk menaikkan atau

menurunkan satu satuan nilai pada sebuah variabel. Contoh penggunaannya adalah pada contoh dibawah ini :

```
1 ...
2 a++;
3 a += 1;
4 a = a + 1;
5 ...
```

Ada dua macam penulisan operator ini, yaitu simbol dapat ditulis sebelum nama variabel dan setelah variabel. Adapun perbedaan antara keduanya adalah :

1 B = 3;	1 B = 3;
2 A = ++B;	2 A = B++;
3 // A = 4, B = 4	3 // A = 3, B = 4

• **Operator relasional**

Operator ini membandingkan dua operan dan hasilnya berupa nilai boolean (BENAR atau SALAH). Operasi relasional dapat dilakukan pada dua nilai dengan tipe data yang sama: tipe data integer, riil, char, string, maupun boolean. Berikut ini adalah operator relasional:

Tabel 4. Operator relasional

Lambang	Deskripsi	Contoh
==	Sama dengan	x == y
!=	Tidak sama dengan	x != y
>	Lebih dari	x > y
<	Kurang dari	x < y
>=	Lebih dari atau sama dengan	x >= y
<=	Kurang dari atau sama dengan	x <= y

Contoh penggunaan operator relasional dalam algoritma:

```
1 // Program Operator Relasional
2 KAMUS DATA {awal deklarasi variabel}
3   iAngka1, iAngka2 : integer
4 BEGIN {awal algoritma}
5   iAngka1 = 6 {pengisian variabel langsung}
6   Input(iAngka2) {pengisian dgn inputan}
```



```
7      IF (iAngka1 <> iAngka2) THEN
8          Output ('Tebakan Anda salah')
9      ELSE
10         Output ('Horee! Tebakan Anda benar')
11     ENDIF
12 END
```

Output dari operasi relasional bertipe boolean (*true/ false*). Pada contoh di atas, *iAngka1 != iAngka2* bernilai benar/ *true* jika *iAngka1* tidak sama dengan *iAngka2* *iAngka1 != iAngka2* bernilai salah/ *false* jika *iAngka1* sama dengan *iAngka2* Program di atas akan mengeluarkan pesan sesuai inputan pengguna. Jika pengguna menginputkan angka selain 6 (*'iAngka1 != iAngka2'* bernilai benar), program akan mengeluarkan pesan 'Tebakan Anda salah'. Jika pengguna menginputkan angka 6 (*'iAngka1 != iAngka2'* bernilai salah), program akan mengeluarkan pesan 'Horee! Tebakan Anda benar'.

• **Operator logika**

Operator logika adalah operator yang digunakan untuk mengkombinasikan hasil ekspresi yang mengandung operator relasional.

Tiga macam operator logika adalah:

Tabel 5. Operator logika

Lambang	Deskripsi	Contoh
&&	And / Dan	$x > 7 \ \&\& \ x = y$
	Or / Atau	$x \neq y \ \ x > 3$
!	Not / Tidak	$! (x > y)$

Pola penggunaan operator logika adalah:
ekspresi1 **OPERATOR** *ekspresi2*

Output dari penggunaan operator AND dan OR adalah sebagai berikut:

Tabel 6. Output operator logika

ekspresi1	ekspresi2	kombinasi dengan	
		AND	OR
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

Pola yang mudah untuk mengingat output kedua operator logika tersebut adalah: True AND True = True, False OR False = False.

Beberapa contoh penggunaan operator logika:

- `(x > 7) && (x = y)`
Jika ternyata nilai x adalah 8 dan y adalah 5, maka
`(8 > 7) && (8 = 5)`
True AND False
False (output operasi)
- `(x != y) || (x > 3)`
Jika ternyata nilai x adalah 4 dan y adalah 4, maka
`(4 != 4) || (4 > 3)`
False OR True
True (output operasi)
- `NOT (x > y)`
Jika ternyata nilai x adalah 3 dan y adalah 3, maka
`NOT (3 > 3)`
`NOT (False)`
True (output operasi)

3.5 Urutan Operasi

Sebuah ekspresi mungkin terdiri atas beberapa operasi sekaligus. Misalnya:

```
iHasil = x * 2 % 2 > y && (x != 3)
```

Untuk menentukan operasi mana yang dilakukan terlebih dahulu daripada operasi lainnya, setiap operator

memiliki level urutan. Level urutan ini terdiri atas lima kelompok, level 1 hingga 5.

Operator yang memiliki level lebih tinggi (ditunjukkan dengan angka yang semakin kecil) akan dioperasikan terlebih dahulu dibandingkan operator lain yang levelnya lebih rendah. Sedangkan pada operator-operator yang berada pada level yang sama, operasi dilakukan secara berurutan dari kiri ke kanan. Hal ini disebut dengan asosiativitas.

Pada beberapa ekspresi diperlukan pengubahan urutan eksekusi operasi-operasi. Untuk memungkinkan pemrogram melakukan hal tersebut, tersedia sebuah operator tambahan yang memiliki level eksekusi paling tinggi, yaitu (). Operasi apapun yang ada dalam tanda kurung () akan dieksekusi pertama kali oleh program.

Level urutan operator-operator tersebut adalah sebagai berikut:

Tabel 7. Urutan operasi

Operator	Deskripsi	Asosiativitas	Level Urutan
()	Tanda kurung		1
!	Logika NOT		2
*	Perkalian	Kiri ke kanan	3
/	Pembagian		
%	Modulo		
+	Penjumlahan	Kiri ke kanan	4
-	Pengurangan		
<	Kurang dari	Kiri ke kanan	5
<=	Kurang dari/ sama dengan		
>=	Lebih dari/ sama dengan		
>	Lebih dari		

=	Sama dengan		
!=	Tidak sama dengan	Kiri ke kanan	6
&&	Logika AND	Kiri ke kanan	7
	Logika OR	Kiri ke kanan	8

Misalnya pada ekspresi berikut ini:

```
iHasil = x * 2 % 2 > y && (x <> 3)
```

Jika inputannya adalah x = 5 dan y = 3 maka urutan pengerjaannya adalah:

Level	Pengerjaan
1	iHasil \leftarrow x * 2 % 2 > y && (5 != 3) iHasil \leftarrow x * 2 % 2 > y && True
3	iHasil \leftarrow 5 * 2 % 2 > y && True iHasil \leftarrow 0 > y && True
5	iHasil \leftarrow 0 > 3 && True iHasil \leftarrow False && True
7	iHasil \leftarrow False

3.6 Runtunan

Secara umum, program akan dibaca dan dieksekusi secara berurutan baris demi baris. Misalnya pada algoritma berikut ini:

```
1. Algoritma Runtunan;
2. {IS:Tersedia empat bilangan yang akan
   dioperasikan
   FS:Mengoutputkan dua bilangan setelah
   dioperasikan }
3. Kamus data
4.   a,b,c,d : integer
5. BEGIN
6.   a  $\leftarrow$  3
7.   b  $\leftarrow$  2
8.   c  $\leftarrow$  a * b
9.   a  $\leftarrow$  5
```

```
10.  d ← a + b
11.  Output (c, d)
12.  END.
```

Maka, output dari algoritma di atas adalah:

6, 7

Perhatikan bahwa pada saat membaca baris ke-3, program akan mengalikan 3 dan 2 (a dan b). Kemudian, saat membaca baris ke-5, program akan menjumlahkan 5 dan 2 (a dan b). Nilai a berubah karena di baris ke-4 variabel a diisi dengan 5. Ini merupakan akibat dari sifat program yang membaca dan mengeksekusi per baris. Setelah baris ke-4 dieksekusi, nilai a yang diisikan pada baris pertama sudah tidak berlaku lagi (tertumpuk dengan nilai baru yang diisikan).

Di bab-bab selanjutnya akan ditunjukkan bahwa sifat program membaca dan mengeksekusi berurut terus per baris ini dapat diubah, dengan memberikannya perintah untuk tidak membaca sesuai urutan. Hal ini dapat dilakukan dengan struktur pemilihan, struktur pengulangan, dan lain-lain.

Jika algoritma runtunan di atas dituliskan dalam bahasa Pascal, maka akan tampak sebagai berikut:

```
1.  // Program Runtunan;
2.  /*IS:Tersedia empat bilangan yang akan
    dioperasikan
    FS:Menampilkan dua bilangan setelah
    dioperasikan */
3.  #include <stdio.h>
4.
5.  main () {
6.      int a,b,c,d;
7.      a = 3;
8.      b = 2;
9.      c = a * b;
10.     a = 5;
11.     d = a + b;
12.     printf("Nilai C : %i",c);
13.     printf("Nilai D : %i",d);
```

```
14. }
```

Jika Program dijalankan, maka hasil keluaran program adalah seperti berikut :

```
Nilai C : 6  
Nilai D : 7
```



Rangkuman

1. Tipe data dasar adalah tipe data yang dapat langsung digunakan, dan memiliki ukuran tertentu sesuai dengan tipe data masing-masing. Beberapa tipe data dasar: integer, real, char, string, dan boolean.
2. Variabel adalah obyek yang nilainya dapat berubah-ubah dalam sebuah program, sedangkan konstanta memiliki nilai yang tetap sejak dideklarasikan hingga program berakhir.
3. Pada saat mendeklarasikan sebuah variabel, pemrogram harus menyebutkan nama variabel dan tipe data dari variabel tersebut.
4. Operator adalah pengendali operasi yang akan dilakukan pada beberapa operan sehingga membentuk sebuah ekspresi. Tiga macam operator dalam pemrograman: operator aritmatik, operator relasional, dan operator logika.
5. Penggunaan beberapa operator sekaligus dalam sebuah ekspresi mengakibatkan perlunya menentukan urutan operasi yang diatur dalam level urutan operator.
6. Operator dengan level yang lebih tinggi akan dieksekusi lebih dahulu daripada operan dengan level yang lebih rendah. Operator-operator yang memiliki level yang sama akan dieksekusi menurut urutan penulisan (dari kiri ke kanan).



Kuis Benar Salah

1. Deklarasi tipe data mempengaruhi besarnya tempat dalam memori yang disediakan untuk menyimpan data tersebut.
2. Tipe data real memiliki cakupan nilai antara 2.9×10^{-39} hingga 1.1×10^{4932} .
3. Penentuan nama variabel dan cara mengaksesnya dilakukan oleh program secara otomatis.
4. Output dari operasi aritmatik akan memiliki tipe data yang sama dengan kedua operannya.
5. Operator relasional dapat membandingkan dua nilai boolean.

(Untuk nomor 6-10) Perhatikan potongan algoritma berikut ini:

```
...
Kamus data
  a, b, d, e : integer
  c, f : real
  g : char
  h : boolean
Konstanta
  d = 100;
BEGIN
  a ← 5
  b ← 3
  c ← 7.0
  a ← a + c
  e ← d / a
  ReadLn (g)
  h ← (g = 'k') OR (d MOD b > 3)
  f ← g + c
  Output (a, b, c, d, e, f, g, h)
END
```


6. Nilai akhir variabel a adalah 12.
7. Operasi $e \leftarrow d / a$ tidak dapat dilakukan sehingga nilai e tidak dapat diketahui.
8. Jika pengguna memberikan input 'k' untuk variabel g, maka nilai akhir h adalah TRUE.
9. Variabel h tidak dapat diketahui nilainya.
10. Output program tersebut akan mengeluarkan empat nilai variabel dan empat error.
11. Tipe data word memiliki cakupan nilai yang lebih luas daripada tipe data integer.
12. Salah satu dari empat tipe data real dapat menyimpan nilai negatif.
13. String adalah kumpulan karakter dengan panjang tertentu.
14. Nilai dari sebuah konstanta tidak dapat diubah dalam badan program.
15. Pengisian nilai ke dalam sebuah variabel dapat dilakukan dengan dua cara, yaitu secara langsung atau dengan inputan.

(Untuk nomor 16-20) Perhatikan ekspresi berikut ini:

Hasil = $(50 - 37) + 50 * 2 \% 35 > 21 / 3 \&\& ! (200 - 7 != 25 * 7 + 18)$

16. Operasi yang pertama kali dieksekusi adalah $(50 - 37)$.
17. Operator yang terakhir dieksekusi adalah '>'.
18. Hasil bertipe integer.
19. Hasil = 116.
20. Hasil = True.



Pilihan Ganda

1. Berikut ini adalah pernyataan-pernyataan yang benar tentang tipe data, kecuali
 - A. Tipe data adalah himpunan nilai yang dapat dimiliki oleh sebuah data.
 - B. Tipe data menentukan operasi apa yang dapat dilakukan pada data tersebut.
 - C. Pemilihan tipe data untuk sebuah variabel atau konstanta harus mempertimbangkan kapasitas memori yang tersedia.
 - D. Tipe data dari setiap variabel dan konstanta harus dideklarasikan di awal program.
 - E. Tipe data dasar dapat langsung digunakan dalam sebuah program.
2. Berikut ini adalah karakter yang sebaiknya tidak digunakan dalam penamaan variabel/ konstanta, kecuali
 - A. '#'
 - B. '/'
 - C. ':'
 - D. '_'
 - E. '<'
3. Perhatikan ekspresi berikut ini:
 $5 + 2 * 3 \geq 100 \text{ MOD } 45 \text{ OR } 75 - 30 / 2 < 10$
Apakah output dari ekspresi tersebut?
 - A. 25
 - B. True
 - C. 11
 - D. False
 - E. 60
4. Perhatikan kedua kalimat tentang urutan eksekusi

operator berikut ini:

Kalimat A: Operasi dalam tanda kurung dieksekusi pertama kali.

Kalimat B: Operasi relasional dieksekusi terlebih dahulu dibandingkan operasi aritmatik.

Manakah pernyataan yang tepat tentang kedua kalimat di atas?

- A. Kalimat A dan kalimat B benar.
- B. Kalimat A dan kalimat B salah.
- C. Kalimat A benar dan kalimat B salah.
- D. Kalimat A salah dan kalimat B benar.
- E. Salah satu dari kalimat A atau kalimat B tidak dapat ditentukan benar/ salah.

5. Perhatikan potongan program berikut ini:

```
#include <stdio.h>

main () {
    int bilangan1, bilangan2;
    int bilangan3, bilangan4;

    bilangan1 := 10;
    bilangan1 := bilangan1 * 2;
    bilangan2 := bilangan1 + 10;
    bilangan3 := bilangan2 * bilangan1;

    printf("%i",bilangan3);
}
```

Manakah yang merupakan output dari program di atas?

- A. 200
- B. 300
- C. 400
- D. 600
- E. 800



Latihan

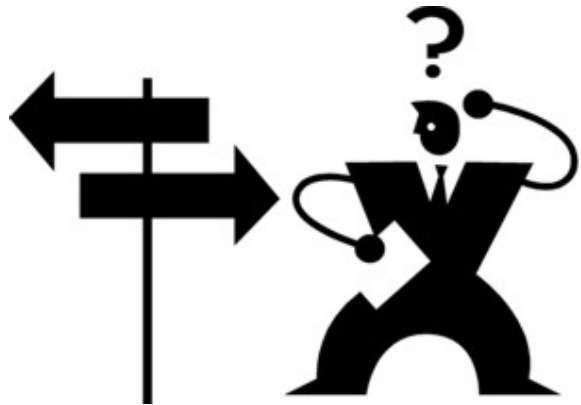
1. Jelaskan perbedaan variabel dan konstanta!
2. Perhatikan ekspresi berikut ini:
 $4 + 7 * 10 - 5 \bmod 13$
Apakah output dari ekspresi di atas?
3. Jelaskan tentang konsep runtunan dalam eksekusi sebuah program!
4. Perhatikan ekspresi berikut ini:
 $x * 3 + (7 \bmod 5) * y$
dengan $x \leftarrow 5$ dan
 $y \leftarrow 3$
Apakah output dari ekspresi di atas?
5. Perhatikan potongan program berikut ini:

```
# include <stdio.h>

main () {
    int iPertama, iKedua;
    iPertama = 50;
    iPertama %= 9;
    iKedua = iPertama - 2;
    printf("%i", iKedua);
}
```

Apakah output dari program di atas?}

4 Pemilihan



Overview

Program dapat merepresentasikan situasi pemilihan yang sering dihadapi dalam dunia nyata. Berdasarkan satu atau beberapa kondisi, dapat ditentukan satu atau sejumlah aksi yang akan dilakukan. Dengan adanya struktur pemilihan, program dapat berjalan dengan jalur yang berbeda, berdasarkan hasil pengecekan kondisi yang dipenuhi.



Tujuan

1. Memahami struktur pemilihan dalam program
2. Mengetahui struktur IF dan CASE yang dapat digunakan

dalam pemilihan

3. Memahami konsep kondisi dan aksi dalam struktur pemilihan
4. Menerapkan pemilihan dalam menyelesaikan berbagai kasus

Dalam kehidupan nyata, seringkali dihadapkan pada beberapa pilihan. Pada saat menghadapi pilihan, satu atau beberapa kondisi menjadi bahan pertimbangan dalam memutuskan untuk melakukan aksi tertentu. Contoh:

Jika cuaca mendung, maka saya membawa payung.

Pada contoh tersebut, 'cuaca mendung' merupakan kondisi yang menjadi bahan pertimbangan untuk melakukan aksi 'saya membawa payung'. Jika kondisi 'cuaca mendung' terpenuhi (bernilai benar), maka aksi 'saya membawa payung' dilakukan

Sebuah program komputer juga dapat mengenali situasi pemilihan. Pernyataan dalam contoh di atas dapat dituliskan dalam struktur pemilihan sebagai berikut:

```
IF cuaca mendung THEN  
    saya membawa payung  
END IF
```

Untuk selengkapnya penggunaan bentuk pemilihan akan dijelaskan berikut ini.

4.1 Bentuk Umum IF dan Variasinya

Bentuk IF yang juga dikenal dengan istilah IF Statement, memiliki bentuk umum sebagai berikut :

```
If kondisi then  
    Aksi-1  
[else  
    Aksi-2]  
End if
```

Kondisi adalah ekspresi boolean yang bernilai benar atau salah, bisa berupa:

Sebuah nilai boolean: true atau false

Sebuah variabel boolean

Sebuah perbandingan data

Dua perbandingan data atau lebih yang digabung

Aksi berupa satu statement beberapa statement, dimana tiap statement

dapat berupa:

Statement pengisian nilai seperti $a \leftarrow 5$

Statement input data

Statement output data

Statement pemilihan (If Statement atau Case Statement)

Statement pengulangan (For, Repeat atau While Statement)

[else Aksi-2], tanda [] menyatakan opsional (boleh ada/tidak ada),

dimana kalau tidak ada, berarti setelah Aksi-1 langsung selesai.

Dari bentuk umum yang telah dijelaskan, maka variasi bentuk IF ini banyak dan tidak berhingga. Di antaranya yang penting dapat disebutkan berikut:

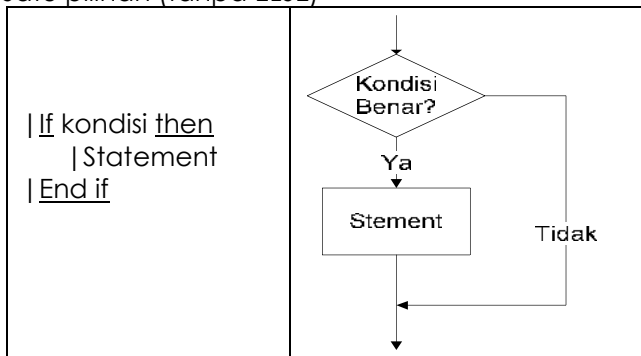
- if tanpa else (satu pilihan, mengerjakan atau tidak)
- if dengan else (dua pilihan)
- if bersarang dimana dalam if ada if lagi, karena Statement dapat berupa

satu perintah pemilihan. Salah satu bentuk if bersarang adalah if untuk

memilih salah satu dari banyak pilihan.

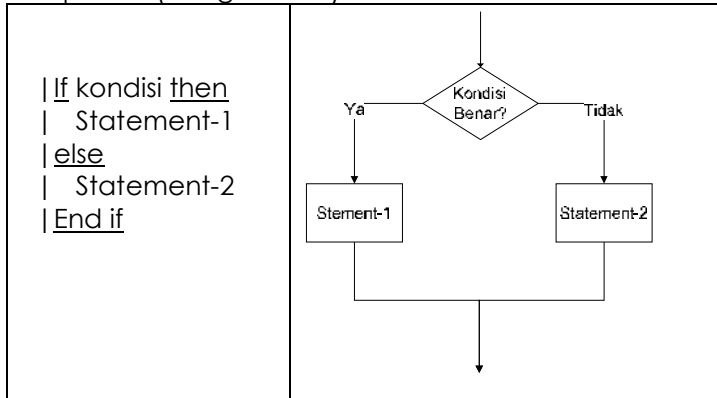
Contoh-contoh variasi:

1. Satu pilihan (tanpa ELSE)



Pada variasi ini, apabila kondisi bernilai benar maka Statement dikerjakan dan apabila kondisi bernilai salah maka Statement tidak dikerjakan.

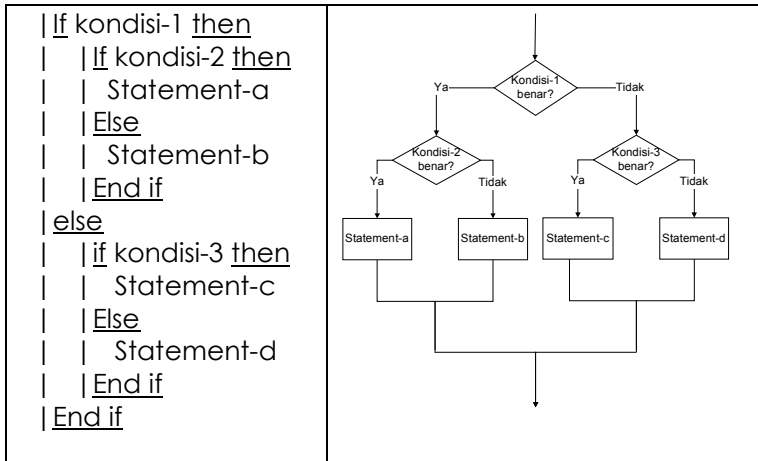
2. Dua pilihan (dengan ELSE)



Pada variasi ini, apabila kondisi bernilai benar maka Statement-1 yang dikerjakan dan apabila kondisi bernilai salah maka Statement-2 yang dikerjakan (tidak pernah 2 statement ini dikerjakan semua).

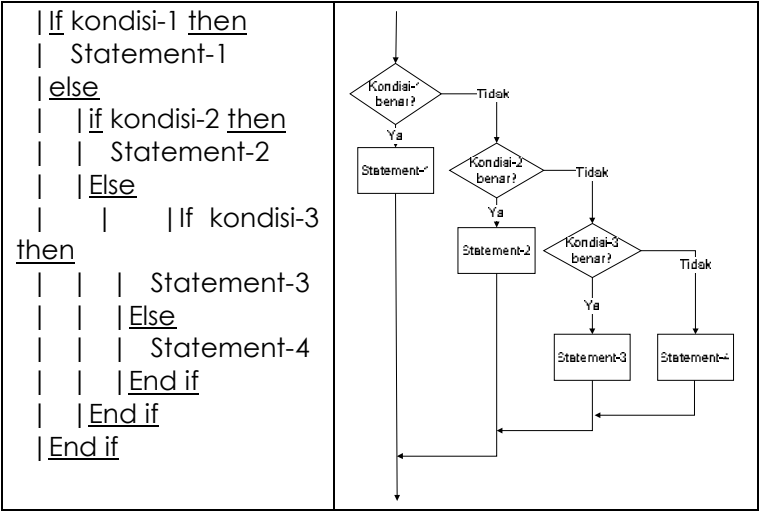
3. Tiga pilihan atau lebih, misal Statement-1 dan Statement-2 pada contoh-2

dikembangkan menjadi bentuk if lagi sehingga jadi 4 pilihan:

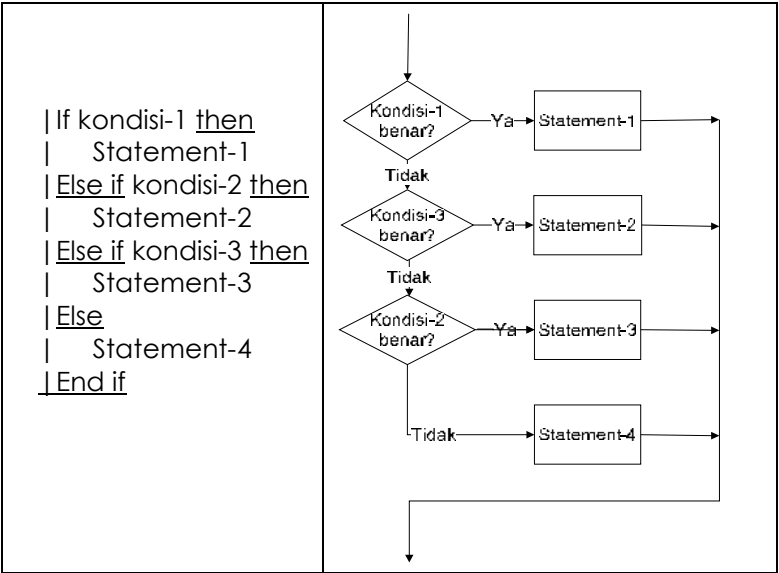


Pada variasi ini, apabila kondisi-1 bernilai benar maka dilanjutkan memeriksa kondisi-2. Apabila kondisi-2 bernilai benar maka Statement-a yang dikerjakan dan apabila kondisi-2 bernilai salah maka Statement-b yang dikerjakan. Sedangkan apabila kondisi-1 salah maka dilanjutkan memeriksa kondisi-3. Apabila kondisi-3 bernilai benar maka Statement-c yang dikerjakan dan apabila kondisi-3 bernilai salah maka Statement-d yang dikerjakan. (dari 4 statement yang ada hanya salah satu yang dikerjakan).

4. Tiga pilihan atau lebih, dengan mengembangkan Statement setelah ELSE



5. Penyederhanaan penulisan contoh-4
(posisi statement tidak semakin ke kanan, END IF hanya satu kali)



Pada variasi 4 dan 5 ini, apabila kondisi-1 bernilai benar maka Statement-1 yang dikerjakan dan selesai, tetapi apabila kondisi-1 bernilai salah maka dilanjutkan memeriksa kondisi-2. Apabila kondisi-2 bernilai benar maka Statement-2 yang dikerjakan dan selesai, tetapi apabila kondisi-2 bernilai salah maka dilanjutkan memeriksa kondisi-3. Apabila kondisi-3 bernilai benar maka Statement-3 yang dikerjakan, tetapi apabila kondisi-3 bernilai salah maka Statement-4 yang dikerjakan. (dari 4 statement yang ada hanya salah satu yang dikerjakan. Penggunaan variasi 5 lebih baik, karena tulisan **tidak makin ke kanan** walaupun **pilihan semakin banyak**.

6. Bentuk bebas (jumlahnya tak terhingga), salah satunya

```
| If kondisi-1 then
|   | Statement-a
|   | If kondisi-2 then
|   |   | If kondisi-3 then
|   |   |   | Statement-b
|   |   |   | Else
|   |   |   | If kondisi-4 then
|   |   |   |   | Statement-c
|   |   |   |   | Else
|   |   |   |   | Statement-d
|   |   |   | End if
|   |   | End if
|   | Else
|   |   | Statement-e
|   |   | End if
|   | Statement-f
| Else
|   | Statement-g
|   | if kondisi-5 then
|   |   | Statement-h
|   |   | End if
|   | if kondisi-6 then
```

```
| | Statement-i
| | End if
| | Statement-j
| End if
```

4.2 Terapan bentuk-bentuk IF

Sebuah masalah terkadang dapat diselesaikan dengan berbagai cara, seperti penggunaan “if tanpa else” dan “if dengan else”. Sebagai contoh dapat dilihat pada kasus berikut:
Kasus 4.1 : Menentukan apakah bilangan yang diinput positif atau negatif
Solusi : ada beberapa cara berikut

Solusi-1 Input(bil) If (bil>=0) then Output('positip') Else Output('negatip') End if	Solusi-2 Input(bil) If (bil<0) then Output('negatip') Else Output('positip') End if
Solusi-3 Input(bil) Ket ← 'positip' If (bil<0) then Ket ← 'negatip' End if Output(Ket)	Solusi-4 Input(bil) Ket ← 'negatip' If (bil>=0) then Ket ← 'positip' End if Output(Ket)
Solusi-5 Input(bil) If (bil>=0) then Output('positip') End if If (bil<0) then Output('negatip') End if	Solusi-6 Input(bil) If (bil<0) then Output('negatip') End if If (bil>=0) then Output('positip') End if
Solusi-7 Input(bil)	Solusi-8 Input(bil)

<pre> positif ← bil >= 0 If (positif=true) then Output('positif') else Output('negatif') End if </pre>	<pre> positif ← bil >= 0 If (positif) then Output('positif') else Output('negatif') End if </pre>
---	--

Ulasan dari beberapa solusi:

Solusi-1 dan Solusi-2 adalah solusi yang sama, digunakan kondisi berkebalikan sehingga posisi perintah tampilan ditukar.

Solusi-3 dan Solusi-4 juga sama, keduanya menggunakan "if tanpa else", dengan cara variabel Ket diinisialisasi (diberi nilai awal) dengan salah satu kemungkinan hasilnya, kemudian diubah bila memenuhi kondisi.

Solusi-5 dan Solusi-6 juga sama, pada solusi ini dibuat 2 buah "if tanpa else" secara terpisah. Dengan cara ini, berarti akan dilakukan pemeriksaan kondisi 2 kali (padahal sebenarnya cukup satu kali).

Solusi-7 dan Solusi-8 keduanya menggunakan variabel bertipe boolean bernama positif untuk mencatat hasil perbandingan $bil \geq 0$. Penulisan "if (positif=true)" sama saja dengan menuliskan "if (positif)" cara yang terakhir lebih cepat waktu eksekusinya.

Berikut beberapa kasus yang lain:

Kasus 4.2 : Terbesar dari 3 bilangan

<p>Solusi-1</p> <pre> Input(A,B,C) <u>If</u> (A>B) <u>then</u> <u>If</u> (A>C) <u>then</u> Output('terbesar =',A) <u>Else</u> Output('terbesar =',C) <u>End if</u> <u>else</u> <u>if</u> (B>C) <u>then</u> Output('terbesar =',B) </pre>	<p>Solusi-2</p> <pre> Input(A,B,C) <u>If</u> (A<B) <u>then</u> <u>If</u> (B<C) <u>then</u> Output('terbesar =',C) <u>Else</u> Output('terbesar =',B) <u>End if</u> <u>else</u> <u>if</u> (A<C) <u>then</u> Output('terbesar =',C) </pre>
---	---

<pre> Else Output('terbesar =',C) End if End if </pre>	<pre> Else Output('terbesar =',A) End if End if </pre>
<p>Solusi-3</p> <pre> Input(A,B,C) If (A>B and A>C) then Output('terbesar =',A) Else if (B>A and B>C) then Output('terbesar =',B) Else if (C>A and C>B) then Output('terbesar =','C) End if End if End if </pre>	<p>Solusi-4</p> <pre> Input(A,B,C) If (A>B and A>C) then Output('terbesar =',A) Else if (B>A and B>C) then Output('terbesar =',B) Else if (C>A and C>B) then Output('terbesar =',C) End if </pre>
<p>Solusi-5</p> <pre> Input(A,B,C) Max ← A If (B>Max) then Max ← B End if If (C>Max) then Max ← C End if Output('terbesar = ',Max) </pre>	<p>Solusi-6</p> <pre> Input(A,B,C) If (A>B) then Max ← A else Max ← B End if If (C>Max) then Max ← C End if Output('terbesar = ',Max) </pre>

Ulasan dari beberapa solusi:

Solusi-1,Solusi-2 dan Solusi-3 menggunakan 3 buah kondisi dan setiap hasil yang didapat akan melalui pemeriksaan 2 buah kondisi.

Solusi-4 menggunakan kondisi yang terdiri dari 2 perbandingan, dengan rata-rata melakukan pemeriksaan 2 kondisi (4 perbandingan)

Solusi-5 dan Solusi-6 digunakan 2 buah if yang terpisah, dimana hasil sementara nilai terbesar dicatat di tempat baru (Max), cara ini lebih praktis terutama kalau dikembangkan untuk mencari terbesar dari banyak bilangan.

Kasus 4.3 : Pembayaran air minum PDAM

PDAM menerapkan pembayaran air minum perumahan dengan cara perhitungan sebagai berikut :

- Tarif per m3 untuk 10 m3 pertama (1-10) adalah 2.000
- Tarif per m3 untuk 10 m3 kedua (11-20) adalah 3.000
- Tarif per m3 untuk 10 m3 ketiga (21-30) adalah 4.000
- Tarif per m3 untuk 10 m3 selanjutnya (31 ke atas) adalah 5.000
- Pemakaian air dihitung minimal 10 m3 (kurang dari 10 m3 dianggap 10 m3)
- Biaya administrasi bulanan sebesar 10.000

Bagaimana membuat algoritma untuk menghitung biaya tersebut?

Contoh kasus

Penggunaan air 5 m3 dengan biaya $10 \times 2.000 + 10.000 = 30.000$

Penggunaan air 15 m3 dengan biaya $10 \times 2.000 + 5 \times 3.000 + 10.000 = 45.000$

Penggunaan air 75 m3 dengan biaya $10 \times 2.000 + 10 \times 3.000 + 10 \times 4.000 + 45 \times 5.000 + 10.000 = 325.000$

Solusi :

Pemakaian air dibagi menjadi 4 area pemakaian (misal area a,b,c,d), baru dihitung total biaya

Solusi-1	Solusi-2
Input(pakai)	Input(pakai)
<u>If (pakai>30) then</u>	$a \leftarrow 10$
$a \leftarrow 10$	$b \leftarrow 0$
$b \leftarrow 10$	$c \leftarrow 0$
$c \leftarrow 10$	$d \leftarrow 0$
$d \leftarrow \text{pakai} - 30$	<u>If (pakai>30) then</u>
<u>Else If (pakai>20) then</u>	$b \leftarrow 10$

<pre> a ← 10 b ← 10 c ← pakai - 20 d ← 0 <u>Else If</u> (pakai > 10) <u>then</u> a ← 10 b ← pakai - 10 c ← 0 d ← 0 <u>Else</u> a ← 10 b ← 0 c ← 0 d ← 0 <u>End if</u> biaya ← a * 2000 + b * 3000 + c * 4000 + d * 5000 + 10000 Output('biaya =', biaya) </pre>	<pre> c ← 10 d ← pakai - 30 <u>Else If</u> (pakai > 20) <u>then</u> b ← 10 c ← pakai - 20 <u>Else If</u> (pakai > 10) <u>then</u> b ← pakai - 10 <u>End if</u> biaya ← a * 2000 + b * 3000 + c * 4000 + d * 5000 + 10000 Output('biaya =', biaya) </pre>
--	--

Ulasan solusi :

Pada solusi-1, tiap aksi dari if , terdiri dari 4 statement mengisi a,b,c dan d. Bentuk solusi ini disederhanakan pada solusi-2 dengan cara memberikan nilai awal sebelum masuk if.

Kasus-4 : Indeks Nilai Kuliah

Indeks nilai sebuah matakuliah didapat dengan cara menghitung nilai akhir berdasarkan prosentase komponen-komponennya kemudian ditentukan indeks nilainya. Misal digunakan ketentuan sebagai berikut:

- Nilai Akhir dihitung dari 30% UTS, 40%UAS, 20% Tugas dan 10% kehadiran
- Indeks Nilai ditentukan berdasarkan Nilai Akhir (NA),
 - Bila $NA \geq 85$ maka Indeksnya A
 - Bila $85 > NA \geq 70$ maka Indeksnya B
 - Bila $70 > NA \geq 55$ maka Indeksnya C

Bila $55 > NA \geq 40$ maka Indeksnya D

Bila $NA < 40$ maka Indeksnya E

Bagaimana membuat algoritma untuk menentukan Indeks Nilai tersebut?

<p>Solusi-1</p> <pre> Input(UTS,UAS,Tugas,Abs) $NAS \leftarrow 0.3*UTS + 0.4*UAS +$ $0.2*Tugas + 0.1*ABS$ <u>If</u> ($NAS \geq 85$) <u>then</u> Indeks \leftarrow 'A' <u>Else If</u> ($NAS \geq 70$ and NAS<85) <u>then</u> Indeks \leftarrow 'B' <u>Else If</u> ($NAS \geq 55$ and NAS<70) <u>then</u> Indeks \leftarrow 'C' <u>Else If</u> ($NAS \geq 40$ and NAS<55) <u>then</u> Indeks \leftarrow 'D' <u>Else</u> Indeks \leftarrow 'E' <u>End if</u> Output('Indeks Nilai =' ,Indeks) </pre>	<p>Solusi-2</p> <pre> Input(UTS,UAS,Tugas,Abs) $NAS \leftarrow 0.3*UTS + 0.4*UAS +$ $0.2*Tugas + 0.1*ABS$ <u>If</u> ($NAS \geq 85$) <u>then</u> Indeks \leftarrow 'A' <u>Else If</u> ($NAS \geq 70$) <u>then</u> Indeks \leftarrow 'B' <u>Else If</u> ($NAS \geq 55$) <u>then</u> Indeks \leftarrow 'C' <u>Else If</u> ($NAS \geq 40$) <u>then</u> Indeks \leftarrow 'D' <u>Else</u> Indeks \leftarrow 'E' <u>End if</u> Output('Indeks Nilai =' ,Indeks) </pre>
--	--

Ulasan solusi :

Pada solusi-2 lebih baik dari solusi-1 karena pemeriksaan kondisi yang serupa tidak dilakukan dua kali. Pada solusi-1, perbandingan NAS dengan 85 dilakukan dua kali, yang pertama $NAS \geq 85$ dan yang kedua $NAS < 85$ adalah perbandingan yang serupa.

4.3 Bentuk Umum CASE dan variasinya

Sebenarnya semua bentuk pemilihan dapat ditulis dengan IF, namun penulisan dengan IF untuk banyak pilihan terasa kurang praktis. Bentuk CASE adalah cara lain penulisan bentuk pemilihan yang lebih sederhana, namun bentuk ini hanya dapat menggantikan IF apabila memenuhi syarat:

- kondisi berupa perbandingan kesamaan (dengan tanda "=")
- nilai yang dibandingkan bertipe ordinal (integer, char dan boolean)

Bentuk CASE yang juga dikenal dengan istilah CASE Statement, memiliki bentuk umum sebagai berikut :

```
Case ekspresi  
  Nilai-1: Aksi-1  
  Nilai-2: Aksi-2  
  ...  
  Nilai-N: Aksi-N  
  [Otherwise : Aksi-X]  
End Case
```

Ekspresi bertipe ordinal, berupa:

Sebuah nilai ordinal: boolean, integer, char (bukan string atau real)

Sebuah variabel bertipe ordinal

Operasi data (nilai atau variabel) yang menghasilkan sebuah nilai ordinal

Nilai harus berupa nilai ordinal (tidak boleh variabel)

Aksi berupa satu statement beberapa statement, dimana tiap statement

dapat berupa:

Statement pengisian nilai seperti $a \leftarrow 5$

Statement input data

Statement output data

Statement pemilihan (If Statement atau Case Statement)

Statement pengulangan (For, Repeat atau While Statement)

[otherwise: Aksi-X], tanda [] menyatakan opsional (boleh ada/tidak ada),

dimana kalau tidak ada, berarti setelah Aksi-1 langsung selesai. Fungsi Otherwise sama dengan ELSE pada IF Statement

Dari bentuk umum yang telah dijelaskan, maka variasi bentuk CASE ini banyak dan tidak berhingga. Di antaranya yang penting dapat disebutkan berikut:

- Case tanpa otherwise
- Case dengan otherwise
- Case dengan Aksi yang sama untuk beberapa Nilai
- Case bersarang dimana dalam case ada case lagi, atau

Statement lain

Contoh-contoh variasi:

1. Case tanpa otherwise

Case ekspresi

Nilai-1: Statement-1

Nilai-2: Statement -2

...

Nilai-N: Statement -N

End Case

2. Case dengan otherwise

Case ekspresi

Nilai-1: Statement -1

Nilai-2: Statement -2

...

Nilai-N: Statement -N

[Otherwise : Aksi-X]

End Case

3. Case dengan Aksi yang sama untuk beberapa Nilai

Case ekspresi

Nilai-1,Nilai-2,Nilai-3: Statement -1

Nilai-4,Nilai-5,Nilai-6: Statement -2

Nilai-7..Nilai-10: Statement -3

...

Nilai-N: Statement -N
[Otherwise : Statement -X]

End Case

4. Case bersarang, contohnya :

Case ekspresi-1

Nilai-1: Case ekspresi-2

Nilai-a: Statement -1

Nilai-b: Statement -2

End Case

Nilai-2: if kondisi then

Statement-3

Else

Statement-4

End if

Nilai-3:

...

Nilai-N: Statement -N

End Case

4.4 Terapan bentuk-bentuk CASE

Kasus 4.4 : Menentukan nama hari dari nomor hari yang diinput
Dinput nomor hari, ditampilkan nama harinya, bagaimana
algoritmanya?

Solusi dengan IF dan CASE

Solusi-If	Solusi-Case
Input(NoHari)	Input(NoHari)
If (NoHari=1) then	Case NoHari
NmHari ← 'Senin'	1: NmHari ← 'Senin'
Else If (NoHari=2) then	2: NmHari ← 'Selasa'
NmHari ← 'Selasa'	3: NmHari ← 'Rabu'
Else If (NoHari=3) then	4: NmHari ← 'Kamis'
NmHari ← 'Rabu'	5: NmHari ← 'Jumat'
Else If (NoHari=4) then	6: NmHari ← 'Sabtu'
NmHari ← 'Kamis'	7: NmHari ← 'Minggu'

<pre> <u>Else If</u> (NoHari=5) <u>then</u> NmHari ← 'Jumat' <u>Else If</u> (NoHari=6) <u>then</u> NmHari ← 'Sabtu' <u>Else If</u> (NoHari=7) <u>then</u> NmHari ← 'Minggu' <u>End if</u> <u>Output</u>(NmHari) </pre>	<pre> <u>End Case</u> <u>Output</u>(NmHari) </pre>
--	--

Pada solusi-2 terlihat lebih sederhana dan mudah dibaca dibanding dengan solusi-1.

Kasus 4.5 : Merubah angka menjadi kalimat

Dinput bilangan/angka (angka dibatasi 1-99), ditampilkan kata-kata/kalimat dari bilangan tersebut, bagaimana algoritmanya?

Solusi

<pre> Solusi-Case Input(bil) pul ← bil div 10 sat ← bil mod 10 Kalimat ← '' <u>Case</u> sat 1: Kalimat ← 'Satu' 2: Kalimat ← 'Dua' 3: Kalimat ← 'Tiga' 4: Kalimat ← 'Empat' 5: Kalimat ← 'Lima' 6: Kalimat ← 'Enam' 7: Kalimat ← 'Tujuh' 8: Kalimat ← 'Delapan' 9: Kalimat ← 'Sembilan' <u>End Case</u> <u>Case</u> pul 1: <u>Case</u> sat 0: Kalimat ← 'Sepuluh' 1: Kalimat ← 'Sebelas' Otherwise: Kalimat ← Kalimat + ' belas' </pre>

```

| | | End Case
| | 2: | Kalimat ← 'Dua Puluh' + Kalimat
| | 3: Kalimat ← 'Tiga Puluh' + Kalimat
| | 4: Kalimat ← 'Empat Puluh' + Kalimat
| | 5: Kalimat ← 'Lima Puluh' + Kalimat
| | 6: Kalimat ← 'Enam Puluh' + Kalimat
| | 7: Kalimat ← 'Tujuh Puluh' + Kalimat
| | 8: Kalimat ← 'Delapan Puluh' + Kalimat
| | 9: Kalimat ← 'Sembilan Puluh' + Kalimat
| End Case
| Output(Kalimat)

```

Pada solusi di atas, satuan diproses dengan case pertama, selanjutnya puluhan diproses CASE kedua. Pada puluhan=1 (angka belasan) dibagi lagi menjadi 3 kemungkinan, karena bunyi kalimatnya ada 3 macam,

4.5 Konversi Struktur IF dan CASE ke Bahasa C

Berikut ini diberikan pedoman konversi dari algoritma ke dalam bahasa C untuk struktur IF dan CASE:

Algoritma	Bahasa C
If kondisi then Aksi End if	if (kondisi) { Aksi; }
If kondisi then Aksi1 Else Aksi2 End if	If (kondisi) { Aksi1; } else { Aksi2; }
If kondisi1 then Aksi1 Else if kondisi2 Aksi2 Else	if (kondisi1) { Aksi1; } else if (kondisi2){ Aksi2; }

Aksi3 End if	} else { Aksi3; }
Case ekspresi Nilai1: Aksi1 Nilai2: Aksi2 Nilai3: Aksi3 End case	switch (ekspresi) { case Nilai1: Aksi1; Break; case Nilai2: Aksi2; Break; case Nilai3: Aksi3; }
Case ekspresi Nilai1: Aksi1 Nilai2: Aksi2 Nilai3: Aksi3 Otherwise: Aksi4 End case	switch (ekspresi) { case Nilai1: Aksi1; Break; case Nilai2: Aksi2; Break; case Nilai3: Aksi3; Break; default: Aksi4; }
<u>Case</u> ekspresi Nilai-1,Nilai-2,Nilai-3: Aksi1 Nilai-4,Nilai-5: Aksi2 Nilai-6..Nilai-8: Aksi3 Otherwise: Aksi4 <u>End Case</u>	switch (ekspresi) { case Nilai1: case Nilai2: case Nilai3: Aksi1; Break; case Nilai4: case Nilai5: Aksi2; Break; case Nilai6: case Nilai7: case Nilai8: Aksi3; Break; default: Aksi4; }

Catatan:

- penulisan kondisi pada IF dan ekspresi pada CASE dalam bahasa C harus digunakan tanda kurung ().
- aksi berupa satu perintah atau lebih, masing-masing diakhiri titik koma.
- apabila aksi hanya berupa satu perintah, penggunaan { } dapat dihilangkan.
- kata "if", "else", "switch", "case" dan "default" dalam bahasa C, harus ditulis dengan huruf kecil semua.
- dalam bahasa C tidak ada kata "then", "end if" dan "end case" tetapi digantikan pasangan kurung kurawal { dan }
- hati-hati dengan penggunaan kesamaan, yaitu dengan "==" bukan "=".
- string digunakan kutip dua (seperti "test") bukan kutip satu ('test').

Contoh penulisan algoritma selengkapnya dan hasil konversinya ke bahasa C.

Diambil dari contoh pada Kasus 4.3

Algoritma	Bahasa C
<u>Algoritma PDAM</u> /* menghitung biaya pemakaian air*/ <u>Kamus Data</u> pakai,a,b,c,d : integer biaya : integer <u>Begin</u> Input(pakai) a←10 b← 0 c← 0 d← 0 <u>If (pakai>30) then</u> b← 10 c← 10 d← pakai - 30	#include <stdio.h> #include <conio.h> /* menghitung biaya pemakaian air*/ int main() { //Kamus Data int pakai,a,b,c,d; int biaya; //Begin printf("Masukkan pemakaian air: "); scanf("%d",&pakai); a=10; b=0; c=0;

<pre> Else If (pakai>20) then b ← 10 c ← pakai - 20 Else If (pakai>10) then b ← pakai - 10 End if biaya ← a * 2000 + b * 3000 + c * 4000 + d * 5000 + 5000 Output('biaya =',biaya) End </pre>	<pre> d=0; if (pakai>30) { b=10; c=10; d=pakai - 30; } else if (pakai>20) { b=10; c=pakai - 20; } else if (pakai>10) { b=pakai - 10; } biaya = a * 2000 + b * 3000 + c * 4000 + d * 5000 + 10000; printf("biaya = %d",biaya); getche(); return 0; //End } </pre>
---	---



Rangkuman

1. Struktur pemilihan dapat digunakan untuk membuat program melakukan aksi tertentu sesuai nilai dari kondisi yang dipertimbangkan.
2. Struktur pemilihan dapat berupa struktur IF ... THEN ..., struktur IF ... THEN ... ELSE ..., struktur CASE, maupun pemilihan bersarang IF atau CASE.
3. Struktur IF ... THEN ... (tanpa ELSE) digunakan pada situasi dengan pilihan mengerjakan aksi atau tidak.

4. Struktur IF ... THEN ... ELSE ... digunakan untuk memilih salah satu aksi dari berdasarkan nilai kondisi.
5. Struktur CASE merupakan bentuk penyederhanaan dari struktur IF dengan persyaratan tertentu, yaitu kondisi berupa perbandingan kesamaan dan nilai yang dibandingkan harus ordinal (integer, char atau boolean).
6. Klausula OTHERWISE pada struktur CASE bersifat opsional, seperti halnya ELSE pada struktur CASE.
7. Struktur pemilihan bersarang dan struktur CASE dapat menyederhanakan program yang menyelesaikan kasus dengan nilai kondisi berjumlah lebih dari dua.



Kuis Benar Salah

1. Klausula ELSE dalam struktur pemilihan IF... THEN... ELSE... berarti 'jika tidak'.
2. Klausula THEN dalam struktur pemilihan IF... THEN... ELSE... berarti 'maka'.
3. Struktur pemilihan dapat membantu pengambilan keputusan dalam program.
4. Struktur pemilihan dapat membuat program berjalan dengan alur yang berbeda sesuai kondisi yang sedang berlaku.
5. Struktur CASE hanya dapat digunakan pada pemilihan dengan satu ekspresi.
6. Pada struktur pemilihan IF... THEN... ELSE..., jika kondisi yang disebutkan setelah klausula IF bernilai benar maka program akan menjalankan aksi setelah klausula ELSE.

(Untuk soal no 7 – 9) Perhatikan potongan algoritma berikut ini:

```
Input (nilai)
IF nilai > 85 AND nilai <= 100 THEN
    Output ('Nilai mutu = A')
ELSE
    IF nilai > 60 AND nilai <= 85 THEN
        Output ('Nilai mutu = B')
    ELSE
        IF nilai > 45 AND nilai <= 60 THEN
            Output ('Nilai mutu = C')
        ELSE
            IF nilai > 30 AND nilai <= 45 THEN
                Output ('Nilai mutu = D')
            ELSE
                Output ('Nilai mutu = E')
            ENDIF
        ENDIF
    ENDIF
ENDIF
```

ENDIF

ENDIF

7. Jika inputan nilai = 60, maka outputnya adalah 'Nilai mutu = B'.
8. Jika inputan nilai = 101, maka outputnya adalah 'Nilai mutu = E'
9. Jika inputan nilai = -5, maka outputnya adalah 'Nilai tidak dapat ditentukan'.

(Untuk soal no 10 – 13) Perhatikan potongan algoritma berikut ini:

CASE *nilai* OF

'A' : Output ('Sangat memuaskan')

'B' : Output ('Memuaskan')

'C' : Output ('Cukup')

'D' : Output ('Kurang')

'E' : Output ('Sangat kurang')

OTHERWISE : Output ('Inputkan A,B,C,D atau E')

ENDCASE

10. Jika inputan nilai = 'B', maka outputnya adalah 'Sangat memuaskan'
11. Jika inputan nilai = 'E', maka outputnya adalah 'Sangat kurang'
12. Jika inputan nilai = '65', maka outputnya adalah 'Cukup'
13. Potongan algoritma berikut ini tepat sama dengan potongan program pada soal:

Input (nilai)

IF *nilai* = 'A' THEN

Output ('Sangat memuaskan')

ELSE

IF *nilai* = 'B' THEN

Output ('Memuaskan')

ELSE

IF *nilai* = 'C' THEN

Output ('Cukup')

ELSE

IF *nilai* = 'D' THEN

Output ('Kurang')

ELSE

```
        IF nilai = 'E' THEN
            Output ('Sangat kurang')
        ELSE
            Output ('Inputkan A,B,C,D atau E')
        ENDIF
    ENDIF
ENDIF
ENDIF
ENDIF
```

14. Pada struktur pemilihan IF... THEN..., pemrogram tidak perlu menentukan parameter kondisi yang digunakan.
15. Pada struktur pemilihan CASE, pemrogram tidak perlu menentukan parameter kondisi yang digunakan.
16. Struktur pemilihan CASE dapat menyederhanakan pemilihan yang ditulis dengan struktur IF... THEN... ELSE... .
17. Dalam flowchart, pengecekan kondisi pada struktur pemilihan digambarkan dengan bentuk belah ketupat.
18. Dua garis keluar dari struktur pemilihan dalam flowchart menunjukkan kemungkinan nilai dari kondisi.
19. Kondisi dan aksi pada struktur pemilihan harus berupa ekspresi boolean.
20. Perhatikan potongan algoritma berikut ini:

```
Input (x)
IF  $x \bmod 2 = 0$  THEN
    Output ('siswa bagi adalah nol')
ELSE
    Output ('siswa bagi bukan nol')
ENDIF
```

Algoritma di atas dapat digunakan untuk membedakan bilangan genap dan ganjil.



Pilihan Ganda

1. Struktur pemilihan terdiri atas komponen-komponen berikut ini, kecuali
A. kondisi pemilihan D. reaksi
B. alternatif aksi dr nilai E. parameter kondisi
C. aksi
2. Perhatikan potongan algoritma berikut ini:

```
IF badan belum bersih THEN  
    teruskan mandi  
ELSE  
    mandi selesai  
ENDIF
```

Menurut potongan algoritma tersebut, mandi selesai jika

A. tangan capek D. mandi tidak diteruskan
B. sudah terlalu lama E. pilihan A,B,C,dan D salah
C. badan sudah bersih
3. Pernyataan manakah yang SALAH tentang struktur pemilihan CASE... END CASE?
A. Tersedia aksi default dalam struktur pemilihan ini
B. Dalam struktur ini, kondisi pemilihan tidak perlu ditentukan
C. Struktur ini dapat digunakan untuk menggantikan struktur pemilihan bersarang.
D. Struktur ini memiliki komponen parameter kondisi, alternatif nilai, dan aksi
E. Pernyataan A,B,C maupun D benar semua.
4. Potongan algoritma manakah yang tepat untuk membedakan bilangan genap dan ganjil?
a. *Input (x)*
 IF $x \bmod 2 \geq 0$ THEN

```
        Output ('x adalah bilangan genap')
    ELSE
        IF x mod 2 < 0 THEN
            Output ('x adalah bilangan ganjil')
        ENDIF
    ENDIF
b. Input (x)
    IF x div 2 >= 0 THEN
        Output ('x adalah bilangan genap')
    ELSE
        IF x div 2 < 0 THEN
            Output ('x adalah bilangan ganjil')
        ENDIF
    ENDIF
c. Input (x)
    IF x mod 2 = 0 THEN
        Output ('x adalah bilangan genap')
    ELSE
        Output ('x adalah bilangan ganjil')
    ENDIF
d. Input (x)
    IF x div 2 = 0 THEN
        Output ('x adalah bilangan genap')
    ELSE
        Output ('x adalah bilangan ganjil')
    ENDIF
```

e. Pilihan A,B,C,D salah

5. Berikut ini adalah contoh kondisi yang tidak dapat digunakan dalam pemilihan, yaitu
- | | |
|--------------------|--------------------------|
| A. angka1 > angka2 | D. $a * b \leq 0$ |
| B. gaji = 1000000 | E. Pilihan A,B,C,D salah |
| C. baju = baru | |
6. Apakah fungsi klausa OTHERWISE pada struktur pemilihan CASE... END CASE?
- A. Menentukan aksi yang harus dilakukan jika kondisi bernilai benar
 - B. Menentukan aksi yang harus dilakukan jika kondisi bernilai salah
 - C. Menentukan aksi tambahan yang dilakukan apapun

- nilai kondisi
- D. Menentukan aksi yang harus dilakukan jika tidak ada nilai yang sesuai dengan ekspresi
- E. Mengakhiri struktur pemilihan
7. Perhatikan potongan algoritma berikut ini:
- ```
Input (harga)
IF harga = 12500 then
 Beli
ENDIF
```
- Pernyataan manakah yang benar tentang program di atas?
- A. Jika harga <12500 maka lakukan aksi beli
- B. Jika harga >12500 maka lakukan aksi beli
- C. Jika harga = 12500 maka program tidak melakukan apapun
- D. Jika harga <> 12500 maka program tidak melakukan apapun
- E. Jika harga <> 12500 maka program error
8. Dalam struktur IF... THEN... ELSE..., klausa IF dapat diartikan ... .
- A. Jika
- B. Maka
- C. Jika tidak
- D. Sebaiknya
- E. Apapun yang terjadi
9. Perhatikan potongan algoritma berikut ini:
- ```
Input (n)
p = n*2
IF p mod 2 = 0 THEN
    Output (p/2)
ELSE
    Output (p*2)
ENDIF
```
- Pernyataan manakah yang benar tentang algoritma di atas?
- A. Berapapun nilai yang diinputkan, outputnya sama dengan nilai inputan
- B. Jika inputannya adalah bilangan ganjil, outputnya adalah nilai inputan dikali dua
- C. Jika inputannya adalah bilangan genap, outputnya

adalah nilai inputan dibagi dua

D. Berapapun nilai yang diinputkan, outputnya adalah nilai inputan dikali dua

E. Pilihan A,B,C maupun D salah

10. Perhatikan potongan algoritma berikut ini:

Input (kelas)

SWITCH kelas

BEGIN

 CASE '1' :

Output ('Pengajar = Ana')

 break

 CASE '2' :

Output ('Pengajar = Ani')

 break

 CASE '3' :

Output ('Pengajar = Ina')

 break

END

Apakah output dari algoritma di atas jika inputnya kelas = '4'?

A. 'Pengajar = Ana'

D. Program error

B. 'Pengajar = Ani'

E. Pilihan A,B,C dan D

C. 'Pengajar = Ina'

salah

5 Pengulangan



Overview

Pengulangan (*Loop*) merupakan sebuah konsep yang penting dalam pemrograman. Dengan struktur pengulangan, program dapat berjalan beberapa kali sesuai inisialisasi, jumlah iterasi dan kondisi berhenti yang ditentukan. Hal ini dapat menyederhanakan program yang kompleks menjadi lebih sederhana. Dalam C disediakan berbagai perintah *Loop*, dimana setiap perintah *loop* memiliki keunikan tersendiri. Di dalam bab ini akan dijelaskan tentang struktur pengulangan dalam algoritma serta implementasi struktur pengulangan dengan perintah *loop* yang ada di dalam C.



Tujuan

1. Memahami konsep dasar dan struktur pengulangan
2. Memahami perintah pengulangan dalam C

3. Menerapkan sintaks-sintaks pengulangan dalam menyelesaikan persoalan

5.1 Konsep Pengulangan

Program yang efisien adalah program yang memungkinkan pengguna bekerja sesedikit mungkin dan komputer bekerja sebanyak mungkin. Salah satu cara melakukan hal tersebut adalah dengan menggunakan kembali sekumpulan baris program yang terdapat pada bagian lain dari program tersebut atau baris program yg terdapat di dalam program lain.

Pengulangan merupakan sebuah konsep pemrograman yang penting karena konsep ini memungkinkan pengguna menggunakan sekumpulan baris program berulang kali dengan tiga komponen yang mengendalikannya, yaitu:

- **Inisialisasi;** menentukan kondisi awal dilakukannya pengulangan.
- **Jumlah iterasi;** menunjukkan berapa kali pengulangan akan dilakukan.
- **Kondisi berhenti;** menentukan kondisi yang dapat mengakhiri pengulangan.

Contoh kasus dunia nyata yang dapat digunakan untuk menggambarkan ketiga komponen ini adalah cerita ibu mengupas sepuluh (10) butir kentang. Ibu akan mengumpulkan dulu 10 butir kentang yang akan dikupas, kemudian Ibu akan mengambil sebuah kentang kemudian mengupasnya, setelah selesai mengupas Ibu akan mengambil kentang berikutnya dan melakukan aksi mengupas lagi. Ibu akan melakukan pengupasan kentang satu persatu hingga mencapai kentang ke-10, dan seluruh kentang tersebut telah terkupas. Ibu akan melakukan sederetan aksi yang tepat sama terhadap kesepuluh butir kentang tersebut. Maka,

- **Inisialisasi:** 10 butir kentang yang masih utuh dengan kulitnya
- **Jumlah iterasi:** 10 (sesuai jumlah kentang)
- **Kondisi berhenti:** 10 butir kentang sudah terkupas.

Ketika mengimplementasikan dalam program, ketiga komponen ini tidak selalu dapat didefinisikan dalam struktur pengulangan. Mungkin saja salah satu komponen tersebut tidak didefinisikan. Pengulangan tetap dapat berjalan, asal komponen yang tidak didefinisikan tersebut dapat diketahui secara tersirat berdasarkan komponen lain yang didefinisikan. Hal lain yang perlu diperhatikan adalah bahwa **pengulangan harus berhenti**. Jika pengulangan tidak pernah berhenti, maka logika program salah. Pengulangan akan berhenti jika jumlah iterasi yang diminta sudah tercapai atau kondisi berhenti bernilai benar. Maka, dalam setiap pengulangan, pemrogram perlu menentukan jumlah iterasi atau kondisi berhenti dan langkah pencapaian menuju kondisi berhenti tersebut.

Pada bab ini akan dijelaskan 3 struktur perulangan dan implementasinya di dalam C, yaitu struktur perulangan While , For, dan Do While

5.2 Sintaks WHILE

Pengulangan dengan menggunakan WHILE merupakan sebuah pengulangan yang dikendalikan oleh suatu kondisi tertentu, dimana kondisi tersebut yang akan menentukan apakah perulangan itu akan terus dilaksanakan atau dihentikan. Kondisi tersebut akan dicek disetiap awal iterasi, apakah sebuah kondisi terpenuhi atau tidak. Jika kondisi terpenuhi (bernilai benar), maka iterasi akan dilanjutkan. Jika kondisi tidak terpenuhi, maka iterasi dihentikan.

Perulangan dengan WHILE dapat digunakan pada struktur perulangan yang diketahui jumlah iterasinya dan juga pada struktur perulangan yang tidak diketahui jumlah iterasinya, tetapi harus selalu terdapat kondisi berhenti. Struktur pengulangan WHILE adalah:

```
1 {inisialisasi}
2 WHILE (kondisi)
3     aksi
4     ubah pencacah (pencapaian kondisi berhenti)
5 ENDWHILE
```

Pencacah adalah variabel pengendali iterasi yang harus diinisialisasi, dicek dalam kondisi, dan terus berubah nilainya setiap iterasi dilakukan. Pencacah inilah yang akan membuat sebuah kondisi berhenti tercapai. Pada struktur pengulangan dengan sintaks WHILE, nilai pencacah akan diubah di akhir aksi pengulangan.

Contoh: Ibu mengupas 10 butir kentang dapat direpresentasikan dengan pengulangan WHILE sebagai berikut :

ALGORITMA Kupas Kentang	
IS :	Terdapat 10 kentang belum dikupas
FS :	10 kentang telah terkupas
KAMUS DATA	
kentang : integer	
1	BEGIN
	{inisialisasi jumlah kentang yang sudah dikupas}
2	kentang ← 0
3	WHILE (kentang < 10){kondisi iterasi dilakukan}
4	Ambil sebuah kentang
5	Kupas kulit kentang
6	kentang←kentang+1 {pencapaian kondisi berhenti}
7	ENDWHILE
8	END

Telah diketahui bahwa Ibu akan melakukan pengupasan sebanyak 10 kentang, maka sebelum masuk struktur pengulangan, variabel kentang berisi 0 {menunjukkan bahwa di awal iterasi belum ada kentang yang dikupas atau jumlah kentang yg telah dikupas = 0}.

Pada iterasi pertama, terjadi pengecekan kondisi (kentang < 10) dimana artinya “apakah jumlah kentang yang dikupas belum mencapai 10”, karena nilai **kentang = 0** maka kondisi bernilai benar, ibu akan mengambil sebuah kentang kemudian mengupas kulitnya. Di akhir iterasi pertama, jumlah kentang yang sudah dikupas menjadi (0+1) = 1. Pada titik ini, variabel kentang berisi 1 (artinya, 1 kentang sudah dikupas).

Pada iterasi kedua, terjadi pengecekan kondisi (kentang < 10), sedangkan kentang = 1, maka kondisi bernilai benar, ibu akan mengambil sebuah kentang kemudian mengupas kulitnya.

Di akhir iterasi kedua, jumlah kentang yang sudah dikupas menjadi $(1+1)=2$.

Iterasi ini terus berulang sampai iterasi ke 10, variabel kentang berisi 9. Saat terjadi pengecekan kondisi ($\text{kentang} < 10$) bernilai benar, maka ibu mengambil sebuah kentang kemudian mengupas kulitnya. Di akhir iterasi ke sepuluh, jumlah kentang yang sudah dikupas menjadi $(9+1) = 10$.

Kemudian, pada iterasi ke 11, terjadi pengecekan kondisi ($\text{kentang} < 10$), karena kentang = 10 maka kondisi bernilai salah, sehingga iterasi diakhiri.

Hasil akhirnya, variabel kentang berisi 10 (artinya, 10 kentang sudah dikupas).

Jalannya iterasi ini dapat ditulis dalam bentuk tabel berikut:

Iterasi ke-	Kentang	kentang < 10	ambil & kupas	kentang
1	0	Ya	Ya	1
2	1	Ya	Ya	2
3	2	Ya	Ya	3
4	3	Ya	Ya	4
5	4	Ya	Ya	5
6	5	Ya	Ya	6
7	6	Ya	Ya	7
8	7	Ya	Ya	8
9	8	Ya	Ya	9
10	9	Ya	Ya	10
11	10	Tidak	Tidak	-

Dari contoh di atas, variabel kentang merupakan pengendali iterasi. Iterasi dapat terus dilakukan atau tidak, bergantung pada nilai variabel kentang ini. Selanjutnya, variabel penentu iterasi ini disebut dengan pencacah. Pencacah harus berupa nilai yang memiliki urutan, yaitu dapat bertipe integer atau karakter. Di setiap struktur pengulangan, pencacah selalu ada dan jangan lupa untuk menginisialisasi pencacah. Nilai pencacah akan berubah pada setiap iterasi.

Hal lain yang perlu diperhatikan adalah bahwa di akhir iterasi, variabel *kentang* bernilai 10. Nilai ini tidak berubah lagi karena iterasi tidak dilakukan lagi, dan disebut sebagai *loop invariant*.

Contoh lain struktur pengulangan ini:

- 1) Algoritma untuk menampilkan karakter ' * ' sebanyak 5 kali.

ALGORITMA Tampil Bintang
IS : - FS : Jumlah bintang yg tampil = 5
KAMUS DATA i : integer
1 BEGIN 2 i ← 0 {inisialisasi pencacah i} 3 WHILE (i < 5) {jumlah iterasi} 4 Output ('*') {aksi} 5 i ← i + 1 {pencapaian kondisi berhenti} 6 ENDWHILE 7 END

Output dari algoritma ini:

*
*
*
*
*

- 2) Algoritma untuk menampilkan iterasi ke 1 sampai 5 dengan pengulangan.

ALGORITMA Iterasi Angka
IS : - FS : Tampil angka 1 hingga 5
KAMUS DATA i : integer
1 BEGIN 2 i ← 1 {inisialisasi pencacah i} 3 WHILE (i <= 5) 4 Output ('Ini adalah iterasi ke-',i)


```

5  i ← i + 1
6  ENDWHILE
7  END

```

Output dari algoritma ini:

```

Ini adalah iterasi ke-1
Ini adalah iterasi ke-2
Ini adalah iterasi ke-3
Ini adalah iterasi ke-4
Ini adalah iterasi ke-5

```

3) Algoritma untuk memasukkan nilai tiga (3) orang mahasiswa

ALGORITMA Input_Nilai

IS : -

FS : Menerima inputan nilai dari user

KAMUS DATA

i : integer

nilai : integer

```

1  BEGIN
2  i ← 1
3  WHILE (i <= 3)
4  Output ('Nilai mahasiswa ke-',i,'adalah:')
5  Input (nilai)
6  i ← i + 1
7  ENDWHILE
8  END

```

Output dari algoritma ini:

```

Nilai mahasiswa ke-1 adalah: _
Nilai mahasiswa ke-2 adalah: _
Nilai mahasiswa ke-3 adalah: _

```

4) Algoritma untuk menghitung nilai rata-rata dari tiga bilangan yang diinputkan.

ALGORITMA Nilai Rata Rata

IS : -

FS : Tampil rata-rata dari nilai yang diinputkan

KAMUS DATA

```

jumlah    : float
bilangan  : integer
x         : float
rerata    : float

1 BEGIN
2  jumlah ← 0      {variabel jumlah bilangan}
3  bilangan ← 3    {inisialisasi variabel pencacah}
4  WHILE (bilangan > 0)
5      Output('Masukkan angka : ')
6      Input (x) {masukkan sebuah bilangan}
7      jumlah ← jumlah + x {tambahkan bilangan}
8      bilangan ← bilangan - 1 {pencacah mundur}
9  ENDWHILE
10 rerata ← jumlah / 3 {menghitung rerata}
11 Output ('Rerata : ', rerata)
12 END

```

Pada contoh algoritma yang ke-4, ditunjukkan bahwa iterasi dapat dilakukan dengan pencacah mundur. Selama kondisi `bilangan > 0` terpenuhi, maka pengguna diminta memasukkan sebuah bilangan yang kemudian dijumlahkan dengan bilangan-bilangan sebelumnya (pada iterasi pertama, bilangan dijumlahkan dengan nol). Karena menggunakan iterasi mundur, maka pencacah akan dikurangkan. Algoritma akan memberikan output berupa hasil perhitungan rerata dari ketiga bilangan yang diinputkan.

Jika pada contoh algoritma yang ke-4 diatas diinputkan angka 60, 70, dan 90, maka jalannya iterasi ini dapat ditulis dalam bentuk tabel berikut:

Iterasi ke-	bilangan	x	jumlah	rerata	bilangan
1	3	60	60	-	2
2	2	70	130	-	1
3	1	90	220	-	0
4	0	-	220	73.33	-

Penggunaan sintaks WHILE dalam bahasa pemrograman C :

Algoritma	C
-----------	---

<pre> WHILE (kondisi) aksi ubah pencacah ENDWHILE </pre>	<pre> while (kondisi) { //aksi //ubah pencacah } </pre>
<pre> ... i ← 0 WHILE (i < 5) Output('*') i ← i + 1 ENDWHILE ... </pre>	<pre> ... i = 0; while (i < 5) { printf("*"); i = i + 1; } ... </pre>

Implementasi algoritma contoh ke-4 ke dalam C adalah :

```

1  //Nilai_Rata_Rata.c
2  //Menerima 3 angka dan menampilkan rata-ratanya
3  #include <stdio.h>
4  main ()
5  {
6  //deklarasi variabel yg digunakan
7  float    jumlah;
8  int      bilangan;
9  float    x;
10 float    rerata;
11 //inisialisasi variabel jumlah
12 jumlah = 0;
13 //inisialisasi variabel pencacah bilangan
14 bilangan = 3;
15 while (bilangan > 0 )
16 {
17 //meminta inputan dari user
18 printf("Masukkan angka : ");
19 scanf("%f",&x);
20 //menjumlahkan angka yg diinputkan
21 jumlah = jumlah + x;
22 //mengurangkan pencacah untuk mencapai
23 //kondisi berhenti
24 bilangan = bilangan - 1;
25 }
26 //menghitung rata-rata dari 3 inputan angka

```

```
27 rerata = jumlah / 3;  
28 //menampilkan hasil rata-rata  
29 printf("Rerata : %.2f",rerata);  
30 }
```

Telah dikatakan di awal bahwa perulangan dengan WHILE dapat digunakan untuk struktur pengulangan yang belum diketahui jumlah iterasinya, tetapi tetap mempunyai kondisi berhenti.

Contoh struktur pengulangan ini adalah sebagai berikut :

ALGORITMA Tebak Huruf
IS : - FS : Menampilkan pesan "Maaf Anda salah" jika tebakkan salah, dan menampilkan "Anda Benar" jika tebakkan benar
KAMUS DATA huruf : character
1 BEGIN 2 {inisialisasi huruf tebakkan pertama } 3 Output('Masukkan tebakkan : ') 4 Input(huruf) 5 WHILE (huruf != 'q') {pengecekan kondisi} 6 Output('Maaf Anda salah ') 7 Output('Masukkan tebakkan : ') 8 Input(huruf) {input huruf berikutnya} 9 ENDWHILE 10 Output ('Anda Benar') 11 END

Pada algoritma diatas tidak terlihat adanya deklarasi variabel pencacah dan inisialisasinya, juga tidak terlihat adanya perubahan terhadap variabel pencacah untuk mencapai kondisi berhenti. Pengulangan diatas tetap dapat berjalan, karena komponen yang tidak didefinisikan tersebut dapat diketahui secara tersirat berdasarkan komponen lain yang didefinisikan. Dan yang paling penting adalah perulangan ini mempunyai kondisi berhenti.

Pada baris ke-4 user akan memasukkan sebuah karakter. Karakter inilah yang akan menjadi pemicu dilaksanakan pengulangan atau tidak, jika karakter yang dimasukkan adalah 'q', maka pengulangan tidak dilanjutkan, tetapi jika karakter yang dimasukkan bukan 'q', maka pengulangan dilanjutkan. Dapat dilihat pada baris ke-5 bahwa kondisi berhenti pengulangan adalah ketika user memasukkan (input) huruf 'q'.

Jumlah iterasi tidak ditentukan oleh variabel pencacah, melainkan ditentukan sendiri oleh user yang menginputkan huruf. Berikut adalah implementasi dari algoritma Tebak_Huruf dalam bahasa pemrograman C.

```
1 //Tebak_Huruf.c
2 //User memasukkan sebuah karakter untuk menebak
3 #include <stdio.h>
4 main()
5 {
6 //deklarasi variabel untuk menerima input
7 char huruf;
8 //menginisialisasi huruf awal untuk dicek
9 printf("Tebakan : " );
10 scanf("%s",&huruf);
11 //pengecekan kondisi terhadap huruf inputan
12 while (huruf!='q')
13 {
14 //jika huruf bukan 'q' maka input huruf lain
15 printf("Maaf anda salah");
16 printf("\nTebakan : " );
17 scanf("%s",&huruf);
18 }
19 //jika huruf = 'q' maka tidak masuk ke while
20 printf("Anda Benar");
21 }
```

5.3 Sintaks DO...WHILE

Sintaks DO... WHILE... melakukan pengulangan serupa dengan sintaks WHILE. Penggunaan sintaks ini juga tidak harus menyebutkan jumlah pengulangan yang harus dilakukan, karena dapat digunakan untuk pengulangan dengan jumlah

iterasinya yang belum diketahui, tetapi harus mempunyai kondisi berhenti.

Bedanya, jika pada sintaks WHILE kondisi dievaluasi/ diuji sebelum aksi pengulangan dilakukan, sedangkan pada sintaks DO...WHILE pengujian kondisi dilakukan setelah aksi pengulangan dilakukan.

Struktur pengulangan DO...WHILE yaitu:

```

1  {inisialisasi}
2  DO
3  aksi
4  ubah pencacah
5  WHILE (kondisi)

```

Pada struktur pengulangan dengan sintaks DO... WHILE..., aksi akan terus dilakukan hingga kondisi yang dicek di akhir pengulangan, bernilai benar. Dengan sintaks ini, pengulangan pasti dilakukan minimal satu kali, yakni pada iterasi pertama sebelum pengecekan kondisi. WHILE dengan DO WHILE seringkali memberikan hasil yang sama, tetapi ada kalanya hasilnya akan berbeda, sehingga harus berhati-hati dalam penggunaan kondisi antara WHILE dengan DO WHILE.

Beberapa contoh penerapan struktur pengulangan DO... WHILE... :

5) Algoritma ibu mengupas kentang

ALGORITMA Kupas Kentang
IS : Terdapat 10 kentang belum dikupas FS : 10 kentang telah terkupas
KAMUS DATA kentang : integer
<pre> 1 BEGIN 2 {inisialisasi jml kentang yang sudah dikupas} 3 kentang ← 0 4 DO 5 Ambil sebuah kentang 6 Kupas kulit kentang 7 {pencapaian kondisi berhenti} 8 kentang←kentang+1 9 WHILE (kentang < 10) {kondisi berhenti} 10 END </pre>

Pada potongan algoritma ini, aksi pasti dilakukan minimal satu kali, tanpa memperhatikan nilai pencacah. Di akhir iterasi pertama, baru dilakukan pengecekan jumlah kentang yang sudah terkupas (pencacah). Jalannya iterasi ini dapat ditulis dalam bentuk tabel berikut:

Iterasi ke-	{aksi}	kentang	Kentang < 10
1	Ya	1	Ya
2	Ya	2	Ya
3	Ya	3	Ya
4	Ya	4	Ya
5	Ya	5	Ya
6	Ya	6	Ya
7	Ya	7	Ya
8	Ya	8	Ya
9	Ya	9	Ya
10	Ya	10	Tidak

Pengulangan dihentikan pada iterasi ke- 10 karena kondisi kentang < 10 bernilai salah.

6) Algoritma untuk menampilkan karakter * sebanyak 5 kali.

ALGORITMA Tampil Bintang
IS : Jumlah bintang yg tampil = 0 FS : Jumlah bintang yg tampil = 5
KAMUS DATA i : integer
1 BEGIN 2 i ← 0 {inisialisasi pencacah i} 3 DO 4 Output ('*') {aksi} 5 i ← i + 1 {pencapaian kondisi berhenti} 6 WHILE (i < 5) {jumlah iterasi} 7 END

Output dari algoritma ini:

*

```
*
*
*
*
```

Pengulangan dihentikan pada iterasi ke- 5 karena kondisi $i < 5$ salah.

- 7) Algoritma untuk menampilkan iterasi ke 1 sampai 5 dengan pengulangan.

ALGORITMA Iterasi Angka
IS : - FS : Tampil angka 1 hingga 5
KAMUS DATA i : integer
1 BEGIN 2 i ← 1 {inisialisasi pencacah i} 3 DO 4 Output ('Ini adalah iterasi ke-',i){aksi} 5 i ← i + 1 6 WHILE (i <= 5) {kondisi berhenti} 7 END

Output dari algoritma ini:

```
Ini adalah iterasi ke-1
Ini adalah iterasi ke-2
Ini adalah iterasi ke-3
Ini adalah iterasi ke-4
Ini adalah iterasi ke-5
```

Pengulangan dihentikan pada iterasi ke- 5 dimana nilai $i = 6$ sehingga kondisi $i <= 5$ salah.

- 8) Algoritma untuk memasukkan nilai tiga (3) orang mahasiswa

ALGORITMA Input Nilai
IS : - FS : Menerima inputan nilai dari user
KAMUS DATA i : integer nilai : integer
1 BEGIN


```

2  i ← 1 {inisialisasi}
3  DO
4  Output ('Nilai mahasiswa ke-',i,'adalah:')
5  Input (nilai)
6  i ← i + 1
7  WHILE(i <= 3) {kondisi berhenti}
8  END

```

Output dari algoritma ini:

```

Nilai mahasiswa ke-1 adalah: _
Nilai mahasiswa ke-2 adalah: _
Nilai mahasiswa ke-3 adalah: _

```

Pengulangan dihentikan pada iterasi ke- 3, ketika nilai $i = 4$ sehingga kondisi $i \leq 3$ salah.

- 9) Algoritma untuk menghitung nilai rata-rata dari tiga bilangan yang diinputkan.

ALGORITMA Nilai Rata Rata	
IS : -	
FS : Tampil rata-rata dari nilai yang diinputkan	
KAMUS DATA	
jumlah : float	
bilangan : integer	
x : float	
rerata : float	
<pre> 1 BEGIN 2 jumlah ← 0 {variabel jumlah bilangan} 3 bilangan ← 3 {inisialisasi variabel pencacah} 4 DO 5 Output('Masukkan angka : ') 6 Input (x) {masukkan sebuah bilangan} 7 jumlah ← jumlah + x {tambahkan bilangan} 8 bilangan ← bilangan - 1 {pencacah mundur} 9 WHILE (bilangan > 0) 10 rerata ← jumlah / 3 {menghitung rerata} 11 Output ('Rerata : ',rerata) 12 END </pre>	

Pada algoritma ini juga ditunjukkan bahwa iterasi pada pengulangan dengan sintaks DO... WHILE... dapat dilakukan dengan pencacah mundur. Pengguna terus diminta memasukkan sebuah bilangan yang kemudian dijumlahkan dengan bilangan-bilangan sebelumnya (pada iterasi pertama, bilangan dijumlahkan dengan nol) hingga pencacah bernilai 0. Program akan memberikan output berupa hasil perhitungan rerata dari ketiga bilangan yang diinputkan.

Penggunaan DO...WHILE dalam pemrograman C :

Algoritma	C
DO aksi ubah pencacah WHILE (kondisi)	do { //aksi //ubah pencacah } while (kondisi);

Implementasi algoritma contoh ke-8 ke dalam C adalah :

```

1  //InputNilai.c
2  //User diminta untuk input nilai sebanyak 3 kali
3  #include <stdio.h>
4  main()
5  {
6  int i; //deklarasi pencacah
7  int nilai;
8  i=1; //inisialisasi pencacah
9  do
10 {
11 printf("Nilai mahasiswa ke-%d adalah : ",i);
12 scanf("%d",&nilai); //input nilai dari user
13 i=i+1; //penambahan nilai pencacah
14 }
15 while (i<=3); //kondisi berhenti
16 }

```

Tidak semua implementasi kondisi pada DO...WHILE sama dengan implementasi pada WHILE, contohnya adalah algoritma berikut :

WHILE
ALGORITMA Tebak Huruf
IS : - FS : Menampilkan pesan "Maaf Anda salah" jika tebakkan salah, dan menampilkan "Anda Benar" jika tebakkan benar
KAMUS DATA huruf : character
<pre> 1 BEGIN 2 Output('Masukkan tebakkan : ') 3 Input(huruf) 4 WHILE (huruf != 'q') 5 Output('Maaf Anda salah ') 6 Output('Masukkan tebakkan : ') 7 Input(huruf) 8 ENDWHILE 9 Output ('Anda Benar') 10 END </pre>
DO...WHILE
ALGORITMA Tebak Huruf
IS : - FS : Menampilkan pesan "Maaf Anda salah" jika tebakkan salah, dan menampilkan "Anda Benar" jika tebakkan benar
KAMUS DATA huruf : character
<pre> 1 BEGIN 2 Output('Masukkan tebakkan : ') 3 Input(huruf) 4 DO 5 Output('Maaf Anda salah ') 6 Output('Masukkan tebakkan : ') 7 Input(huruf) 8 WHILE (huruf != 'q') 9 Output ('Anda Benar') 10 END </pre>

Perbandingan output dari algoritma diatas adalah sebagai berikut

WHILE

Masukkan tebakan : q

Anda Benar

DO...WHILE

Masukkan tebakan : q

Maaf Anda Salah

Masukkan tebakan :

Bila user memasukkan inputan 'q' pada pertanyaan yang kedua ini, maka pengulangan akan dihentikan, tetapi jika user memasukkan huruf yang lain maka pengulangan akan dilanjutkan.

Dari contoh diatas dapat dilihat bahwa penggunaan sintaks WHILE dan DO...WHILE kadang akan memberikan output yang berbeda.

Sama seperti pada penggunaan sintaks WHILE, sintaks DO...WHILE dapat digunakan untuk struktur pengulangan yang belum diketahui jumlah iterasinya, tetapi tetap mempunyai kondisi berhenti.

Contoh struktur pengulangan tersebut adalah sebagai berikut :

ALGORITMA Menu

IS : -

FS : Menampilkan menu yang dipilih user

KAMUS DATA

pilihan : integer

```

1 BEGIN
2 DO
3   Output('MENU : ')
4   Output('1. Ulang')
5   Output('2. Keluar')
6   Output('Pilihan : ')
7   Input(pilihan)
8   WHILE (pilihan != 2) {pengecekan kondisi}
9   Output ('Anda Pilih Keluar')
10  END

```

Karena pada struktur DO...WHILE tidak terdapat pengecekan kondisi di awal, maka isi dari DO...WHILE akan

langsung dijalankan, pengecekan akan dilakukan setelah user memasukkan pilihan. Yang paling penting adalah perulangan ini mempunyai kondisi berhenti, yaitu ketika user input angka 2. Jumlah iterasi tidak ditentukan oleh variabel pencacah, melainkan ditentukan sendiri oleh user yang menginputkan angka. Berikut adalah implementasi dari algoritma Menu dalam bahasa pemrograman C.

```
1 //Menu.c
2 //User memasukkan pilihan menu 1 atau 2
3 #include <stdio.h>
4 main()
5 {
6     int pilihan;
7     do
8     {
9         printf("MENU");
10        printf("\n1. Ulang");
11        printf("\n2. Keluar");
12        printf("\nPilihan : ");
13        scanf("%d",&pilihan);
14    }
15    while (pilihan != 2);
16    printf("\nAnda pilih keluar");
17 }
```

Output dari sintaks diatas adalah

```
MENU
1. Ulang
2. Keluar
Pilihan : 1
MENU
1. Ulang
2. Keluar
Pilihan : 1
MENU
1. Ulang
2. Keluar
Pilihan : 2
Anda pilih keluar
```

5.4 Sintaks FOR

Sintaks pengulangan FOR merupakan sintaks yang relatif paling mudah digunakan. Sintaks ini serupa dengan sintaks WHILE... DO... dalam hal pengecekan kondisi dilakukan di awal. Dalam menggunakan struktur pengulangan dengan sintaks FOR, pemrogram harus mendefinisikan nilai awal dan nilai akhir pencacah yang menunjukkan jumlah iterasi. Setiap kali iterasi berlangsung, nilai pencacah akan diubah. Jika pencacah sudah mencapai nilai akhir yang ditentukan, maka pengulangan akan berhenti.

Bila contoh 'Ibu mengupas kentang' ingin diubah ke dalam struktur pengulangan dengan sintaks FOR, pemrogram harus menentukan nilai awal dan akhir pencacah, yaitu variabel kentang. Karena ibu akan mengupas kentang pertama hingga kentang ke sepuluh, maka:

- Nilai awal pencacah: kentang = 1
- Nilai akhir pencacah: kentang = 10
- Selama kondisi $\text{kentang} \geq 1$ dan $\text{kentang} \leq 10$ terpenuhi, aksi pengulangan akan dilakukan.

Struktur umum pengulangan dengan sintaks FOR adalah:

```
FOR(inisialisasi;KondisiPengulangan;PerubahNilaiPenc  
acah)  
    {pernyataan/perintah pengulangan}
```

ENDFOR

Dimana :

- Inisialisasi : untuk memberikan nilai awal untuk variabel pencacah.
- Kondisi Pengulangan : kondisi pengulangan akan berhenti atau tidak.
- Perubah Nilai Pencacah : pengubahan nilai variabel pencacah untuk mencapai kondisi berhenti, dapat berupa kenaikan ataupun penurunan. Pengubah variabel pencacah tidak harus selalu naik atau turun satu, tetapi dapat dilakukan pengubahan variabel pencacah lebih dari satu.
- Pernyataan perintah : aksi yang akan diulang

Maka, pada contoh-contoh sebelumnya dapat diubah dalam struktur FOR menjadi :

10) Algoritma ibu mengupas kentang

ALGORITMA Kupas Kentang	
IS : Terdapat 10 kentang belum dikupas	
FS : 10 kentang telah terkupas	
KAMUS DATA	
kentang : integer	
1	BEGIN
2	//inisialisasi,kondisi, dan pengubah pencacah
3	//dideklarasikan dalam sintaks for
4	FOR(kentang ← 0; kentang < 10; kentang++)
5	{aksi pengulangan}
6	Ambil sebuah kentang
7	Kupas kulit kentang
8	ENDFOR
9	END

Perhatikan bahwa potongan algoritma di atas tidak mencantumkan aksi pengubah pencacah kentang ← kentang + 1. Inisialisasi, pengecekan kondisi, dan pengubah variabel pencacah sudah terdapat dalam argumen FOR. Pada posisi pengubah variabel, pernyataan kentang++ sama dengan kentang←kentang + 1,penambahan akan dilakukan setelah aksi pengulangan dilaksanakan.

Jalannya iterasi ini dapat ditulis dalam bentuk tabel berikut:

Iterasi ke-	kentang	kentang< 10?	{aksi}
1	0	Ya	Ya
2	1	Ya	Ya
3	2	Ya	Ya
4	3	Ya	Ya
5	4	Ya	Ya
6	5	Ya	Ya
7	6	Ya	Ya

8	7	Ya	Ya
9	8	Ya	Ya
10	9	Ya	Ya
11	10	Tidak	-

Pengulangan dihentikan pada iterasi ke- 11 karena kondisi $kentang < 10$ bernilai salah.

- 11) Algoritma untuk menampilkan karakter '*' sebanyak 5 kali.

ALGORITMA Tampil Bintang
IS : Jumlah bintang yg tampil = 0 FS : Jumlah bintang yg tampil = 5
KAMUS DATA i : integer
1 BEGIN 2 FOR(i ← 0; i < 5; i++) 3 Output ('*') {aksi} 4 ENDFOR 5 END

Output dari algoritma ini:

* * * * *

Pengulangan dihentikan pada iterasi ke- 6 karena kondisi $i < 5$ bernilai salah. Perhatikan bahwa pencacah dapat dimulai dari angka berapapun hingga berapapun sesuai kebutuhan program.

- 12) Algoritma untuk menampilkan iterasi ke 1 sampai 5 dengan pengulangan.

ALGORITMA Iterasi Angka
IS : - FS : Jumlah bintang yg tampil = 5
KAMUS DATA i : integer
1 BEGIN


```

2  FOR(i ← 1; i ≤ 5;i++)
3  Output ('Ini adalah iterasi ke-',i){aksi}
4  ENDFOR
5  END

```

Output dari algoritma ini:

```

Ini adalah iterasi ke-1
Ini adalah iterasi ke-2
Ini adalah iterasi ke-3
Ini adalah iterasi ke-4
Ini adalah iterasi ke-5

```

Pengulangan dihentikan pada iterasi ke- 6 karena kondisi $i \leq 5$ bernilai salah.

13) Algoritma untuk memasukkan nilai tiga (3) orang mahasiswa

```

ALGORITMA Input_Nilai
IS : -
FS : Menerima inputan nilai dari user
KAMUS DATA
  i      : integer
  nilai  : integer
1  BEGIN
2  FOR(i ← 1; i ≤ 3;i++)
3  Output ('Nilai mahasiswa ke-',i,'adalah:')
4  Input (nilai)
5  ENDFOR
6  END

```

Output dari algoritma ini:

```

Nilai mahasiswa ke-1 adalah: _
Nilai mahasiswa ke-2 adalah: _
Nilai mahasiswa ke-3 adalah: _

```

Pengulangan dihentikan pada iterasi ke- 4 karena kondisi $i \leq 3$ bernilai salah.

Serupa dengan struktur pengulangan dengan sintaks lain, struktur pengulangan dengan sintaks FOR juga dapat dilakukan dengan pencacah mundur. Berikut ini adalah beberapa

contoh penggunaan struktur pengulangan FOR dengan pencacah mundur :

- 14) Program untuk menghitung rerata dari tiga bilangan yang diinputkan.

ALGORITMA Nilai Rata Rata	
IS : -	
FS : Tampil rata-rata dari nilai yang diinputkan	
KAMUS DATA	
jumlah : float	
bilangan : integer	
x : float	
rerata : float	
1 BEGIN	
2 jumlah \leftarrow 0 {variabel jumlah bilangan}	
3 {inisialisasi variabel pencacah}	
4 FOR(bilangan \leftarrow 3; bilangan > 0;bilangan--)	
5 Output('Masukkan angka : ')	
6 Input (x)	
7 jumlah \leftarrow jumlah + x	
8 ENDFOR	
9 rerata \leftarrow jumlah/ 3 {menghitung rerata}	
10 Output ('Rerata : ',rerata)	
11 END	

Pada algoritma, pengguna terus diminta memasukkan sebuah bilangan yang kemudian dijumlahkan dengan bilangan-bilangan sebelumnya (pada iterasi pertama, bilangan dijumlahkan dengan nol) hingga pencacah bernilai 0. Pengulangan dihentikan pada iterasi ke-4 karena kondisi `bilangan>0` bernilai salah. Program akan memberikan output berupa hasil perhitungan rerata dari ketiga bilangan yang diinputkan.

- 15) Algoritma petasan: program akan menghitung mundur dengan menampilkan sejumlah bilangan tertentu, kemudian mengeluarkan pesan 'DOR!!!' di akhir perhitungan mundur.

ALGORITMA Hitung_Mundur
IS : FS : Menampilkan angka dari 5 hingga 1
KAMUS DATA hitung : integer
1 BEGIN 2 FOR(hitung ← 5; hitung > 0;bilangan--) 3 Output(hitung) 4 ENDFOR 5 Output ('DOR!!!') 6 END

Output dari algoritma ini:

5 4 3 2 1 DOR!!!

- 16) Algoritma ibu mengupas kentang dengan perhitungan mundur

ALGORITMA Kupas Kentang
IS : Terdapat 10 kentang belum dikupas FS : 10 kentang telah terkupas
KAMUS DATA kentang : integer
1 BEGIN 2 //inisialisasi,kondisi, dan pengubah pencacah 3 //dideklarasikan dalam sintaks for 4 FOR(kentang ← 10; kentang > 0; kentang--) 5 Ambil sebuah kentang 6 Kupas kulit kentang 7 ENDFOR 8 END

Pada algoritma di atas, iterasi dilakukan 10 kali dan akan dihentikan pada iterasi ke 11 karena kondisi `kentang>0` bernilai salah.

Penulisan sintaks FOR dalam bahasa pemrograman C:

For dengan satu aksi

```
int i;  
for(i=0;i<5;i++)  
    printf("%d",i);
```

For dengan banyak aksi

```
int i;  
for(i=0;i<5;i++)  
{  
    printf("Masukkan angka ke-%d",i);  
    scanf("%d",&nilai);  
}
```

Implementasi algoritma contoh ke-11 ke dalam bahasa pemrograman C :

```
1 //bintang.c  
2 #include <stdio.h>  
3 main()  
4 {  
5     int i;  
6     for(i=0;i<5;i++)  
7         printf("\n*");  
8 }
```

Seperti pada penggunaan sintaks WHILE dan DO...WHILE, dimana kita dapat mengimplementasikan struktur pengulangan yang belum diketahui jumlah iterasinya, tetapi tetap mempunyai kondisi berhenti. Pada sintaks FOR, kita dapat menggunakannya untuk pengulangan dengan jumlah iterasi tak berhingga.

Contoh struktur pengulangan tersebut adalah sebagai berikut :

ALGORITMA Menu
IS : -

FS : Menampilkan menu yang dipilih user	
KAMUS DATA	
pilihan	: integer
1 BEGIN	
2 FOR(;pilihan!=2;)	
3 Output('MENU : ')	
4 Output('1. Ulang')	
5 Output('2. Keluar')	
6 Output('Pilihan : ')	
7 Input(pilihan)	
8 ENDFOR	
9 Output ('Anda Pilih Keluar')	
10 END	

Argumen yang digunakan di dalam FOR tidak harus selalu lengkap, kadang bagian inialisasi tidak digunakan

```
FOR(;KondisiPengulangan;PerubahNilaiPencacah)  
    {pernyataan/perintah pengulangan}
```

```
ENDFOR
```

Atau bagian Kondisi Pengulangan tidak kita perlukan

```
FOR(Inisialisai;;PerubahNilaiPencacah)  
    {pernyataan/perintah pengulangan}
```

```
ENDFOR
```

Ataupun bagian Perubah Nilai Pencacah dihilangkan seperti

```
FOR(Inisialisai;KondisiPengulangan;)  
    {pernyataan/perintah pengulangan}
```

```
ENDFOR
```

Juga bisa digunakan kombinasi seperti contoh algoritma di atas, dimana bagian inialisasi dan perubah nilai pencacah dihilangkan, tetapi meskipun dihilangkan, pengulangan harus tetap mempunyai kondisi berhenti

5.5 Sintaks Pengulangan Bersarang

Sama halnya dengan struktur pemilihan, struktur pengulangan juga dapat disusun bersarang. Sebuah struktur pengulangan bisa berada dalam struktur pengulangan lainnya. Atau, sebuah struktur pengulangan bisa mengandung struktur pengulangan lain di dalamnya.

Dari contoh ibu mengupas kentang, misalnya dengan struktur pengulangan WHILE berikut ini:

ALGORITMA Kupas Kentang
IS : Terdapat 10 kentang belum dikupas FS : 10 kentang telah terkupas
KAMUS DATA kentang : integer
1 BEGIN {inisialisasi jumlah kentang yang sudah dikupas} 2 kentang ← 0 3 WHILE (kentang < 10){kondisi iterasi dilakukan} 4 Ambil sebuah kentang 5 Kupas kulit kentang 6 kentang←kentang+1 {pencapaian kondisi berhenti} 7 ENDWHILE 8 END

Setiap kali ibu mengambil sebuah kentang, ibu akan mengupas kulit kentang kemudian langsung memotongnya menjadi empat bagian. Berdasarkan kondisi ini, potongan algoritma di atas dapat dilengkapi menjadi:

ALGORITMA Kupas Kentang Potong 4
IS : Terdapat 10 kentang belum dikupas FS : 10 kentang telah terkupas dan teriris menjadi 4 bagian
KAMUS DATA kentang : integer potongan: integer
1 BEGIN {inisialisasi jumlah kentang yang sudah dikupas} 2 kentang ← 0 3 WHILE (kentang < 10){kondisi iterasi dilakukan} 4 Ambil sebuah kentang 5 Kupas kulit kentang 6 potongan ← 1 {inisialisai jml potongan kentang} 7 DO 8 Potong kentang 9 potongan = potongan + 1

```

10 WHILE (potongan<=4)
11 kentang←kentang+1 {pencapaian kondisi berhenti}
12 ENDWHILE
13 END

```

Pada contoh di atas, pengulangan luar (WHILE) merupakan pengulangan untuk aksi mengupas kentang, sedangkan pengulangan dalam (DO...WHILE) merupakan pengulangan untuk memotong kentang ke dalam 4 bagian. Pengulangan dalam akan berhenti ketika jumlah potongan = 4.

Contoh lain penggunaan sintaks pengulangan bersarang ini adalah sebagai berikut:

- 17) Algoritma untuk menampilkan karakter * sebanyak 5 kali, masing-masing sebanyak tiga kali.

ALGORITMA Tampil Bintang
IS : - FS : Menampilkan bintang sebanyak 3 kolom dan 5 baris
KAMUS DATA i : integer j : integer
<pre> 1 BEGIN 2 FOR(i ← 0; i < 5; i++) 3 FOR(j ← 0; j < 3; j++) 4 Output ('*') {aksi} 5 ENDFOR 6 ENDFOR 7 END </pre>

Output dari algoritma ini:

```

***
***
***
***
***

```

- 18) Algoritma untuk memasukkan nilai tiga (3) orang mahasiswa, yang masing-masing memiliki dua nilai

ALGORITMA Input_Nilai
IS :

FS : Nilai mahasiswa telah diinputkan
KAMUS DATA
i : integer
j : integer
nilai : integer
1 BEGIN
2 FOR(i ← 1; i ≤ 3;i++)
3 Output ('Nilai mahasiswa ke-',i,'adalah:')
4 FOR(j←1; j≤2; j++)
5 Output('Nilai ke-',j,':')
6 Input (nilai)
7 ENDFOR
8 ENDFOR
9 END

Output dari algoritma ini:

Nilai mahasiswa ke-1 adalah
Nilai ke-1: _
Nilai ke-2: _
Nilai mahasiswa ke-2 adalah
Nilai ke-1: _
Nilai ke-2: _
Nilai mahasiswa ke-3 adalah
Nilai ke-1: _
Nilai ke-2: _

Pengulangan bersarang dapat juga dilakukan dengan beberapa struktur pengulangan yang berbeda, bahkan dapat juga digabungkan dengan struktur pemilihan. Contoh:

- 19) Algoritma untuk menampilkan karakter * untuk membentuk garis pembatas kotak sebesar lima kolom dan lima baris.

ALGORITMA Tampil Bintang Kotak
IS : -
FS : Menampilkan tanda bintang berbentuk kotak
KAMUS DATA
i : integer
j : integer
1 BEGIN


```
2  FOR(i ← 1; i ≤ 5; i++){pengulangan baris}
3  IF (i=1 OR i=5)    {baris ke-1 dan ke-5}
4  j←1                {pencacah kolom}
5  DO
6  Output ('*')
7  j←j+1
8  WHILE (j≤5)
9  ELSE                {baris ke-2,3 dan 4}
10 j←1
11 DO
12 IF (j=1 OR j=5)
13 Output('*')
14 ELSE
15 Output(' ')
16 ENDIF
17 j←j+1
18 WHILE (j≤5)
19 ENDIF
20 ENDFOR
21 END
```

Implementasi algoritma diatas ke dalam C

```
1  //Bintang_Kotak.c
2  //menggambar * membentuk kotak
3  #include <stdio.h>
4  main()
5  {
6  int i,j;
7  for(i=1;i≤5;i++) //perulangan baris
8  {
9  //jika kursor di posisi baris 1,5
10 if ((i==1) || (i==5))
11 {
12 j=1;
13 do //perulangan kolom utk baris 1,5
14 {
15 printf("*");
16 j=j+1;
17 }
18 while (j≤5);
```

```
19 }
20 //jika kursor di posisi baris 2,3,4
21 else
22 {
23 j=1;
24 do //perulangan kolom utk baris 2,3,4
25 {
26 //jika cursor berada di kolom 1,5
27 if ((j==1) || (j==5))
28 printf("*");
29 //jika cursor berada di kolom 2,3,4
30 else
31 printf(" ");
32 j=j+1;
33 }
34 while(j<=5);
35 }
36 //ke baris berikutnya
37 printf("\n");
38 }
39 }
```

Output dari algoritma ini:

```
*****
*      *
*      *
*      *
*****
```

5.6 Sintaks *BREAK* dan *CONTINUE*

Sintaks *BREAK* dan *CONTINUE* merupakan sintaks yang digunakan untuk menghentikan pengulangan dan melanjutkan ke perintah atau aksi berikutnya. Sintaks *BREAK* dan *CONTINUE* dapat digunakan baik di dalam struktur pengulangan *WHILE*, *DO...WHILE*, dan *FOR*.

Sintaks *BREAK* digunakan untuk menghentikan pengulangan kemudian keluar dari struktur pengulangan tanpa melanjutkan perintah di dalam struktur pengulangan.

Berikut adalah pengulangan dengan *FOR* tanpa menggunakan *BREAK* atau *CONTINUE* :

ALGORITMA Iterasi_Angka
IS : - FS : Menampilkan angka 1 hingga 6
KAMUS DATA i : integer
1 BEGIN 2 FOR(i ← 1; i ≤ 6;i++) 3 Output ('Ini adalah iterasi ke-',i){aksi} 4 ENDFOR 5 OUTPUT('Akhir pengulangan') 6 END

Jika algoritma diatas dijalankan, maka akan menghasilkan output sebagai berikut :

Ini adalah iterasi ke-1 Ini adalah iterasi ke-2 Ini adalah iterasi ke-3 Ini adalah iterasi ke-4 Ini adalah iterasi ke-5 Ini adalah iterasi ke-6 Akhir pengulangan

Contoh penggunaan sintaks *BREAK* di dalam pengulangan dengan menggunakan *FOR*

ALGORITMA Iterasi Angka Break
IS : - FS : Menampilkan angka 1 hingga 6 tetapi berhenti di angka 3
KAMUS DATA i : integer
1 BEGIN 2 FOR(i ← 1; i ≤ 6;i++) 3 IF (i=4) 4 BREAK

```
5  ENDIF
6  Output ('Ini adalah iterasi ke-',i){aksi}
7  ENDFOR
8  OUTPUT('Akhir pengulangan')
9  END
```

Jika algoritma diatas dijalankan, maka akan menghasilkan output sebagai berikut :

```
Ini adalah iterasi ke-1
Ini adalah iterasi ke-2
Ini adalah iterasi ke-3
Akhir pengulangan
```

Ketika variabel *i* mencapai angka 4 maka akan memenuhi syarat untuk masuk ke struktur IF, kemudian perintah BREAK dijalankan yaitu keluar dari struktur pengulangan FOR, dan menampilkan perintah pada baris ke-8.

Sintaks *CONTINUE* digunakan untuk kembali ke awal pengulangan tanpa menjalankan perintah berikutnya. Contoh penggunaan sintaks *CONTINUE* di dalam pengulangan dengan menggunakan FOR

```
ALGORITMA Iterasi Angka Continue
IS : -
FS : Menampilkan angka 1 hingga 6
KAMUS DATA
    i : integer
1  BEGIN
2  FOR(i ← 1; i <= 6;i++)
3  IF (i=4)
4  CONTINUE
5  ENDIF
6  Output ('Ini adalah iterasi ke-',i){aksi}
7  ENDFOR
8  OUTPUT('Akhir pengulangan')
9  END
```

Jika algoritma diatas dijalankan, maka akan menghasilkan output sebagai berikut :

```
Ini adalah iterasi ke-1
Ini adalah iterasi ke-2
Ini adalah iterasi ke-3
```

```
Ini adalah iterasi ke-5  
Ini adalah iterasi ke-6  
Akhir pengulangan
```

Ketika variabel *i* mencapai angka 4 maka akan memenuhi syarat untuk masuk ke struktur IF, kemudian perintah CONTINUE dijalankan yaitu kembali ke awal pengulangan dimana akan dilakukan penambahan nilai variabel pencacah dan pengecekan kondisi. Ketika perintah CONTINUE dijalankan, maka perintah berikutnya yaitu perintah pada baris ke-6 tidak dijalankan, sehingga tampilan Ini adalah iterasi ke-4 tidak keluar. Setelah kembali ke awal pengulangan, variabel *i* akan ditambah menjadi 5 kemudian masuk lagi ke struktur pengulangan.

Berikut adalah implementasi algoritma di atas ke dalam bahasa pemrograman C :

```
1 //TampilAngkaBreak.c  
2 #include <stdio.h>  
3 main()  
4 {  
5     int i;  
6     for(i=1; i<=6; i++)  
7     {  
8         if (i==4)  
9         {  
10            break;  
11        }  
12        printf("\nIni adalah iterasi ke-%d",i);  
13    }  
14    printf("\nAkhir pengulangan");  
15 }
```

```
1 //TampilAngkaContinue.c  
2 #include <stdio.h>  
3 main()  
4 {  
5     int i;  
6     for(i=1; i<=6; i++)
```

```
7 {
8   if (i==4)
9   {
10    continue;
11   }
12   printf("\nIni adalah iterasi ke-%d",i);
13   }
14   printf("\nAkhir pengulangan");
15 }
```

Jika tidak terdapat kondisi berhenti yang tepat untuk suatu pengulangan, maka kita dapat menggunakan sintaks BREAK untuk keluar dari suatu pengulangan. Berikut contoh pengulangan FOR dimana semua argumen di dalam FOR dihilangkan, hal ini akan memberikan pengulangan tak berhingga.

```
//Menu.c
//User memasukkan pilihan menu 1 atau 2
#include <stdio.h>
main()
{
    int pilihan;
    for (;;)
    {
        printf("MENU");
        printf("\n1. Ulang");
        printf("\n2. Keluar");
        printf("\nPilihan : ");
        scanf("%d",&pilihan);
        if (pilihan == 2)
            break;
    }
    printf("\nAnda pilih keluar");
}
```



1. Struktur pengulangan memungkinkan program melakukan satu atau lebih aksi beberapa kali.
2. Tiga komponen pengendali aksi adalah inisialisasi, jumlah iterasi, dan kondisi berhenti.
3. Tiga struktur pengulangan yang dapat digunakan adalah struktur pengulangan dengan sintaks WHILE, DO...WHILE, dan FOR.
4. Struktur pengulangan dapat dibuat bersarang dengan sintaks pengulangan yang sama atau berbeda, bahkan dapat digabungkan dengan struktur pemilihan.
5. Untuk keluar dari struktur pengulangan sebelum kondisi berhenti, kita dapat menggunakan sintaks BREAK
6. Untuk melanjutkan struktur pengulangan ke awal pengulangan maka dapat digunakan sintaks CONTINUE
7. Hal yang terpenting dari struktur pengulangan adalah kondisi berhenti yang akan memberikan kondisi apakah pengulangan dilakukan atau tidak.



Kuis Benar Salah

1. Struktur pengulangan dengan sintaks WHILE tidak mencantumkan inisialisasi.
2. Struktur pengulangan dengan sintaks DO...WHILE tidak mencantumkan kondisi berhenti.
3. Struktur pengulangan dengan sintaks FOR tidak mencantumkan jumlah iterasi.
4. Aksi yang dituliskan didalam sintaks WHILE akan dijalankan jika kondisi pada WHILE... bernilai benar.
5. Aksi yang dituliskan setelah DO... pada sintaks pengulangan DO... WHILE... dijalankan jika kondisi yang dituliskan setelah WHILE bernilai salah.
6. Aksi yang dituliskan didalam sintaks pengulangan FOR dijalankan jika kondisi yang dituliskan didalam argumen FOR... bernilai benar.
7. Pengulangan dengan sintaks WHILE dapat diubah dengan sintaks DO...WHILE.
8. Pengulangan dengan sintaks DO... WHILE... dapat diubah dengan sintaks FOR.
9. Pengulangan dengan sintaks FOR dapat diubah dengan sintaks WHILE.
10. Struktur pengulangan bersarang mensyaratkan bahwa variabel pencacah dari seluruh pengulangan sama.

(Untuk soal nomor 11 – 15) Perhatikan potongan algoritma di bawah ini:

ALGORITMA	
IS	:
FS :	
KAMUS DATA	
a : integer	
nilai : integer	


```
1  a ← 0
2  WHILE (a < 15)
3  Output ('Nilai mahasiswa ke-',a,'adalah:')
4  Input (nilai)
5  IF ((nilai >= 0) AND (nilai <= 100))
6  a ← a + 1
7  ELSE
8  Output ('Nilai harus antara 0-100')
9  ENDIF
10 ENDWHILE
```

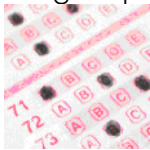
- 11. Variabel a pada contoh di atas merupakan pencacah.
- 12. Pengulangan di atas akan mengalami sebanyak 16 kali iterasi.
- 13. Jika nilai yang diinputkan tidak berada dalam range 0-100, maka program akan diakhiri.
- 14. Nilai pencacah akan bertambah 1 jika nilai yang diinputkan sudah berada dalam range 0-100.
- 15. Inisialisasi variabel a membuat program tidak dapat berjalan dengan semestinya.

(Untuk soal nomor 6-10) Perhatikan potongan algoritma berikut ini:

ALGORITMA	
IS	:
FS :	
KAMUS DATA	
a : integer	
b : integer	
1 a ← 1	
2 DO	
3 b ← a mod 2	
4 IF (b = 0)	
5 Output ('o')	
6 ELSE	
7 Output ('x')	
8 ENDIF	
9 a ← a + 1	

```
10 WHILE (a <= 5)
```

16. Struktur pengulangan di atas tidak tepat karena tidak mencantumkan jumlah iterasi yang akan dilakukan.
17. Iterasi akan dilakukan tepat lima kali.
18. Output dari algoritma di atas adalah 'xoxox'.
19. Struktur pengulangan di atas menggunakan pencacah mundur.
20. Pada algoritma di atas, terdapat perbedaan aksi untuk nilai a berupa bilangan ganjil dan a berupa bilangan genap.



Pilihan Ganda

1. Perhatikan potongan algoritma berikut ini:

ALGORITMA	
IS	:
FS :	
KAMUS DATA	
a : integer	
1 WHILE (a < 7)	
2 Output ('Ini contoh program')	
3 a ← a + 1	
4 ENDWHILE	

Jika output yang diinginkan adalah menampilkan 'Ini contoh program' sebanyak tujuh kali, maka inisialisasi yang tepat untuk pengulangan di atas yaitu

- A. $a \leftarrow 0$
 - B. $a \leftarrow 1$
 - C. $a > 0$
 - D. $a > 1$
 - E. pilihan A,B,C,D salah
2. Apakah yang dimaksud dengan kondisi berhenti dalam struktur pengulangan?
 - A. Kondisi yang menyebabkan inisialisasi berhenti

- B. Kondisi yang menyebabkan iterasi berhenti
 - C. Kondisi yang menyebabkan program berhenti
 - D. Kondisi yang akan berhenti saat iterasi berhenti
 - E. Kondisi yang akan berhenti saat program berhenti
3. Perhatikan potongan algoritma berikut ini:

ALGORITMA		
IS		:
FS	:	
KAMUS DATA		
e : integer		
huruf : character		
1	e ← 1	
2	DO	
3	Input (huruf)	
4	e ← e + 1	
5	WHILE ((huruf != 'a') OR (e != 10))	

- Apakah yang dilakukan oleh algoritma di atas?
- A. Meminta inputan sebuah huruf hingga 10 kali
 - B. Meminta inputan sebuah huruf hingga huruf yang diinputkan 'a'
 - C. Meminta inputan sebuah huruf hingga huruf yang diinputkan 'e'
 - D. Meminta inputan sebuah huruf hingga huruf yang diinputkan 'a' atau iterasi sudah 10 kali
 - E. Meminta inputan sebuah huruf hingga huruf yang diinputkan 'e' atau iterasi sudah 10 kali.
4. Pernyataan manakah yang salah tentang variabel pencacah?
- A. Variabel pencacah mengendalikan jumlah iterasi dalam struktur pengulangan.
 - B. Nilai variabel pencacah terus berubah setiap kali iterasi.
 - C. Nilai variabel pencacah akan berhenti berubah di akhir pengulangan.
 - D. Variabel pencacah dapat bertipe integer maupun

char.

E. Nilai variabel pencacah tidak perlu diinisialisasi di awal iterasi.

5. Pada sebuah konser, panitia memberikan bonus 1 tiket untuk setiap pembelian 3 tiket. Harga 1 tiket adalah Rp. 100.000,-. Panitia membuat algoritma untuk menghitung harga yang harus dibayar setiap orang yang membeli tiket. Perhatikan potongan algoritma berikut ini:

ALGORITMA		
IS		:
FS	:	
KAMUS DATA		
jmlTiket : integer		
jmlBayar : integer		
harga : integer		
i : integer		
1	Input (jmlTiket)	
2	jmlBayar \leftarrow 0	
3	IF (jmlTiket < 1)	
4	Output ('Jumlah tiket minimal 1')	
5	ELSE	
6	BEGIN	
7	FOR (i \leftarrow 1; i<= jmlTiket; i++)	
8	IF (jmlTiket > 3)	
9	BEGIN	
10	jmlTiket \leftarrow jmlTiket - 3	
11	jmlBayar \leftarrow jmlBayar + 3	
12	END	
13	ELSE	
14	jmlBayar \leftarrow jmlBayar + jmlTiket	
15	ENDIF	
16	ENDFOR	
17	harga \leftarrow jmlBayar * 100000	
18	Output (harga)	
19	END	
20	ENDIF	

Pernyataan manakah yang benar tentang potongan algoritma di atas?

- A. Algoritma di atas tidak dapat menghitung harga untuk 1 tiket
 - B. Algoritma di atas memberikan output salah jika input $jmlTiket > 3$
 - C. Pengulangan tidak pernah dilakukan jika input $jmlTiket = 0$
 - D. Pengulangan tidak pernah dilakukan jika input $jmlTiket > 3$
 - E. Tidak ada masalah dalam algoritma di atas
6. Apakah output dari algoritma nomor 5 jika input $jmlTiket = 10$?
- A. 600000
 - B. 800000
 - C. 100000
 - D. 120000
 - E. Algoritma error
7. Variabel $jmlBayar$ pada algoritma nomor 5 digunakan untuk menyimpan nilai ...
- A. Harga 1 tiket
 - B. Harga total tiket
 - C. Jumlah pengulangan
 - D. Jumlah tiket yang harus dibayar
 - E. Jumlah tiket gratis
8. Sintaks pengulangan manakah yang melakukan pengecekan kondisi di akhir pengulangan?
- A. DO... WHILE...
 - B. WHILE... ..
 - C. FOR...
 - D. IF... THEN... ELSE...
9. Tipe data apakah yang dapat digunakan untuk variabel pencacah?
- A. Integer
 - B. Text
 - C. Float
 - D. Real
 - E. Array
10. Perhatikan potongan algoritma berikut ini:

ALGORITMA	
IS	:
FS :	
KAMUS DATA	
i : integer	
j : character	
1 FOR (i ← 1 ; i ≤ 2 ; i++)	
2 IF (i mod 2 = 0)	

```
3  FOR (j ← 1 ; j<= 4 ; j++)
4  IF (j mod 2 = 1)
5  Output ('x')
6  ELSE
7  Output ('o')
8  ENDFOR
9  ELSE
10 Output (i)
11 ENDIF
12 ENDFOR
```

Apakah output dari algoritma di atas?

- | | |
|---------|---------|
| A. 1 | D. oxox |
| xoxo | 1 |
| B. xoxo | E. 1 |
| 1 | 1 |
| C. 1 | |
| Oxox | |



Latihan

1. Buatlah program dengan struktur pengulangan untuk menampilkan tulisan "Saya suka mata kuliah ini" sebanyak 5 kali!

2. Bandingkan tiga sintaks pengulangan: WHILE, DO...WHILE, dan FOR kemudian sebutkan kekurangan dan kelebihan masing-masing sintaks!
3. Buatlah program pengulangan untuk menghitung jumlah sederet bilangan berurut yang dimulai dari 1 hingga bilangan inputan. Contoh:

```
INPUT      : 7
PROSES     : 1+2+3+4+5+6+7
OUTPUT     : 28
```

4. Buatlah program pengulangan bersarang dengan sintaks FOR untuk menampilkan output sebagai berikut:

```
*
**
***
****
```

5. Buatlah program pengulangan bersarang dengan sintaks FOR untuk menampilkan output sebagai berikut:

```
*****
****
***
**
*
```

6. Buatlah program pengulangan bersarang dengan sintaks FOR untuk menampilkan output sebagai berikut:

```
1
22
333
4444
55555
```

Dengan jumlah baris sesuai inputan. (Pada contoh di atas, input = 5)

7. Ubahlah program nomor 7 dengan sintaks WHILE !
8. Ubahlah program nomor 7 dengan sintaks DO...WHILE !
9. Apakah fungsi inisialisasi dalam sebuah struktur pengulangan?
10. Apakah fungsi pencacah dalam sebuah struktur pengulangan?

6 Array dan Tipe Data Bentukan



Overview

Dalam dunia nyata, struktur data yang dihadapi sangat beragam dan penggunaan variabel dengan tipe data dasar memiliki keterbatasan pada banyaknya nilai yang dapat disimpan. Dengan menggunakan *array* dan tipe data bentukan, dapat dilakukan pemodelan struktur data dengan lebih baik bahkan untuk struktur data yang relatif kompleks.



Tujuan

1. Memahami tipe data *array* dan keuntungan yang dapat diberikan
2. Memahami *array* yang memiliki dimensi lebih dari satu
3. Dapat meng-implementasikan tipe data *array* dalam program
4. Memahami cara menentukan tipe data bentukan dan

menggunakannya dalam program

6.1 Array

Tipe data *array* adalah tipe data terstruktur yang merujuk kepada sebuah atau sekumpulan elemen yang mempunyai tipe data yang sama melalui indeks. *Array* biasanya disebut juga sebagai **tabel, vektor atau larik**.

Elemen dari *array* dapat diakses langsung **jika dan hanya jika** indeks terdefinisi (telah ditentukan nilainya sesuai dengan domain yang didefinisikan untuk indeks tersebut). Struktur data *array* disimpan dengan urutan yang sesuai dengan definisi indeks **secara kontigu (berurutan) dalam memori** komputer. Karena itu indeks haruslah merupakan suatu tipe data yang memiliki keterurutan (ada suksesor dan predesesor), misal tipe integer dan karakter.

Dilihat dari dimensinya, *array* dapat dibagi menjadi *Array Satu Dimensi*, *Array Dua Dimensi* dan *Array Multi-Dimensi*

6.1.1 Array Satu Dimensi



Untuk mendeklarasikan variabel dengan tipe data *array* satu dimensi pada notasi algoritma, digunakan pola sebagai berikut:

```
...  
KAMUS DATA  
  Nama_variabel : array [x..y] of tipe_data  
...
```

Keterangan:

Nilai x merupakan nilai awal indeks pada *array*, dan nilai y merupakan nilai akhir pada indeks *array*.

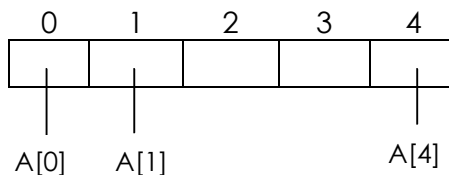
Contoh :

Algoritma

```
...  
Kamus data  
  arrHari : array [1..7] of string  
  arrJmlBulan : array [1..12] of integer  
  arrFrekuensi : array ['a'..'z'] of integer  
...
```

Mengakses data array satu dimensi:

Array satu dimensi diakses melalui indeksnya. Misal akan disiapkan array satu dimensi A bertipe integer dengan 5 elemen yang diberi nomor indeks dari 0 sampai 4, yang dapat diilustrasikan dengan gambar berikut:



Karena *array* tersebut mempunyai nama yang sama, yaitu A, maka setiap elemen diberi sebutan nama yang berbeda dengan memberikan nomor indeks, sehingga masing-masing menjadi: A[0], A[1], sampai dengan A[4], yang dapat dibaca dengan:

A dengan indeks 0 atau A nol

A dengan indeks 1 atau A satu


```

1. ALGORITMA
2. /* Menyiapkan dan memasukkan nilai dalam array satu dimensi
3.   I.S: array dalam keadaan kosong
4.   F.S: menampilkan nilai yang disimpan dalam array dengan menggunakan struktur
5.     pengulangan */
6. KAMUS DATA
7.   A : array[0..4] of integer
8.   i : integer
9. BEGIN
10.  A[0] ← 4   /*simpan 4 dalam array A indeks 0*/
11.  A[1] ← 8   /*simpan 8 dalam array A indeks 1*/
12.  A[2] ← 6
13.  A[3] ← A[0] + A[1]
14.  A[4] ← A[2]
15.  /*menampilkan kembali nilai dalam array*/
16.  For (i=0; i<=4; i++)
17.    output("A[" , i, "]" = " , A[i])
18.  EndFor
19. END.

```

Dalam Bahasa C

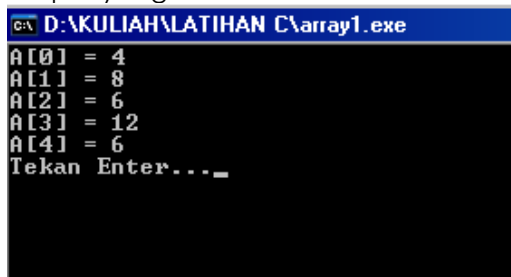
```

1. /*Menyiapkan dan memasukkan nilai dalam array satu dimensi
2.   I.S: array dalam keadaan kosong
3.   F.S: menampilkan nilai yang disimpan dalam array dengan menggunakan struktur
4.     pengulangan */
5.
6. #include <stdio.h>
7. #include <conio.h>
8.
9. void main()
10. {
11.   int A[5];   /*deklarasi array A dengan 5 elemen*/
12.   int i;
13.   A[0] = 4;   /*simpan 4 dalam array A indeks 0*/
14.   A[1] = 8;
15.   A[2] = 6;
16.   A[3] = A[0] + A[1];
17.   A[4] = A[2];
18.   /*menampilkan kembali nilai dalam array*/

```

```
19. for(i=0;i<=4;i++)
20. {
21.     printf("A[%i] = %i\n",i,A[i]);
22. }
23. printf("Tekan Enter...");
24. getch(); /* menahan tampilan pada layar */
25.}
```

Output yang dihasilkan:



6.1.2 Array Dua Dimensi

Array dua dimensi merupakan *array* yang terdiri dari *m* buah baris (*row*) dan *n* buah kolom (*column*). Bentuk *array* semacam ini menggunakan 2 (dua) buah kelompok indeks yang masing-masing direpresentasikan sebagai indeks baris dan kolom. Jika ingin memasukkan atau membaca sebuah nilai pada matriks maka, harus diketahui terlebih dahulu indeks baris dan kolomnya.

Untuk mendeklarasikan variabel dengan tipe data *array* dua dimensi pada notasi algoritma, digunakan pola sebagai berikut:

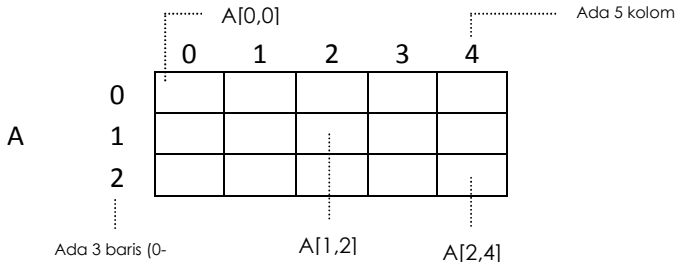
```
...
KAMUS DATA
    Nama_variabel : array [x..y,t..u] of tipe_data
...
```

Keterangan:

Nilai *x* merupakan nilai awal indeks baris pada *array*, dan nilai *y* merupakan nilai akhir indeks baris *array*. Nilai *t* merupakan nilai

awal indeks kolom pada *array*, dan nilai *u* merupakan nilai akhir indeks kolom *array*.

Representasi *array* dua dimensi:



Gambar di atas merepresentasikan *array* yang terdiri dari 3 baris dan 5 kolom, dan jumlah elemennya = $3 \times 5 = 15$ elemen. Karena terdiri dari baris (*row*) dan kolom (*column*), maka *array* dua dimensi sering juga disebut **matrix**.

Mengakses data *array* dua dimensi:

Seperti *array* satu dimensi, *array* dua dimensi juga diakses melalui indeksnya. Contoh: *A*[1,2], menunjuk pada posisi nilai *array* pada baris 1, kolom 2.

Untuk menyimpan nilai dalam *array* dua dimensi, dapat dilakukan dengan cara sebagai berikut:

```

A[0,0] ← 2    /*simpan 2 pada array A baris 0,kolom 0*/
A[0,1] ← 4    /*simpan 3 pada array A baris 0,kolom 1*/
A[1,2] ← 8    /*simpan 5 pada array A baris 1,kolom 2*/
A[2,2] ← A[0,0] + A[1,2] /*tambahkan nilai pada array A baris
                                0,kolom 0 dengan nilai pada array A
                                baris 1,kolom 2 dan simpan hasilnya
                                pada array A baris 2,kolom 2 */
  
```

Sehingga gambaran array A menjadi seperti berikut ini:

Ada 5 kolom

	0	1	2	3	4
0	2	4			
1			8		
2			10		

Ada 3 baris (0-2)

$A[0,0]$

$A[1,2]$

$A[2,4]$

Contoh *pseudocode* untuk menyimpan dan menampilkan nilai pada *array* dua dimensi.

1. ALGORITMA

2. /* Menyiapkan dan memasukkan nilai dalam array dua dimensi

3. I.S : array dalam keadaan kosong

4. *F.S: menampilkan nilai yang disimpan dalam array dengan menggunakan struktur*
5. *pengulangan */*

6. KAMUS DATA

7. A : array[0..2,0..4] of integer

```
8.  i, j, k : integer
```

9. BEGIN

```
10. k=0;
```

```
11. /*memasukkan nilai dalam array*/
```

```
12. for (i=0; i<=3; i++)
```

```
13.   for (j=0; j<=4; j++)
```

14. $A[i, j] = k + 2;$

```
15.      k=k+2;
```

```
16.     endfor
```

```
17. endfor
```

18. /*menampilkan kembali nilai pada array*/

```
19. for (i=0; i<=3; i++)
```

```
20.   for (j=0; j<=4; j++)
```

```
21.      output('A[',i,j,'] = ',A[i,j])
```

```
22.     endfor
```

```
23. endfor
```

24 .END .

Pseudocode di atas menggambarkan proses menyimpan array dua dimensi, dimana nilai yang dimasukkan merupakan penambahan dengan 2.

Dalam Bahasa C

```
1. /*Menyiapkan dan memasukkan nilai dalam array dua dimensi
2.  I.S : array dalam keadaan kosong
3.  F.S : menampilkan nilai yang disimpan dalam array dengan menggunakan struktur
4.      pengulangan */
5. #include <stdio.h>
6. #include <conio.h>
7. void main()
8. {
9.     int A[3][5]; /*deklarasi array dua dimensi*/
10.    int i,j,k;
11.    k=0;
12.    /*memasukkan data dalam array dua dimensi*/
13.    for(i=0;i<=2;i++)
14.    { for(j=0;j<=4;j++)
15.        { A[i][j] = k + 2;
16.          k+=2; } /* endfor loop j */
17.    } /* endfor loop i */
18.    /*menampilkan kembali nilai array dua dimensi*/
19.    for(i=0;i<=2;i++)
20.    { for(j=0;j<=4;j++)
21.        { printf("A[%i,%i] = %i\n",i,j,A[i][j]);
22.          } /*endfor loop j*/
23.    } /*endfor loop i*/
24.    printf("tekan Enter...");
25.    getch(); /* menahan tampilan pada layar*/
26. }
```

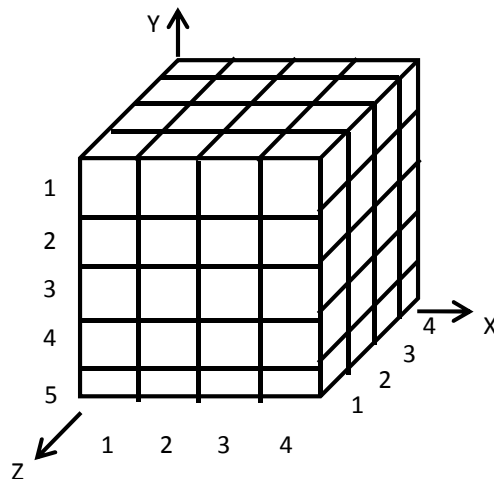
Output yang dihasilkan:


```
C:\D:\KULIAH\LATIHAN C\array2.exe
A[0,0] = 2
A[0,1] = 4
A[0,2] = 6
A[0,3] = 8
A[0,4] = 10
A[1,0] = 12
A[1,1] = 14
A[1,2] = 16
A[1,3] = 18
A[1,4] = 20
A[2,0] = 22
A[2,1] = 24
A[2,2] = 26
A[2,3] = 28
A[2,4] = 30
tekan Enter...
```

6.1.3 Array Multi-Dimensi

Dalam menggambarkan array multidimensi, hanya terbatas hingga dimensi ke-3, yakni dengan menggunakan bangun ruang, namun dalam kenyataannya, tipe data array ini dapat dibentuk menjadi lebih dari tiga dimensi atau menjadi n-dimensi.

Representasi array 3 (tiga) dimensi



Penulisan notasi algoritma untuk mendeklarasikan tipe data array multidimensi cukup dengan memodifikasi deklarasi

array satu dimensi, yakni dengan menambahkan tanda koma “,” pada bagian definisi banyaknya elemen *array* dan menambahkan ukuran elemen yang diinginkan.

Untuk mendeklarasikan variabel dengan tipe data *array* *n*-dimensi pada notasi algoritma, digunakan pola sebagai berikut:

```
...  
KAMUS DATA  
  Nama_variabel: array [a..b,t..u,x..y,...] of tipe_data  
...
```

6.2 Tipe Data Bentukkan

Dalam membuat program, kadangkala akan dihadapkan dengan struktur data yang tidak sederhana dan apabila hanya ditangani dengan tipe data dasar saja, maka pembuat program akan kesulitan merumuskan komposisinya.

Sebagai contoh, program yang akan dibuat melibatkan data tentang mahasiswa, maka untuk variabel mahasiswa akan sulit ditentukan tipe datanya karena pada mahasiswa terdapat beberapa elemen yaitu, nama, nomor induk mahasiswa, jenis kelamin, alamat, dan elemen-elemen yang lainnya.

Tantangan berikutnya adalah bagaimana cara menyimpan data-data mahasiswa tersebut jika jumlah mahasiswa lebih dari satu? Tentunya hal ini akan sangat sulit jika harus diselesaikan dengan tipe data dasar saja. Oleh karena itu diperlukan adanya suatu tipe data baru yang digunakan untuk menangani kasus di atas, yaitu dengan menggunakan tipe data bentukkan.

Tipe data bentukkan merupakan suatu tipe data yang dirancang/dibentuk (dan diberi nama) dari beberapa elemen bertipe tertentu yang sudah dikenal. Jadi di dalam tipe data bentukkan akan terdapat elemen dengan tipe data dasar dan dapat juga terdapat tipe data bentukkan lain yang telah didefinisikan sebelumnya.

Tujuan digunakannya tipe data bentukkan adalah supaya perancang program mendapatkan suatu tipe data dimana

seluruh komponennya secara keseluruhan memiliki makna semantik dan di dalamnya terdapat keterkaitan antar komponen. Pada data mahasiswa telah dijabarkan beberapa elemen yang ada maka, dengan menggunakan tipe data bentukan ini, perancang program dapat mendefinisikannya ke dalam program.

Implementasi tipe data bentukan dalam bahasa pemrograman sangat bervariasi tergantung dari struktur bahasa pemrograman itu sendiri. Dalam notasi algoritmik, sebuah tipe bentukan (tipe komposisi) dapat disusun sebagai berikut :

```
type nama_type < elemen1 : type_data1,
                  elemen2 : type_data2,
                  ...>
```

Contoh pada data mahasiswa dapat dijabarkan elemen-elemennya sebagai berikut:

1. Nim bertipe longint
2. Nama bertipe string
3. Umur bertipe word

Jika dituliskan dalam notasi algoritmik maka, akan menjadi:

```
type Mahasiswa : <nim : integer,
                  nama : string,
                  umur : integer>
```

Operasi dalam menggunakan tipe data bentukan memiliki perilaku yang sama dengan operasi pada tipe data dasar, hanya perbedaannya adalah pada cara mengaksesnya. Tipe data bentukan memiliki beberapa variabel/elemen yang berada di dalamnya, oleh karena itu cara mengaksesnya menggunakan tanda dot/titik '.'

Contoh:

Jika akan mendefinisikan nama variabel Mhs dengan tipe data Mahasiswa maka pendeklarasiannya adalah:

```
Kamus data
Mhs : Mahasiswa
```

Jika akan mengisi elemen nim pada variabel Mhs maka:

```
Mhs.nim ← 30107001
```

Atau dengan perintah masukan sebagai berikut:

```
input (Mhs.nim)
```

Jika akan menampilkan isi dari elemen nim pada variabel Mhs maka:

```
output (Mhs.nim)
```

Berikut adalah contoh penggunaan tipe data bentukan secara lengkap:

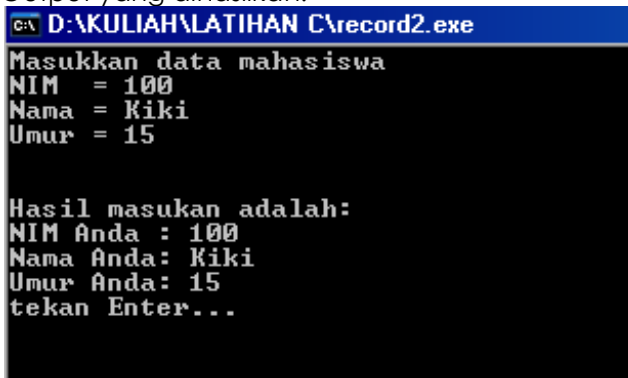
```
1. ALGORITMA
2. /*contoh algoritma penggunaan tipe data bentukan sederhana*/
3. KAMUS DATA
4. type Mahasiswa : <nim : integer,
5.                     nama: string,
6.                     umur: integer>
7. /*Menggunakan tipe data Mahasiswa pada variabel Mhs*/
8. Mhs : Mahasiswa
9. Begin
10. /*mengisi elemen-elemen dalam variabel Mhs*/
11. input (Mhs.nim)
12. input (Mhs.nama)
13. input (Mhs.umur)
14. /*menampilkan isi elemen-elemen dalam Mhs*/
15. output (Mhs.nim)
16. output (Mhs.nama)
17. output (Mhs.umur)
18. End.
```

Dalam bahasa C

```
1. #include <stdio.h>
2. #include <conio.h>
3. /*deklarasi record mahasiswa*/
4. struct mahasiswa{ long nim;
5.                   char nama[20];
6.                   short umur; };
7. struct mahasiswa Mhs;
8.
9. main()
10. {
11. printf("Masukkan Data Mahasiswa\n");
12. printf("NIM = "); scanf("%i",&Mhs.nim);
13. printf("Nama = "); scanf("%s",&Mhs.nama);
14. printf("Umur = "); scanf("%i",&Mhs.umur);
15.
```

```
16. /*menampilkan isi elemen-elemen dalam Mhs*/
17. printf("\n\nHasil masukan Anda adalah: \n");
18. printf("Nim Anda = %i\n",Mhs.nim);
19. printf("Nama Anda = %s\n",Mhs.nama);
20. printf("Umur Anda = %i\n",Mhs.umur);
21. printf("tekan Enter...");
22. getch(); /*menahan tampilan pada layar*/
23. }
```

Output yang dihasilkan:



```
C:\ D:\KULIAH\LATIHAN C\record2.exe
Masukkan data mahasiswa
NIM = 100
Nama = Kiki
Umur = 15

Hasil masukan adalah:
NIM Anda : 100
Nama Anda: Kiki
Umur Anda: 15
tekan Enter...
```

6.3 Kombinasi Tipe Bentuk dan Array

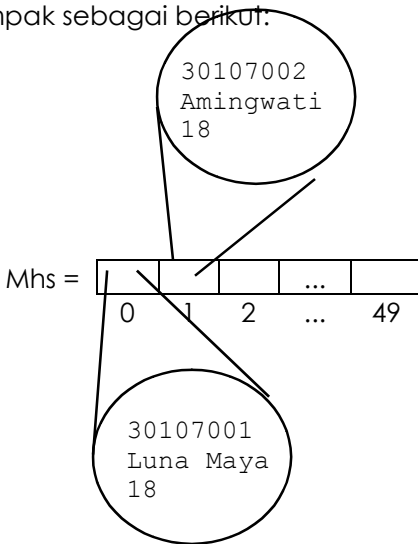
6.3.1 Tipe Data Bentuk di dalam array

Permasalahan yang berikutnya adalah bagaimana caranya memasukkan data mahasiswa dengan jumlah yang banyak? Di sini dapat digunakan array sebagai sarana untuk menyimpan data mahasiswa dalam satu variabel. Cara mendeklarasikannya adalah sebagai berikut:

KAMUS DATA

```
type Mahasiswa : <nim : integer,  
                  nama : string,  
                  umur : integer  
                  >  
Mhs : array [0..49] of Mahasiswa
```

Jika direpresentasikan dalam bentuk gambar maka, akan tampak sebagai berikut:



Untuk mengimplementasikan penggambaran data di atas maka, cara pengisian array `Mhs` adalah sebagai berikut:

```
...  
Mhs[0].nim ← 30107001  
Mhs[0].nama ← "Luna Maya"  
Mhs[0].umur ← 18  
  
Mhs[1].nim ← 30107002  
Mhs[1].nama ← "Amingwati"  
Mhs[1].umur ← 18  
...
```

Atau dengan perintah masukan sebagai berikut

```
input (Mhs[0].nim)
input (Mhs[0].nama)
input (Mhs[0].umur)

input (Mhs[1].nim)
input (Mhs[1].nama)
input (Mhs[1].umur)
.....
```

Dengan demikian data mahasiswa dapat disimpan dalam satu buah variabel bertipe array dengan tipe Mahasiswa.

Contoh pseudocode lengkapnya

```
1. ALGORITMA
2. /*contoh algoritma penggunaan tipe data bentukan dalam array*/
3. KAMUS DATA
4.   Type Mahasiswa : <nim : integer,
5.                       nama : string,
6.                       umur : integer >
7. /*Menggunakan tipe data Mahasiswa pada variabel Mhs*/
8.   Mhs : array [0..2] Mahasiswa
9.   i : integer
10. Begin
11. /*mengisi elemen-elemen dalam variabel Mhs*/
12.   for (i=0; i<=2; i++)
13.     output ("Nim = "); input (Mhs[i].nim)
14.     output ("Nama = "); input (Mhs[i].nama)
15.     output ("Umur = "); input (Mhs[i].umur)
16.   EndFor
17. /*menampilkan isi elemen-elemen dalam Mhs*/
18.   for (i=0; i<=2; i++)
19.     output (Mhs[i].nim)
20.     output (Mhs[i].nama)
21.     output (Mhs[i].umur)
22.   EndFor
23. End.
```

Dalam bahasa C

```
1. /*contoh program penggunaan tipe data bentukan dalam array*/
2. #include <stdio.h>
```

```
3.#include <conio.h>
4./*deklarasi record mahasiswa*/
5.struct mahasiswa { long nim;
6.                    char nama[20];
7.                    short umur; };
8.struct mahasiswa mhs[3];
9. /* main program */
10.main()
11.{
12.  int i;
13.  int a;
14.  printf("Input Data Mahasiswa\n");
15.  a=1;
16.  for(i=0;i<=2;i++)
17.  {
18.    printf("Data ke-%d\n",a);
19.    printf("NIM = "); scanf("%i",&mhs[i].nim);
20.    printf("Nama = "); scanf("%s",&mhs[i].nama);
21.    printf("Umur = "); scanf("%i",&mhs[i].umur);
22.    printf("\n");
23.    a++;
24.  } //endfor
25. /*menampilkan data mahasiswa*/
26.  printf("\nData Yang Telah Di inputkan\n");
27.  for(i=0;i<=2;i++)
28.  {
29.    printf("%i%10s%3i\n",mhs[i].nim,mhs[i].nama,
30.            mhs[i].umur);
31.  } //endfor
32.  getch(); /*menahan tampilan pada layar*/
33.}
```

Output yang dihasilkan:


```
C:\ D:\KULIAH\LATIHAN C\recarr.exe
Input Data Mahasiswa

Data ke-1
NIM = 100
Nama = Dede
Umur = 12

Data ke-2
NIM = 200
Nama = Rina
Umur = 14

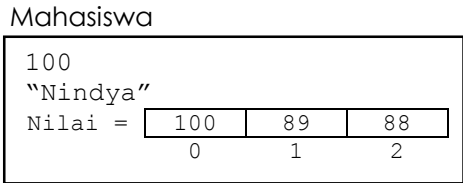
Data ke-3
NIM = 300
Nama = Nuri
Umur = 15

Data Yang Telah Di inputkan
100      Dede 12
200      Rina 14
300      Nuri 15
```

6.3.2 Array di dalam Tipe Data Bentukan

Pada contoh sebelumnya telah ditunjukkan bahwa di dalam sebuah elemen *array* dapat diisi dengan suatu nilai yang memiliki tipe data bentukan. Jika kondisi yang dihadapi oleh perancang program adalah kondisi yang sebaliknya, yaitu memasukkan variabel bertipe *array* menjadi salah satu atau beberapa elemen di dalam variabel yang memiliki tipe data bentukan, bagaimanakah cara mendefinisikan dan mengoperasikan variabel tersebut?

Di dalam tipe data bentukan, satu atau beberapa elemennya diperbolehkan untuk menggunakan tipe data *array*. Salah satu contoh kasusnya adalah bagaimana mendefinisikan data mahasiswa yang mempunyai beberapa nilai. Jika digambarkan akan tampak sebagai berikut:



Dari gambar di atas dapat dilihat bahwa seorang mahasiswa memiliki beberapa nilai, maka cara mendeklarasikan tipe datanya adalah:

```
type Mahasiswa : <nim : integer,
```

```
nama : string,  
nilai: array [0..2] of integer  
>
```

Kamus data

Mhs : Mahasiswa

Dalam hal ini yang memiliki tipe data *array* adalah elemen di dalam tipe data bentukan, oleh karena itu cara mengakses elemen *array* yang terdapat di dalam tipe Mahasiswa adalah sebagai berikut:

```
Mhs.nilai[0] ← 100  
Mhs.nilai[1] ← 89  
Mhs.nilai[2] ← 88
```

Atau dengan perintah masukan sebagai berikut:

```
input (Mhs.nilai[0])  
input (Mhs.nilai[1])  
input (Mhs.nilai[2])
```

Contoh pseudocode lengkapnya

```
1. ALGORITMA
2. /*contoh algoritma penggunaan array dalam tipe data bentukan*/
3. KAMUS DATA
4. type Mahasiswa: <nim : integer,
5.                 nama : string,
6.                 nilai: array[0..2] of integer>
7. /*Menggunakan tipe data Mahasiswa pada variabel Mhs*/
8. Mhs : Mahasiswa
9. i, a : integer
10. Begin
11. /*mengisi elemen-elemen dalam variabel Mhs*/
12. output("Memasukkan nilai dalam array")
13. output("Nim = "); input(Mhs.nim)
14. output("Nama= "); input(Mhs.nama)
15. a=1;
16. for(i=0;i<=2;i++)
17.     output("Nilai ke ",a," = ")
18.     input(Mhs.nilai[i])
19.     a=a+1
20. endfor
21. /*menampilkan isi elemen-elemen dalam Mhs*/
22. output("Nim Anda : ",Mhs.nim," dan Nama
23.       Anda: ",Mhs.nama)
24. output("Nilai Anda adalah:")
25. for(i=0;i<=2;i++)
26.     output(Mhs.nilai[i])
27. endFor
28. End.
```

Dalam bahasa C

```
1. /*contoh program array dalam tipe data bentukan*/
2. #include <stdio.h>
3. #include <conio.h>
4. /*deklarasi record dan variabel*/
5. struct mahasiswa {long nim;
6.                  char nama[20];
7.                  int nilai[3];};
8. struct mahasiswa mhs;
9. int i, a;
10.
11. main()
```

```
12.{
13.  printf("Memasukkan nilai dalam array\n");
14.  printf("NIM = "); scanf("%i",&mhs.nim);
15.  printf("Nama = "); scanf("%s",&mhs.nama);
16.  a=1;
17.  for(i=0;i<=2;i++)
18.  {
19.      printf("Nilai ke-%i = ",a);
20.      scanf("%i",&mhs.nilai[i]);
21.      a++;
22.  }
23.  /*menampilkan kembali data dalam array*/
24.  printf("\nNIM Anda : %i dan Nama Anda :
25.          %s\n",mhs.nim,mhs.nama);
26.  printf("\nNilai Anda adalah:\n");
27.  a=1;
28.  for(i=0;i<=2;i++)
29.  {
30.      printf("Nilai ke-%i : %i\n",a,mhs.nilai[i]);
31.      a++;
32.  }
33.  printf("\nTekan Enter.....");
34.  getch();
35.}
```

Output yang dihasilkan:

```

C:\D:\KULIAH\LATIHAN C\arrrec.exe
Memasukkan nilai dalam array
NIM = 100
Nama = Nindya
Nilai ke-1 = 90
Nilai ke-2 = 89
Nilai ke-3 = 88

NIM Anda : 100 dan Nama Anda : Nindya

Nilai Anda adalah:
Nilai ke-1 : 90
Nilai ke-2 : 89
Nilai ke-3 : 88

Tekan Enter....._

```

6.3.3 Array dari Tipe Bentukan yang Mengandung Array

Pada kasus ini tujuannya adalah mendefinisikan tipe data untuk menyimpan data dengan tipe data bentukan dan di dalam tipe data bentukan tersebut terdapat elemen dengan tipe *array*.

Pada contoh kasus di sub bab 6.3.2 ditunjukkan seorang mahasiswa memiliki nilai lebih dari satu. Pertanyaan berikutnya adalah bagaimana jika jumlah mahasiswanya lebih dari satu?

Caranya adalah dengan membuat variabel bertipe *array* dimana *array* tersebut memiliki tipe data bentukan yang di dalamnya terdapat *array*.

```

type Mahasiswa : <nim : integer,
                  nama : string,
                  nilai: array [0..2] of integer>

```

Kamus data

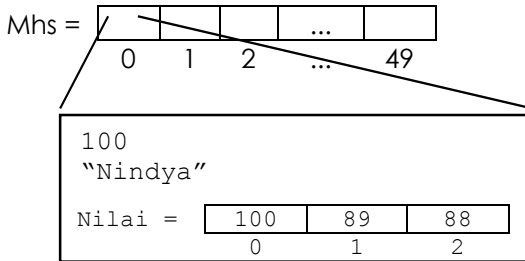
```

Mhs : array [0..49] of Mahasiswa

```

Dapat dilihat bahwa dalam tipe data Mahasiswa terdapat elemen 'nilai' yang memiliki tipe *array*, kemudian pada bagian kamus data didefinisikan bahwa variabel Mhs merupakan tipe *array* dengan tipe data Mahasiswa.

Jika digambarkan akan tampak sebagai berikut:



Cara untuk mengisi satu elemen 'Nilai' dalam variabel `Mhs` adalah sebagai berikut:

```
...
Mhs[0].Nilai[0] ← 100
Mhs[0].Nilai[1] ← 89
Mhs[0].Nilai[2] ← 88
...
```

Demikian pula untuk mengakses suatu nilai pada elemen `Nilai` di variabel `Mhs` adalah:

```
...
{Menampilkan isi elemen nilai ke-0}
output(Mhs[0].Nilai[0])
...
```

Contoh pseudocode lengkapnya

```
1. ALGORITMA
2. /*contoh algoritma penggunaan array dari tipe data bentukan yang mengandung
3. array*/
4. KAMUS DATA
5. type Mahasiswa : <nim : integer,
6. nama : string,
7. nilai: array[0..2] of integer>
8. /*Menggunakan tipe data Mahasiswa pada variabel array Mhs*/
9. Mhs : array of [0..3] of Mahasiswa
10. i,a,j,b : integer
11. Begin
12. /*mengisi elemen-elemen dalam variabel array Mhs*/
13. output("Memasukkan nilai dalam array")
14. a=1
15. for (i=0; i<=3; i++)
16. output("Data mahasiswa ke-",a)
17. output("Nim = "); input(Mhs[i].nim)
18. output("Nama= "); input(Mhs[i].nama)
19. b=1
20. for (j=0; j<=2; j++)
21. output("Nilai ke ",b," = ")
22. input(Mhs[i].nilai[j])
23. b=b+1
24. endfor /*akhir loop j*/
25. a=a+1
26. endfor /*akhir loop i*/
27. /*menampilkan isi elemen-elemen dalam Mhs*/
28. for (i=0; i<=3; i++)
29. output(Mhs[i].nim, Mhs[i].nama)
30. for (j=0; j<=2; j++)
31. output(Mhs[i].nilai[j])
32. endfor
33. endFor
34. End.
```

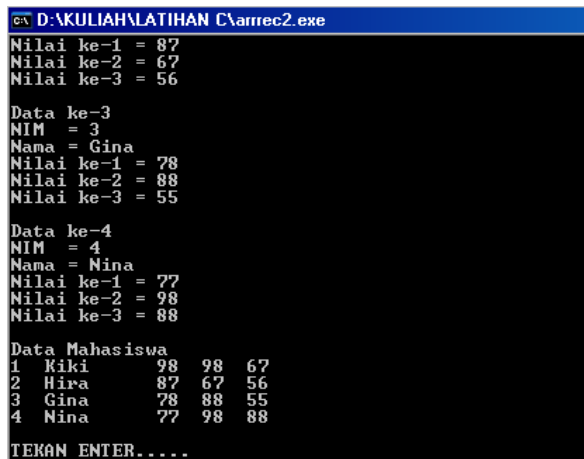
Dalam bahasa C

```
1.  /*contoh algoritma penggunaan array dari tipe data bentukan yang mengandung
2.      array*/
3.  #include <stdio.h>
4.  #include <conio.h>
5.  /*deklarasi record dan variabel*/
6.  struct mahasiswa {long nim;
7.                      char nama[20];
8.                      int nilai[3];};
9.  struct mahasiswa mhs[4];
10. int i,j,a,b;
11. /*main program*/
12. main()
13. {
14.     printf("Memasukkan data pada Array");
15.     a=1;
16.     for(i=0;i<=3;i++)
17.     {
18.         printf("\nData ke-%i\n",a);
19.         printf("NIM = "); scanf("%i", &mhs[i].nim);
20.         printf("Nama = "); scanf("%s", &mhs[i].nama);
21.         b=1;
22.         for(j=0;j<=2;j++)
23.         {
24.             printf("Nilai ke-%i = ",b);
25.             scanf("%i",&mhs[i].nilai[j]);
26.             b++;
27.         } //end loop j
28.         a++;
29.     } //end loop i
30. /*proses menampilkan kembali data pada array*/
31.     printf("\nData Mahasiswa\n");
32.     for(i=0;i<=3;i++)
33.     {
34.         printf("%i %-10s",mhs[i].nim,mhs[i].nama);
35.         for(j=0;j<=2;j++)
36.         {
37.             printf("%i ",mhs[i].nilai[j]);
38.         } //end loop j
39.         printf("\n");
```



```
40.  } //end loop i
41.  printf("\nTEKAN ENTER.....");
42.  getch();
43. }
```

Output yang dihasilkan:



```
D:\KULIAH\LATIHAN C\arrec2.exe
Nilai ke-1 = 87
Nilai ke-2 = 67
Nilai ke-3 = 56

Data ke-3
NIM = 3
Nama = Gina
Nilai ke-1 = 78
Nilai ke-2 = 88
Nilai ke-3 = 55

Data ke-4
NIM = 4
Nama = Nina
Nilai ke-1 = 77
Nilai ke-2 = 98
Nilai ke-3 = 88

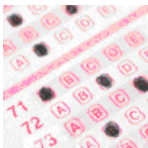
Data Mahasiswa
1 Kiki      98  98  67
2 Hira     87  67  56
3 Gina     78  88  55
4 Nina     77  98  88

TEKAN ENTER.....
```



Rangkuman

1. Tipe data *array* digunakan untuk menampung/menyimpan banyak nilai pada satu variabel.
2. Setiap elemen pada tipe data *array* ditandai dengan indeks.
3. Indeks penanda elemen pada *array* menggunakan tipe data yang memiliki keterurutan.
4. Tipe data *array* memiliki dimensi minimal satu hingga n-dimensi.
5. Pada tipe data *array* satu dimensi memiliki satu indeks, kemudian pada *array* dua dimensi memiliki dua indeks, demikian seterusnya dimana jumlah indeks mengikuti banyaknya dimensi *array* yang dibentuk.
6. Tipe data bentukan adalah tipe data yang dirancang/dibentuk (dan diberi nama) dari beberapa elemen bertipe tertentu.
7. Tipe data bentukan dapat disimpan dalam variabel bertipe *array*.
8. Elemen dalam tipe data bentukan dapat menggunakan variabel bertipe *array*.
9. Tipe data bentukan yang di dalamnya terdapat elemen bertipe *array*, dapat disimpan dalam variabel bertipe *array*.



Pilihan Ganda

Petunjuk: Pilihlah jawaban yang paling tepat!

- Manakah deklarasi array yang benar berikut ini dalam
1. bahasa C:

A. `#include <stdio.h>`
 `void main()`
 `{ A[7] integer; }`

B. `#include <stdio.h>`
 `void main()`
 `{integer A[7]; }`

C. `#include <stdio.h>`
 `void main()`
 `{int A[7]; }`

D. Bukan salah satu di atas

2. Kumpulan elemen-elemen yang teratur dan memiliki tipe data yang sama disebut:

A. Rekursif

C. Array

B. Record

D. File

3. `struct siswa mahasiswa[500]`. Dari array ini yang merupakan NAMA dari suatu ARRAY adalah:

A. `struct`

C. `mahasiswa`

B. `siswa`

D. bukan salah satu di atas

4. Di bawah ini merupakan hal-hal yang harus dikemukakan dalam mendeklarasikan suatu bentuk Array, **kecuali**:

A. nama array

C. record

B. banyak elemen array

D. tipe data

5. Pada array 2 dimensi dengan ordo 4x4, dengan kondisi $A[i,j] = 1$, jika $i \leq j$, dan $A[i,j] = j$, jika $i > j$. Dari pernyataan di atas nilai dari $A[3,2]$ adalah:

A. 1

C. 3

B. 2

D. 4



Latihan

1. KAMUS

```
nilai : array[0..9] of integer
p : integer
```

ALGORITMA

```
for(p=0;p<10;p++)
    nilai[p] ← p + 2
endfor

for(p=0;p<10;p++)
    if (p mod 4) = 0 THEN
        Output("    ")
    else
        Output(nilai[p])
    endif
endfor
```

Tentukan output yang dihasilkan algoritma di atas !!!

2. KAMUS :

```
X, Y : array [0..2,0..2] of integer
i, j : integer
```

ALGORITMA:

```
for(i=0;i<2;i++)
    for(j=2;j>=0;j--)
        X[i,j] ← i + j
        Y[i,j] ← x[i,j]
    endfor
endfor
```

Tentukan berapa harga matriks Y?

- Buat algoritma untuk menghitung nilai total dan nilai rata-rata dari sejumlah nilai yang diinputkan dalam *array*. (Asumsi *array* memiliki 5 elemen)
- Buat algoritma untuk menyimpan matriks segitiga bawah berikut ! (Gunakan skema **WHILE**)

```
1      0      0      0
1      2      0      0
```

1	2	3	0
1	2	3	4

5. Buat algoritma untuk memasukkan sejumlah nilai dalam suatu *array*, kemudian tampilkan nilai terbesar dari nilai-nilai yang diinputkan dalam *array* tersebut. (Asumsikan *array* memiliki 10 elemen data)

7 Pemrograman Modular



Overview

Pemrograman modular memungkinkan perancang program menyederhanakan persoalan didalam program dengan memecah atau membagi persoalan tersebut menjadi sub-sub persoalan yang lebih kecil agar mudah diselesaikan. Secara umum dikenal dua cara yang dapat digunakan untuk memecah persoalan dalam modul-modul, yaitu dengan menggunakan struktur fungsi dan prosedur. Pemahaman tentang perbedaan dan karakteristik masing-masing struktur tersebut perlu diimbangi pula dengan kemampuan mengimplementasikannya dalam program.



Tujuan

1. Memahami konsep pemrograman modular
2. Mengetahui dua cara pemrograman modular: fungsi dan prosedur

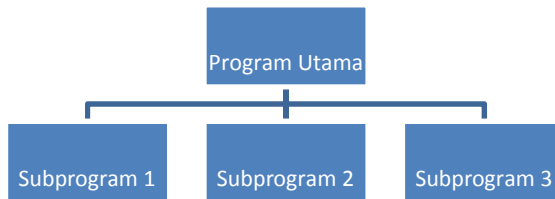
3. Mengetahui cara mengimplementasikan fungsi dan prosedur dalam program
4. Mengetahui dan dapat menerapkan pemanggilan subprogram dari program utama.

7.1 Definisi Pemrograman Modular

Dalam sebuah program, seringkali pemrogram perlu memecah persoalan yang kompleks menjadi beberapa bagian yang lebih mudah diselesaikan. Ide inilah yang mencetuskan struktur pemrograman modular, yaitu memecah persoalan menjadi sub-sub persoalan yang biasa disebut subprogram.

Bayangkan sebuah program yang dibuat untuk menghitung nilai rata-rata dari sekumpulan nilai integer. Dalam prosesnya, program melakukan perhitungan tersebut dalam dua langkah, yaitu menjumlahkan seluruh nilai, kemudian membaginya dengan banyaknya nilai yang tersedia. Dengan demikian program tersebut dapat dipecah menjadi dua subprogram, yaitu subprogram penjumlahan dan subprogram pembagian.

Selain itu, pemrograman modular memungkinkan pemrogram memanggil kembali subprogram yang telah didefinisikannya setiap kali diperlukan dalam program tersebut. Pemrogram tidak perlu berulang kali mendefinisikan sekumpulan instruksi yang diperlukan beberapa kali dalam sebuah program maupun dalam program lainnya. Dengan pemrograman modular, sebuah subprogram dapat dianggap sebagai program kecil dengan sebuah tujuan spesifik yang umumnya berisi operasi sederhana dan apabila terdapat kesalahan dapat dilokalisir pada subprogram itu sendiri. Sub-sub program tersebut kemudian disatukan oleh bagian program utama yang dapat memanggil subprogram tersebut sesuai kebutuhan dalam program.



Gambar 7.1 Ilustrasi pemrograman modular

Dalam pemrograman, dikenal dua tipe subprogram yang biasa digunakan untuk memecah persoalan kompleks menjadi lebih sederhana, yaitu fungsi (*function*) dan prosedur (*procedure*). Kedua tipe subprogram ini dapat digunakan bersamaan maupun salah satunya saja dalam sebuah program. Masing-masing tipe subprogram memiliki karakteristik dan perilaku yang berbeda sehingga penggunaannya dalam program juga berbeda-beda.

Subprogram sebagai bagian dari program utama wajib mendefinisikan kondisi awal (*initial state/I.S.*) sebelum proses dalam subprogram dieksekusi dan juga mendefinisikan kondisi akhir (*final state/F.S.*) yang berupa hasil proses (*output*) atau perubahan nilai dalam variabel tertentu (khusus untuk fungsi saja).

Beberapa fungsi dan prosedur telah terdefinisi dan dapat langsung digunakan oleh pemrogram dalam sebuah program dengan mendefinisikan variabel-variabel yang diperlukan. Selain fungsi dan prosedur yang telah terdefinisi tersebut, pemrogram juga dapat membuat sendiri fungsi dan prosedur yang diperlukannya dalam sebuah program.

Dalam membuat sebuah subprogram, pemrogram dapat menyimpannya dalam salah satu dari dua lokasi berikut ini:

- dalam file yang sama dengan program utama: dapat dilakukan jika subprogram sedikit dan berukuran kecil sehingga relatif mudah dikelola dalam sebuah file
- dalam file yang terpisah: biasanya dilakukan jika subprogram sudah terlalu banyak sehingga sulit dikelola,

atau jika pemrogram menginginkan supaya subprogram dapat digunakan di beberapa program utama sekaligus

7.2 Variabel Lokal dan Variabel Global

7.2.1 Variabel Lokal

Dalam mendeklarasikan sebuah fungsi/ prosedur, dapat dideklarasikan pula variabel-variabel yang akan digunakan dalam fungsi/ prosedur tersebut. Variabel semacam ini disebut **variabel lokal** atau **variabel internal**, artinya variabel ini hanya dikenali secara lokal dalam sebuah subprogram (fungsi atau prosedur). Variabel lokal tidak dapat dipanggil, diakses dan diubah oleh prosedur atau fungsi yang lain, bahkan oleh program utama sekalipun karena hanya dapat dikenali oleh prosedur atau fungsi dimana variabel ini didefinisikan.

7.2.2 Variabel Global

Sedangkan variabel yang didefinisikan dalam program utama dan dapat digunakan di program utama maupun sub-sub program lainnya disebut dengan **variabel global**. Nilai dari variabel ini dapat dipanggil, diakses dan diubah oleh prosedur atau fungsi apapun yang terdapat dalam program tersebut.

7.3 Fungsi

Fungsi adalah subprogram yang menerima data masukan, melakukan beberapa perhitungan dari data tersebut, kemudian mengembalikan output berupa sebuah data baru. Dengan kata lain, sebuah fungsi memetakan sebuah nilai (dalam *domain*) menjadi nilai lain (dalam *range*) dengan operasi/proses tertentu. Pendeklarasian fungsi merupakan salah satu cara memecah persoalan ke dalam beberapa sub persoalan yang lebih mudah diselesaikan.

Dalam pembuatan sebuah fungsi, pemrogram harus mendefinisikan:

- nama fungsi
- Tipe data yang dibuat/ dihasilkan oleh fungsi

- Daftar parameter yang menyatakan data yang diperlukan oleh fungsi
 - Satu atau lebih instruksi yang melakukan perhitungan
- Selanjutnya, fungsi yang sudah didefinisikan dapat digunakan dalam program utama maupun dalam fungsi lainnya dengan cara memanggil nama fungsi dan memberikan parameter yang diperlukan oleh fungsi tersebut.

Fungsi bekerja menurut mekanisme pemanggilan-pengembalian (*call-return mechanism*). Tahapan dalam mekanisme tersebut adalah:

1. Fungsi dipanggil dari program utama maupun fungsi lainnya
2. Sekumpulan operasi dalam fungsi dieksekusi
3. Hasil eksekusi dikembalikan ke program utama atau fungsi lain yang memanggilnya.

Struktur umum sebuah fungsi:

```
FUNCTION nama_fungsi (daftar_parameter) → tipe_data
BEGIN
    {instruksi dalam fungsi}
    return
ENDFUNCTION
```

Sedangkan penulisan dalam bahasa pemrograman C++ adalah sebagai berikut :

```
Tipe_data_kembali nama_fungsi(daftar_parameter)
{
    /* instruksi dalam fungsi */
    return value;
}
```

Dengan catatan bahwa daftar parameter boleh kosong (tidak ada parameter yang perlu diinputkan pada saat pemanggilan fungsi). Jika daftar parameter (disebut juga parameter formal) tidak kosong maka parameter tersebut harus berupa nama parameter dan tipe datanya. Dalam sebuah fungsi, instruksi terakhir harus merupakan perintah untuk mengirimkan nilai/ data keluaran dari fungsi ke program utama

(bagian program yang memanggilnya). Nilai yang dikembalikan oleh sebuah fungsi dapat bertipe skalar, record, array, maupun set.

Jika kita melihat struktur penulisan fungsi, strukturnya hampir atau bahkan sama dengan program utama. Pada dasarnya, pemrograman dengan menggunakan C++ adalah pemrograman dengan struktur fungsi, dimana setiap kode yang dituliskan harus dalam bentuk fungsi, tak terkecuali program utama. Program utama merupakan suatu fungsi dengan nama `main()` yang tidak memiliki nilai kembali atau nilai kembalinya adalah kosong atau 0. Oleh karenanya, kita juga dapat menuliskan program utama dengan `void main()` atau dengan `int main()`, dengan *return value*-nya 0.

Saat program pertama dijalankan, kode yang pertama dieksekusi adalah fungsi `main()`. Oleh karenanya, setiap program minimal harus memiliki satu fungsi yaitu `main()`, dimana isi dari fungsi ini adalah inti dari program.

Parameter dalam sebuah fungsi merupakan antarmuka (penghubung) antara fungsi dengan kode pemanggilnya. Fungsi menerima satu atau beberapa nilai melalui parameter-parameter yang telah didefinisikan. Setiap kali fungsi dipanggil, kode pemanggil harus menyediakan parameter yang dibutuhkan oleh fungsi. Beberapa karakteristik parameter dalam fungsi:

- Parameter hanya muncul di dalam fungsi yang mendefinisikannya dan tidak dapat diakses di luar fungsi tersebut.
- Parameter menyimpan nilai hingga fungsi dieksekusi.
- Parameter diinisialisasi setiap kali fungsi dipanggil oleh program utama maupun fungsi lainnya.

Setelah didefinisikan, fungsi dapat dipanggil dari program utama maupun dari fungsi lainnya dengan perintah:

`<nama_variabel> ← <nama_fungsi>(<daftar_parameter>)`

Contoh fungsi:

```
1. FUNCTION luas_segiempat (P:integer, L:integer) →  
   integer  
2. { IS : Dua buah bilangan Bulat  
3.   FS : Hasil Kali dua bilangan }  
4. KAMUS DATA  
5.     hasil : integer  
6.     P,L : integer;  
7. BEGIN  
8.     hasil ← P * L  
9.     return (hasil)  
10. ENDFUNCTION
```

Pada contoh di atas, fungsi `luas_segiempat` memerlukan dua parameter inputan, yaitu `P` dan `L` yang keduanya bertipe `integer`. Kemudian, fungsi ini akan menghitung `hasil` dengan perkalian nilai `P` dan `L`. Fungsi akan mengembalikan nilai variabel `hasil` ke kode pemanggil.

Berikut ini adalah contoh algoritma program utama/ fungsi lain (selanjutnya disebut kode pemanggil) yang memanggil fungsi `luas_segiempat`:

```
1. Kamus data  
2.     panjang, lebar, luas: integer  
3. BEGIN  
4.     { meminta inputan 2 nilai}  
5.     Input (panjang, lebar)  
6.     {pemanggilan fungsi luas_segiempat}  
7.     luas ← luas_segiempat(panjang, lebar)  
8.     Output (luas) {menampilkan luas}  
9. END
```

Pada potongan program di atas didefinisikan tiga variabel, yaitu `panjang`, `lebar`, dan `luas`. Pemanggilan fungsi `luas_segiempat` dilakukan dalam tiga tahap sesuai mekanisme `call-return`, yaitu:

1. Kode pemanggil memberikan (*passing*) dua parameter yang sudah didefinisikan sebagai variabel dalam program tersebut, yaitu `panjang` dan `lebar`.
2. Fungsi `luas_segiempat` dijalankan dengan menerima dua parameter dari kode pemanggil. Secara berurutan,

variabel panjang dikenali sebagai parameter *P* dan variabel lebar dikenali sebagai parameter *L*. Fungsi melakukan perkalian dan menyimpan hasilnya dalam variabel *hasil* yang merupakan parameter output dari fungsi.

3. Fungsi *luas_segiempat* mengembalikan nilai dalam variabel *hasil* ke kode pemanggil. Kode pemanggil akan menyimpannya ke dalam variabel *luas* yang telah didefinisikan, kemudian menampilkannya ke pemakai.

Perhatikan pula bahwa variabel *P*, *L* dan *hasil* hanya didefinisikan dan digunakan di fungsi *luas_segiempat* dan tidak dapat digunakan dalam program utama maupun fungsi lainnya (variabel lokal).

Berikut ini adalah contoh lain dari definisi fungsi dan pemanggilannya:

```
1. {===Program Utama===}
2. Kamus data           {deklarasi variabel global}
3.   nilai1, nilai2 : integer
4.   rataa : real
5. BEGIN
6.   input (nilai1, nilai2) {input 2 bilangan}
7.   {pemanggilan fungsi1}
8.   rataa ← hitung_rataan(nilai1, nilai2)
9.   output (rataa) {menampilkan nilai rata-rata}
10. END

11. {===Fungsi 1===}
12. FUNCTION           hitung_rataan(var1:      real, var2:
    real) → real
13. {I.S.: var1 dan var2 berisi bilangan
14. F.S.: mengembalikan nilai rataa bertipe real}
15. Kamus data           {deklarasi variabel lokal}
16.   jumlah : integer
17.   rata : real
18. BEGIN
19.   {pemanggilan fungsi2}
20.   jumlah ← hitung_jumlah(var1, var2)
21.   rata ← jumlah/2
22.   return (rata) {kembali ke program utama}
```

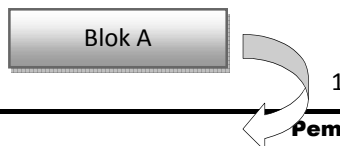
```
23. ENDFUNCTION

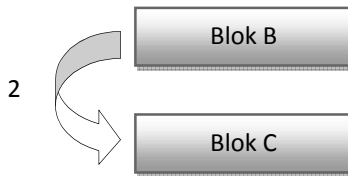
24. {===Fungsi 2===}
25. FUNCTION      hitung_jumlah(pertama, kedua      :
      integer)→integer
26. {I.S.: pertama dan kedua berisi bilangan
27. F.S.: mengembalikan hasil penjumlahan}
28. Kamus data      {deklarasi variabel lokal}
29.      hasil : integer
30. BEGIN
31.      hasil ← pertama + kedua
32.      return (hasil)      {kembali ke fungsi1}
33. ENDFUNCTION
```

Pada contoh di atas, program bekerja sebagai berikut:

1. Program utama menerima inputan dua bilangan integer
2. Program memanggil fungsi `hitung_rataan` dengan memberikan dua bilangan inputan sebagai parameter
3. Fungsi `hitung_rataan` menghitung variabel jumlah dengan memanggil fungsi `hitung_jumlah` dan memberikan parameter `var1` dan `var2`
4. Fungsi `hitung_jumlah` dieksekusi dengan menjumlahkan 2 parameter input yang diberikan dan mengembalikan hasil ke `hitung_rataan`
5. Fungsi `hitung_rataan` menyimpan nilai hasil, menghitung rata dengan (`jumlah/2`), lalu mengembalikan rata ke program utama
6. Program utama menyimpannya dalam variabel `rataan`, kemudian menampilkannya kepada pemakai.

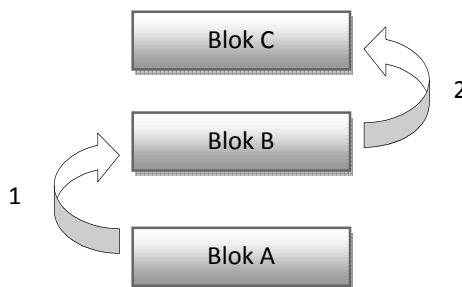
Sebagai ilustrasi algoritma di atas, program utama dimisalkan sebagai blok A, fungsi 1 sebagai blok B dan fungsi 2 sebagai blok C. Jika blok-blok tersebut disusun sesuai dengan penulisannya, maka akan tampak sebagai berikut:





Arah panah menggambarkan alur pemanggilan fungsi, urutan pemanggilan ditandai dengan menggunakan angka. Panah dengan angka 1 menggambarkan blok A (program utama) memanggil Blok B (fungsi 1), kemudian panah dengan angka 2 menggambarkan Blok B memanggil Blok C (fungsi 2).

Pada saat implementasi menggunakan bahasa Pascal, penulisan deklarasi function harus diletakkan sebelum blok pemanggilnya, dengan kata lain blok yang akan dipanggil harus diletakkan di atas blok yang akan memanggil. Ilustrasinya adalah sebagai berikut:



Dari ilustrasi tersebut, maka penerapan (implementasi) algoritma di atas, jika ditulis dalam bahasa C++ akan menjadi sebagai berikut :

```
1. #include <stdio.h>
2.
3. //deklarasi variabel global
4. int nilai1, nilai2;
5. float rataa;
```

```
6.
7. {===Program Utama===}
8. void main () { // program Utama
9.     printf("Masukan Nilai pertama : ");
10.    scanf("%i",&nilai1);
11.    printf("Masukan Nilai kedua   : ");
12.    scanf("%i",&nilai2);
13.
14.    {pemanggil fungsi 1}
15.    rataan = hitung_rataan(nilai1,nilai2);
16.    //menampilkan nilai rata-rata
17.    printf("Rata-rata dari %i dan %i adalah
    %f",nilai1,nilai2,rataan);
18. }
19.
20. {===Fungsi 1===}
21. float hitung_rataan(int var1,int var2) {
22.     // deklarasi variabel lokal
23.     int jumlah;
24.     float rata;
25.
26.     // panggil fungsi 2
27.     jumlah = hitung_jumlah(var1,var2);
28.     rata = jumlah / 2;
29.     return rata; // nilai Kembali Fungsi 1
30. }
31.
32. {===Fungsi 2===}
33. int hitung_jumlah(int pertama,int kedua){
34.     // deklarasi variabel lokal
35.     int hasil;
36.
37.     hasil = pertama + kedua;
38.     return hasil; // nilai kembali Fungsi 2
39. }
```

Jika program tersebut dijalankan dan user menginputkan nilai 12 dan 18, maka akan mengeluarkan hasil :

```
Masukan Nilai pertama : 12
Masukan Nilai kedua   : 18
Rata-rata dari 12 dan 18 adalah 15
```


7.4 Prosedur

Cara lain memecah persoalan pemrograman ke dalam sub-sub persoalan pemrograman adalah dengan mendeklarasikan prosedur. Prosedur adalah sederetan instruksi yang diberi nama, dan melakukan tujuan tertentu. Seperti halnya pada fungsi, prosedur bekerja dengan mekanisme pemanggilan-pengembalian (*call-return mechanism*), yaitu dengan urutan langkah:

1. Prosedur dipanggil oleh kode pemanggil (program utama maupun prosedur lainnya)
2. Sekumpulan operasi yang disimpan dalam prosedur dieksekusi
3. Kontrol dikembalikan ke kode pemanggil

Struktur umum deklarasi prosedur adalah sebagai berikut:

```
PROCEDURE <nama_prosedur> (input <daftar parameter
input>, output <daftar parameter output>)
{I.S.: [kondisi awal]
 F.S.: [kondisi akhir/hasil yang diharapkan]}
BEGIN
    {sekumpulan instruksi dalam prosedur}
ENDPROCEDURE
```

Sedangkan dalam pemrograman bahasa C++, penulisan prosedur seperti berikut :

```
1. void    nama_procedure    (<daftar_parameter_input>
    <,daftar_parameter_output>)
2. {
3.     /* instruksi */
4. }
```

Dengan catatan bahwa nama prosedur dan nama parameternya harus disebutkan dalam blok kode pemanggil. Berbeda dengan fungsi, daftar parameter pada procedure terbagi menjadi dua yaitu parameter input dan parameter output. Daftar parameter boleh kosong (tidak ada parameter input maupun output). Jika parameter tidak kosong (minimal ada satu parameter) maka harus dituliskan nama parameter beserta tipe datanya.

Prosedur tanpa parameter memanfaatkan nilai dari variabel yang terdefinisi dalam kode program utama/prosedur

lain yang memanggilnya. Prosedur tanpa parameter ini hanya dapat dieksekusi jika nilai dari variabel yang diperlukan dalam prosedur sudah didefinisikan dalam kode program utama/prosedur lain yang memanggilnya.

Prosedur dengan parameter dibuat untuk mengeksekusi sekumpulan instruksi dengan parameter yang berbeda-beda. Nama parameter yang dituliskan pada definisi / spesifikasi prosedur disebut dengan **parameter formal**. Sedangkan parameter yang dituliskan pada pemanggilan prosedur disebut **parameter aktual**.

Parameter formal adalah nama-nama variabel yang dipakai dalam mendefinisikan prosedur, dan membuat prosedur tersebut dapat dieksekusi dengan variabel yang berbeda saat pemanggilan. Terdapat tiga tipe parameter formal:

- parameter input, yaitu parameter yang diperlukan prosedur sebagai masukan untuk melakukan aksi yang efektif
- parameter output, yaitu parameter yang akan menyimpan nilai yang dihasilkan oleh prosedur
- parameter input/output, yaitu parameter yang diperlukan prosedur sebagai masukan untuk melakukan aksi tertentu, yang pada akhir prosedur akan diisi dengan nilai baru sebagai hasil eksekusi prosedur

Parameter aktual adalah variabel / konstanta yang dipakai ketika prosedur dipanggil oleh program utama / prosedur lain. Parameter aktual dapat berupa variabel / konstanta, tapi parameter output harus berupa variabel karena akan menyimpan hasil eksekusi prosedur.

Struktur pemanggilan prosedur dari program utama maupun prosedur lain adalah hanya dengan menuliskan nama procedurennya kemudian diikuti daftar parameternya sebagai berikut:

nama_prosedur

Sama halnya dengan fungsi, prosedur dapat terhubung dengan program utama maupun prosedur lain melalui pertukaran parameter. Bedanya, deklarasi prosedur harus

menyertakan nama dan tipe data dari seluruh parameter input dan outputnya.

Contoh algoritma penghitung luas segi empat dalam subbab fungsi dapat diubah menjadi sebagai berikut (dengan prosedur):

```
1. VAR
2.     panjang, lebar, luas: integer
3. BEGIN
4.     Input (panjang, lebar) {meminta inputan 2
        nilai}
5.     {pemanggilan fungsi luas_segiempat}
6.     call luas_segiempat(panjang, lebar, luas)
7.     Output (luas) {menampilkan luas}
8. END

9. PROCEDURE luas_segiempat (input: p, l: integer,
    output: hasil: integer)
10. {I.S.: panjang dan lebar berisi bilangan
11. F.S.: luas berisi nilai luas segiempat}
12. BEGIN
13.     hasil  $\leftarrow$  p * l
14. ENDPROCEDURE
```

Contoh algoritma penghitung nilai rata-rata dari dua buah bilangan inputan dengan pendeklarasian prosedur:

```
1. {===Program Utama===}
2. VAR {deklarasi variabel global}
3.     nilai1, nilai2, rata : real
4. BEGIN
5.     input (nilai1, nilai2) {input 2 bilangan}
6.     {pemanggilan prosedur1}
7.     call hitung_rataan(nilai1, nilai2)
8.     output(rataan) {menampilkan nilai rata-rata}
9. END

10. {===Prosedur1 hitung_rataan===}
11. PROCEDURE hitung_rataan(input: var1, var2:
    integer, output: rata: real)
12. VAR
13.     jml : integer
14. BEGIN
15.     Call hitung_jumlah(var1, var2) {panggil
        fungsi2}
16.     rata  $\leftarrow$  jml / 2
```

```
17. ENDPROCEDURE

18. {==Prosedur2 hitung_jumlah==}
19. PROCEDURE hitung_jumlah(input: pertama, kedua:
    integer, output: jumlah: integer)
20. BEGIN
21.     jumlah  $\leftarrow$  pertama + kedua
22. ENDPROCEDURE
```

Pola pemanggilan *procedure* juga mengikuti aturan yang diterapkan pada *function*. Dari algoritma di atas, dapat dituliskan ke dalam bahasa pemrograman C++ sebagai berikut:

```
1.  #include <stdio.h>
2.
3.  //Variabel Global
4.  float nilai1, nilai2, rataan;
5.
6.  {===Program Utama===}
7.  void main () { // program Utama
8.      // Inisialisasi dua buah nilai
9.      nilai1 = 8;
10.     nilai2 = 16;
11.
12.     /* pemanggilan prosedur1 */
13.     hitung_rataan(nilai1, nilai2, rataan);
14.     /* menampilkan nilai rata-rata */
15.     cout << rataan;
16. }
17.
18. {==Prosedur2 hitung_jumlah==}
19. void hitung_jumlah(int pertama, int kedua, int&
    jumlah)
20. {
21.     jumlah = pertama + kedua;
22. }
23.
24. {===Prosedur1 hitung_rataan===}
25. void hitung_rataan(int var1, int var2, int& rata)
26. {
27.     int jml;
28.     // panggil procedure 2
29.     hitung_jumlah(var1, var2, jml);
30.     rata = jml / 2;
31. }
```

Perbedaan utama yang terlihat antara fungsi dan prosedur adalah bahwa prosedur tidak perlu mengembalikan sebuah nilai, sedangkan fungsi harus selalu mengembalikan nilai (ditunjukkan dengan perintah `return ...` di akhir blok fungsi). Selain itu, kode pemanggil fungsi perlu mendefinisikan sebuah variabel untuk menyimpan nilai yang dikembalikan oleh fungsi, sedangkan pada prosedur tidak demikian. Pengisian variabel dilakukan oleh prosedur sehingga kode pemanggil tidak perlu lagi mempersiapkan sebuah variabel penyimpan hasil eksekusi prosedur.

Pada procedure pertama dan kedua terdapat lambang “&” setelah penulisan tipe data pada parameter procedure. Hal itu maksudnya adalah variabel tersebut merupakan parameter input/output, dengan kata lain akan terjadi pemetaan dua arah antara variabel pada parameter procedure dengan variabel pada parameter pemanggilan procedure.

7.5 Fungsi dan Prosedur yang telah terdefinisi

Selain dapat membuat sendiri fungsi atau prosedur yang diperlukan dalam sebuah program, bahasa pemrograman juga sudah menyediakan beberapa fungsi dan prosedur yang sudah terdefinisi dan dapat langsung digunakan / dipanggil dalam program. Penggunaan fungsi maupun prosedur yang telah terdefinisi tersebut dapat mempermudah perancang program menyelesaikan sub persoalan tertentu.

Beberapa fungsi yang telah terdefinisi, antara lain:

FUNGSI	CONTOH
- Fungsi <i>ceil</i> (untuk membulatkan keatas nilai pecahan).	<code>var-int ← ceil(ekspresi float)</code>
- Fungsi <i>min/ max</i> (menentukan nilai minimal atau maksimal)	<code>var-int ← min(3,5)</code> <code>var-int ← max(3,5)</code>

dari dua bilangan)	
- Fungsi <i>random</i> (mendapatkan nilai secara acak dari rentang tertentu)	$\text{var-int} \leftarrow \text{random}(10)$
- Fungsi <i>sin</i> (memperoleh nilai sinus dari suatu bilangan)	$\text{Var-float} \leftarrow \sin(\text{int})$

Beberapa prosedur yang telah terdefinisi, antara lain:

PROSEDUR	CONTOH
- Prosedur <i>assert</i> (mengecek error pada ekspresi boolean)	<i>assert</i> (eks-boolean [,string])
- Prosedur <i>arg</i> (mengembalikan argumen ke-i dari program dan meyimpannya dalam sebuah string)	<i>arg</i> (eks-integer, var- string)
- Prosedur <i>date</i> (menampilkan tanggal sekarang)	<i>date</i> (var-string)
- Fungsi <i>time</i> (menampilkan jam sekarang dengan format jj:mm:dd)	<i>time</i> (var-string)

7.6 Fungsi Rekursif

Fungsi dapat dipanggil oleh program utama dan juga oleh fungsi yang lain, selain kedua metode pemanggilan tersebut, fungsi dapat juga dipanggil oleh dirinya sendiri. Yang dimaksud disini adalah pemanggilan fungsi itu didalam fungsi itu sendiri, bukan pada fungsi yang lain. Fungsi yang melakukan pemanggilan terhadap dirinya sendiri disebut dengan fungsi rekursif.

Berikut adalah contoh program dalam program yang menerapkan metode rekursif untuk menghitung nilai faktorial dari suatu bilangan.

Struktur umum deklarasi prosedur adalah sebagai berikut:

```
1. // Program Hitung_Faktorial
2. #include <stdio.h>
3.
4. int angka;
5. int hasil;
6.
7. {===Program Utama===}
8. void main () { // program Utama
9.     printf("Masukan Angka Batas Atas Faktorial :");
10.    scanf("%i",&angka);
11.    hasil = faktorial(angka);
12.    printf("Faktorial Dari %i adalah %i.", angka,
        hasil);
13.}
14.
15.int faktorial(int bil)
16.{
17.    if bil = 0 then
18.        return 1
19.    else
20.        return (bil * faktorial(bil - 1));
21.}
```

Perhatikan pada baris ke-20 kode diatas yang diberi tanda arsis. Kode pemanggilan fungsi faktorial tersebut berada pada bloknya sendiri degan parameter yang diubah. Hal yang harus diperhatikan dalam pembuatan rekursif adalah fungsi tersebut harus berhenti dimana didalam fungsi tersebut harus ada pengkondisian bahwa fungsi harus berhenti. Pada contoh diatas, code untuk menghentikan fungsi tersebut ada pada baris ke 17 dan 18, dimana apabila inputan parameternya 0, maka akan menghasilkan 1. Namun jika tidak, proses akan memanggil fungsi yaitu dirinya sendiri.

Sebagai ilustrasi program diatas, mari kita coba memanggil fungsi faktorial dengan Faktorial(5), maka proses yang akan dilakukan adalah :

```
1. 5 * Faktorial(4)
2. 5 * 4 * Faktorial(3)
3. 5 * 4 * 3 * Faktorial(2)
4. 5 * 4 * 3 * 2 * Faktorial(1)
5. 5 * 4 * 3 * 2 * 1 * Faktorial(0)
6. 5 * 4 * 3 * 2 * 1 * 1
7. 5 * 4 * 3 * 2 * 1
8. 5 * 4 * 3 * 2
9. 5 * 4 * 6
10. 5 * 24
11. 120
Hasil Akhir : 120
```

7.7 Unit

Fungsi dan prosedur dapat disimpan dalam file yang terpisah dari program utamanya. File-file semacam ini disebut sebagai unit. Fungsi, prosedur, dan variabel dapat disimpan dalam sebuah unit tanpa blok program utama. Pada saat dilakukan kompilasi, program utama akan mendeklarasikan unit-unit yang digunakan.

```
// nama File external.c
#include <stdio.h>

#define vector_size = 100;

void pTambah(int a,int b,int& c);
int fTambah(int a,int b);
```

```
#include <"external.c">
// Program Utama
void pTambah(int a,int b,int& c)
{
    ...
}
```


Gambar 7.4. Contoh deklarasi unit dalam program utama



Rangkuman

1. Pemrograman modular adalah upaya memecah program yang kompleks ke dalam sub-subprogram yang lebih kecil untuk menyederhanakan penyelesaian persoalan.
2. Setelah didefinisikan, subprogram dapat dipanggil berkali-kali dengan parameter yang berbeda-beda.
3. Dua tipe subprogram yang biasa digunakan adalah fungsi (*function*) dan prosedur (*procedure*).
4. Sebuah subprogram dapat disimpan dalam file yang sama dengan program utama, ataupun dalam file terpisah.
5. Dalam penggunaan fungsi dan prosedur, dikenal 2 tipe variabel, yaitu variabel lokal dan variabel global.
6. Sebuah fungsi memetakan sebuah nilai (dalam *domain*) menjadi nilai lain (dalam *range*) dengan operasi / proses tertentu.
7. Dalam penggunaan prosedur, dikenal dua tipe parameter, yaitu parameter formal dan parameter aktual.
8. Fungsi dan prosedur bekerja menurut mekanisme pemanggilan-pengembalian (*call-return mechanism*).
9. Terdapat beberapa fungsi dan prosedur yang telah terdefinisi dan dapat langsung digunakan dalam program.
10. Fungsi, prosedur dan variabel yang disimpan dalam sebuah file yang terpisah dari program utamanya membentuk unit yang dapat digunakan oleh program utama.



Kuis Benar Salah

1. Prosedur adalah fungsi yang tidak mengembalikan nilai apapun ke kode pemanggilnya.
2. Pemrograman modular adalah pemrograman dengan modul-modul.
3. Sebuah subprogram dapat dipanggil oleh program utama maupun subprogram lainnya.
4. Subprogram harus disimpan dalam file yang sama dengan program utama atau subprogram lain yang akan memanggilnya.
5. Variabel global adalah variabel yang dideklarasikan digunakan dalam program utama dan tidak dapat dikenali dalam subprogram manapun.
6. Salah satu keuntungan pemrograman modular adalah bahwa jika terjadi kesalahan pada sebuah modul, perbaikan dapat dilokalisasi.
7. Dalam mendefinisikan sebuah fungsi, pemrogram harus menentukan nama fungsi, tipe data keluaran dari fungsi, dan daftar parameter yang diperlukan oleh fungsi.
8. Tahapan terakhir dalam mekanisme pemanggilan-pengembalian fungsi adalah pengembalian hasil eksekusi ke program utama atau fungsi lain yang memanggilnya.
9. Fungsi selalu memiliki daftar parameter yang perlu diinputkan saat pemanggilan.
10. Program / subprogram pemanggil fungsi / prosedur memegang kendali terhadap data yang dihasilkan oleh fungsi / prosedur yang dipanggilnya.



Pilihan Ganda

1. Fungsi adalah
 - A. Salah satu jenis subprogram
 - B. Sekumpulan instruksi yang diberi nama
 - C. Blok program yang menerima data masukan, tanpa harus mengembalikan nilai apapun ke kode pemanggil
 - D. Deklarasi dari prosedur
 - E. Bagian program yang dapat memanggil program utama berulang kali sesuai kebutuhan
2. Perhatikan potongan program berikut ini:

```
void main() {  
    int st;  
    st = sesuatu(4, 5);  
    printf("%i",st);  
}  
  
int sesuatu(int P, int T)  
{  
    int hasil;  
    hasil ← (P * T) / 2  
    return (hasil)  
}
```

Apakah nama fungsi yang ada di dalam potongan program di atas?

- A. luas_segitiga
 - B. hasil
 - C. integer
 - D. luas
 - E. sesuatu
3. Di antara pilihan berikut ini, manakah yang tidak harus didefinisikan pada saat membuat sebuah subprogram?

- A. Nama subprogram
 - B. Tipe data yang dihasilkan oleh subprogram
 - C. Satu atau lebih instruksi yang melakukan perhitungan
 - D. Program utama/ subprogram lain yang akan menggunakannya
 - E. Daftar parameter yang menyatakan data yang diperlukan oleh subprogram
4. Berikut ini adalah pernyataan-pernyataan yang benar tentang subprogram, kecuali
- F. Subprogram adalah cara yang digunakan pemrogram untuk memecah persoalan menjadi sub-sub persoalan yang lebih sederhana.
 - G. Subprogram dapat dipanggil oleh program utama maupun subprogram lainnya.
 - H. Subprogram adalah fungsi/ prosedur yang ada di dalam file yang sama dengan program utama.
 - I. Subprogram dapat memanggil subprogram lain dalam file yang berbeda.
 - J. Subprogram memungkinkan pemrogram menyelesaikan persoalan secara parsial.
5. Perhatikan potongan algoritma berikut ini:

```
FUNCTION hitung_rataan(var1,var2)
BEGIN
    rata ← (var1+var2)/ 2
    return (rata)
ENDFUNCTION
```

Bagian apakah yang kurang dari fungsi di atas?

- F. deklarasi variabel global
- I. deklarasi parameter output
- G. deklarasi variabel lokal
- J. deklarasi output fungsi
- H. deklarasi parameter input



Latihan

1. Jelaskan perbedaan antara fungsi dan prosedur!

(Untuk soal nomor 2-4) Perhatikan potongan algoritma berikut ini

```
VAR
    sNama, sNamaOrtu : string
    iUmur, iUmurOrtu, iSelisih : integer
    bValid : boolean
BEGIN
    Input (sNama, iUmur)
    Input (sNamaOrtu, iUmurOrtu)
    iSelisih ← hitung_selisih(iUmurOrtu, iUmur)
    IF iSelisih >= 15 THEN
        Output ('Valid. Silakan masuk!')
    ELSE
        Output ('Tidak valid. Dilarang masuk!')
    ENDIF
END

FUNCTION hitung_selisih (a,b)
BEGIN
    beda ← a-b
    return (beda)
END FUNCTION
```

2. Apakah output program di atas jika pengguna menginputkan $iUmur \leftarrow 10$ dan $iUmurOrtu \leftarrow 23$?
3. Sebutkan variabel-variabel lokal dan global dalam potongan program di atas!
4. Sebutkan tiga macam parameter dalam subprogram!
Ubahlah algoritma tersebut kedalam bahasa C++!"

8 Mesin Karakter



Overview

Dalam menyelesaikan masalah yang relatif kompleks perlu dilakukan penggambaran atau analogi dengan suatu hal yang bersifat nyata. Tujuan penggambaran ini agar mempermudah menentukan kebutuhan proses atau mekanisme tertentu, sehingga pada akhirnya seluruh mekanisme dapat teridentifikasi untuk menyelesaikan masalah.



Tujuan

1. Memahami perlunya analogi dalam mendefinisikan suatu permasalahan
2. Mempelajari mekanisme mesin abstrak
3. Mempelajari mekanisme mesin pencacah (mesin integer)

4. Mempelajari mekanisme mesin karakter
5. Mempelajari penggunaan mesin integer dan mesin karakter dalam menyelesaikan suatu kasus

Mesin

Mesin merupakan mekanisme yang terdefinisi dan mengerti serta mampu untuk mengeksekusi aksi-aksi primitif yang terdefinisi untuk mesin tersebut.

Aksi-aksi primitif disini dapat dianalogikan sebagai fungsi dan prosedur yang terdefinisi dari segi input, proses dan output. Sebagai contoh sederhana adalah mesin kendaraan bermotor. Pada mesin tersebut terdapat karburator, busi, blok mesin, dan katup mesin sebagai jalur masuk bahan bakar atau sebagai jalur keluar gas sisa pembakaran.

Jika karburator bekerja dengan baik maka, karburator dapat dengan lancar mengalirkan bahan bakar menuju ke blok mesin. Jika katup mesin dapat bekerja dengan baik maka, jalur masuk bahan bakar dan jalur keluar gas sisa pembakaran dapat dialirkan dengan lancar. Jika busi dapat bekerja dengan baik maka, busi dapat menghasilkan percikan api yang berguna untuk meletupkan bahan bakar yang ada dalam blok mesin, sehingga gerigi-gerigi yang ada dalam blok mesin akan bekerja.

Oleh karena itu, sebuah mesin dapat dikatakan sebagai mekanisme karena memiliki primitif-primitif yang terdefinisi dengan baik fungsi-fungsinya sehingga dapat menghasilkan sesuatu.

Mesin Abstrak

Mesin abstrak adalah mesin yang dianggap ada dan diasumsikan dapat melakukan mekanisme yang didefinisikan untuk mesin tersebut. Mesin abstrak ini digunakan untuk memodelkan suatu mekanisme tertentu supaya dapat lebih mudah dipelajari. Dengan menggunakan mesin abstrak, perancang program dapat dengan mudah membuat suatu mekanisme dari mesin yang akan dibuat.

Dalam pemrograman, mesin abstrak ini diciptakan pada tahap konseptual dan belum menjadi sesuatu yang riil.

Perancang program seringkali harus mendefinisikan mesin-mesin abstrak untuk memecahkan masalah secara bertahap, sehingga pada akhirnya nanti seluruh primitif serta mekanisme dapat terdefinisi dengan baik. Setelah mesin abstrak ini terdefinisi dengan baik (termasuk fungsi dan prosedur yang terlibat), barulah kode-kode program dituliskan untuk menerapkan sesuatu yang abstrak menjadi produk yang nyata (riil) yaitu yang disebut sebagai mesin riil.

Dalam buku ini akan dibahas mengenai suatu bentuk mesin yang umum digunakan, yakni mesin integer dan mesin karakter.

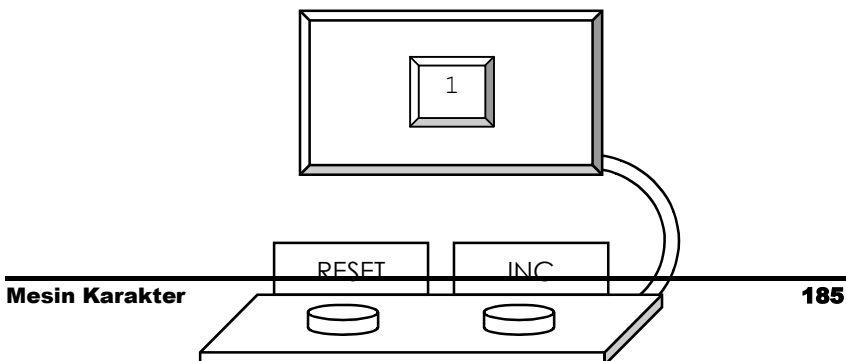
Mesin Integer (Pencacah)

Mesin integer merupakan sebuah mesin yang terdiri dari :

1. Satu buah tombol RESET
2. Satu buah tombol INC (singkatan dari increment yang berarti menambahkan)
3. Sebuah jendela yang menunjukkan angka integer yang sedang diingat, oleh karena itu angka yang sedang muncul di jendela disebut sebagai *Current Integer (CI)*.

Masing-masing tombol merupakan analogi dari *procedure*, jika tombol RESET ditekan artinya *procedure* RESET dipanggil, demikian juga untuk tombol INC.

Tombol RESET berguna untuk mengembalikan CI pada angka nol. Sedangkan tombol INC berguna untuk menambahkan angka 1 pada CI, jika CI bernilai 0 maka setelah tombol INC ditekan maka, CI akan bernilai satu. Nilai pada CI akan terus bertambah jika tombol INC selalu ditekan.



Gambar 8.1 Mesin Pencacah

Berikut adalah prosedur untuk kedua tombol tersebut:

```
PROCEDURE RESET
{mengembalikan isi dari pencacah CI menjadi nol
 CI merupakan variabel global
 I.S.: sembarang
 F.S.: CI = 0}
BEGIN
    CI ← 0
ENDPROCEDURE
```

```
PROCEDURE INC
{menambahkan isi variabel CI dengan satu
 I.S.: CI = nilai saat ini
 F.S.: CI = CI + 1}
BEGIN
    CI ← CI + 1
ENDPROCEDURE
```

Mesin Karakter

Mesin karakter merupakan mesin abstrak yang di dalamnya terdiri dari beberapa komponen, yaitu:

1. Pita yang berisi deretan karakter dan diakhiri dengan tanda titik '.'. Pita yang hanya berisi tanda titik '.' akan disebut sebagai pita kosong. Pita dalam mesin ini sebagai penggambaran dari *array* dengan tipe data *char* (karakter). Dalam lingkungan pemrograman dengan bahasa Pascal, tipe data 'string' dapat diperlakukan sama dengan *array* dengan tipe data karakter.
2. Dua buah tombol yakni tombol START dan ADV (singkatan dari kata *advance* yang berarti memajukan)

3. Sebuah lampu *EOP (End Of Position)*. Lampu ini akan menyala jika tanda titik '.' sudah terbaca, artinya sudah berada pada posisi terakhir. Penggambaran lampu menyala adalah kondisi dimana status pada saat itu bernilai TRUE dan lampu padam adalah FALSE.
4. Sebuah "jendela" yang ukurannya sebesar satu karakter saja. Hanya karakter yang sedang berada di jendela disebut sebagai *Current Character (CC)* dan dapat dibaca sedangkan karakter lain tidak terlihat.



Gambar 8.2 Mesin Karakter

Mesin karakter ini memiliki skenario mekanisme atau cara kerja mesin saat digunakan untuk mengamati suatu pita karakter. Skenarionya adalah sebagai berikut :

1. Kondisi (*state*) awal mesin adalah pada jendela akan tampak CC. Jika CC merupakan titik, maka EOP akan menyala yang berarti isi pita adalah kosong. Jika CC tidak sama dengan titik, maka lampu EOP akan padam
2. Tombol ADV digunakan untuk memajukan pita karakter sebanyak satu karakter. Pada kondisi ini tetap akan diperiksa apakah CC merupakan tanda titik atau bukan.

Berikut adalah procedure untuk tombol START dan ADV:

PROCEDURE START

```
{Mesin siap digunakan. Pita disiapkan untuk dibaca.
Karakter pertama pada pita berada di jendela
I.S.: sembarang
F.S.: CC = karakter pertama pada pita}
```

```
Jika CC='.' maka EOP padam (FALSE)
Jika CC≠'.' maka EOP menyala (TRUE)}
```

PROCEDURE ADV

{Pita dimajukan satu karakter

I.S.: CC = karakter pada jendela, CC≠'.'

F.S.: CC adalah karakter berikutnya, kemungkinan
CC='.' Jika CC='.' Maka EOP menyala (TRUE)}

Catatan:

Jika EOP menyala, maka mesin tidak dapat digunakan lagi (sudah sampai akhir pita).

Penggunaan Mesin

Menghitung Jumlah Karakter

Kedua mesin ini (mesin integer dan mesin karakter) dapat digunakan secara bersama-sama untuk menyelesaikan beberapa kasus. Sebagai contoh jika terdapat sebuah pita karakter yang berisi data sebagai berikut:

"P"	"O"	"L"	"I"	"T"	"E"	"K"	"N"	"J"	"K"	"."
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

Jika jumlah karakter dalam pita tersebut akan dihitung, maka dalam pengoperasian kedua mesin ini adalah:

Tombol yang ditekan	CC	CI
START, RESET	"P"	0
ADV, INC	"O"	1
ADV, INC	"L"	2
ADV, INC	"I"	3
ADV, INC	"T"	4
ADV, INC	"E"	5
ADV, INC	"K"	6
ADV, INC	"N"	7
ADV, INC	"J"	8
ADV, INC	"K"	9
ADV, INC	"."	10

Atau dapat juga dilakukan dengan cara menekan tombol INC terlebih dahulu:

Tombol yang ditekan	CC	CI
START, RESET	"P"	0
INC, ADV	"O"	1
INC, ADV	"L"	2
INC, ADV	"I"	3
INC, ADV	"T"	4
INC, ADV	"E"	5
INC, ADV	"K"	6
INC, ADV	"N"	7
INC, ADV	"I"	8
INC, ADV	"K"	9
INC, ADV	". "	10

Berikut ini adalah algoritma proses penghitungan huruf:

```

Algoritma Hitung_Huruf
{Menghitung banyaknya huruf dalam pita karakter}
Kamus data
  CI : integer, CC : char, EOP : boolean
BEGIN
  START
  RESET
  WHILE (cc ≠ '. ') DO
    INC
    ADV
  ENDWHILE
  OUTPUT (CI)
END

```

Bahasa C

```

#include <stdio.h>
#include <conio.h>
#include "mesinkar.inc"

/*Menghitung banyaknya huruf dalam pita karakter*/

```

```
void main() {  
    START();  
    RESET();  
    while (!EOP) {  
        INC();  
        ADV();  
    }  
    printf("Banyak huruf dalam pita = %d \n",ci);  
    getche();  
}
```

Catatan:

Jika CC='.', maka EOP menyala, karena itu CC='.' bisa diganti EOP saja dan CC≠'.' dapat diganti not EOP, demikian juga sebaliknya.

Menghitung Jumlah Karakter Tertentu

Kombinasi penggunaan mesin karakter dan mesin integer dapat juga dimanfaatkan untuk menghitung jumlah karakter tertentu, misalnya untuk menghitung jumlah huruf 'A' dalam suatu pita karakter.

Cara menangani kasus ini yaitu dengan mengkombinasikannya dengan operasi if..then..else. Jika CC menunjukkan huruf 'A', maka tombol INC pada mesin integer akan ditekan, jika bukan huruf 'A', maka akan memajukan pita karakter menggunakan tombol ADV.

Berikut adalah algoritma dan program untuk menghitung banyaknya huruf 'A':

Algoritma
Algoritma Hitung_Huruf_A {Menghitung banyaknya huruf A dalam pita karakter} Kamus data CI : integer, CC : char, EOP : boolean BEGIN RESET START WHILE (CC≠".") DO IF CC = "A" THEN


```
        INC
    ENDIF
    ADV
ENDWHILE
OUTPUT(CI)
END
```

Bahasa C

```
#include <conio.h>
#include <stdio.h>
#include "mesinkar.inc"

/*Menghitung banyaknya huruf A dlam pita karakter*/
void main() {
    START();
    RESET();
    while (cc!='.') {
        if (cc=='A') {
            INC();
        }
        ADV();
    }
    printf("Banyak huruf A = %d \n",ci);
    getche();
}
```

Menghitung Jumlah Kata

Sekilas terbayang menghitung jumlah kata adalah sesuatu yang amat sulit, padahal jumlah kata tidak lain adalah jumlah spasi ditambah 1.

Contoh isi pita:

HARI INI HARI SENIN	3 spasi = 4 kata
HARI INI HUJAN TURUN LAGI	4 spasi = 5 kata

Namun persoalan akan makin rumit bila dimungkinkan antar kata lebih dari 1 spasi. Untuk ini dapat dibuat prosedur mengabaikan spasi yang berlebihan.

Contoh isi pita karakter

" HARI INI HUJAN TURUN LAGI ."

Algoritma berikut prosedurnya sbb:

Algoritma

```
PROCEDURE IGNORE_BLANK
{mengabaikan/membuang spasi berlebihan}
BEGIN
    while (cc==' ') do
        ADV
    endwhile
ENDPROCEDURE

Algoritma Hitung_Kata
{Menghitung banyaknya kata dalam pita karakter}
Kamus data
    CI : integer, CC : char, EOP : boolean
BEGIN
    RESET
    START
    WHILE (not EOP) DO
        IF CC=' ' THEN
            INC
            IGNORE_BLANK
        ELSE
            ADV
        ENDIF
    ENDWHILE
    INC
    OUTPUT(CI)
END
```

Bahasa C

```
void IGNORE_BLANK() {
    while (cc==' ') {
        ADV();
    }
}

int main() {
    START();
```

```
RESET();
IGNORE_BLANK();
while (cc!='.') {
    if (cc==' ') {
        INC();
        IGNORE_BLANK();
    }
    else {
        ADV();
    }
}
/* banyak kata = banyak spasi + 1 */
INC();
printf("Banyak kata = %d \n",ci);
getche();
}
```

Studi Kasus Mesin Karakter : Palindrom

Palindrom adalah istilah yang digunakan untuk kata atau kalimat yang apabila dibaca dari depan ke belakang atau sebaliknya, memiliki arti yang sama.

Contoh palindrom:

KATAK

KASUR RUSAK

KASUR NABABAN RUSAK

Untuk memeriksa apakah kata yang dimasukkan merupakan palindrom maka, dapat dibuat sebuah function yang memiliki tipe data boolean. Function ini akan mengembalikan nilai TRUE jika kata termasuk palindrom, dan akan mengembalikan nilai FALSE untuk kondisi sebaliknya.

Algoritma

```
FUNCTION IsPalindrom (kt : string) → boolean
{akan mengembalikan nilai TRUE jika k adalah palindrom}
Kamus data
    i,j : integer
    temp : string

BEGIN
```

```
{mengisi temporer disingkat temp dengan string kosong}
temp ← ""

{mengisi j dengan lebar kata}
j ← length(kt)
WHILE (j>0) DO
    {operasi konkatenasi}
    temp ← temp + kt[j]
    j ← j - 1
ENDWHILE

{membandingkan isi temporer, dengan kt}
IF temp = kt THEN
    return TRUE
ELSE
    return FALSE
ENDIF
ENDFUNCTION
```

Bahasa C

```
bool IsPalindrom(char kt[]){
    char ss[]={0,0};
    int i,j;
    char temp[30];

    strcpy(temp,"");
    j=strlen(kt)-1;
    while (j>=0) {
        *ss=kt[j];
        strcat(temp,ss);
        j=j-1;
    }
    if (strcmp(temp,kt)==0)
        return true;
    else {
        return false;
    }
}
```

Catatan:

Operasi konkatenasi berfungsi untuk menggabungkan dua data bertipe string. Contoh:

Kata1 dan Kata2 bertipe string.

Bila Kata1 berisi "algo" dan Kata2 berisi "ritma"

Maka operasi Kata1+Kata2 akan menghasilkan kata "algoritma"



Rangkuman

1. Dengan menganalogikan suatu permasalahan maka, dengan mudah dalam mempelajari mekanisme penyelesaian masalah.
2. Mesin abstrak merupakan mesin yang didefinisikan di tingkat konseptual sebelum diimplementasikan menjadi produk yang riil.
3. Mesin integer berguna untuk melakukan pencacahan.
4. Mesin karakter berguna untuk melakukan penyelesaian masalah yang berkaitan dengan string.
5. Dalam lingkungan pemrograman dengan bahasa Pascal, string merupakan tipe data yang ekivalen dengan *array* bertipe data karakter (*array of char*).
6. Mesin integer dapat digunakan secara bersama-sama dengan mesin karakter untuk menyelesaikan masalah.
7. Palindrom adalah istilah untuk kata atau kalimat yang memiliki arti yang sama pada saat kata atau kalimat itu dibaca dari arah yang berlawanan.



Kuis Benar Salah

1. Mesin integer tidak memiliki jendela untuk menampilkan *Current Character*.
2. Fungsi dari tombol RESET pada mesin integer adalah untuk mengembalikan nilai *current integer* ke nilai satu.
3. INC dapat dilakukan untuk menambahkan CI sebanyak satu angka
4. Pita karakter pada mesin karakter harus diakhiri dengan tanda titik "." atau penanda lain sebagai end-of-position
5. Pita karakter merupakan penggambaran dari tipe data string atau *array of karakter*.
6. Jika S adalah variabel dengan tipe data string dan $S \leftarrow \text{"algoritma"}$ maka, output dari $S[9]$ adalah huruf 'A'.
7. Operasi konkatenasi berguna untuk menggabungkan dua buah data bertipe karakter atau string.
8. Perhatikan algoritma berikut:

```
Kamus data
S1, S2, S3 : string
BEGIN
    S1  $\leftarrow$  "selamat"
    S2  $\leftarrow$  "pagi"
    S3  $\leftarrow$  S1 + S2
END
```

Output dari variabel S3 adalah "selamat pagi"

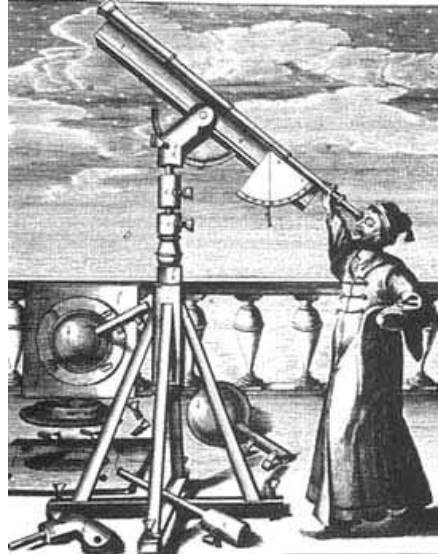
9. "Nama Papa Aman" merupakan kalimat yang palindrom.
10. Jika diketahui anagram adalah kata atau kalimat yang tersusun dari huruf-huruf yang sama, maka pernyataan berikut bernilai benar:
"Kata yang disebut palindrom sudah pasti anagram, sedangkan kata yang anagram belum tentu palindrom"



Latihan

6. Buatlah algoritma untuk menghitung frekuensi huruf "A" yang terdapat pada pita karakter yang berisi :
"ada apa dengan cinta." outputnya: 6/20.
7. Buatlah algoritma untuk menentukan frekuensi huruf hidup (vokal) mana yang paling banyak muncul :
"hari ini hari libur." outputnya: vokal terbanyak "i".
8. Buatlah prosedur untuk melakukan invers kata atau kalimat. Invers adalah kebalikan, artinya fungsi akan menghasilkan kata atau kalimat secara terbalik dari semula.
Contoh:
"POLITEKNIK" setelah diinvers akan menjadi "KINKETILOP".
9. Buatlah fungsi (function) baru, yang berbeda dengan contoh, untuk memeriksa sebuah kata atau kalimat termasuk kategori palindrom atau bukan! (Ketentuan: tidak menggunakan temporer dan operasi konkatenasi)
10. Buatlah algoritma yang menerima masukan berupa dua buah kata, kemudian memeriksa apakah kedua kata tadi termasuk anagram atau tidak (anagram artinya huruf-hurufnya sama tetapi diacak).
Contoh:
"SEBAB" dan "BEBAS" → anagram
"KAPAS" dan "PASAK" → anagram

8. Pencarian



Overview

Pencarian merupakan sebuah algoritma dasar yang sering diperlukan dalam pembuatan program. Berbagai algoritma pencarian telah diciptakan dan dapat digunakan. Pemahaman tentang beberapa algoritma pencarian dasar perlu diketahui, termasuk cara penggunaannya dalam program.



Tujuan

1. Memahami konsep pencarian
2. Mengetahui beberapa algoritma pencarian
3. Menerapkan algoritma pencarian dalam program

8.1 Konsep Pencarian

Pencarian adalah proses menemukan nilai (data) tertentu dari dalam sekumpulan nilai yang bertipe sama (tipe dasar maupun tipe bentukan). Dengan kata lain, algoritma pencarian adalah algoritma yang mengambil input berupa persoalan dan mengembalikan penyelesaian berupa penemuan nilai yang dicari dalam persoalan inputan.

Proses pencarian seringkali diperlukan pada saat program perlu mengubah atau menghapus nilai tertentu (sebelum bisa mengubah atau menghapus, perlu mencari dulu apakah nilai tersebut ada dalam kumpulan nilai tersebut). Kasus lain yang memerlukan algoritma pencarian adalah penyisipan data ke dalam kumpulan data (perlu dimulai dengan pencarian apakah data tersebut telah ada sehingga terhindar dari duplikasi data).

8.2 Pencarian Sekuensial

Pencarian sekuensial (*sequential search*) adalah proses membandingkan setiap elemen larik (array) satu persatu dengan nilai yang dicari secara beruntun, mulai dari elemen pertama sampai elemen yang dicari sudah ditemukan, atau sampai seluruh elemen sudah diperiksa.

Algoritma pencarian sekuensial ini cocok untuk pencarian nilai tertentu pada sekumpulan data terurut maupun tidak. Keunggulan algoritma ini adalah dalam mencari sebuah nilai dari sekumpulan kecil data. Algoritma ini termasuk algoritma yang sederhana dan cepat karena tidak memerlukan proses persiapan data (misalnya: pengurutan).

Algoritma pencarian sekuensial:

```
PROCEDURE SeqSearch(input L:array[1..N] of integer,  
input N:integer,input X:integer, output idx :
```

integer)		
IS : Terdapat array berisi data angka		
FS : Memberikan hasil data ketemu atau tidak ketemu		
KAMUS DATA		
k : integer		
BEGIN		
k \leftarrow 0		
WHILE ((k<N) AND (L[k] !=X))		
k \leftarrow k+1		
ENDWHILE		
IF ((L[k]=X) && (k<N))		
idx \leftarrow k+1		
ELSE		
idx	\leftarrow	-1
ENDIF		
END		

Prosedur di atas memerlukan parameter input berupa:

- L yaitu sebuah larik (array) tempat menyimpan data yang diinginkan,
- N yaitu jumlah elemen larik (atau index terbesar),
- x yaitu nilai data yang ingin dicari

serta sebuah parameter output berupa variabel idx bertipe integer yang akan mengembalikan posisi ditemukannya data yang dicari apabila ketemu dan akan mengembalikan nilai -1 jika data tidak ditemukan.

Prosedur dimulai dengan inisialisasi pencacah iterasi ($k \leftarrow 0$). Kemudian pencarian dilakukan dengan perulangan yang membandingkan setiap elemen larik secara berurutan dari elemen pertama hingga terakhir dengan nilai data yang diinginkan. Perulangan berakhir jika elemen yang dibandingkan bernilai sama dengan data yang dicari (mengembalikan posisi data), atau jika elemen larik telah dibandingkan semua namun data yang dicari tidak ditemukan (mengembalikan nilai -1).

Pada algoritma diatas $idx=k+1$, hal ini hanya untuk mempermudah dalam melihat urutan, biasanya user melihat elemen pertama sebagai indeks ke - 1, tetapi pada array elemen pertama merupakan indeks ke - 0. Maka variabel idx

untuk menyesuaikan dengan user, sedangkan variabel k menyesuaikan dengan array.

Berikut ini adalah contoh pencarian sekuensial:

elemen	13	16	14	21	76	15
index	0	1	2	3	4	5

Dari data diatas, larik L mempunyai 6 elemen. Pencarian akan dimulai dari indeks ke-0 yaitu posisi pertama.

Misalkan elemen yang dicari: $X = 21$. Urutan elemen yang dibandingkan:

```
13 <> 21
16 <> 21
14 <> 21
21 = 21 (ditemukan idx = 4)
```

Indeks array yang dikembalikan: $k = 3$

Misalkan elemen yang dicari: $X = 13$. Urutan elemen yang dibandingkan:

```
13 = 13 (ditemukan idx = 1)
```

Indeks array yang dikembalikan: $k = 0$

Misalkan elemen yang dicari: $X = 17$. Urutan elemen yang dibandingkan:

```
13 <> 17
16 <> 17
14 <> 17
21 <> 17
76 <> 17
15 <> 17 (tidak ditemukan idx = -1)
```

Indeks array yang dikembalikan: $k = 6$, maka akan mengeluarkan output tidak ketemu karena tidak memenuhi syarat $((L[k]=X) \ \&\& \ (k < N))$.

Berikut adalah konversi algoritma pencarian sekuensial dalam bahasa C:

```
1 #include <stdio.h>
```

```
2 void seqSearch(int L[10],int N,int X, int *idx);

3 void main()
4 {
5     int pos;
6     int arr[10]= {6,7,3,8,2,5,4,1,8,10};
7     seqSearch(arr,10,5,&pos);
8     if (pos!=-1)
9         printf("Ketemu di posisi %d",pos);
10    else
11        printf("Tidak Ketemu");
12 }

13 void seqSearch(int L[10],int N,int X, int *idx)
14 {
15     int k;
16     k=0;
17     while ((k<N) && (L[k] != X))
18     {
19         k = k+1;
20     }
21     if ((L[k] == X) && (k<N))
22         *idx=k+1;
23     else
24         *idx=-1;
25 }
```

Pencarian sequential tidak efektif jika digunakan pada data yang banyak atau data yang dicari berada pada posisi terakhir pencarian.

8.3 Pencarian Biner

Pencarian biner adalah proses mencari data dengan membagi data atas dua bagian secara terus menerus sampai elemen yang dicari sudah ditemukan, atau indeks kiri lebih besar dari indeks kanan.

Algoritma ini lebih efisien daripada algoritma pencarian sekuensial, tetapi pencarian ini mempunyai syarat yaitu bahwa kumpulan data yang harus dilakukan pencarian harus sudah terurut terlebih dahulu, baik terurut secara menaik (ascendant)

atau menurun (descendant). Karena data sudah terurut, algoritma dapat menentukan apakah nilai data yang dicari berada sebelum atau sesudah elemen larik yang sedang dibandingkan pada suatu saat. Dengan cara ini, algoritma dapat lebih menghemat waktu pencarian.

Pencarian dalam data terurut bermanfaat misalnya pada penyimpanan data dengan beberapa komponen, program dapat mencari sebuah indeks terurut. Setelah menemukan indeks yang dicari, program dapat membaca data lain yang bersesuaian dengan indeks yang ditemukan tersebut.

Algoritma pencarian biner dengan elemen larik terurut **menaik**:

PROCEDURE BinSearch(input L:array[1..N] of integer, input N:integer,input X:integer, output idx : integer)
IS : Terdapat array berisi data angka terurut menaik FS : Mengembalikan posisi data yang dicari jika ketemu, dan Mengembalikan -1 jika tidak ketemu
KAMUS DATA i,j,k : integer ketemu : boolean
1 BEGIN 2 i←0 3 j←N 4 ketemu ← false 5 WHILE ((!ketemu) and (i<=j)) 6 k←(i+j) div 2 7 IF (L[k] = X) 8 ketemu = true 9 ELSE 10 IF (L[k] < X) 11 i←k+1 12 ELSE 13 j←k-1 14 ENDIF 15 ENDIF 16 ENDWHILE 17 IF(ketemu)

```
18 idx ← k+1
19 ELSE
20 idx ← -1
21 ENDIF
22 END
```

Prosedur di atas dapat bekerja pada larik yang telah terurut menaik (*ascending*). Prosedur memerlukan parameter input berupa:

- *L* yaitu sebuah larik (array) tempat menyimpan data yang diinginkan,
- *N* yaitu jumlah elemen larik,
- *x* yaitu nilai data yang ingin dicari

serta sebuah parameter output berupa nilai *idx* (mengembalikan posisi nilai jika nilai ditemukan, dan mengembalikan -1 jika nilai tidak ditemukan).

Prosedur dimulai dengan inisialisasi pencacah ($i \leftarrow 0$) dan menyimpan jumlah elemen larik dalam variabel *j*. Variabel *ketemu* akan diberi nilai *false*, hal ini memberitahukan bahwa data yang dicari belum ditemukan. Perulangan diawali dengan membandingkan elemen tengah (elemen dengan indeks *k*) dengan nilai data yang dicari.

- Jika elemen larik yang dibandingkan bernilai sama dengan data yang dicari, maka variabel boolean *ketemu* bernilai *true* dan prosedur mengembalikan nilai indeks elemen tersebut.
- Jika elemen larik bernilai lebih kecil dari data yang dicari, maka pencarian akan bergeser ke kanan dengan cara mengubah nilai *i* (awal pencacah) dengan indeks di sebelah kanan nilai tengah ($i \leftarrow k+1$).
- Jika elemen larik bernilai lebih besar dari data yang dicari, maka pencarian akan bergeser ke kiri dengan cara mengubah nilai *i* (awal pencacah) dengan indeks di sebelah kiri nilai tengah ($i \leftarrow k-1$).

Selanjutnya, perulangan terus dilakukan sampai *ketemu* bernilai *true* atau nilai $i > j$ (semua elemen larik sudah dibandingkan dengan nilai yang dicari). Jika semua elemen larik

sudah dibandingkan dengan nilai yang dicari tetapi nilai tidak ditemukan, maka nilai ketemu akan tetap bernilai false sehingga akan dikembalikan nilai $\text{index} = -1$ (penanda bahwa nilai yang dicari tidak ditemukan dalam larik).

Contoh penggunaan algoritma biner untuk elemen terurut menaik adalah sebagai berikut:

Elemen	81	76	21	28	16	13	10	7
(terurut)	7	10	13	16	21	28	76	81
Index	0	1	2	3	4	5	6	7

Misalkan nilai yang dicari: $X=16$, jumlah elemen: $N=8$. Dengan pencarian biner, elemen yang digunakan adalah elemen terurut (baris kedua). Urutan penyelesaian persoalan ini adalah sebagai berikut:

1. Prosedur melakukan inisialisasi perulangan (iterasi pertama)

$i=0;$
 $j=8;$
 $k=(0+8) \text{ div } 2 = 4$

2. Prosedur menguji apakah $L[4]=16?$

Tidak. $L[4] = 21$ Maka prosedur perlu memutuskan apakah pencarian dilanjutkan ke kiri/ ke kanan.
 $L[4] < 16?$ Tidak. $21 > 16$. Maka prosedur melakukan pencarian di sebelah kiri, dengan mengubah batas j menjadi $k-1$.

3. Prosedur melakukan inisialisasi perulangan (iterasi kedua)

$i=0;$
 $j=k-1 = 3;$
 $k=(0+3) \text{ div } 2 = 1$

4. Prosedur menguji apakah $L[1]=16?$

Tidak. $L[1] = 10$. Maka prosedur perlu memutuskan apakah pencarian dilanjutkan ke kiri/ ke kanan.

$L[1] < 16$? Ya. $10 < 16$. Maka prosedur melakukan pencarian di sebelah kanan, dengan mengubah batas i menjadi $k+1$.

- 5 Prosedur melakukan inisialisasi perulangan (iterasi ketiga)

```
i=k+1=2;  
j=3;  
k=(2+3) div 2 = 2
```

- 6 Prosedur menguji apakah $L[2]=16$?
Tidak. $L[2] = 13$. Maka prosedur perlu memutuskan apakah pencarian dilanjutkan ke kiri/ ke kanan.
 $L[2] < 16$? Ya. $13 < 16$. Maka prosedur melakukan pencarian di sebelah kanan, dengan mengubah batas i menjadi $k+1$.

- 7 Prosedur melakukan inisialisasi perulangan (iterasi ketiga)

```
i=k+1=3;  
j=3;  
k=(3+3) div 2 = 3
```

- 8 Prosedur menguji apakah $L[3]=16$?
Ya. Maka variabel `ketemu` = true dan prosedur mengembalikan nilai $k=3$.

Maka, dalam empat iterasi prosedur pencarian biner dapat menemukan nilai $X=16$ yang dicari pada larik tersebut.

Algoritma berikut ini pencarian biner dengan elemen larik terurut **menurun**:

```
PROCEDURE BinSearch(input L:array[1..N] of integer,  
input N:integer,input X:integer, output idx :  
integer)
```

IS : Terdapat array berisi data angka terurut menurun

FS : Mengembalikan posisi data yang dicari jika

ketemu, dan Mengembalikan -1 jika tidak ketemu	
KAMUS DATA	
i,j,k : integer	
ketemu : boolean	
1	BEGIN
2	i←0
3	j←N
4	ketemu ← false
5	WHILE ((!ketemu) and (i<=j))
6	k←(i+j) div 2
7	IF (L[k] = X)
8	ketemu = true
9	ELSE
10	IF (L[k] < X)
11	j←k-1
12	ELSE
13	i←k+1
14	ENDIF
15	ENDIF
16	ENDWHILE
17	IF(ketemu)
18	idx←k+1
19	ELSE
20	idx←-1
21	ENDIF
22	END

Prosedur di atas bekerja untuk pencarian data di dalam larik, dimana data terurut menurun (dari besar ke kecil). Jika nilai ditemukan dalam larik, prosedur mengembalikan nilai idx (indeks elemen larik yang nilainya sama dengan nilai yang dicari). Jika semua elemen larik sudah dibandingkan dengan nilai yang dicari namun tidak ditemukan, prosedur akan mengembalikan nilai idx←-1 (penanda bahwa nilai yang dicari tidak ditemukan dalam larik).

Contoh penggunaan algoritma biner untuk elemen terurut menurun adalah sebagai berikut:

Elemen	81	76	21	28	16	13	10	7
--------	----	----	----	----	----	----	----	---

(terurut)	81	76	28	21	16	13	10	7
Index	0	1	2	3	4	5	6	7

Misalkan nilai yang dicari: $X=13$ jumlah elemen: $N=8$. Dengan pencarian biner, elemen yang digunakan adalah elemen terurut (baris kedua). Urutan penyelesaian persoalan ini adalah sebagai berikut:

- 1 Prosedur melakukan inisialisasi perulangan (iterasi pertama)

```
i=0;
j=8;
k=(0+8) div 2 = 4
```

- 2 Prosedur menguji apakah $L[4]=13$?
Tidak. $L[4]=16$ Maka prosedur perlu memutuskan apakah pencarian dilanjutkan ke kiri/ ke kanan.
 $L[4]<13$? Tidak. $16 > 13$. Maka prosedur melakukan pencarian di sebelah kanan, dengan mengubah batas i menjadi $k+1$.

- 3 Prosedur melakukan inisialisasi perulangan (iterasi kedua)

```
i=k+1=5;
j=8;
k=(5+8) div 2 = 6
```

- 4 Prosedur menguji apakah $L[6]=13$?
Tidak. $L[6]=10$. Maka prosedur perlu memutuskan apakah pencarian dilanjutkan ke kiri/ ke kanan.
 $L[6]<13$? Ya. $10 < 13$. Maka prosedur melakukan pencarian di sebelah kiri, dengan mengubah batas j menjadi $k-1$.

- 5 Prosedur melakukan inisialisasi perulangan (iterasi ketiga)

```
i=5;
j=k-1=5;
k=(5+5) div 2 = 5
```

6. Prosedur menguji apakah $L[5]=13$?

Ya. Maka variabel `ketemu = true` dan prosedur mengembalikan nilai `k=5`.

Maka, dalam tiga iterasi prosedur pencarian biner dapat menemukan nilai $X=16$ yang dicari pada larik tersebut.

Berikut adalah konversi dari algoritma pencarian biner dengan larik terurut **menaik**, ke dalam bahasa C :

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
typedef int bool;

void BinSearch(int L[7], int N, int X, int *idx);

void main()
{
    int pos, arr[7] = {0, 11, 23, 31, 43, 45, 65};
    BinSearch(arr, 7, 11, &pos);
    if (pos != -1)
        printf("Ketemu di posisi ke-%d", pos);
    else
        printf("Tidak Ketemu");
}

void BinSearch(int L[7], int N, int X, int *idx)
{
    int i, j, k;
    bool ketemu;
    i = 0;
    j = N;
    ketemu = FALSE;
    while ((!ketemu) && (i <= j))
    {
        k = (i + j) / 2;
```

```
        if (L[k]==X)
            ketemu=TRUE;
        else
        {
            if (L[k]<X)
                i=k+1;
            else
                j=k-1;
        }
    }
    if (ketemu)
        *idx = k+1;
    else
        *idx = -1;
}
```

Berikut adalah konversi dari algoritma pencarian biner dengan larik terurut **menurun**, ke dalam bahasa C :

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
typedef int bool;

void BinSearch(int L[7],int N, int X,int *idx)

void main()
{
    int pos,arr[7] = {45,22,16,10,6,2,0};
    BinSearch(arr,7,0,&pos);
    if(pos!= -1)
        printf("Ketemu di posisi ke-%d",pos);
    else
        printf("Tidak Ketemu");
}

void BinSearch(int L[7],int N, int X,int *idx)
{
    int i,j,k;
    bool ketemu;
```

```
i=0;
j=N;
ketemu = FALSE;
while ((!ketemu) && (i<=j))
{
    k=(i+j)/2;
    if (L[k]==X)
        ketemu=TRUE;
    else
    {
        if (L[k]<X)
            j=k-1;
        else
            i=k+1;
    }
}
if (ketemu)
    *idx = k+1;
else
    *idx = -1;
}
```

8.4 Pencarian Lain

Pencarian sekuensial dan pencarian biner merupakan algoritma pencarian dasar yang termasuk ke dalam kelompok pencarian daftar (*list search*). Terdapat pula beberapa algoritma lain yang termasuk pula dalam kelompok pencarian daftar, antara lain:

- pencarian interpolasi (*interpolation search*): melakukan pencarian lebih baik daripada pencarian biner pada larik berukuran besar dengan distribusi seimbang, tapi waktu pencariannya buruk
- pencarian Grover (*Grover's search*): melakukan pencarian dalam waktu singkat, yang merupakan pengembangan dari pencarian linier biasa pada larik dengan elemen tidak berurut

Selain algoritma pencarian dalam kelompok pencarian daftar, terdapat pula beberapa kelompok algoritma lain. Beberapa di antaranya adalah sebagai berikut:

Kelompok Algoritma	Penjelasan dan Contoh Algoritma
Pencarian tanpa informasi (<i>uninformed search</i>)	Algoritma ini mencari data tanpa ada batasan tipe data persoalan.
Pencarian pohon (<i>tree search</i>)	Algoritma ini mencari data dengan bantuan struktur pohon (eksplisit maupun implisit). <ul style="list-style-type: none"> • <i>breadth-first search</i> (mencari level demi level) • <i>depth-first search</i> (mencari dengan meraih kedalaman pohon terlebih dahulu) • <i>iterative-deepening search</i> • <i>depth-limited search</i> • <i>bidirectional search</i> • <i>uniform-cost search</i>
Pencarian grafik (<i>graph search</i>)	Algoritma ini mencari data dengan algoritma penelusuran grafik, misalnya <ul style="list-style-type: none"> • <i>Dijkstra's algorithm</i> • <i>Kruskal's algorithm</i> • <i>nearest neighbour algorithm</i> • <i>Prim's algorithm</i>
Pencarian dengan informasi (<i>informed search</i>)	Algoritma ini mencari data dengan fungsi heuristik yang spesifik pada persoalan tertentu. <ul style="list-style-type: none"> • <i>best-first search</i> • <i>A*</i>
Jenis lain	<ul style="list-style-type: none"> • Algoritma pencarian string • Algoritma genetik • Algoritma <i>minimax</i>



Rangkuman

1. Algoritma pencarian adalah algoritma yang mengambil input berupa persoalan dan mengembalikan penyelesaian berupa penemuan nilai yang dicari dalam persoalan inputan.
2. Dua contoh algoritma pencarian dasar adalah pencarian sekuensial dan pencarian biner.
3. Pencarian sekuensial (*sequential search*) membandingkan setiap elemen larik (array) satu persatu dengan nilai yang dicari secara beruntun, mulai dari elemen pertama sampai elemen yang dicari sudah ditemukan, atau sampai seluruh elemen sudah diperiksa.
4. Pencarian biner adalah proses mencari data dengan membagi data atas dua bagian secara terus menerus sampai elemen yang dicari sudah ditemukan.
5. Pencarian biner mempunyai prasyarat yaitu data harus terurut baik menaik atau menurun.



Kuis Benar Salah

1. Algoritma pencarian hanya dapat dilakukan pada sekumpulan data yang terurut menaik.
2. Pencarian sekuensial membandingkan setiap elemen larik satu persatu dengan nilai yang dicari secara beruntun, mulai dari elemen pertama sampai elemen yang dicari sudah ditemukan, atau sampai seluruh elemen sudah diperiksa.
3. Pencarian biner melakukan pencarian dengan membagi elemen larik ke dalam dua bagian, sehingga kemungkinan keberhasilan pencarian ini hanya 50%.
4. Pencarian sekuensial mensyaratkan elemen larik diurutkan terlebih dahulu supaya pencarian dapat dilakukan secara beruntun.
5. Pencarian biner lebih efisien daripada pencarian sekuensial, tapi pencarian biner mengharuskan data telah terurut.

(Untuk soal nomor 6-10) Perhatikan larik berikut ini:

elemen	25	38	46	72	12	3	90
(terurut)	3	12	25	38	46	72	90
Index	0	1	2	3	4	5	6

6. Dengan memperhatikan algoritma Pencarian Sekuensial, maka pencarian sekuensial untuk $X=46$ dapat diselesaikan dalam 2 iterasi.
7. Pencarian biner pada elemen terurut untuk $X=12$ dapat diselesaikan dalam 2 iterasi.
8. Pencarian biner pada elemen terurut untuk $X=90$ tidak akan menemukan solusi.
9. Pencarian sekuensial pada elemen terurut untuk $X=25$ dapat diselesaikan dengan jumlah iterasi yang sama dengan pencarian biner pada elemen terurut untuk $X=25$.

10. Iterasi maksimal pada pencarian biner dalam larik tersebut adalah 3 iterasi.



Pilihan Ganda

1. Berikut ini adalah pernyataan-pernyataan yang benar tentang pencarian sekuensial, kecuali
 - K. Dapat digunakan untuk mencari nilai tertentu dalam larik.
 - L. Melakukan pencarian secara berurutan dari elemen pertama hingga elemen terakhir larik.
 - M. Melakukan pencarian dengan mengurutkan isi elemen larik terlebih dahulu.
 - N. Melakukan pencarian dengan lebih cepat jika nilai yang dicari ada di elemen awal larik.
 - O. Melakukan pencarian dengan langsung membandingkan elemen larik dengan nilai yang dicari.
2. Berikut ini adalah pernyataan-pernyataan yang benar tentang pencarian biner, kecuali
 - F. Hanya dapat melakukan pencarian pada data yang terurut.
 - G. Melakukan pencarian dengan membagi elemen data menjadi dua bagian dan membandingkan elemen tengahnya dengan nilai yang dicari.
 - H. Hanya dapat melakukan pencarian pada setengah data awal.
 - I. Kemungkinan lebih efisien daripada pencarian beruntun karena tidak perlu membandingkan seluruh elemen larik secara berurutan.

- J. Disebut juga pencarian bagidua.
3. Pencarian sekuensial adalah
- F. Pencarian yang hanya dapat digunakan pada elemen data yang terurut menaik/ menurun.
- G. Pencarian yang mencari data berurutan dari elemen pertama hingga elemen terakhir dalam larik.
- H. Pencarian yang paling efisien dibandingkan pencarian lainnya.
- I. Pencarian yang membagi elemen larik ke dalam dua bagian pada awal setiap iterasi
- J. Pencarian yang bisa jadi tidak menemukan solusi jika data yang dicari ada dalam elemen larik.

4. Perhatikan larik berikut ini:

elemen	5	8	12	22	37	86	88	97
index	0	1	2	3	4	5	6	7

Misalkan ingin dicari $X=22$, dalam berapa kali iterasi-kah pencarian sekuensial dapat menemukan data yang dicari?

- F. 1 iterasi
- I. 4 iterasi
- G. 2 iterasi
- J. 5 iterasi
- H. 3 iterasi
5. Berdasarkan elemen larik pada soal nomor 4, dalam berapa kali iterasi-kah pencarian biner dapat menemukan data $X=22$?
- A. 1 iterasi
- D. 4 iterasi
- B. 2 iterasi
- E. 5 iterasi
- C. 3 iterasi
6. Pada persoalan pencarian nilai dalam sebuah larik dengan elemen terurut, jika nilai yang dicari tidak ada dalam larik, maka pernyataan-pernyataan berikut ini benar, kecuali... .
- A. Pencarian berurut dapat menemukan bahwa nilai yang dicari tidak ada dalam larik.
- B. Pencarian biner dapat menemukan bahwa nilai yang dicari tidak ada dalam larik.
- C. Pencarian berurut memerlukan jumlah iterasi yang lebih sedikit dibandingkan jumlah iterasi yang

- diperlukan oleh pencarian biner.
- D. Pencarian berurut memerlukan jumlah iterasi yang lebih banyak dibandingkan jumlah iterasi yang diperlukan oleh pencarian biner.
- E. Pilihan A, B, C, D salah
7. Berikut ini adalah contoh persoalan yang memerlukan algoritma pencarian, kecuali
- K. mengubah nilai salah satu elemen tertentu dalam larik
- L. menyisipkan sebuah nilai di sebelah salah satu elemen tertentu dalam larik
- M. menghapus salah satu elemen tertentu dalam larik
- N. mencari data dengan indeks tertentu
- O. menghitung rata2 dari seluruh elemen dari sebuah larik
8. Perhatikan larik berikut ini:
- | | | | | | | | | |
|--------|-----|-----|----|-----|---|---|----|----|
| elemen | 134 | 100 | 25 | 132 | 8 | | 72 | 23 |
| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
- Misalkan ingin dicari elemen kosong, dalam berapa kali iterasi-kah pencarian sekuensial dapat menemukan data yang dicari?
- A. 2 iterasi
- B. 3 iterasi
- C. 4 iterasi
- D. 5 iterasi
- E. 6 iterasi
9. Jika elemen larik pada soal nomor 8 telah diurutkan menurun dan elemen kosong diletakkan di akhir larik (pada index nomor 7), pada iterasi ke berapakah pencarian sekuensial dapat menemukan data yang dicari?
- F. 3 iterasi
- G. 4 iterasi
- H. 6 iterasi
- I. 7 iterasi
- J. 8 iterasi
10. Jika elemen larik pada soal nomor 8 telah diurutkan menurun dan elemen kosong diletakkan di akhir larik (pada index nomor 7), pada iterasi ke berapakah pencarian biner dapat menemukan elemen kosong? (gunakan algoritma pencarian binary untuk data yang terurut menurun)
- F. 3 iterasi
- I. 6 iterasi

G. 4 iterasi
H. 5 iterasi

J. 7 iterasi



Latihan

1. Apakah yang dimaksud dengan pencarian?
2. Buatlah prosedur pencarian sekuensial yang dapat mengembalikan nilai indeks dari elemen larik yang berisi sama dengan nilai yang dicari!
3. Jelaskan perbedaan pencarian sekuensial dan pencarian biner!
4. Buatlah prosedur penyisipan sebuah nilai di sebelah kanan sebuah elemen tertentu dalam larik!

(Untuk soal nomor 5-8) Perhatikan larik berikut ini:

elemen	56	34	32	29	29	25	19	15	3
indeks	0	1	2	3	4	5	6	7	8

5. Ujilah prosedur yang Anda buat pada soal nomor 1 dengan menguraikan langkah-langkah yang dilakukan prosedur tersebut untuk mencari $X=29$ pada larik di atas! Berapakah nilai indeks yang dikembalikan?
6. Ujilah prosedur yang Anda buat pada soal nomor 3 dengan menguraikan langkah-langkah yang dilakukan prosedur tersebut untuk mencari $X=29$ pada larik di atas! Berapakah nilai indeks yang dikembalikan?
7. Ujilah prosedur yang Anda buat pada soal nomor 1 dengan menguraikan langkah-langkah yang dilakukan prosedur tersebut untuk mencari $X=30$ pada larik di atas! Dalam berapa kali iterasi-kah prosedur pencarian sekuensial dapat menemukan bahwa nilai yang dicari tidak ada dalam larik tersebut?
8. Ujilah prosedur yang Anda buat pada soal nomor 1 dengan menguraikan langkah-langkah yang dilakukan prosedur tersebut untuk mencari $X=30$ pada larik di atas! Dalam berapa kali iterasi-kah prosedur pencarian biner dapat

menemukan bahwa nilai yang dicari tidak ada dalam larik tersebut?

9. Jelaskan alasan pencarian biner dinyatakan sebagai algoritma pencarian yang lebih efisien dibandingkan pencarian sekuensial!

Buatlah prosedur untuk menghapus sebuah elemen tertentu di dalam larik!

9 Pengurutan (Sorting)



Overview

Seringkali perancang program perlu mengurutkan sekumpulan data yang dimiliki untuk memudahkan pemrosesan selanjutnya terhadap data tersebut. Pengurutan adalah sebuah algoritma dasar yang sering diperlukan dalam pembuatan program. Berbagai algoritma pengurutan telah diciptakan dan dapat digunakan. Pemahaman tentang beberapa algoritma pengurutan dasar perlu diketahui, termasuk cara penggunaannya dalam program.



Tujuan

1. Memahami konsep pengurutan
2. Mengetahui beberapa algoritma pengurutan

3. Menerapkan algoritma pengurutan dalam program

9.1 Pengertian Sort

Sorting atau pengurutan data adalah proses yang sering harus dilakukan dalam pengolahan data. **Sort** dalam hal ini diartikan mengurutkan data yang berada dalam suatu tempat penyimpanan, dengan urutan tertentu baik urut menaik (*ascending*) dari nilai terkecil sampai dengan nilai terbesar, atau urut menurun (*descending*) dari nilai terbesar sampai dengan nilai terkecil. **Sorting** adalah proses pengurutan.

Terdapat dua macam pengurutan:

- **Pengurutan internal (*internal sort*)**, yaitu pengurutan terhadap sekumpulan data yang disimpan dalam media internal komputer yang dapat diakses setiap elemennya secara langsung. Dapat dikatakan sebagai pengurutan tabel
- **Pengurutan eksternal (*external sort*)**, yaitu pengurutan data yang disimpan dalam memori sekunder, biasanya data bervolume besar sehingga tidak mampu untuk dimuat semuanya dalam memori.

Dalam courseware ini, hanya akan dibahas algoritma pengurutan internal, dengan data berada dalam *array* satu dimensi.

Algoritma pengurutan internal yang utama antara lain:

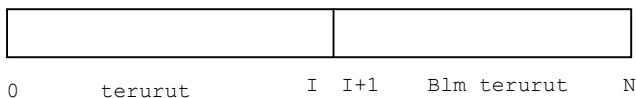
1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Shell Sort
5. Merge Sort
6. Radix Sort
7. Quick Sort
8. Heap Sort

Dalam *courseware* ini hanya akan dibahas tiga metode *sort* yang pertama yang dianggap mudah, yaitu: **Bubble Sort**, **Selection Sort** dan **Insertion Sort**

9.2 Bubble Sort

Bubble sort adalah proses pengurutan sederhana yang bekerja dengan cara berulang kali membandingkan dua elemen data pada suatu saat dan menukar elemen data yang urutannya salah. Ide dari *Bubble sort* adalah gelembung air yang akan "**mengapung**" untuk table yang terurut menaik (*ascending*). Elemen bernilai kecil akan "**diapungkan**" (ke indeks terkecil), artinya diangkat ke "**atas**" (indeks terkecil) melalui pertukaran. Karena algoritma ini melakukan pengurutan dengan cara membandingkan elemen-elemen data satu sama lain, maka *bubble sort* termasuk ke dalam jenis algoritma *comparison-based sorting*.

Proses dalam *Bubble sort* dilakukan sebanyak $N-1$ langkah (*pass*) dengan N adalah ukuran *array*. Pada akhir setiap langkah ke $- I$, *array* $L[0..N]$ akan terdiri atas dua bagian, yaitu bagian yang sudah terurut $L[0..I]$ dan bagian yang belum terurut $L[I+1..N-1]$. Setelah langkah terakhir, diperoleh *array* $L[0..N-1]$ yang terurut menaik.



Untuk mendapatkan urutan yang menaik, algoritmanya dapat ditulis secara global sebagai berikut :

Untuk setiap *pass* ke $- I = 0, 1, \dots, N-2$, lakukan :

Mulai dari elemen $J = N-1, N-2, \dots, I + 1$, lakukan :

- Bandingkan $L[J-1]$ dengan $L[J]$
- Pertukarkan $L[J-1]$ dengan $L[J]$ jika $L[J-1] > L[J]$

Rincian setiap *pass* adalah sebagai berikut :

Pass 1: $I = 0$. Mulai dari elemen $J = N-1, N-2, \dots, 1$, bandingkan $L[J-1]$ dengan $L[J]$. Jika $L[J-1] > L[J]$, pertukarkan

$L[J-1]$ dengan $L[J]$. Pada akhir langkah 1, elemen $L[0]$ berisi harga minimum pertama.

Pass 2: $I = 1$. Mulai dari elemen $J = N-1, N-2, \dots, 2$, bandingkan $L[J-1]$ dengan $L[J]$. Jika $L[J-1] > L[J]$, pertukarkan $L[J-1]$ dengan $L[J]$. Pada akhir langkah 2, elemen $L[1]$ berisi harga minimum kedua dan array $L[0..1]$ terurut, sedangkan $L[2..(N-1)]$ belum terurut.

Pass 3: $I = 2$. Mulai dari elemen $J = N-1, N-2, \dots, 3$, bandingkan $L[J-1]$ dengan $L[J]$. Jika $L[J-1] > L[J]$, pertukarkan $L[J-1]$ dengan $L[J]$. Pada akhir langkah 3, elemen $L[2]$ berisi harga minimum ketiga dan array $L[0..2]$ terurut, sedangkan $L[3..(N-1)]$ belum terurut.

.....

Pass N-1: Mulai dari elemen $J = N-1$, bandingkan $L[J-1]$ dengan $L[J]$. Jika $L[J-1] > L[J]$, pertukarkan $L[J-1]$ dengan $L[J]$.

Pada akhir langkah N-2, elemen $L[N-2]$ berisi nilai minimum ke $[N-2]$ dan array $L[0..N-2]$ terurut menaik (elemen yang tersisa adalah $L[N-1]$, tidak perlu diurut karena hanya satu-satunya).

Misal array L dengan $N = 5$ buah elemen yang belum terurut. Array akan diurutkan secara *ascending* (menaik).

8	9	7	6	1
0	1	2	3	4

Pass 1 :

$I = 0$; $J = N-1 = 4$	8	9	7	1	6
$J = 3$	8	9	1	7	6
$J = 2$	8	1	9	7	6
$J = 1$	<u>1</u>	8	9	7	6

Hasil akhir langkah 1 :

<u>1</u>	8	9	7	6
0	1	2	3	4

Pass 2 :

$I = 1$; $J = N-1 = 4$	<u>1</u>	8	9	6	7
$J = 3$	<u>1</u>	8	6	9	7
$J = 2$	<u>1</u>	<u>6</u>	8	9	7

Hasil akhir langkah 2 :

<u>1</u>	<u>6</u>	8	9	7
----------	----------	---	---	---

0 1 2 3 4

Pass 3 :

$$I = 2 ; J = N-1 = 4$$
$$J = 3$$
$$\frac{1}{1}$$
$$\frac{6}{6}$$
8
7
 7
8

99

Hasil akhir langkah 3 :

<u>1</u>	<u>6</u>	<u>7</u>	8	9
0	1	2	3	4

Pass 4 :

$$I = 3 ; J = N-1 = 4$$

1

6

7

8

9

Hasil akhir langkah 4 :

<u>1</u>	<u>6</u>	<u>7</u>	<u>8</u>	<u>9</u>
0	1	2	3	4

Selesai. Array L sudah terurut !!

Pseudocode prosedur algoritma Bubble Sort secara Ascending

1. //prosedur algoritma Bubble Sort secara Ascending
2. //I.S:array sudah berisi nilai integer yang belum terurut
3. //F.S:nilai-nilai dalam array terurut secara Ascending
4. **procedure** v_Bubble(input/output A:array[0..4] of integer,
input N:integer)

5. KAMUS :

```
6. i, j, temp: integer
```

7. ALGORITMA:

```
8. for (i=0; i<= (N-2) ; i++)
```

```
9.  for (j = (N-1) ; j >= (i+1) ; j--)
```

```
10.      if (A[j-1]>A[j])
```

```
11.      temp ← A[j-1]
```

12. $A[i-1] \leftarrow A[i]$

```
13. A[j] ← temp
```

```
14.      endif
```

```
15.   endfor
```

```
16. endfor
```

17. end procedure

Program lengkap penerapan algoritma *Bubble Sort* dalam bahasa C

```

1.  #include <stdio.h>
2.  #include <conio.h>
3.
4.  void v_Bubble(int A[],int N);
5.  void main()
6.  {   int L[5];
7.      int i,N;
8.      //proses untuk memasukkan data array
9.      printf("Banyak data : ");scanf("%i",&N);
10.     for(i=0;i<N;i++)
11.     { printf("Data ke-%i: ",i+1);
12.       scanf("%i",&L[i]); } //end loop i
13.     //memanggil procedure bubble sort
14.     v_Bubble(L,N);
15.
16.     //proses menampilkan kembali data array
17.     printf("\nData Array Terurut\n");
18.     for(i=0;i<N;i++)
19.     { printf("%3i",L[i]); };
20.     getch();
21. } //end main program
22.
23. void v_Bubble(int A[5],int N)
24. {   int a,b,temp;
25.     //proses sortir dengan bubble sort
26.     for(a=0;a<=(N-2);a++)
27.     {   for(b=(N-1);b>=(a+1);b--)
28.         {   if (A[b-1] > A[b])
29.             {   temp = A[b-1];
30.                 A[b-1]= A[b];
31.                 A[b] = temp; } //endif
32.         } //end loop j
33.     } //end loop i
34. } //end procedure v_Bubble

```

Output yang dihasilkan:

```

c:\D:\KULIAH\LATIHAN C\pioc3bubble.exe
Banyak data : 5
Data ke-1: 8
Data ke-2: 9
Data ke-3: 7
Data ke-4: 6
Data ke-5: 1
Data Array Terurut
1 6 7 8 9

```

9.3 Selection Sort

Algoritma *Selection sort* memilih elemen maksimum/minimum *array*, lalu menempatkan elemen maksimum/minimum itu pada awal atau akhir *array* (tergantung pada urutannya *ascending/descending*). Selanjutnya elemen tersebut tidak disertakan pada proses selanjutnya. Karena setiap kali *selection sort* harus membandingkan elemen-elemen data, algoritma ini termasuk dalam *comparison-based sorting*.

Seperti pada algoritma *Bubble Sort*, proses memilih nilai maksimum /minimum dilakukan pada setiap pass. Jika *array* berukuran N , maka jumlah pass adalah $N-1$.

Terdapat dua pendekatan dalam metode pengurutan dengan *Selection Sort* :

1. **Algoritma pengurutan maksimum** (*maximum selection sort*), yaitu memilih **elemen maksimum sebagai basis pengurutan**.
2. **Algoritma pengurutan minimum** (*minimum selection sort*), yaitu memilih **elemen minimum sebagai basis pengurutan**.

9.3.1 Maximum Selection Sort Ascending

Untuk mendapatkan *array* yang terurut menaik (*ascending*), algoritma *maximum selection sort* dapat ditulis sebagai berikut :

1. Jumlah Pass = $N-1$ (*jumlah pass*)
2. Untuk setiap pass ke – $I = 0, 1, \dots$, jumlah pass lakukan :
 - cari elemen maksimum (*maks*) mulai dari elemen ke – I sampai elemen ke – $(N-1)$
 - pertukarkan *maks* dengan elemen ke – I
 - kurangi N dengan satu

Rincian setiap pass adalah sebagai berikut :

Langkah 1 : Cari elemen maksimum di dalam $L[0..(N-1)]$
Pertukarkan elemen maksimum dengan elemen $L[N-1]$

Langkah 2 : Cari elemen maksimum di dalam $L[0..N-2]$
 Pertukarkan elemen maksimum dengan elemen $L[N-2]$

Langkah 3 : Cari elemen maksimum di dalam $L[0..N-3]$
 Pertukarkan elemen maksimum dengan elemen $L[N-3]$

.....

Langkah N-1 : Tentukan elemen maksimum di dalam $L[0..1]$
 Pertukarkan elemen maksimum dengan elemen $L[0]$

(elemen yang tersisa adalah $L[0]$, tidak perlu diurut karena hanya satu-satunya).

Jadi, pada setiap *pass* pengurutan terdapat proses mencari harga maksimum dan proses pertukaran dua buah elemen *array*.

Misal, terdapat *array* L dengan $N = 5$ buah elemen yang belum terurut. *Array* akan diurutkan secara **Ascending** (menaik), dengan algoritma ***maximum selection sort***.

9	7	12	6	1
0	1	2	3	4

Pass 1 :

- Cari elemen maksimum di dalam *array* $L[0..4]$. Maks= $L[2]=12$
- Tukar Maks dengan $L[4]$, diperoleh :

9	7	1	6	12
0	1	2	3	4

Pass 2 :

(berdasarkan susunan *array* pada Pass 1)

- Cari elemen maksimum di dalam *array* $L[0..3]$. Maks= $L[0]=9$
- Tukar Maks dengan $L[3]$, diperoleh :

6	7	1	9	12
0	1	2	3	4

Pass 3:

(berdasarkan susunan *array* pada Pass 2)

- Cari elemen maksimum di dalam *array* $L[0..2]$. Maks= $L[1]=7$
- Tukar Maks dengan $L[2]$, diperoleh :

6	1	7	9	12
0	1	2	3	4

Pass 4 :

(berdasarkan susunan array pada Pass 3)

- Cari elemen maksimum di dalam array L[0..1]. Maks=L[0]=6
- Tukar Maks dengan L[1], diperoleh :

1	6	7	9	12
0	1	2	3	4

Selesai, array L sudah terurut secara Ascending.

Berikut ini akan diberikan pseudocode procedure Maximum Selection Sort Ascending dan pseudocode procedure untuk tukar tempat.

Pseudocode Algoritma Maximum Selection Sort secara Ascending :

```

1. //prosedur algoritma Maximum Selection Sort secara Ascending
2. //I.S:array sudah berisi nilai integer yang belum terurut
3. //F.S:nilai-nilai dalam array terurut secara Ascending
4. procedure v_SelAsc(input/output A:array[0..4] of integer,
                    input N:integer)
5. KAMUS:
6.   maks,k,j,temp:integer
7. ALGORITMA:
8.   for (k=(N-1); k>=0; k←k-1)
9.     maks←0;
10.    // cari elemen maksimum
11.    for (j=0; j<=k; j←j+1)
12.      if (A[j] > A[maks])
13.        maks←j;
14.      endif
15.    endfor
16.    v_Tukar(A[k],A[maks]) //panggil procedure v_Tukar
17.  endfor
18. end procedure

```

Pseudocode Algoritma Tukar Tempat :

```

1. //prosedur algoritma Tukar Tempat
2. //I.S:nilai-nilai yang dikirimkan sudah terdefinisi sebelumnya
3. //F.S:nilai yang dikirimkan bertukar nilainya
4. procedure v_Tukar(input/output P:integer,
                  input/output M:integer)
5. KAMUS:
6.   temp:integer
7. ALGORITMA:

```

```
8.   temp ← P
9.   P ← M
10.  M ← temp
11. endprocedure
```

Program lengkap penerapan algoritma *Maximum Selection Sort Ascending* dalam bahasa C

```
#include <stdio.h>
#include <conio.h>
void v_SelAsc(int A[],int N);
void v_Tukar(int *P,int *M);

main()
{ int L[5];
  int i,N;
  //input data array
  printf("Banyak Data: ");scanf("%i",&N);
  for(i=0;i<N;i++)
  { printf("Data ke-%i: ",i+1);
    scanf("%i",&L[i]); } //end loop i
  //memanggil procedure v_SelAsc
  v_SelAsc(L,N);
  //menampilkan kembali data array
  printf("\nData Terurut:\n");
  for(i=0;i<N;i++)
  { printf("%3i",L[i]); } //end loop i
  getch();
}

void v_SelAsc(int A[5],int N)
{ int maks,k,j,temp;
  for(k=(N-1);k>=0;k--)
  { maks=0;
    for(j=0;j<=k;j++)
    { if (A[j] > A[maks])
      { maks=j; } //endif
    } //end loop j
    v_Tukar(&A[k],&A[maks]);
  } //end loop k
} //end procedure v_SelAsc

void v_Tukar(int *P,int *M)
{ int temp;
  temp = *P;
  *P = *M;
```



```
*M = temp;
} //end procedure v_Tukar
```

Output yang dihasilkan:

```
D:\KULIAH\LATIHAN C\procMaxAsc.exe
Banyak Data: 5
Data ke-1: 9
Data ke-2: 7
Data ke-3: 12
Data ke-4: 6
Data ke-5: 1
Data Terurut:
1 6 7 9 12_
```

9.3.2 Maximum Selection Sort Descending

Misal, terdapat array L dengan $N = 5$ buah elemen yang belum terurut. Array akan diurutkan secara **Descending** (menurun), dengan **algoritma maximum selection sort**.

9	8	11	7	12
0	1	2	3	4

Pass 1 :

- Cari elemen maksimum di dalam array $L[0..4]$. Maks= $L[4]=12$
- Tukar Maks dengan $L[0]$, diperoleh :

12	8	11	7	9
0	1	2	3	4

Pass 2 :

(berdasarkan susunan array pada Pass 1)

- Cari elemen maksimum di dalam array $L[1..4]$. Maks= $L[2]=11$
- Tukar Maks dengan $L[1]$, diperoleh :

12	11	8	7	9
----	----	---	---	---

0 1 2 3 4

Pass 3 :

(berdasarkan susunan array pada Pass 2)

- Cari elemen maksimum di dalam array L[2..4]. Maks=L[4]=9
- Tukar Maks dengan L[2], diperoleh :

12	11	9	7	8
0	1	2	3	4

Pass 4 :

(berdasarkan susunan array pada Pass 3)

- Cari elemen maksimum di dalam array L[3..4]. Maks=L[4]=8
- Tukar Maks dengan L[3], diperoleh :

12	11	9	8	7
0	1	2	3	4

Selesai array L sudah terurut secara Descending (menurun)

Pseudocode Algoritma Maximum Selection Sort secara Descending :

```

1. //prosedur algoritma Maximum Selection Sort secara Descending
2. //I.S:array sudah berisi nilai integer yang belum terurut
3. //F.S:nilai-nilai dalam array terurut secara Descending
4. procedure v_SelDesc(input/output A:array[0..4]of integer,
                       input N:integer)
5. KAMUS:
6.   k,maks,j,temp:integer
7. ALGORITMA:
8.   for (k=0; k<= (N-2) ; k←k+1)
9.     //cari elemen maksimum
10.    maks←k
11.    for (j= (k+1) ; j<= (N-1) ; j←j+1)
12.      if (A[j] > A[maks])
13.        maks←j
14.      endif
15.    endfor
16.    temp←A[k]
17.    A[k]←A[maks]
18.    A[maks]←temp
19.  endfor

```

Program lengkap penerapan algoritma *Maximum Selection Sort Descending* dalam bahasa C

```
1. #include <stdio.h>
2. #include <conio.h>
3. void v_Tukar(int *P,int *M);
4. void v_SelDesc(int A[5],int N);
5. main()
6. { int L[5];
7.   int i,k,j,maks,temp,N;
8.   printf("Banyak Data: ");scanf("%i",&N);
9.   //input data array
10.  printf("Input Data Array\n");
11.  for(i=0;i<N;i++)
12.  { printf("Data ke-%i = ",i+1);
13.    scanf("%i",&L[i]); } //endloop i
14.    //panggil procedure v_SelDesc
15.    v_SelDesc(L,N);
16.    printf("\nOutput Data Array Terurut:\n");
17.    for(i=0;i<N;i++)
18.    { printf(" %5i",L[i]); } //endloop i
19.
20.    printf("\nTekan Enter...\n");
21.    getch();
22. } //end main program
23.
24. void v_SelDesc(int A[5],int N)
25. { int k,maks,j,temp;
26.   //proses sorting max descending
27.   for(k=0;k<=(N-2);k++)
28.   { //cari elemen maksimum
29.     maks=k;
30.     for(j=(k+1);j<=(N-1);j++)
31.     { if (A[j] > A[maks])
32.       maks=j; } //endfor loop j
33.     v_Tukar(&A[k],&A[maks]);
34.   } //endfor loop k
35. } //end procedure v_SelDesc
36.
37. void v_Tukar(int *P,int *M)
38. { int temp;
39.   temp = *P;
40.   *P = *M;
41.   *M = temp;
42. } //end procedure v_Tukar
```

Output yang dihasilkan:

```

D:\KIRI\AHM ALIHAN\T\proMax\test.exe
Banyak Data: 5
Input Data Array
Data ke-1 = 9
Data ke-2 = 8
Data ke-3 = 11
Data ke-4 = 7
Data ke-5 = 12

Output Data Array Terurut:
    12    11    9    8    7
Tekan Enter...

```

9.3.3 Minimum Selection Sort Ascending

Untuk mendapatkan array yang terurut menaik (*ascending*), algoritma *minimum selection sort* dapat ditulis sebagai berikut :

1. Jumlah Pass = $N-1$ (*jumlah pass*)
2. Untuk setiap pass ke – $I = 0, 1, \dots, N-1$, lakukan :
 - a. cari elemen minimum (*min*) mulai dari elemen ke – I sampai elemen ke – $(N-1)$
 - b. pertukarkan *min* dengan elemen ke – I

Rincian setiap *pass* adalah sebagai berikut :

Langkah 1 : Cari elemen minimum di dalam $L[0..(N-1)]$
 Pertukarkan elemen terkecil dengan elemen $L[0]$

Langkah 2 : Cari elemen minimum di dalam $L[1..(N-1)]$
 Pertukarkan elemen terkecil dengan elemen $L[1]$

Langkah 3 : Cari elemen minimum di dalam $L[2..(N-1)]$
 Pertukarkan elemen terkecil dengan elemen $L[2]$

.....

Langkah $N-1$: Tentukan elemen minimum di dalam $L[(N-2)..(N-1)]$
 Pertukarkan elemen terkecil dengan elemen $L[N-2]$
 (elemen yang tersisa adalah $L[N-1]$, tidak perlu diurut karena hanya satu-satunya).

Jadi, pada setiap *pass* pengurutan terdapat proses mencari harga minimum dan proses pertukaran dua buah elemen *array*.

Misal, terdapat *array* L dengan $N = 5$ buah elemen yang belum terururt. *Array* akan diurutkan secara Ascending (menaik), dengan **algoritma minimum selection sort**.

9	7	12	6	1
0	1	2	3	4

Pass 1 :

- Cari elemen terkecil di dalam array L[0..4]. Min=L[4]=1
- Tukar Min dengan L[0], diperoleh :

1	7	12	6	9
0	1	2	3	4

Pass 2 :

(berdasarkan susunan array pada Pass 1)

- Cari elemen terkecil di dalam array L[1..4]. Min=L[3]=6
- Tukar Min dengan L[1], diperoleh :

1	6	12	7	9
0	1	2	3	4

Pass 3:

(berdasarkan susunan array pada Pass 2)

- Cari elemen terkecil di dalam array L[2..4]. Min=L[3]=7
- Tukar Min dengan L[2], diperoleh :

1	6	7	12	9
0	1	2	3	4

Pass 4 :

(berdasarkan susunan array pada Pass 3)

- Cari elemen terkecil di dalam array L[3..4]. Min=L[4]=9
- Tukar Min dengan L[3], diperoleh :

1	6	7	9	12
0	1	2	3	4

Selesai, array L sudah terurut secara Ascending.

Pseudocode Algoritma Minimum Selection Sort secara Ascending :

1. //prosedur algoritma Minimum Selection Sort secara Ascending
2. //I.S:array sudah berisi nilai integer yang belum terurut
3. //F.S:nilai-nilai dalam array terurut secara Ascending
4. **procedure** v_minAsc(input/output A:array[0..4] of integer,
input N:integer)
5. **KAMUS:**
6. k,min,j,temp:integer
7. **ALGORITMA:**
8. **for** (k=0; k<=(N-2); k←k+1)

```

9.      //cari elemen terkecil
10.     min ← k
11.     for (j=(k+1); j<=(N-1); j←j+1)
12.         if (A[j] < A[min])
13.             min ← j
14.         endif
15.     endfor
16.     v_Tukar(A[k],A[min])
17. endfor

```

Program lengkap penerapan algoritma *Minimum Selection Sort Ascending* dalam bahasa C

```

1.  #include <stdio.h>
2.  #include <conio.h>
3.  void v_minAsc(int A[5],int N);
4.  void v_Tukar(int *P,int *M);
5.  main()
6.  { int L[5];
7.    int i,j,k,min,temp,N;
8.    //input data array
9.    printf("Input Data Array\n");
10.   printf("\nBanyak Data : "); scanf("%i",&N);
11.   for(i=0;i<N;i++)
12.   { printf(" Data ke-%i = ",i+1);
13.     scanf("%i",&L[i]); } //end loop i
14.   //panggil procedure v_minAsc
15.   v_minAsc(L,N);
16.   //output data array
17.   printf("\n Data Sortir:\n");
18.   for(i=0;i<N;i++)
19.   { printf(" %5i",L[i]); } //end loop i
20.   printf("\n Tekan Enter\n");
21.   getch();
22. } //end main program
23.
24. void v_minAsc(int A[5],int N)
25. { int k,min,j,temp;
26.   //proses minimum ascending selection sort
27.   for(k=0;k<=(N-2);k++)
28.   { min = k;
29.     for(j=(k+1);j<=(N-1);j++)
30.     { if (A[j] < A[min])
31.       min = j; } //end loop j
32.     v_Tukar(&A[k],&A[min]); } //end loop k

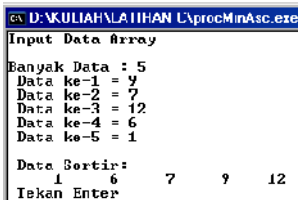
```

```

33. } //end procedure
34.
35. void v_Tukar(int *P,int *M)
36. { int temp;
37.   temp = *P;
38.   *P = *M;
39.   *M = temp;
40. } //end procedure v_Tukar

```

Output yang dihasilkan:



```

D:\KULIAHLATIHAN\AprocMinAsc.exe
Input Data Array
Banyak Data : 5
Data ke-1 = 9
Data ke-2 = 8
Data ke-3 = 11
Data ke-4 = 7
Data ke-5 = 12
Data Berturut:
1 6 7 9 12
Tekan Enter

```

9.3.4 Minimum Selection Sort Descending

Misal, terdapat array *L* dengan $N = 5$ buah elemen yang belum terurut. Array akan diurutkan secara **Descending** (menurun), dengan **algoritma minimum selection sort**.

9	8	11	7	12
0	1	2	3	4

Pass 1 :

- Cari elemen terkecil di dalam array $L[0..4]$. $\text{Min}=L[3]=7$
- Tukar Min dengan $L[4]$, diperoleh :

9	8	11	12	7
0	1	2	3	4

Pass 2 :

(berdasarkan susunan array pada Pass 1)

- Cari elemen terkecil di dalam array $L[0..3]$. $\text{Min}=L[1]=8$
- Tukar Min dengan $L[3]$, diperoleh :

9	12	11	8	7
0	1	2	3	4

Pass 3 :

(berdasarkan susunan array pada Pass 2)

- Cari elemen terkecil di dalam array L[0..2]. Min=L[0]=9
- Tukar Min dengan L[2], diperoleh :

11	12	9	8	7
0	1	2	3	4

Pass 4 :

(berdasarkan susunan array pada Pass 3)

- Cari elemen terkecil di dalam array L[0..1]. Min=L[0]=11
- Tukar Min dengan L[1], diperoleh :

12	11	9	8	7
0	1	2	3	4

Selesai array L sudah terurut secara Descending (menurun)

Pseudocode Algoritma Minimum Selection Sort secara Descending :

```

1. //prosedur algoritma Minimum Selection Sort secara Descending
2. //I.S:array sudah berisi nilai integer yang belum terurut
3. //F.S:nilai-nilai dalam array terurut secara Descending
4. procedure v_minDesc (input/output A:array[0..4] of integer,
                        input N:integer)
5. KAMUS:
6.   k,j,temp,min : integer
7. ALGORITMA:
8.   //minimum selection sort descending
9.   for (k=(N-1) ; k>=1 ; k←k-1)
10.    min←0
11.    //cari nilai terkecil
12.    for (j=0 ; j<=k ; j←j+1)
13.      if (A[j] < A[min])
14.        min←j
15.      endif
16.    endfor
17.    v_Tukar(A[k],A[min])
20. endfor

```

Program lengkap penerapan algoritma *Minimum Selection Sort Descending* dalam bahasa C


```
1. #include <stdio.h>
2. #include <conio.h>
3. void v_minDesc(int A[5],int N);
4. void v_Tukar(int *P,int *M);
5. main()
6. { int L[5];
7.   int i,N;
8.   //input data array
9.   printf("Input Data Array\n");
10.  printf("\nBanyak Data : ");scanf("%i",&N);
11.  for(i=0;i<N;i++)
12.  { printf(" Data ke-%i = ",i+1);
13.    scanf("%i",&L[i]); } //endloop i
14.    //panggil procedure v_minDesc
15.    v_minDesc(L,N);
16.    //output data array
17.    printf("\n Data Sortir:\n");
18.    for(i=0;i<N;i++)
19.    { printf(" %5i",L[i]); } //endloop i
20.    printf("\n Tekan Enter...\n");
21.    getch();
22. } //end main program
23.
24. void v_minDesc(int A[5],int N)
25. { int k,j,temp,min;
26.   //minimum selection sort descending
27.   for(k=(N-1);k>=1;k--)
28.   { min = 0;
29.     for(j=0;j<=k;j++)
30.     { if (A[j] < A[min])
31.       min=j; } //endloop j
32.     v_Tukar(&A[k],&A[min]); } //endloop k
33. } //end procedure v_minDesc
34.
35. void v_Tukar(int *P,int *M)
36. { int temp;
37.   temp = *P;
38.   *P = *M;
39.   *M = temp;
40. } //end procedure v_Tukar
```

Output yang dihasilkan:

ca D:\KULIAH\KULIAHAN C\procNinDesc.exe

Input Data Array

Banyak Data : 5

Data ke-1 = 9

Data ke-2 = 8

Data ke-3 = 11

Data ke-4 = 7

Data ke-5 = 12

Data Sortir:

12 11 9 8 7

Tekan Enter...

9.4 Insertion Sort

Insertion sort adalah sebuah algoritma pengurutan yang membandingkan dua elemen data pertama, mengurutkannya, kemudian mengecek elemen data berikutnya satu persatu dan membandingkannya dengan elemen data yang telah diurutkan. Karena algoritma ini bekerja dengan membandingkan elemen-elemen data yang akan diurutkan, algoritma ini termasuk pula dalam *comparison-based sort*.

Ide dasar dari algoritma *Insertion Sort* ini adalah mencari tempat yang "tepat" untuk setiap elemen array, dengan cara *sequential search*. Proses ini kemudian menyisipkan sebuah elemen array yang diproses ke tempatnya yang seharusnya. Proses dilakukan sebanyak $N-1$ tahapan (dalam *sorting* disebut sebagai "*pass*"), dengan indeks dimulai dari 0.

Proses pengurutan dengan menggunakan algoritma *Insertion Sort* dilakukan dengan cara membandingkan data ke- i (dimana i dimulai dari data ke-2 sampai dengan data terakhir) dengan data berikutnya. Jika ditemukan data yang lebih kecil maka data tersebut disisipkan ke depan sesuai dengan posisi yang seharusnya.

Misal terdapat *array* satu dimensi L , yang terdiri dari 7 elemen *array* ($n=7$). *Array* L sudah berisi data seperti dibawah ini dan akan diurutkan secara *ascending* dengan algoritma *Insertion Sort*.

$L[]$	15	10	7	22	17	5	12
	0	1	2	3	4	5	6

Tahapan *Insertion Sort*:

- Dimulai dari $L[1]$: Simpan nilai $L[1]$ ke variabel X.
(Pass-1) Geser masing-masing satu langkah ke kanan semua nilai yang ada disebelah kiri $L[1]$ satu persatu apabila nilai tersebut lebih besar dari X.
Setelah itu **insert**-kan (sisipkan) X di bekas tempat nilai yang terakhir digeser.
- Dilanjutkan ke $L[2]$: Simpan nilai $L[2]$ ke variabel X
(Pass-2) Geser masing-masing satu langkah ke kanan semua nilai yang ada disebelah kiri $L[2]$ satu persatu apabila nilai tersebut lebih besar dari X.
Setelah itu **insert**-kan (sisipkan) X di bekas tempat nilai yang terakhir di geser.
- Demikian seterusnya untuk $L[3]$, $L[4]$, $L[5]$, dan terakhir $L[6]$ bila $n = 7$. Sehingga untuk $n = 7$ ada 6 pass proses pengurutan.

Berikut ilustrasi dari 6 pass tersebut:

Data awal:

15	10	7	22	17	5	12
0	1	2	3	4	5	6

Pass-1:

15	10	7	22	17	5	12	10
0	1	2	3	4	5	6	X

Pass 1 dimulai dari kolom $L[1]$, $X=L[1]=10$

15 lebih besar dari 10, maka geser 15 ke kanan.
Proses selesai karena sudah sampai kolom 1.
Kemudian insert X menggantikan 15.

15	15	7	22	17	5	12
0	1	2	3	4	5	6

10	15	7	22	17	5	12
0	1	2	3	4	5	6

Hasil Pass 1:

10	15	7	22	17	5	12
----	----	---	----	----	---	----

0 1 2 3 4 5 6

Pass-2:

10	15	7	22	17	5	12	7
0	1	2	3	4	5	6	X

Pass 2 dimulai dari $L[2]$, $X=L[2]=7$.

15 lebih besar dari 7, maka geser 15 ke kanan. 10 lebih besar dari 7, maka geser 10 ke kanan. Proses selesai karena sudah sampai kolom 1. Kemudian insert X menggantikan 10.

	15	15	22	17	5	12
0	1	2	3	4	5	6

10	10	15	22	17	5	12
0	1	2	3	4	5	6

7	10	15	22	17	5	12
0	1	2	3	4	5	6

Hasil Pass 2:

7	10	15	22	17	5	12
0	1	2	3	4	5	6

Pass-3:

7	10	15	22	17	5	12	22
0	1	2	3	4	5	6	X

Pass 3 dimulai dari $L[3]$, $X=L[3]=22$.

15 tidak lebih besar dari 22, maka proses selesai. Kemudian insert X menggantikan 22.

Proses berlanjut sampai Pass 6. Hasil tiap pass dapat digambarkan sebagai berikut:

Data awal:

15	10	7	22	17	5	12
0	1	2	3	4	5	6

Pass 1:

10	15	7	22	17	5	12
----	----	---	----	----	---	----

	0	1	2	3	4	5	6
Pass 2:	7	10	15	22	17	5	12
	0	1	2	3	4	5	6
Pass 3:	7	10	15	22	17	5	12
	0	1	2	3	4	5	6
Pass 4:	7	10	15	17	22	5	12
	0	1	2	3	4	5	6
Pass 5:	5	7	10	15	17	22	12
	0	1	2	3	4	5	6
Pass 6:	5	7	10	12	15	17	22
	0	1	2	3	4	5	6

Selesai array L sudah terurut secara Ascending (menaik)

Pseudocode Algoritma Insertion Sort secara Ascending :

```

1. //prosedur algoritma Insertion Sort secara Ascending
2. //I.S:array sudah berisi nilai integer yang belum terurut
3. //F.S:nilai-nilai dalam array terurut secara Ascending
4. procedure v_inAsc(input/output A:array[0..6]of integer,
                    input N:integer)
5. KAMUS:
6.   k,X,i:integer
7. ALGORITMA:
8. //insertion sort ascending
9.   k←1
10. while (k<=N-1)
11.   i←k
12.   X←A[i]
13.   while (i>=1 && A[i-1]>X)
14.     A[i]←A[i-1]
15.     i←i-1
16.   endwhile
17.   A[i]←X
18.   k←k+1
19. endwhile

```

Program lengkap penerapan algoritma *Insertion Sort Ascending* dalam bahasa C

```
#include <stdio.h>
#include <conio.h>
main()
{ int L[7];
  int i,N;
  void v_insertAsc(int A[7],int N);

  //input data array
  printf("Input Data Array\n");
  printf("\nBanyak Data: "); scanf("%i",&N);
  for(i=0;i<N;i++)
  { printf("Nilai ke-%i = ",i+1);
    scanf("%i",&L[i]); } //end loop i
  //panggil procedure v_inAsc
  v_insAsc(L,N);
  //output data array
  printf("Data terurut:\n");
  for(i=0;i<N;i++)
  { printf("%5i",L[i]); } //end loop i
  printf("\nTekan Enter...\n");
  getch();
}

void v_insAsc(int A[7],int N)
{ int k,X,i;
  //insertion sort ascending
  k=1;
  while(k<=N-1)
  { i=k;
    X=A[i];
    while(i>=1 && A[i-1]>X)
    { A[i]=A[i-1];
      i--; } //endwhile
    A[i]=X;
    k++; } //endwhile
} //end procedure
```

Output yang dihasilkan:



```
D:\My Documents in D\Maya\Data Maya 280502\LATIHAN C\proclnsAsc.exe
Input: Data Array
Banyak Data: 7
Nilai ke-1 = 15
Nilai ke-2 = 10
Nilai ke-3 = 7
Nilai ke-4 = 22
Nilai ke-5 = 17
Nilai ke-6 = 5
Nilai ke-7 = 12
Data terurut:
5 7 10 12 15 17 22
Tekan Enter...
```



Rangkuman

1. **Proses Sorting** merupakan proses mengurutkan data yang berada dalam suatu tempat penyimpanan, dengan urutan tertentu baikurut menaik (*ascending*) dari nilai terkecil sampai dengan nilai terbesar, atau urut menurun (*descending*) dari nilai terbesar sampai dengan nilai terkecil
2. Terdapat dua macam proses pengurutan, yaitu pengurutan internal (*internal sort*) dan pengurutan eksternal (*external sort*).
3. *Bubble sort* adalah proses pengurutan sederhana yang bekerja dengan cara berulang kali membandingkan dua elemen data pada suatu saat dan menukar elemen data yang urutannya salah.
4. Algoritma *Selection sort* memilih elemen maksimum/minimum *array*, lalu menempatkan elemen maksimum/minimum itu pada awal atau akhir *array* (tergantung pada urutannya *ascending/descending*).
5. Algoritma *Insertion Sort*, mencari tempat yang "tepat" untuk setiap elemen *array*, dengan cara *sequential search*. Proses ini kemudian menyisipkan sebuah elemen *array* yang diproses ke tempatnya yang seharusnya.



Pilihan Ganda

Petunjuk: Pilihlah jawaban yang paling tepat!

1. Usaha untuk mengurutkan kumpulan-kumpulan data dalam suatu *array* disebut:
- A. Searching C. Sorting
B. Divide D. Conquer

Berikut ini adalah metode yang digunakan pada teknik

2. sorting, **kecualli**:
- | | |
|-----------|--------------|
| A. Bubble | C. Fibonacci |
| B. Heap | D. Radix |

3. Data 4 0 8 2, diurutkan secara *ascending* (dari kecil ke besar) dengan metode *bubble sort*. Hasil urutan data pass satu (tahap satu) adalah:
- A. 0 4 2 8 C. 0 8 2 4
B. 0 4 8 2 D. 0 2 4 8

Pada data 0 6 3 2 4, akan dilakukan *maximum selection*

4. sort secara ascending, maka pada langkah pertama, urutan data yang terjadi adalah:
- A. 0 2 6 3 4 C. 0 4 3 2 6
B. 0 2 3 4 6 D. 0 3 6 2 4

5. Pengurutan adalah proses untuk:
 - A. mencari elemen sebuah list
 - B. mengecek harga elemen tertentu
 - C. pengaturan kembali elemen kedalam urutan tertentu
 - D. menyederhanakan persoalan dengan cara memecah ke persoalan yang

lebih kecil



Latihan

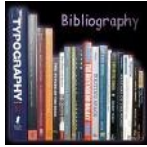
1. Misal terdapat record mahasiswa sebagai berikut :
 - nim mahasiswa (tipe data integer)
 - nama mahasiswa (tipe data string, panjang max 15 char)
 - tahun lahir (tipe integer)
 - umur (tipe real)Buat algoritma dengan menggunakan *procedure* berparameter untuk menampilkan kembali data mahasiswa yang terurut berdasarkan tahun lahir. Asumsikan terdapat 5 mahasiswa.
2. PT. MURAH HATI memberi komisi salesmannya berdasarkan ketentuan sebagai berikut :
 - ❖ Bila salesman berhasil menjual barang **seharga** Rp 500.000,- maka dia akan mendapat komisi sebesar 10 %.
 - ❖ Bila lebih dari Rp 500.000,-, untuk Rp 500.000,- pertama komisinya 10 %, sedangkan sisanya mendapat 15 %.Bila perusahaan tersebut memiliki 5 orang salesman, rancanglah algoritma untuk menentukan komisi yang diterima oleh **setiap** salesmannya, serta **total komisi** yang telah dibayarkan oleh PT. MURAH HATI kepada ke 5 salesman tadi serta total penjualan yang berhasil dilakukan para salesman.

Adapun spesifikasi dari algoritma adalah:

 - a. Algoritma harus menggunakan *procedure* atau *function* **berparameter.**
 - b. Output yang diinginkan adalah memunculkan data karyawan (nama karyawan, besar penjualan, serta komisi) yang **terurut secara descending** berdasarkan **besar penjualan.**
 - c. Output yang diinginkan:

LAPORAN KOMISI KARYAWAN			
NO	NAMA	PENJUALAN	KOMISI
1.			
:			
5.			

--	Total	99999	99999



Daftar Pustaka

1. Algorithm Data Structures and Problem Solving with C++. 1997. Addison Wesley.
 2. Moh. Sjukani, Algoritma dan Struktur Data. Mitra Wacana Media
 3. Inggriani Liem, Diktat Catatan Singkat Bahasa C Agustus 2003. ITB
 4. Inggriani Liem, Diktat Kuliah Dasar Pemrograman April 2007. ITB
 5. Rinaldi Munir, Algoritma dan Pemrograman. Informatika Bandung
 6. Schaum, Programming with C++ 2nd. 2000. McGraw-Hill
 7. Schaum. Teach yourself C++ in 21 days. 2007. McGraw-Hill
 8. http://www.cs.aau.dk/~normark/prog3-03/html/notes/paradigms_themes-paradigms.html akses pada 18 Juli 2009 14.00
 9. <http://encyclopedia.thefreedictionary.com/Programming%20paradigm> akses pada 18 Juli 2009 14.00
-