ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/315773643

Belajar Dasar Algoritma dan Pemrograman C++

Book · January 2016

CITATIONS
5

READS
203,935

3 authors:

Agus Perdana Windarto
Sekolah Tinggi Ilmu Komputer Tunas Bangsa Pemata…
188 PUBLICATIONS **3,521** CITATIONS

SEE PROFILE

Henny Harumy
University of North Sumatra
25 PUBLICATIONS **68** CITATIONS

SEE PROFILE

Indri Sulistianingsih
Politeknik Negeri Medan
20 PUBLICATIONS **53** CITATIONS

SEE PROFILE

All content following this page was uploaded by Indri Sulistianingsih on 29 August 2017.

The user has requested enhancement of the downloaded file.

PEMULA
BELAJAR DASAR
ALGORITMA & PEMROGRAMAN
C++
T. HENNY FEBRIANA HARUMY S.KOM,. M.SI,. M.KOM
AGUS PERDANA WINDARTO S.KOM,. M.KOM
INDRI SULISTIANINGSIH S.KOM

BIOGRAFI PENULIS

T.Henny Febriana Harumy S.Kom.,M.Kom.,M.Si Lahir di Banda Aceh 19 Februari 1988 . Menyelesaikan S1 di Universitas Pembangunan Panca Budi Medan, Kemudian menyelesaikan Program Magister Pada Program Magister Ilmu Komputer, Universitas Putra Indonesia "YPTK" Padang pada tahun 2014 Dan menyelesaikan Program Magister Ekonomi Pembangunan Pada Universitas Sumatera Utara pada Tahun 2012. Mulai meniti karir sebagai dosen pada Universitas Pembangunan Pancabudi Medan sejak tahun 2011 dengan kompetensi biang kajian adalah tentang teknologi informasi dan artificial inteligent dan komputer grafik.

Agus Perdana Windarto S.Kom.,M.Kom, Lahir di Pematangsiantar, 30 Agustus 1986. Menamatkan SMU di SMU Negeri 1 Porsea. Menyelesaikan program S1 dari Jurusan Komputer Program Teknik Informatika, Sekolah Tinggi Teknik Poliprofesi pada tahun 2010. Menyelesaikan program S2 dari Jurusan Komputer Program Magister Ilmu Komputer, Universitas Putra Indonesia "YPTK" Padang pada tahun 2014. Kompetensi awalnya adalah bidang basis data, sistem informasi, Pemograman visual, sistem pakar dan saat ini sedang bergerak ke arah open source.

Indri Sulistianingsih S.Kom, Lahir di Pematangsiantar 04 Mei 1991. Menamatkan pendidikan di SMK Negeri 9 Medan. Menyelesaikan program S1 di Universitas Pembangunan Panca Budi Medan pada tahun 2012. Kemudian saat ini masih aktif sebagai Mahasiswi PJJ APTIKOM – AMIKOM Yogyakarta dan mulai meniti karir sebagai dosen pada Universitas Pembangunan Pancabudi Medan sejak tahun 2013 dengan kompetensi biang kajian adalah tentang pemrograman, sistem informasi dan basis data.

KATA PENGANTAR

Puji Syukur Kehadirat Allah SWT, berkat rahmat Allah SWT, akhirnya penulis dapat menyelesaikan penyusunan Buku belajar dasar – dasar algoritma & pemrograman C++. Buku ini membahas tentang bahasa pemrograman tingkat menengah yang sanga mudah untuk dipelajari dan di pahami. Buku ini akan membantu untuk mengenal bahasa pemrograman C++ beserta dengan contoh penyelesaian kasus sehingga bagi pemula dapat menguasai bahasa pemrograman ini dengan cepat dan mudah

Buku ini sangat cocok untuk mahasiswa dan pemula yang baru mempelajari dan ingin lebih menguasai pemrograman C++. Materi buku ini dibuat tahap demi tahap dan latihan untuk mempercepat pemahaman.

Dalam kesempatan ini penulis mengucapkan terima kepada semua pihak yang telah membantu dan banyak memberikan motivasi sehinga buku ini dapat diselesaikan. Ucapan terimakasih khususnya penulis ucapkan kepada Kedua orang tua kami yang terus berkontribusi dalam pendidikan dan selalu memotivasi agar kami selalu berkarya. Dan kepada seluruh teman teman yang tidak dapat disebutkan satu persatu Kepada Akhirnya dengan segala kerendahan hati, penulis sangat mengharapkan kritik dan saran yang sifatnya membangun dari pembaca demi kesempurnaan buku ini sehingga akan lebih bermanfaat. Dan semua pihak yang telah banyak membantu , dan akhirnya kami berharap semoga buku ini bias bermanfaat. Amin.

Medan, Februari 2016
Penulis,

DAFTAR ISI

KATA PENGANTAR ..... *i*

DAFTAR ISI ..... *ii*

PENDAHULUAN *1*
1.1 Komputer Elektronik 2
1.2 Komponen Komputer 2
1.3 Algoritma 3
1.4 Program 4
1.5 Bahasa Pemrograman 4
1.5.1 Klasifikasi Menurut Generasi 5
1.5.2 Klasifikasi Menurut Tingkatan 6
1.6 Paradigma Pemrograman 6
1.6.1 Paradigma Imperatif 6
1.6.2 Paradigma Fungsional 7
1.6.3 Paradigma Logika 7
1.6.4 Paradigma Berorientasi Obyek 8
1.7 Flowchart 8
1.8 Pseudocode 11

FLOWCHART DAN PSEUDOCODE *13*
2.1 Flowchart 14
2.1.1 Pengambilan Keputusan 14
2.1.2 Pengulangan Proses 17
2.2 Pseudocode 19
2.3 Struktur algoritma 19

PENGENALAN BAHASA C++ *22*
3.1 Sejarah C++ 23
3.2 Struktur Bahasa C++ 23
3.3 Elemen Dasar C++ 27
3.3.1 Himpunan Karakter 27
3.3.2 Pengenal (Identifier) 27
3.3.3 Penamaan Pengenal 28
3.3.4 Huruf Kecil dan kapital Berbeda 28

3.3.5 Kata Kunci 28
3.3.6 Tipe Data..... 28
3.3.7 Variabel dan Konstanta 29

VARIABEL,TIPE DATA DAN KONSTANTA.....*30*
4.1 Identifiers 31
4.2 Deklarasi variabel 33
4.2.1 Menentukan Tipe Variabel..... 33
4.2.2 Memberikan Nilai ke Variabel 33
4.2.3 Insialisai Variabel 34
4.3 Lingkup Variabel 35
4.4 Deklarasi Konstanta 36
4.4.1 Nilai Integer 36
4.4.2 Nilai Floating Point 36
4.5 Karakter dan String 37

OPERATOR DAN UNGKAPAN.....*39*
5.1 Pengantar Operator dan Ungkapan 40
5.1.1 Operator Aritmatika 40
5.1.2 Operator Sisa Pembagian..... 42
5.1.3 Operator Penurunan dan Penaikan..... 42
5.1.4 Operator Majemuk 44
5.1.5 Operator Kondisi..... 45
5.2 Ungkapan Kondisi..... 46
5.2.1 Operator Relasi..... 46
5.2.2 Fungsi Pustaka..... 47

OPERASI DASAR MASUKAN DAN KELUARAN.....*49*
6.1 Cout 50
6.2 Manipulator 50
6.2.1 Manipulator endl..... 50
6.2.2 Manipulator setw()..... 51
6.3 cin..... 51
6.3.1 cin dengan sebuah variabel 51
6.3.2 cin dengan lebih dari satu variabel..... 52
6.3.3 Fungsi getc() dan getche()..... 54

PEMILIHAN *60*
7.1 Bentuk Umum IF dan Variasinya 61
7.1.1 Pernyataan if 61
7.1.2 Pernyataan if sederhana 61
7.1.3 Pernyataan if else 63
7.1.4 Pernyataan if dalam if 65
7.1.5 Terapan bentuk-bentuk IF 68
7.1.6 Pernyataan Case 74
7.1.7 Terapan bentuk-bentuk CASE 77
7.2 Konversi Struktur IF dan CASE ke Bahasa C 79

PENGULANGAN (LOOPING) *96*
8.1 Konsep Pengulangan 97
8.1.1 Sintaks WHILE 98
8.1.2 Sintaks DO…WHILE 107
8.1.3 Sintaks FOR 116

ARRAY *134*
9.1 Array 135
9.1.1 *Array* Satu Dimensi 136
9.1.2 Array Dua Dimensi 139

PEMROGRAMAN MODULAR *148*
10.1 Definisi Pemrograman Modular 149
10.2 Variabel Lokal dan Variabel Global 150
10.2.1 Variabel Lokal 150
10.2.2 Variabel Global 151
10.3 Fungsi 151
10.3.1 Struktur Fungsi 152
10.3.2 Prototipe Fungsi 153
10.3.3 Parameter Fungsi 154
10.3.4 Pemanggilan dengan nilai (*Call by Value*) 155
10.3.5 Pemanggilan dengan Referensi (*Call by Reference*) 156
10.4 Prosedur 158
10.5 Fungsi dan Prosedur yang telah terdefinisi 162
10.6 Fungsi Rekursif 163
10.7 Unit 164

PENGURUTAN (SORTING).....*170*
11.1 Pengertian Sort 171
11.1.1 Bubble Sort.....171
11.1.2 Selection Sort.....176
11.1.3 Insertion Sort 190

DAFTAR REFERENSI

PENDAHULUAN

1

Overview

Komputer merupakan istilah yang umum yang sudah dikenal oleh masyarakat banyak pada umumnya. 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. Dengan kata lain komputer dan program merupakan satu kesatuan yang saling berkaitan satu sama lain.

Tujuan

- ☑ Memahami bagaimana komputer menangani data elektronik
- ☑ Memahami komponen yang terlibat dalam memproduksi informasi
- ☑ Memahami perbedaan bahasa pemrograman di setiap tingkatan

1.1 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 1.1 Komputer elektronik

1.2 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.

1.3 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. 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:

- Nyalakan api kompor
- Tuangkan minyak ke dalam wajan
- Pecahkan telur ayam ke dalam mangkok
- Tambahkan garam secukupnya
- Aduk campuran telur dan garam
- Tuang adonan telur ke dalam wajan
- 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*.

1.4 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.

1.5 Bahasa Pemrograman

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

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.

1.5.1 Klasifikasi Menurut Generasi

- First Generation Language (1GL)
  Bahasa pemrograman ini berupa kode-kode mesin yang hanya bisa dipahami oleh mikroprosesor.
- 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 programer tetap harus mengetahui keunikan dari masing-masing mikroprosesor (register dan jenis instruksi).
- 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.
- 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. Contoh bahasa:
  - Pemrograman umum : DataFlex, WinDev, PowerBuilder
  - Basis data : SQL, Progress 4GL
  - Manipulasi data, analisis dan pelaporan : ABAP, Matlab, PL/SQL.
- 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. Bahasa pemrograman ini digunakan untuk membangun sistem kecerdasan buatan dan belum digunakan secara meluas di dunia industri. Contoh bahasa: Prolog, LISP, Mercury.

1.5.2 Klasifikasi Menurut Tingkatan

- 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).
- 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.
- Very high-level programming language (VHLL)
  Bahasa ini memiliki abstraksi yang lebih tinggi dibandingkan HLL, dan digunakan untuk menunjang produktifitas programer profesional. Biasanya VHLL digunakan hanya untuk tujuan yang spesifik, misalnya untuk keperluan bisnis: mengolah data, membuat laporan, dsb.

1.6 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:

1.6.1 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 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

1.6.2 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.

1.6.3 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 sistematik dalam sekumpulan fakta, dengan menggunakan sekumpulan aturan.
- Beberapa contoh bahasa pemrograman: ALF, Fril, Gödel, Mercury, Oz, Ciao, Visual Prolog, XSB, and λProlog

1.6.4 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.

1.7 Flowchart

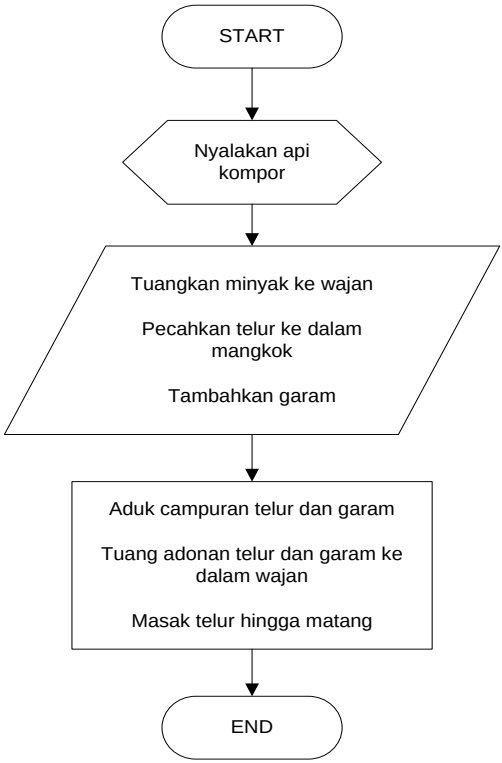
Dalam membuat algoritma, diperlukan suatu mekanisme atau alat bantu untuk menuangkan hasil pemikiran mengenai langkah-langkah penyelesaian masalah yang sistematis dan terurut. Pada dasarnya untuk bisa menyusun solusi diperlukan kemampuan *problem-solving* yang baik. Oleh karena itu, sebagai sarana untuk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

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

Tabel 1.1 Simbol-simbol dalam *flowchart*

	Terminator Sebagai simbol 'START' atau 'END' untuk memulai atau mengakhiri flowchart.
	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 *monitor*

Berikut ini adalah flowchart untuk menggambarkan kegiatan membuat telur dadar:



Gambar 1.2 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.

1.8 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 :

- lebih mudah dibaca oleh manusia
- lebih mudah untuk dipahami
- 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 programer yang ingin 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 biasnya 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 programer 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
bonus ← sales * 25%
salary ← 2000000 + bonus
endif
output(salary)
	int sales;
	sales=1001;
	if (sales > 1000)
	{
	bonus = sales * 0.25;
	salary = 2000 + bonus;
	}
	cout << "Salary : "<<salary;

If totalbelanja > 500000 then
diskon ← 0.2 * totalbelanja
bayar ← totalbelanja – diskon
endif
output(bayar)
	int totalbelanja;
	totalbelanja =500001;
	if (totalbelanja > 500000)
	{
	diskon = 0.25 * totalbelanja;
	bayar = totalbelanja - diskon;
	}
	cout << "Bayar : "<<bayar;

FLOWCHART DAN PSEUDOCODE

2

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

- ☑ Mengenal bentuk pengambilan keputusan menggunakan flowchart
- ☑ Mengenal operasi boolean
- ☑ Mengenal bentuk pengulangan menggunakan flowchart
- ☑ Memahami tujuan penggunaan pseudocode dalam menyusun algoritma

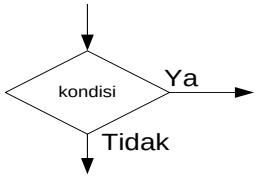
2.1 Flowchart

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.

2.1.1 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 :

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. 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 2.2 Tabel Kebenaran AND

X	Y	Hasil
T	T	T
T	F	F
F	T	F
F	F	F

Tabel 2.3 Tabel Kebenaran OR

X	Y	Hasil
T	T	T
T	F	T
F	T	T
F	F	F

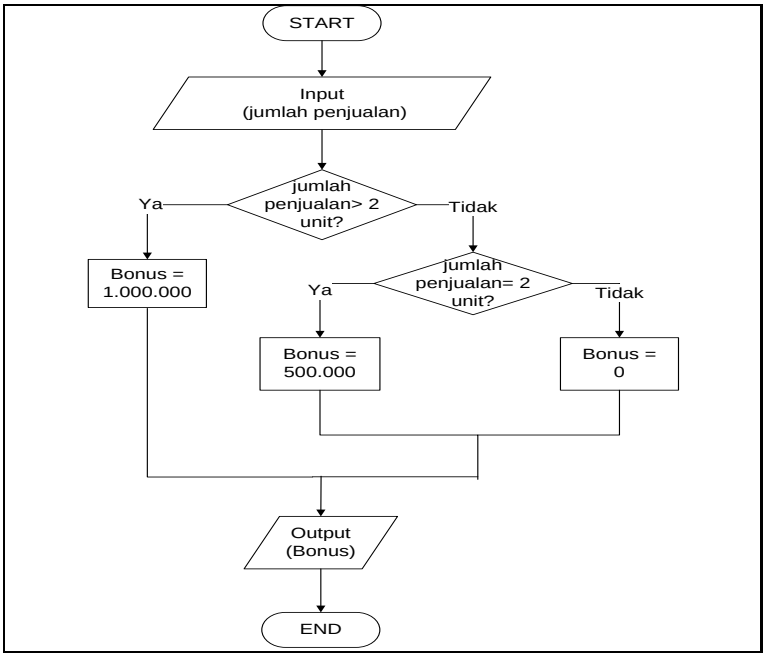
Tabel 2.4 Tabel Kebenaran OR

X	Hasil
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

2.1.2 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:

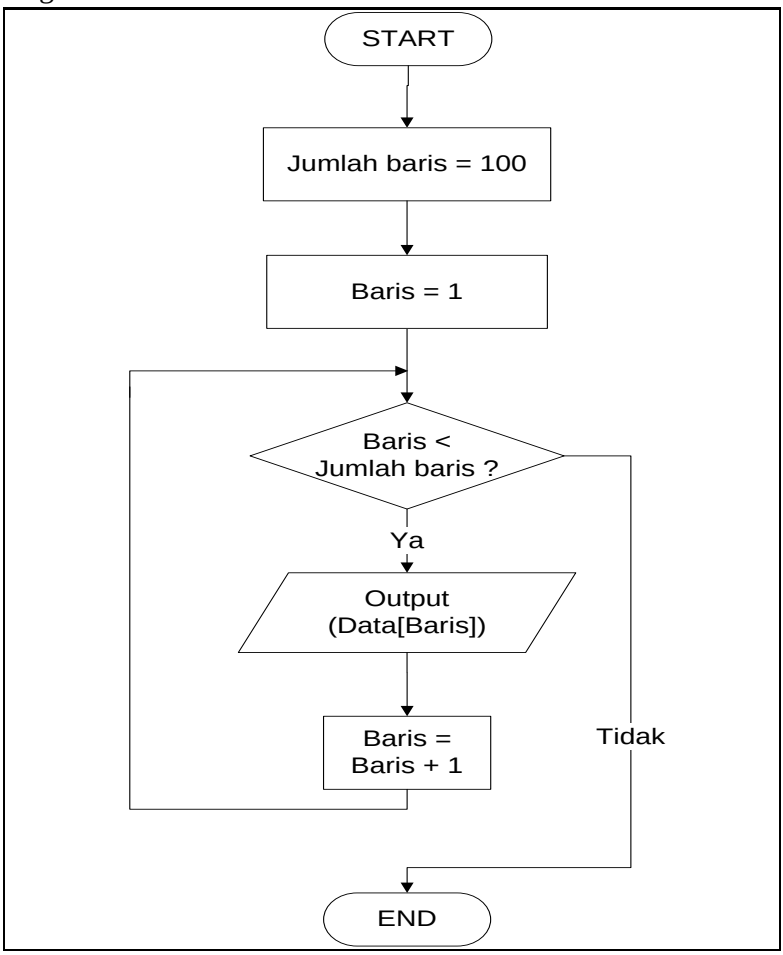
Dalam kasus ini yang diminta adalah bagaimana menampilkan data sebanyak 100 baris tanpa harus menggunakan proses output sebanyak 100 kali. Metode yang digunakan adalah pengulangan.

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

- Inisialisasi (penentuan kondisi/nilai awal)
- Proses
- 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 tehadap 10 baris data dapat dilakukan.

2.2 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 pemroraman tertentu. Dalam buku ini akan digunakan aturan penulisan pseudocode yang mendekati bahasa pemrograman Pascal.

2.3 Struktur algoritma

Struktur algoritma yang digunakan mengacu pada struktur pemrograman bahasa C++ 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:

- Tidak diawali dengan angka atau karakter selain alphabet
- 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.

Rangkuman

- ☑ Flowchart digunakan untuk menggambarkan algoritma atau proses
- ☑ Dalam melakukan pengujian kondisi digunakan notasi relasional
- ☑ Simbol dalam flowchart yang digunakan untuk pengambilan keputusan adalah belah ketupat
- ☑ Analisa yang benar sangat dibutuhkan saat membuat flowchat
- ☑ Aljabar boolean merupakan kalkulus logika yang digunakan untuk menentukan nilai kebenaran dari suatu ekspresi logika
- ☑ Program yang baik adalah program yang bisa mengoptimalkan kinerja komputer, dengan cara menggunakan kembali program atau sekumpulan program dengan proses tertentu
- ☑ Pseudocode cocok digunakan untuk menuliskan algoritma dengan kasus yang kompleks dan berskala besar

PENGENALAN BAHASA C++

Overview

Berbicara tentang C++ tak lepas dari C, sebagai bahasa pendahulunya. C adalah bahasa pemrograman yang dapat dikatakan berada antara bahasa beraras rendah (bahasa yang berorientasi pada mesin) dan bahasa beraras tinggi (bahasa yang berorientasi pada manusia). Seperti diketahui bahasa tingkat tinggi mempunyai kompatibilitas yang tinggi antar *platform.*

Tujuan

- ☑ Mengetahui sejarah bahasa C++
- ☑ Mengenal struktur dasar bahasa C++
- ☑ Mengenal elemen elemen C++
- ☑ Memahami tujuan penggunaan bahasa C++

3.1 Sejarah C++

Tahun 1978, Brian W. Kerninghan & Dennis M. Ritchie dari AT & T Laboratories mengembangkan bahasa B menjadi bahasa C. Bahasa B yang diciptakan oleh Ken Thompson sebenarnya merupakan pengembangan dari bahasa BCPL (Basic Combined Programming Language) yang diciptakan oleh Martin Richard.

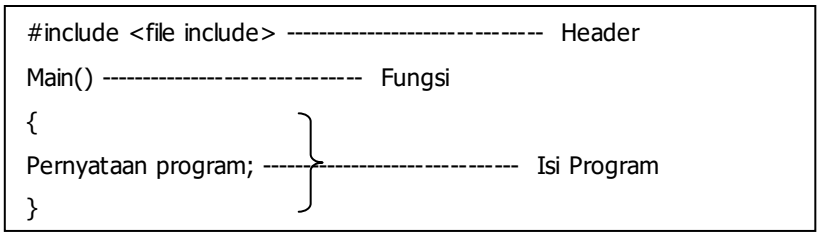
Sejak tahun 1980, bahasa C banyak digunakan pemrogram di Eropa yang sebelumnya menggunakan bahasa B dan BCPL. Dalam perkembangannya, bahasa C menjadi bahasa paling populer diantara bahasa lainnya, seperti PASCAL, BASIC, FORTRAN.

Tahun 1989, dunia pemrograman C mengalami peristiwa penting dengan dikeluarkannya standar bahasa C oleh *American National Standards Institute* (ANSI). Bahasa C yang diciptakan Kerninghan & Ritchie kemudian dikenal dengan nama ANSI C.

Mulai awal tahun 1980, Bjarne Stroustrup dari AT & T Bell Laboratories mulai mengembangkan bahasa C. Pada tahun 1985, lahirlah secara resmi bahasa baru hasil pengembangan C yang dikenal dengan nama C++. Sebenarnya bahasa C++ mengalami dua tahap evolusi. C++ yang pertama, dirilis oleh AT&T Laboratories, dinamakan **cfront**. C++ versi kuno ini hanya berupa kompiler yang menterjemahkan C++ menjadi bahasa C

sebuah kompiler yang mampu mengubah C++ langsung menjadi bahasa mesin (assembly). Sejak evolusi ini, mulai tahun 1990 C++ menjadi bahasa berorientasi obyek yang digunakan oleh sebagian besar pemrogram professional.

3.2 Struktur Bahasa C++



Contoh 1	Hasil

// Program Pertama saya di C++
#include <iostream.h>
int main ()
{
cout << "Hello World!";
return 0;
}
	Hello World!

Sisi kiri merupakan *source code*, yang dapat diberi nama *hiworld*.cpp dan sisi kanan adalah hasilnya setelah di-kompile dan di-eksekusi.

Program diatas merupakan salah satu program paling sederhana dalam C++, tetapi dalam program tersebut mengandung komponen dasar yang selalu ada pada setiap pemrograman C++. Jika dilihat satu persatu :

//Program Pertama saya di C++

Baris ini adalah komentar. semua baris yang diawali dengan dua garis miring (//) akan dianggap sebagai komentar dan tidak akan berpengaruh terhadap program. Dapat digunakan oleh programmer untuk menyertakan penjelasan singkat atau observasi yang terkait dengan program tersebut.

#include <iostream.h>

Kalimat yang diawali dengan tanda (#) adalah are *preprocessor directive*. Bukan merupakan baris kode yang dieksekusi, tetapi indikasi untuk kompiler. Dalam kasus ini kalimat **#include <iostream.h>** memberitahukan preprocessor kompiler untuk menyertakan header file standard **iostream**. File spesifik ini juga termasuk library deklarasi standard I/O pada C++ dan file ini disertakan karena fungsi-fungsinya akan digunakan nanti dalam program.

int main ()

Baris ini mencocokan pada awal dari deklarasi fungsi **main**. fungsi **main** merupakan titik awal dimana seluruh program C++ akan mulai dieksekusi. Diletakan diawal, ditengah atau diakhir program, isi dari fungsi main akan selalu dieksekusi

pertama kali. Pada dasarnya, seluruh program C++ memiliki fungsi **main**.

main diikuti oleh sepasang tanda kurung **()** karena merupakan fungsi. pada C++, semua fungsi diikuti oleh sepasang tanda kurung **()** dimana, dapat berisi argumen didalamnya. Isi dari fungsi **main** selanjutnya akan mengikuti**,**berupa deklarasi formal dan dituliskan diantara kurung kurawal (**{}**), seperti dalam contoh.

cout << "Hello World";

Intruksi ini merupakan hal yang paling penting dalam program contoh. **cout** merupakan standard output stream dalam C++ (biasanya monitor). **cout** dideklarasikan dalam header file iostream.h, sehingga agar dapat digunakan maka file ini harus disertakan.

Perhatikan setiap kalimat diakhiri dengan tanda semicolon (**;**). Karakter ini menandakan akhir dari instruksi dan harus disertakan pada setiap akhir instruksi pada program C++ manapun.

return 0;

Intruksi **return** menyebabkan fungsi **main()** berakhir dan mengembalikan kode yang mengikuti instruksi tersebut, dalam kasus ini **0**. Ini merupakan cara yang paling sering digunakan untuk mengakhiri program.

Tidak semua baris pada program ini melakukan aksi. Ada baris yang hanya berisi komentar (diawali //), baris yang berisi instruksi untuk preprocessor kompiler (Yang diawali #),kemudian baris yang merupakan inisialisasi sebuah fungsi (dalam kasus ini, fungsi **main**) dan baris yang berisi instruksi (seperti, **cout** <<), baris yang terakhir ini disertakan dalam blok yang dibatasi oleh kurung kurawal (**{}**) dari fungsi **main**.

Struktur program dapat dituliskan dalam bentuk yang lain agar lebih mudah dibaca, contoh :

```
int main ()
{
 cout << " Hello World ";
 return 0;
}
```

Atau dapat juga dituliskan :

```
int main () { cout << "Hello World"; return 0; }
```

Dalam satu baris dan memiliki arti yang sama dengan program-program sebelumnya. pada C++ pembatas antar instruksi ditandai dengan semicolon (;) pada setiap akhir instruksi.

Komentar

Komentar adalah bagian dari program yang diabaikan oleh kompiler. Tidak melaksanakan aksi apapun. Mereka berguna untuk memungkinkan para programmer untuk memasukan catatan atau deskripsi tambahan mengenai program tersebut. C++ memiliki dua cara untuk menuliskan komentar :

```
//      Komentar baris
/*      Komentar Blok  */
```

Komentar baris, akan mengabaikan apapun mulai dari tanda (//) sampai akhir dari baris yang sama. Komentar Blok, akan mengabaikan apapun yang berada diantara tanda /* dan */.

File include dasar input output pada C++

- *#include <stdio.h> / #include "stdio.h"*
  Stdio = Standart Input Output
  Untuk menjalankan fungsi input dan cetak
  Input : scanf
  Cetak : printf
- *#include <iostream.h> / #include "iostream.h"*
  Iostream = Input Output Stream
  Untuk menjalankan fungsi input dan cetak
  Input : cin
  Output : cout
- *#include <conio.h> / #include "conio.h"*
  Conio = Console Input Output
  Untuk menjalankan fungsi getch dan clrscr
  Getch : untuk menahan tampilan
  Clrscr : untuk membersihkan layar

Compile File

Menterjemahkan program dari bahasa manusia kedalam bahasa yang dimengerti komputer

Klik **project – compile** atau

Alt + F9

Menjalankan File

Klik **debug – Run** atau

Ctr + F9

3.3 Elemen Dasar C++

Elemen-elemen C++ terdiri dari :

3.3.1 Himpunan Karakter

Himpunan karakter pada C++ terdiri dari huruf, digit maupun simbol-simbol lainnya (termasuk spasi, karakter kontrol).

- **Huruf**
  A B C D E F G H I J K L M N O P Q R S T U V X Y Z
  A b c d e f g h i j k l m n o p q r s t u v w x y z
- **Digit**
  0 1 2 3 4 5 6 7 8 9
- **Simbol dan Lain-lain**
  _ - + * dan sebagainya

3.3.2 Pengenal (Identifier)

Pengenal adalah suatu nama yang biasa dipakai dalam pemrograman untuk menyatakan :

- Variabel
- Fungsi
- Konstata
- Label
- Tipe data
- Obyek

Serta hal-hal lain yang dideklarasikan atau didefinisikan oleh pemrogram.

3.3.3 Penamaan Pengenal

Suatu pengenal berupa satu atau beberapa karakter :

- Huruf
- Garis bawah (_)
- Digit

Dan berawalan dengan huruf atau garis bawah. Disarankan agar pemberian nama pengenal menggunakan kata yang berarti dan mudah dibaca.

3.3.4 Huruf Kecil dan kapital Berbeda

Pada C++, huruf kecil dan huruf kapital pada suatu pengenal dianggap berbeda. sifat ini dikenal dengan istilah *case sensitive*. Sebab itu pengeanal **NAMA**, **Nama** dan **nama** menyatakan tiga pengenal yang berbeda.

3.3.5 Kata Kunci

Kata kunci (*keyword*) adalah pengenal sistim yang mempunyai makna khusus bagi kompiler. Kata kunci tidak dapat dirubah. Karena itu kata kunci tidak dapat digunakan sebagai pengenal.

3.3.6 Tipe Data

Tipe data dasar pada C++ meliputi :

- Char
- Float
- Long
- Int
- double
- short
- long double

Tipe data yang berhubungan dengan bilangan bulat adalah **char**, **int**, **short** dan **long**. Sedangkan lainya berhubungan dengan bilangan pecahan.

3.3.7 Variabel dan Konstanta

Data pada C++ tersusun dari :

- Variabel
- Konstanta

Variabel digunakan dalam program untuk menyimpan suatu nilai, nilai yang ada padanya dapat diubah selama eksekusi program berlangsung. Sementara konstanta adalah kebalikan variabel yang sudah memiliki ketetapan dan nilainya tidak dapat diubah (statis) selama eksekusi program berlangsung.

VARIABEL,TIPE DATA DAN KONSTANTA

Overview

Tipe data, variabel, dan konstanta merupakan suatu kesatuan konsep yang paling mendasar didalam pemprograman komputer, karena tipe-tipe data sangat berpengaruh dalam penggunaan suatu variabel dan kostanta dapat membentuk berbagai macam ekspresi yang akan digunakan dalam program. Apabila seorang programmer salah dalam memilih tipe data untuk sebuah variabel atau konstanta maka hasil yang akan dicapai atau output tidak sesuai dengan harapan. Oleh karena itu pemilihan tipe data sangat penting dalam suatu program,

Tujuan

- ☑ Mengenal dan membedakan tipe-tipe data dasar
- ☑ Memahami penggunaan tipe-tipe data dasar dalam program
- ☑ Memahami penggunaan Variabel dan Konstanta dalam program dan penggunaannya dalam program

Untuk dapat menulis program yang dapat membantu menjalankan tugas-tugas kita, kita harus mengenal konsep dari **variabel**. Sebagai ilustrasi, ingat 2 buah angka, angka pertama adalah 5 dan angka kedua adalah 2. Selanjutnya tambahkan 1 pada angka pertama kemudian hasilnya dikurangi angka kedua (dimana hasil akhirnya adalah 4).

Seluruh proses ini dapat diekspresikan dalam C++ dengan serangkaian instruksi sbb :

```
a = 5;
b = 2;
a = a + 1;
result = a - b;
```

Jelas ini merupakan satu contoh yang sangat sederhana karena kita hanya menggunakan 2 nilai integer yang kecil, tetapi komputer dapat menyimpan jutaan angka dalam waktu yang bersamaan dan dapat melakukan operasi matematika yang rumit.

Karena itu, kita dapat mendefinisikan variable sebagai bagian dari memory untuk menyimpan nilai yang telah ditentukan. Setiap variable memerlukan **identifier** yang dapat membedakannya dari variable yang lain, sebagai contoh dari kode diatas identifier variabelnya adalah **a**, **b** dan **result**, tetapi kita dapat membuat nama untuk variabel selama masih merupakan identifier yang benar.

4.1 Identifiers

Identifier adalah untaian satu atau lebih huruf, angka, atau garis bawah (_). Panjang dari identifier, tidak terbatas, walaupun untuk beberapa kompiler hanya 32 karakter pertama saja yang dibaca sebagai identifier (sisanya diabaikan). Identifier harus selalu diawali dengan huruf atau garis bawah (_).

Ketentuan lainnya yang harus diperhatikan dalam menentukan identifier adalah tidak boleh menggunakan **key word** dari bahasa C++. Dibawah ini adalah **key** word dalam C++ :

Asm	auto	bool	break	case
Catch	char	class	const	const_cast
continue	default	delete	do	double
dynamic_cast	else	enum	explicit	extern
False	float	for	friend	goto
If	inline	int	long	mutable
namespace	new	operator	private	protected
Public	register	reinterpret_cast	return	short
Signed	sizeof	static	static_cast	struct
Switch	template	this	throw	true
Try	typedef	typeid	typename	union
unsigned	using	virtual	void	volatile
wchar_t				

Sebagai tambahan, represetasi alternatif dari operator, tidak dapat digunakan sebagai identifier. Contoh :

and, and_eq, bitand, bitor, compl, not, not_eq, or, or_eq, xor, xor_eq

catatan: Bahasa C++ adalah bahasa yang "case sensitive", ini berarti identifier yang dituliskan dengan huruf kapital akan dianggap berbeda dengan identifier yang sama tetapi dituliskan dengan huruf kecil, sabagai contoh : variabel **RESULT** tidak sama dengan variable **result** ataupun variabel **Result**.

Tipe Data

Tipe data yang ada pada C++, berikut nilai kisaran yang dapat direpresentasikan :

Tabel 4.1. Tipe data pada bahasa C++

Tipe Data	Ukuran Memori	Jangkauan Nilai	Jumlah Digit
Char	1 Byte	-128 s.d 127	
Int	2 Byte	-32768 s.d 32767	
Short	2 Byte	-32768 s.d 32767	
Long	4 Byte	-2,147,435,648 s.d 2,147,435,647	
Float	4 Byte	3.4 x 10-38 s.d 3.4 x 10+38	5 – 7
Double	8 Byte	1.7 x 10-308 s.d 1.7 x 10+308	15 – 16
Long Double	10 Byte	3.4 x 10-4932 s.d 1.1 x 10+4932	19

4.2 Deklarasi variabel

Untuk menggunakan variabel pada C++, kita harus mendeklarasikan tipe data yang akan digunakan. Sintaks penulisan deklarasi variabel adalah dengan menuliskan tipe data yang akan digunakan diikuti dengan identifier yang benar. Variabel yang akan digunakan dalam program haruslah dideklarasikan terlebih dahulu. Pengertian deklarasi disini berarti mengenalkan sebuah pengenal ke program dan menentukan jenis data yang disimpam didalamnya.

Bentuk pendefinisian variabel :

tipe daftar_variabel

Perhatikan contoh cara mendeklarasikan variabel dalam bahasa C++

```
Int a;
Float b;
Char kata;
```

Jika akan menggunakan tipe data yang sama untuk beberapa identifier maka dapata dituliskan dengan menggunakan tanda koma, contoh

```
int a, b, c;
float d, e, f;
```

4.2.1 Menentukan Tipe Variabel

Jika variabel hendak dipakai untuk menyimpan data bilangan bulat saja, maka pilihannya adalah tipe bilangan bulat (seperti **int** , **long**). Jika variabel hendak dipakai untuk data bilangan pecahan, maka variabel harus didefinisikan bertipe bilangan pecahan (seperti **float**).

4.2.2 Memberikan Nilai ke Variabel

Bentuk pernyataan yang digunakan untuk memberikan nilai ke variabel yang telah dideklarasikanatau didefinisikan :

variabel = nilai

Pernyataan diatas sering disebut sebagai pernyataan penugasan.

4.2.3 Insialisai Variabel

Adakalanya dalam penulisan program, variabel langsung diberi nilai setelah didefinisikan. Sebagai contoh :

```
int jumlah;
jumlah = 10;
```

Dua pernyataan seperti diatas sebenarnya dapat disingkat melalui pendefinisian yang disertai penugasan nilai, sebagi berikut :

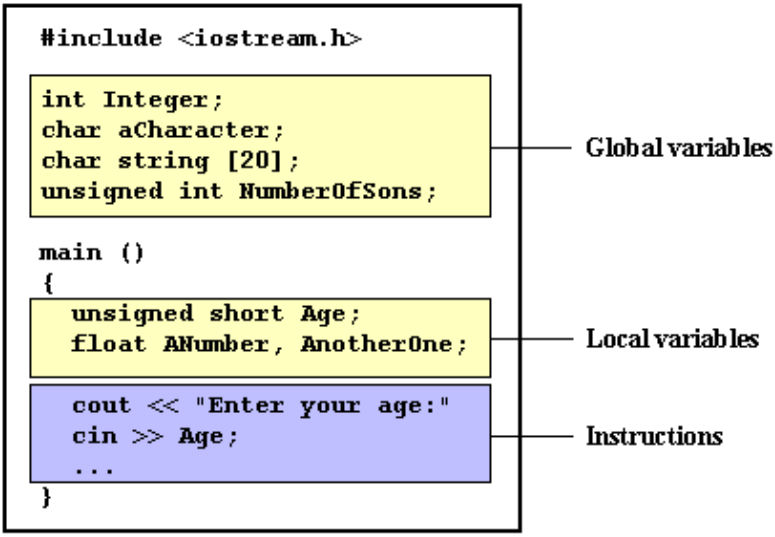
```
int jumlah = 10;
```

Contoh 2	Hasil

`// operating with variables`
`#include <iostream.h>`
 	
`int main ()`
`{`
`// declaring variables:`
`int a, b;`
`int result;`
 	
`// process:`
`a = 5;`
`b = 2;`
`a = a + 1;`
`result = a - b;`
`cout<<result;`
`return 0;`
`}`
	4

4.3 Lingkup Variabel

Pada C++, kita dapat mendeklarasikan variable dibagian mana saja dari program, bahkan diantara 2 kalimat perintah.



Gambar 4.1. Ruang lingkup bahasa C++

variabel Global dapat digunakan untuk setiap bagian dari program, maupun fungsi, walaupun dideklarasikan diakhir program. Lingkup dari **variable local** terbatas. Hanya berlaku dimana variable tersebut dideklarasikan. Jika dideklarasikan diawal fungsi (seperti dalam **main**) maka lingkup dari variable tersebut adalah untuk seluruh fungsi **main**. Seperti contoh diatas, jika terdapat fungsi lain yang ditambahkan pada main(), maka variable local yang dideklarasikan dalam **main** tidak dapat digunakan pada fungsi lainnya dan sebaliknya.

Pada C++, lingkup variable local ditandai dengan blok dimana variable tersebut dideklarasikan (blok tersebut adalah sekumpulan instruksi dalam kurung kurawal {}). Jika dideklarasikan dalam fungsi tersebut, maka akan berlaku sebagai variable dalam fungsi tersebut, jika dideklarasikan dalam sebuah perulangan, maka hanya berlaku dalam perulangan tersebut, dan seterusnya.

4.4 Deklarasi Konstanta

Konstanta adalah ekspresi dengan nilai yang tetap. Terbagi dalam Nilai Integer, Nilai Floating-Point, Karakter and String.

4.4.1 Nilai Integer

Merupakan nilai konstanta numerik yang meng-identifikasikan nilai integer decimal. Karena merupakan nilai numeric, maka tidak memerlukan tanda kutip (") maupun karakter khusus lainnya. Contoh:

```
1776
707
-273
```

C++ memungkinkan kita untuk mempergunakan nilai oktal (base 8) dan heksadesimal (base 16). Jika menggunakan octal maka harus diawali dengan karakter **0** (karakter nol), dan untuk heksadesimal diawali dengan karakter **0x** (nol, x). Contoh :

```
75          // decimal
0113        // octal
0x4b        // hexadecimal
```

Dari contoh diatas, seluruhnya merepresentasikan nilai yang sama : 75.

4.4.2 Nilai Floating Point

Merepresentasikan nilai desimal dan/atau eksponen, termasuk titik desimal dan karakter **e** (Yang merepresentasikan "dikali 10 pangkat n" , dimana n merupakan nilai integer) atau keduanya. Contoh:

```
3.14159     // 3.14159
6.02e23     // 6.02 x 10^23
1.6e-19     // 1.6 x 10^-19
3.0         // 3.0
```

4.5 Karakter dan String

Merupakan konstanta non-numerik, Contoh:

'z'
'p'
"Helloworld"
"How do you do?"

Untuk karakter tunggal dituliskan diantara kutip tunggal (') dan untuk untaian beberapa karakter, dituliskan diantara kutip ganda (").

Kita dapat mendefinisikan sendiri nama untuk konstanta yang akan kita pergunakan, dengan menggunakan preprocessor directive **#define**.

Bentuk pendefinisian konstanta :

#define *identifier value*

Atau

Const tipe *nama_variabel = nilai;*

Contoh :

```
#define PI 3.14159265
#define NEWLINE '\n'
#define WIDTH 100
```

Setelah didefinisikan seperti diatas, maka kita dapat menggunakannya pada seluruh program yang kita buat, contoh :

```
circle = 2 * PI * r;
cout << NEWLINE;
```

Pada dasarnya, yang dilakukan oleh kompiler ketika membaca **#define** adalah menggantikan literal yang ada (dalam contoh, **PI**, **NEWLINE** atau **WIDTH**) dengan nilai yang telah ditetapkan (**3.14159265**, '**\n**' dan **100**). #define bukan merupakan instruksi, oleh sebab itu tidak diakhiri dengan tanda semicolon (;).

Dengan prefix **const** kita dapat mendeklarasikan konstanta dengan tipe yang spesifik seperti yang kita inginkan. contoh:

```
const int width = 100;
const char tab = '\t';
const zip = 12440;
```

Jika tipe data tidak disebutkan, maka kompiler akan meng-asumsikan sebagai **int**.

Contoh 3	Hasil

//*----------*
//* contoh 3 : inisialisasi variabel dengan *
//* nilai konstan *
//*----------*
#include <iostream.h>
#include <conio.h>
int main()
{
int jumlah = 10; // inisialisasi
float harga_per_unit = 17.5; // inisialisasi
clrscr();
cout << "Isi Jumlah = "
<< jumlah << '\n';
cout << "Isi harga per per unit = "
<< harga_per_unit << '\n';
}
	Isi Jumlah = 10
	Isi harga per unit = 17.5

Contoh 4	**Hasil**

//*----------*
//* Contoh 4 : Contoh Inisialisasi variabel dengan *
//* suatu ungkapan *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
float kphi = 2 * 3.14; //inisialisasi dengan ungkapan
 	
clrscr();
cout << "Isi duaphi = " << kphi << '\n';
}
	Isi duaphi = 6.28

OPERATOR DAN UNGKAPAN

5

Overview

Operator merupakan simbol yang biasa dilibatkan dalam program untuk melakukan suatu operasi atau manipulasi, sedangkan ungkapan adalah dasar bagi pernyataan berkondisi (misalnya **if**). Hasil ungkapan berupa 1 kalau ungkapan bernilai benar dan ungkapan berupa 0 kalau ungkapan bernilai salah.

Tujuan

- ☑ Mengenal dan membedakan operator dan ungkapan
- ☑ Mengetahui jenis-jenis operator dan ungkapan
- ☑ Memahami operator dan penggunaannya dalam program
- ☑ Memahami ungkapan dan penggunaannya dalam program

5.1 Pengantar Operator dan Ungkapan

Operator merupakan simbol yang biasa dilibatkan dalam program untuk melakukan suatu operasi atau manipulasi. Sebagaian operator C++ tergolong sebagai operator *binary*, yaitu operator yang dikenakan terhadap dua buah nilai (*operand*).

Contoh :

a + b

simbol " + " merupakan *operand* untuk melakukan penjumlahan dari a dan b. Karena operator penjumlahan melibatkan dua *operand*, operator penjumlahan tergolong sebagai operator *binary*.

Contoh lain :

-c

simbol " – " (minus) merupakan *unary*, karena hanya memiliki sebuah *operand* (yaitu c pada contoh diatas).

Ungkapan (ekspresi) dalam C++ dapat berupa :

a. Pengenal
b. Konstanta
c. Diantara kombinasi elemen diatas dengan operator

Contoh ungkapan :

3 + 2 - 1

Pada ungkapan diatas, 3, 2 dan 1 merupakan operand dan simbol "+" serta "–" adalah operator. Nilai ungkapan sendiri adalah hasil penjumlahan 3 dan 2, dikurangi 1.

5.1.1 Operator Aritmatika

Operator untuk aritmatika yang tergolong sebagai operator *binary*. Contoh penggunan operator aritmatika misalnya untuk memperoleh nilai diskriminan dari suatu persamaan kuadrat.

$d = b^2 - 4ac$

untuk mengimplementasikan contoh diatas adalah seperti berikut:

```
d = b * b – 4 * a * c ;
```

Contoh 5	Hasil

//*----------*
//* contoh 5 : Contoh pemakaian operator *
//* Aritmatika *
//*----------*
#include <iostream.h>
#include <conio.h>
int main()
{
int a, b, c, d;
clrscr();
a = 5;
b = 600;
c = 5;
 	
d = b * b – 4 * a * c;
cout << " d = " << d << '\n';
return 0;
}
	d = 32220

Operator aritmatika mempunyai prioritas pengerjaan. Prioritas yang tinggi akan diuatamakan dalam hal pengerjaan dibandingkan dengan operator yang memiliki prioritas yang lebih rendah. Urutan prioritas dapat dilihat dalam tabel berikut ini :

Operator	Prioritas
+ -- (Khusus yang berkedudukan sebagai awalan)	Tertinggi
- (Unary Minus)	
* / %	
+ -	Terendah

Jika operator memiliki prioritas yang sama, operator sebelah kiri akan diutamakan untuk dikerjakan terlebih dahulu. Tanda kurung biasa digunakan untuk merubah urutan pengerjaan.

Misalnya : x = (2 + 3) * 2 ;

akan memberikan nilai 10 ke x, sebab 2 + 3 dikerjakan terlebih dahulu dan hasilnya baru dikalikan dengan 2.

Contoh 6	Hasil
//*----------* //* Contoh 6 : Penggunaan kurung untuk mengatur * //* prioritas pengerjaan terhadap suatu * //* operasi * //*----------* #include <iostream.h> #include <conio.h> void main() { clrscr()	X = 8 X = 12

```
 int x ;
 x = 2 + 3 * 2 ;
 cout << " x = " << x << '\n';
 x = (2 + 3) * 2 ;
 cout << " x = " << x << '\n';
return 0;
}
```

5.1.2 Operator Sisa Pembagian

Operator sisa pembagian (operator modulus) yang berupa %. Operator ini diterapkan pada operand bertipe integer. Untuk lebih jelasnya perhatikan contoh berikut :

7 % 2 → 1	Sisa pembagian bilangan 7 dengan 2 adalah 1
6 % 2 → 0	Sisa pembagian bilangan 6 dengan 2 adalah 0
8 % 3 → 2	Sisa pembagian bilangan 8 dengan 3 adalah 2

Contoh 7

```
//*-----------------------------------------------*
//* Contoh 7 : Melihat sisi pembagian dengan *
//* menggunakan operator %. *
//*-----------------------------------------------  *
#include <iostream.h>
#include <conio.h>

void main()
{
 clrscr();
 cout << 5 % 7 << '\n'; //sisa 5
 cout << 6 % 7 << '\n'; //sisa 6

 cout << 7 % 7 << '\n'; //sisa 0
 cout << 8 % 7 << '\n'; //sisa 1
 cout << 9 % 7 << '\n'; //sisa 2
return 0;
}
```

Hasil

```
5
6
0
1
2
```

Kegunaan operator % diantaranya bisa dipakai untuk menentukan suatu bilangan bulat termasuk ganjil atau genap.

5.1.3 Operator Penurunan dan Penaikan

Kedua operator ini digunakan pada operand bertipe bilangan bulat. Operator penaikan digunakan untuk menaikan nilai variabel

sebesar satu, sedangkan operator penurunan dipakai untuk menurunkan nilai variabel sebesar satu. Sebagai contoh :

```
x = x + 1 ;
y = y – 1 ;
```

bisa ditulis menjadi :

```
++ x ;
-- y ;
```

atau :

```
x ++ ;
y -- ;
```

5.1.3.1 Penaikan dibelakang

Efek peletakkan tanda ++ dibelakang variabel ditunjukkan pada program berikut :

Contoh 8	Hasil

//*----------*
//* Contoh 8 : Pemakaian operator penaikan di *
//* belakang variabel *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int r = 10;
int s;
clrscr();
s = 10 + r++ ;
cout << "R = " << r << '\n' ;
 	
cout << "S = " << s << '\n' ;
return 0;
}
	R = 11
	S = 20

Pada contoh diatas s diisi dengan penjumlahan nilai 10 dan r. Dengan demikian s akan bernilai 20. setelah s diisi dengan 20, nilai r baru dinaikan karena operator ++ ditulis dibelakang r. Disebut *post-increment* yang artinya dinaikkan dibelakang setelah penjumlahan anatara r dan 10 dilaksanakan.

5.1.3.2 Penaikan di Depan

Efek peletakkan tanda ++ di depan variabel ditunjukkan pada program berikut ini :

Contoh 9	Hasil

//*----------*
//* Contoh 9: Pemakaian operator penaikan di *
//* belakang variabel *
//*----------*
 	
#include <iostream.h>
#include <conio.h>
void main()
{
int r = 10;
int s;
clrscr();
s = 10 + ++r ;
cout << "R = " << r << '\n' ;
cout << "S = " << s << '\n' ;
return 0;
}
	R = 11
	S = 21

Pada contoh ini, nilai r mula-mula dinaikan terlebih dahulu karena operator ++ ditempatkan didepan r. Disebut *pre-increment* kemudian nilainnya dijumlahkan dengan 10 dan diberikan ke s. Dengan demikian s bernilai 21 dan r sama dengan 11.

5.1.4 Operator Majemuk

Operator majemuk digunakan untuk memendekkan penulisan operasi penugasan semacam :

```
x = x + 2 ;
y = y * 4 ;
```

menjadi :

```
x += 2;
y *= 4;
```

Contoh 10	Hasil

//*----------*
//* Contoh 10 : penggunaan operator majemuk *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int x = 2; // Mula-mula x bernilai 2
clrscr();
cout << "x = " << x << '\n' ;
x += 3 ;
cout << "Setelah x += 3, x = " << x << '\n' ;
x *= 2 ;
cout << "Setelah x *= 2, x = " << x << '\n' ;
return 0;
}
	x = 2
	Setelah x += 3, x = 5
	Setelah x += 3, x = 5

5.1.5 Operator Kondisi

Operator kondisi biasa dipakai untuk mendapatkan sebuah nilai dari dua buah kemungkinan, berdasarkan suatu kondisi. Format pemakaiannya :

ungkapan1 ? ungkapan 2 : ungkapan 3

Contoh 11	Hasil

//*----------*
//* Contoh 11 : Penggunaan operator kondisi untuk *
//* memperoleh bilangan terkecil *
//* diantara dua buah bilangan *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int bil1, bil2, minim;
clrscr();
bil1 = 53;
bil2 = 6;
minim = bil1 < bil2 ? bil1 : bil2;
cout << " Bilangan terkecil = " << minim << '\n';
 	
return 0;
}
	Bilangan terkecil =6

minim = bil1 < bil2 ? bil1 : bil2;

akan menyebabkan minim bernilai bil1 kalau ungkapan :

bil1 < bil2

bernilai benar. Untuk keadaan sebaliknya, minim akan bernilai bil2.

5.2 Ungkapan Kondisi

Ungkapan adalah ungkapan yang menjadi dasar bagi pernyataan berkondisi (misalnya **if**). Hasil ungkapan berupa 1 kalau ungkapan bernilai benar dan ungkapan berupa 0 kalau ungkapan bernilai salah.

5.2.1 Operator Relasi

Operator biasa digunakan untuk membandingkan dua buah nilai. Macam operator relasi dapat dilihat dalam tabel berikut :

Operator	Keterangan
==	Sama dengan (bukan penugasan)
!=	Tidak sama dengan
>	Lebih dari
<	Kurang dari
>=	Lebih dari atau sama dengan
<=	Kurang dari atau sama dengan

Agar tidak salah dalam menuliskan suatu ungkapan, pengetahuan tentang prioritas operator perlu diketahui.

Contoh 1 :

a = b = c

pada pernyataan diatas, operator yang dilibatkan (=) mempunyai sifat pengerjaan dimulai dari kanan. Berarti :

b = c

akan dikerjakan terlebih dahulu, barulah kemudian mengerjakan :

a = b

Contoh 2 : x = 2 * 3 * 4 ;

pada pernyataan diatas, 2 * 3 akan dikerjakan terlebih dahulu, barulah kemudian mengerjakan perkalian hasil 6 dengan 4. Adapun prioritas = lebih rendah dari *, maka 2 * 3 * 4 dikerjakan lebih dahulu. Selanjutnya hasilnya baru diberikan ke x.

5.2.2 Fungsi Pustaka

Dalam melakukan operasi seperti memeperoleh akar kuadrat ataupun memeperoleh logaritma alamiah dari suatu nilai. Pada C++ memang tidak terdapat operator-operator yang khusus untuk melaksanakan operasi-oerasi seperti itu. Tetapi tidak berarti C++ tidak dapat melakukan operasi itu. C++ menyediakan sejumlah fungsi pustaka (*library fuctions*) yang dirancang untuk memenuhi solusi dari berbagai persoalan.

Misalkan kita akan menghitung sebuah akar kuadrat, pemrogram bisa menggunakan fungsi **sqrt()**.

Jika program ingin menggunakan fungsi pustaka, perlulah untuk mencatumkan deklarasi dari fungsi bersangkutan. Untuk keperluan ini program mesti menyertakan baris :

```
#include <nama_file>
```

dengan *nama_file* adalah nama *header*, yaitu file yang berakhiran **.h**. sebagai contoh program diatas menyertakan **#include <math.h>** disebabkan file *header* tersebut berisi deklarasi (prototipe) dari fungsi **sqrt()**.

Rangkuman

- ☑ 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.
- ☑ Variabel adalah obyek yang nilainya dapat berubah-ubah dalam sebuah program, sedangkan konstanta memiliki nilai yang tetap sejak dideklarasikan hingga program berakhir.
- ☑ Pada saat mendeklarasikan sebuah variabel, pemrogram harus menyebutkan nama variabel dan tipe data dari variabel tersebut.
- ☑ 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.
- ☑ Penggunaan beberapa operator sekaligus dalam sebuah ekspresi mengakibatkan perlunya menentukan urutan operasi yang diatur dalam level urutan operator.
- ☑ 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).

OPERASI DASAR MASUKAN DAN KELUARAN

6

Overview

Dalam sebuah bahasa pemograman terdapat suatu masukan (input) dan keluaran (output). Masukan dan keluaran ini digunakan sebagai interaksi pengguna terhadap program yang akan dieksekusinya. Masukkan adalah suatu inputan yang diberikan oleh pengguna suatu program dimana masukan tersebut akan diteruskan kedalam proses dengan aturan-aturan yang disepakatin oleh bahasa pemograman dan akan ditampilkan hasil berupa keluaran (output) dari proses yang telah dilakukan.

Tujuan

- ☑ Dapat membedakan mana operasi masukan dan keluaran
- ☑ Dapat membuat program dengan operasi masukan dan keluaran
- ☑ Memahami penggunaan variabel atau konstanta dalam operasi masukan dan keluaran

6.1 Cout

Dengan cout pemrogram dapat meletakkan suatu informasi ke *standart output* (normalnya berupa layar). Sebagai contoh, pernyatan berikut akan menampilkan tulisan "Pilihan Anda salah"

cout << "Pilihan Anda salah !\a\n";

Untuk lebih jelasnya dapat memeperhatikan program dibawah ini :

Contoh 12	Hasil

//*----------*
//* Contoh 12 : Memperkenalkan cout *
//*----------*
#include <iostream.h>
void main()
{
cout << "Pilihan Anda Salah !";
return 0;
}
	Pilihan Anda Salah!

6.2 Manipulator

Manipulator umumnya digunakan untuk mengatur tampilan data. Misalnya untuk mengatur agar suatu nilai ditampilkan dengan lebar 10 karakter dan diatur rata kanan terhadap lebar tersebut.

6.2.1 Manipulator endl

Manipulator **endl** digunakan untuk menyisipkan karakter *newline*. Dengan kata lain manipulato ini identik dengan '\n'. Contoh program berikut menunjukkan penggunan endl.

Contoh 13	Hasil

//*----------*
//* Contoh 13 : Menampilkan 3 jumlah barang dan *
//* menggunakan manipulator endl *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int jumbar1 = 150;
int jumbar2 = 23;
int jumbar3 = 1401;
clrscr(); // Hapus layar
cout << "Barang 1 = " << jumbar1 << endl;
cout << "Barang 2 = " << jumbar2 << endl;
cout << "Barang 3 = " << jumbar3 << endl;
return 0; }
	Barang 1=150
	Barang 2=23
	Barang 3=1401

6.2.2 Manipulator setw()

Manipulator **setw()** bermanfaat untuk mangatur lebar dari suatu tampilan data. Untuk lebih jelasnya dapat dilihat dalam program berikut :

Contoh 14	Hasil

//*----------*
//* Contoh 14 : Menampilkan 3 jumlah barang dgn *
//* menggunakan manipulator setw() *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int jumbar1 = 150;
int jumbar2 = 23;
int jumbar3 = 1401;
clrscr(); // Hapus layar
cout << "Barang 1 = " << setw(4) <<jumbar1 << endl;
cout << "Barang 2 = " << setw(4) <<jumbar2 << endl;
cout << "Barang 3 = " << setw(4) <<jumbar3 << endl;
return 0; }
	Barang 1= 150
	Barang 2= 23
	Barang 3= 1401

Dengan menggunakan setw(), terlihat hasil eksekusi adalah rata kanan. Ini dapat di bedakan dengan hasil eksekusi pada contoh 13 yang rata kiri.

6.3 cin

Obyek **cin** bermanfaat untuk untuk membaca data dari *standart input* (normalnya adalah keyboard).

6.3.1 cin dengan sebuah variabel

Bentuk pernyataan **cin** untuk membaca data dari keyboard dan meletakkan ke sebuah variabel variabel bernama *var* :

```
cin >> var
```

Contoh program yang menunjukkan pemakaian **cin** untuk membaca data bertipe **int** dan **float**.

Contoh 15	Hasil

//*---------- *
 //* Contoh 15 : Membaca data bertipe int dan *
 //* float dari keyboard *
 //*---------- *
 #include <iostream.h>
 void main ()
 {
 int bil_x ; // Definisi bilangan bulat
 float bil_y ; // Definisi bilangan pecahan
 cout << "Masukkan sebuah bilangan bulat = " ;
 cin >> bil_x ;
 cout << "Masukkan sebuah bilangan pecahan = " ;
 cin >> bil_y ;
 	
 cout << " Bilangan Bulat = " << bil_x << endl;
 cout << " Bilangan Pecahan = " << bil_y << endl;
 	
 return 0;
 }
	Masukkan sebuah bilangan bulat = 123
	 Masukkan sebuah bilangan pecahan = 23.1
	 Bilangan Bulat = 123
	 Bilangan Pecahan 23.1

Tampak bahwa bentuk perintah untuk membaca data bertipe **int** ataupun **float** sama saja. Hal ini berlaku untuk semua jenis data dasar (**char**, **int**, **long**, **float** ataupun **double**). Salah satu manfaat dari adanya fasilitas pemasukkan dari keyboard adalah memungkinkan untuk membuat program yang membaca data yang berubah-ubah.

6.3.2 cin dengan lebih dari satu variabel

Bentuk cin dapat juga sebagai berikut :

```
cin >> var1 >> var2
```

Pada contoh ini kedua bilangan dapat bilangan dapat dimasukkan dari keyboard dengan pemisah berupa spasi, tab atau enter.

Contoh 16	Hasil

//*----------*
//* Contoh 16 : Untuk membaca data dengan lebih *
//* dari satu variabel *
//*----------*
#include <iostream.h>
void main ()
{
int bil_x ; // Definisi bilangan bulat
float bil_y ; // Definisi bilangan pecahan
cout << "Masukkan sebuah bilangan bulat dan " << endl;
cout << "Masukkan sebuah bilangan pecahan " << endl ;
cin >> bil_x >> bil_y;
 	
cout << " Bilangan Bulat = " << bil_x << endl;
cout << " Bilangan Pecahan = " << bil_y << endl;
return 0;
}
	Masukkan sebuah bilangan bulat dan
	Masukkan sebuah bilangan pecahan
	20 23.2 ↵
	Bilangan Bulat = 20
	Bilangan Pecahan 23.2

6.2.1. cin untuk membaca karakter

cin juga dapat dipakai untuk membaca sebuah karakter. Sebagai contoh :

Contoh 17	Hasil

//*----------*
//* Contoh 17 : Membaca karakter dengan cin *
//*----------*
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
void main ()
{
char karakter;
clrscr();
cout << "Masukkan sebuah karakter ";
cin >> karakter;
cout << "Anda mengetik " << karakter << endl;
cout << "Masukkan sebuah karakter ";
cin >> karakter;
cout << "Anda mengetik " << karakter << endl;
return 0;
}
	Masukkan sebuah karakter a ↵
	Anda mengetik a
	Masukkan sebuah karakter b ↵
	Anda mengetik b

Tampak pada contoh program diatas, untuk setiap karakter yang dimasukkan harus diakhiri dengan enter (↵). Penekanan enter pada pemasukkan sebuah karakter terkadang tidak dikehendaki oleh pemakai. Jika demikian, anda dapat menggunakan fungsi **getch()** ataupun **getche()**.

6.3.3 Fungsi getc() dan getche()

Fungsi **getc()** dan **getche()** berguna untuk membaca sebuah karakter tanpa perlu menekan enter. Selain itu, fungsi ini juga dapat dipakai untuk membaca tombol seperti spasi, tab ataupun enter.

Bentuk pemakaiannya :

karakter = ***getch()***
karakter = ***getche()***

Perbedaan kedua fungsi ini adalah :
a. getch() : tidak menampilkan karakter dari tombol yang ditekan
b. getche() : menampilkan karakter dari tombol yang ditekan

Contoh 18	Hasil

//*----------*
//* Contoh 18 : Membaca karakter dengan getch() *
//*----------*
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
void main()
{
char karakter;
clrscr();
cout << "Masukkan sebuah karakter ";
karakter = getch();
cout << "Anda mengetik " << karakter << endl;
cout << "Masukkan sebuah karakter ";
karakter = getch();
cout << "Anda mengetik " << karakter << endl;
return 0;
}
	Masukkan sebuah karakter a ↵
	Tidak ditampilkan
	Anda mengetik a
	Masukkan sebuah karakter b ↵
	Tidak ditampilkan
	Anda mengetik b

Sebuah **getch()** bisa pula dimanfaatkan untuk menunggu sembarang tombol ditekan. Pada keadaan seperti ini tidak perlu diletakkan ke variabel.

Contoh 19	Hasil

//*----------*
//* Contoh 19 : getch() untuk memebaca sembarang *
//* tombol *
//*----------*
#include <iostream.h >
#include <conio.h>
void main()
{
cout << "Tekanlah sembarang tombol" << endl;
cout << "Untuk mengakhiri program ini " ;
getch();
}
	Tekanlah sembarang tombol
	Untuk mengakhiri program ini

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 $2.9x10^{-39}$ hingga $1.1x10^{4932}$.
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;
    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)
```

6. Nilai akhir variabel a adalah 12.
7. Operasi e ← 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 >= 100 MOD 45 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 mod 13
   Apakah output dari ekspresi di atas?
3. Jelaskan tentang konsep runtunan dalam eksekusi sebuah program!
4. Perhatikan ekspresi berikut ini:
   x * 3 + (7 mod 5) * y
   dengan x ← 5 dan
   y ← 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?

PEMILIHAN

7

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

- ☑ Memahami struktur pemilihan dalam program
- ☑ Mengenal struktur IF dan CASE yang dapat digunakan dalam pemilihan
- ☑ Memahami konsep kondisi dan aksi dalam struktur pemilihan
- ☑ 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.

7.1 Bentuk Umum IF dan Variasinya

7.1.1 Pernyataan if

Pernyataan if dapat dipakai untuk mengambil keputusan berdasarkan suatu kondisi. Bentuk pernyataan ada dua macam :

- **if**
- **if else**

7.1.2 Pernyataan if sederhana

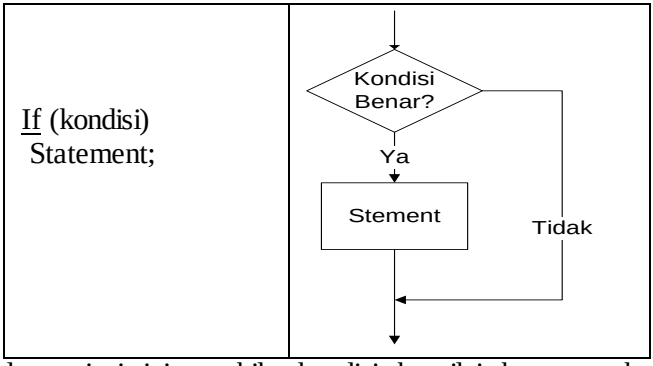
Bentuk IF yang juga dikenal dengan istilah IF Statement, memiliki bentuk umum sebagai berikut. Pernyataan **if** paling sederhana berbentuk :

```
if (kondisi)
pernyataan
```

Keterangan :

- Kondisi digunakan untuk menentukan pengambilan keputusan
- Pernyataan dapat berupa sebuah pernyataan-pernyataan majemuk.
- Nilai yang dihasilkan adalah True (Benar)

Flowchat if sederhana berikut :



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

Penerapan **if** misalnya untuk menentukan seseorang boleh atau tidak menonton pertunjukkan bioskop. Kondisi yang digunakan seseorang boleh menonton kalau sudah berusia 17 tahun.

Contoh 20	Hasil

//*----------*
//* Contoh 20 : Penggunaan if dalam pengambilan *
//* keputusan *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int usia;
clrscr(); // Hapus layar
cout << "Berapa usia anda ? ";
cin >> usia;
if (usia < 17)
cout << " Anda tidak diperkenankan menonton" <<
endl;
}
	Barapa usia anda ? 16 ↵
	Anda Tidak diperkenankan menonton

Apabila program dieksekuisi lagi untuk memasukkan usia diatas 17 maka :

Tampak diatas bila dimasukkan usia diatas 17 maka pesan tidak akan ditampilkan. Untuk mengatasi hal ini dapat dilakukan dengan menggunakan pernyataan kondisi **if else**.

Contoh 20	Hasil

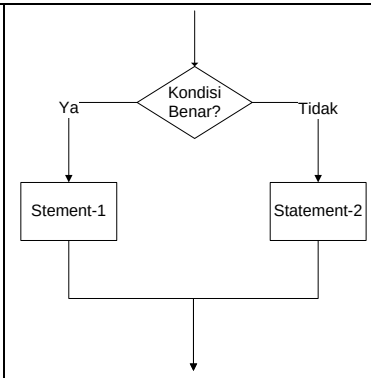
//*----------*
//* Contoh 20 : Penggunaan if dalam pengambilan *
//* keputusan *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int usia;
clrscr(); // Hapus layar
cout << "Berapa usia anda ? ";
cin >> usia;
if (usia < 17)
cout << " Anda tidak diperkenankan menonton" << endl;
}
	Barapa usia anda ? 17 ↵

7.1.3 Pernyataan if else

Pernytaan **if else** mempunyai bentuk sebagai berikut :

```
if (kondisi)
  Pernyataan 1;
else
  Pernyataan 2;
```

Flowchat if else berikut :

If (kondisi)
Statement-1;
else
Statement-2;
	

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).

Untuk **if else** kita dapat menggunakan contoh 21 untuk melihat perbedaan dengan if sederhana.

Contoh 21	Hasil

//*----------
-----* 	
//* Contoh 21 : Penggunaan if else dalam *
//* pengambilan keputusan *
//*----------
-----* 	
#include <iostream.h>
#include <conio.h>
void main()
{
int usia;
clrscr(); // Hapus layar
cout << "Berapa usia anda ? ";
cin >> usia;
if (usia < 17)
cout << " Anda tidak diperkenankan menonton" << endl;
else
cout << " Selamat menonton" << endl;
}
	Barapa usia anda ? 16 ↵
	Anda Tidak diperkenankan menonton

Apabila kita memasukkan umur lebih dari 17 maka hasil eksekusi yang didapat adalah :

Contoh 21	Hasil

//*----------*
//* Contoh 21 : Penggunaan if else dalam *
//* pengambilan keputusan *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int usia;
clrscr(); // Hapus layar
cout << "Berapa usia anda ? ";
cin >> usia;
if (usia < 17)
cout << " Anda tidak diperkenankan menonton" << endl;
else
cout << " Selamat menonton" << endl;
}
	Barapa usia anda ? 17 ↵
	Selamat menonton

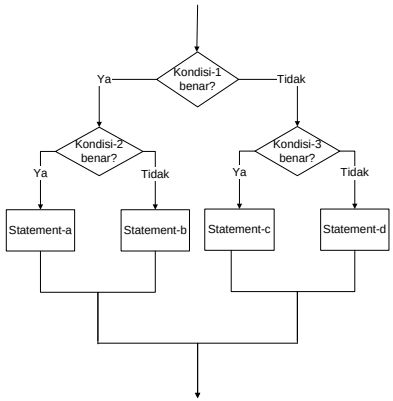
7.1.4 Pernyataan if dalam if

Pernyataan **if** yang terletak dalam **if** sering disebut *nested if* atau **if** bersarang. Salah satu bentuknya adalah :

```
if (kondisi1)
 pernyataan1;
else if (kondisi2)
 pernyataan2;
else if (kondisi3)
 pernyataan3;
if (kondisiM)
 pernyataanM;
else /*Opsional*/
 pernyataanN; /*Opsional*/
```

Bentuk pernyatan if seperti ini bermanfaat untuk menyeleksi sejumlah kemungkinan tindakan. Penyeleksian dilakukan secara bertingkat. Flowchat if dalam if berikut :

```
If (kondisi-1)
If (kondisi-2)
Statement-a
Else
Statement-b
else
if (kondisi-3)
Statement-c
Else
Statement-d
Else
...
```



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.

Contoh 22	Hasil
//*---------- ---------* //* Contoh 22 : Pemakaian if bertingkat untuk * //* menentukan nama hari * //*---------- ---------* #include <iostream.h> #include <conio.h> void main () { int kode_hari;	Menentukan hari 1 = Senin 3 = Rabu 5 = Jum'at 7 = Minggu 2 = Selasa 4 = Kamis 6 = Sabtu Kode hari [1..7] : 2 ↵ Selasa

```
clrscr();  // Hapus layar
cout << "Menentukan hari " << endl;
cout << "1 = Senin 3 = Rabu 5 = Jum'at 7
= Minggu "<< endl;
cout << "2 = Selasa 4 = Kamis 6 = Sabtu "
<< endl;
cout << "Kode hari [1..7] : " ;
cin >> kode_hari;
// Proses seleksi;
if (kode_hari = = 1)
 cout << "Senin" << endl;
if (kode_hari = = 2)
 cout << "Selasa" << endl;
if (kode_hari = = 3)
 cout << "Rabu" << endl;
if (kode_hari = = 4)
 cout << "Kamis" << endl;
if (kode_hari = = 5)
 cout << "Jum'at" << endl;
if (kode_hari = = 6)
 cout << "Sabtu" << endl;
if (kode_hari = = 7)
 cout << "Minggu" << endl;
else
 cout << "Kode hari salah" << endl;
}
}
```

Program diatas pertama-tama meminta kode hari dimasukkan dari keyboard. Kemudian **if** dan **else** secara bertingkat akan menyeleksi nilai tersebut dan memeberikan nama hari. Bila anda memasukkan kode hari yang salah maka :

Contoh 22

```
//*---------------------------------------------------
---------*
//* Contoh 22 : Pemakaian if bertingkat
untuk *
//* menentukan nama hari *
//*---------------------------------------------------
---------*
#include <iostream.h>
#include <conio.h>
void main ()
{
int kode_hari;
clrscr();  // Hapus layar
```

Hasil

```
Menentukan hari
1 = Senin 3 = Rabu 5 = Jum'at
7 = Minggu
2 = Selasa 4 = Kamis 6 =
Sabtu
Kode hari [1..7] : 9 ↵
Kode hari salah
```

```
cout << "Menentukan hari " << endl;
cout << "1 = Senin 3 = Rabu 5 = Jum'at 7
= Minggu "<< endl;
cout << "2 = Selasa 4 = Kamis 6 = Sabtu "
<< endl;
cout << "Kode hari [1..7] : " ;
cin >> kode_hari;
// Proses seleksi;
if (kode_hari = = 1)
 cout << "Senin" << endl;
if (kode_hari = = 2)
 cout << "Selasa" << endl;
if (kode_hari = = 3)
 cout << "Rabu" << endl;
if (kode_hari = = 4)
 cout << "Kamis" << endl;
if (kode_hari = = 5)
 cout << "Jum'at" << endl;
if (kode_hari = = 6)
 cout << "Sabtu" << endl;
if (kode_hari = = 7)
 cout << "Minggu" << endl;
else
 cout << "Kode hari salah" << endl;
}
}
```

7.1.5 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 7.1 : Menentukan apakah bilangan yang diinput positip atau negatip

Solusi : ada beberapa cara berikut

Solusi-1	*Solusi-2*

Input(bil)
If (bil>=0)
Output('positip')
Else
Output('negatip')
	Input(bil)
	If (bil<0)
	Output('negatip')
	Else
	Output('positip')

Solusi-3	*Solusi-4*

Input(bil) Ket ← 'positip' If (bil<0) Ket ← 'negatip' Output(Ket)	Input(bil) Ket ← 'negatip' If (bil>=0) Ket ← 'positip Output(Ket)

Solusi-5
Input(bil)
If (bil>=0)
Output('positip')
 	
If (bil<0)
Output('negatip')
	Solusi-6
	Input(bil)
	If (bil<0)
	Output('negatip')
	
	If (bil>=0)
	Output('positip')

Solusi-7
Input(bil)
positip ← bil>=0
If (positip=true)
Output('positip')
else
Output('negatip')
	Solusi-8
	Input(bil)
	positip ← bil>=0
	If (positip)
	Output('positip')
	else
	Output('negatip')

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 positip untuk mencatat hasil pembandingan bil>=0. Penulisan "if (positip=true)" sama saja dengan menuliskan "if (positip)" cara yang terakhir lebih cepat waktu eksekusinya.

Berikut beberapa kasus yang lain:
Kasus 7.2 : Terbesar dari 3 bilangan

Solusi-1	*Solusi-2*

Input(A,B,C)
If (A>B)
If (A>C)
Output('terbesar =',A)
Else
Output('terbesar =',C)
else
if (B>C)
Output('terbesar =',B)
Else
Output('terbesar =',C)
	Input(A,B,C)
	If (A<B)
	If (B<C)
	Output('terbesar =',C)
	Else
	Output('terbesar =',B)
	else
	if (A<C)
	Output('terbesar =',C)
	Else
	Output('terbesar =',A)

Solusi-3	*Solusi-4*

Input(A,B,C)
If (A>B and A>C)
Output('terbesar =',A)
Else
if (B>A and B>C)
Output('terbesar =',B)
Else
if (C>A and C>B)
Output('terbesar =',C)
	Input(A,B,C)
	If (A>B and A>C)
	Output('terbesar =',A)
	Else if (B>A and B>C)
	Output('terbesar =',B)
	Else if (C>A and C>B)
	Output('terbesar =',C)

Solusi-5	*Solusi-6*

Input(A,B,C)
Max ←A
If (B>Max)
Max ←B
End if
If (C>Max)
Max ←C
Output('terbesar = ',Max)
	Input(A,B,C)
	If (A>B)
	Max ←A
	else
	Max ←B
	If (C>Max)
	Max ←C
	Output('terbesar = ',Max)

Ulasan dari beberapa solusi:
Solusi-1,Solusi-2 dan Solusi-3 mengguanakan 3 buah kondisi dan setiap hasil yang didapat akan melalui pemeriksaan 2 buah kodisi.
Solusi-4 menggunakan kondisi yang terdiri dari 2 pembandingan, dengan rata-rata melakukan pemeriksaan 2 kondisi (4 pembandingan)
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 7.3 : 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 x 2.000 + 10.000 = 30.000
Penggunaan air 15 m3 dengan biaya :
10 x 2.000 + 5 x 3.000 + 10.000 = 45.000
Penggunaan air 75 m3 dengan biaya :
10 x 2.000 + 10 x 3.000 + 10 x 4.000 + 45 x 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)
If (pakai>30)
a← 10
b← 10
c← 10
d← pakai - 30
Else If (pakai>20)
a← 10
b← 10
c← pakai - 20
d← 0
Else If (pakai>10)
a← 10
b← pakai - 10
c← 0
d← 0
Else
a← 10
b← 0
c← 0
d← 0
biaya← a * 2000 + b * 3000 + c * 4000 + d * 5000 + 10000
Output('biaya =',biaya)
	Input(pakai)
	a←10
	b← 0
	c← 0
	d← 0
	If (pakai>30) then
	b← 10
	c← 10
	d← pakai - 30
	Else If (pakai>20)
	b← 10
	c← pakai - 20
	Else If (pakai>10) then
	b← pakai - 10
	biaya← a * 2000 + b * 3000 + c * 4000 + d * 5000 + 10000
	Output('biaya =',biaya)

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 7.4 : 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 >= 85 maka Indeksnya A
  Bila 85 > NA >= 70 maka Indeksnya B
  Bila 70 > NA >= 55 maka Indeksnya C
  Bila 55 > NA >= 40 maka Indeksnya D
  Bila NA < 40 maka Indeksnya E

Bagaimana membuat algoritma untuk menentukan Indeks Nilai tersebut?

Solusi-1	*Solusi-2*

Input(UTS,UAS,Tugas,Abs)
NAS ← 0.3*UTS + 0.4*UAS + 0.2*Tugas + 0.1*ABS
If (NAS>=85)
Indeks ← 'A'
ElseIf (NAS>=70 and NAS<85)
Indeks ← 'B'
ElseIf (NAS>=55 and NAS<70)
Indeks ← 'C'
ElseIf (NAS>=40 and NAS<55)
Indeks ← 'D'
Else
Indeks ← 'E'
Output('Indeks Nilai =',Indeks)
	Input(UTS,UAS,Tugas,Abs)
	NAS ← 0.3*UTS + 0.4*UAS + 0.2*Tugas + 0.1*ABS
	If (NAS>=85)
	Indeks ← 'A'
	Else If (NAS>=70)
	Indeks ← 'B'
	Else If (NAS>=55)
	Indeks ← 'C'
	Else If (NAS>=40)
	Indeks ← 'D'
	Else
	Indeks ← 'E'
	Output('Indeks Nilai =',Indeks)

Ulasan solusi :

Pada solusi-2 lebih baik dari solusi-1 karena pemeriksaan kondisi yang serupa tidak dilakukan dua kali. Pada solusi-1, pembandingan NAS dengan 85 dilakukan dua kali, yang pertama NAS>=85 dan yang kedua NAS<85 adalah pembandingan yang serupa.

7.1.6 Pernyataan Case

Case adalah pernyataan yang digunakan untuk menjalankan salah satu pernyataan dari beberapa kemungkinan pernyataan, berdasarkan nilai dari sebuah ungkapan dan nilai penyeleksi. 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 pembandingan 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 :

```
switch (ungkapan)
{
 case ungkapan1;
 pernyataan_1;
 break;
 case ungkapan2;
 pernyataan_2;
 break;
 …….
 default : /*Opsinal*/
 pernyataan_x; /*Opsinal*/
}
```

Pada pernyataan switch, *ungkapan* dapat berupa ungkapan, konstanta ataupun variabel. Adapun *ungkapan1*, *ungkapan2* dan seterusnya dapat berupa sembarang konstanta bertipe **int** atau **char**.

Contoh 23	Hasil
//*---------- -----* //* Contoh 23 : Pemakaian switch untuk menentukan * //* nama hari * //*---------- -----* #include <iostream.h> #include <conio.h> void main() { int kode_hari; clrscr(); // Hapus layar cout << "Menentukan Hari"" << endl; cout << "1 = Senin 3 = Rabu 5 = Jum'at 7 = Minggu "<< endl; cout << "2 = Selasa 4 = Kamis 6 = Sabtu " << endl; cout << "Kode hari [1..7] : " ; cin >> kode_hari; // Proses seleksi dengan switch switch (kode_hari) { case 1: cout << "Senin" << endl; break; case 2 : cout << "Selasa" << endl; break; case 3 : cout << "Rabu" << endl; break; case 4 : cout << "Kamis" << endl; break; case 5 : cout << "Jum'at" << endl; break; case 6 : cout << "Sabtu" << endl; break; case 7 : cout << "Minggu" << endl; break; default : cout << "Kode hari salah" << endl; break; } // akhir switch }	Menentukan hari 1 = Senin 3 = Rabu 5 = Jum'at 7 = Minggu 2 = Selasa 4 = Kamis 6 = Sabtu Kode hari [1..7] : 2 ↵ Selasa

Kehadiran **break** pada setiap **case** sangat penting. Sebab **break** akan menyebabkan keluar dari **switch**.

Program diatas pertama-tama meminta kode hari dimasukkan dari keyboard. Kemudian **if** dan **else** secara bertingkat akan menyeleksi nilai tersebut dan memeberikan nama hari. Bila anda memasukkan kode hari yang salah maka :

Contoh 23	Hasil

//*----------*
//* Contoh 23 : Pemakaian switch untuk menentukan *
//* nama hari *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int kode_hari;
clrscr(); // Hapus layar
cout << "Menentukan Hari"" << endl;
cout << "1 = Senin 3 = Rabu 5 = Jum'at 7 = Minggu "<< endl;
cout << "2 = Selasa 4 = Kamis 6 = Sabtu " << endl;
cout << "Kode hari [1..7] : " ;
cin >> kode_hari;
// Proses seleksi dengan switch
switch (kode_hari)
{
case 1:
cout << "Senin" << endl;
break;
case 2 :
cout << "Selasa" << endl;
break;
case 3 :
cout << "Rabu" << endl;
break;
case 4 :
cout << "Kamis" << endl;
break;
case 5 :
cout << "Jum'at" << endl;
break;
case 6 :
cout << "Sabtu" << endl;
break;
case 7 :
cout << "Minggu" << endl;
	Menentukan hari
	1 = Senin 3 = Rabu 5 = Jum'at 7 = Minggu
	2 = Selasa 4 = Kamis 6 = Sabtu
	Kode hari [1..7] : 9 ↵
	Kode hari salah

```
 break;
default :
 cout << "Kode hari salah" << endl;
 break;
} // akhir switch
}
```

7.1.7 Terapan bentuk-bentuk CASE

Kasus 7.5 : 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) If (NoHari=1) then NmHari ← 'Senin' Else If (NoHari=2) NmHari ← 'Selasa' Else If (NoHari=3) NmHari ← 'Rabu' Else If (NoHari=4) NmHari ← 'Kamis' Else If (NoHari=5) NmHari ← 'Jumat' Else If (NoHari=6) NmHari ← 'Sabtu' Else If (NoHari=7) then NmHari ← 'Minggu' Output(NmHari)	Input(NoHari) Case NoHari 1: NmHari ← 'Senin';Break; 2: NmHari ← 'Selasa';Break; 3: NmHari ← 'Rabu';Break; 4: NmHari ← 'Kamis';Break; 5: NmHari ← 'Jumat';Break; 6: NmHari ← 'Sabtu';Break; 7: NmHari ← 'Minggu';Break; Output(NmHari)

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

Kasus 7.6 : Merubah angka menjadi kalimat
Dinput bilangan/angka (angka dibatasi 1-99), ditampilkan kata-kata/kalimat dari bilangan tersebut, bagaimana algoritmanya?
Solusi

Solusi-Case

```
Input(bil)
pul ← bil div 10
sat ← bil mod 10
Kalimat ← ‘’
Case 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’
Case pul
1: Case sat
0: Kalimat ←‘Sepuluh’
1: Kalimat ←‘Sebelas’
Otherwise: Kalimat ← Kalimat + ‘ belas’
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
Output(Kalimat)
```

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

7.2 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
Aksi3
End if
	if (kondisi1) {
	Aksi1;
	}
	else if (kondisi2){
	Aksi2;
	}
	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;

	}

Case ekspresi
Nilai-1,Nilai-2,Nilai-3: Aksi1
Nilai-4,Nilai-5: Aksi2
Nilai-6..Nilai-8: Aksi3
Otherwise: Aksi4
End Case
	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 7.3

Algoritma	Bahasa C

Algoritma PDAM
/* menghitung biaya pemakaian air*/
Kamus Data
pakai,a,b,c,d : integer
biaya : integer
Begin
Input(pakai)
a← 10
b← 0
c← 0
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 +
5000
Output('biaya =',biaya)
	#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;
	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
	}

Rangkuman

- ☑ Struktur pemilihan dapat digunakan untuk membuat program melakukan aksi tertentu sesuai nilai dari kondisi yang dipertimbangkan.
- ☑ Struktur pemilihan dapat berupa struktur IF ... THEN ..., struktur IF ... THEN ... ELSE ..., struktur CASE, maupun pemilihan bersarang IF atau CASE.
- ☑ Struktur IF ... THEN ... (tanpa ELSE) digunakan pada situasi dengan pilihan mengerjakan aksi atau tidak.
- ☑ Struktur IF ... THEN ... ELSE ... digunakan untuk memilih salah satu aksi dari berdasarkan nilai kondisi.
- ☑ Struktur CASE merupakan bentuk penyederhanaan dari struktur IF dengan persyaratan tertentu, yaitu kondisi berupa pembandingan kesamaan dan nilai yang dibandingkan harus ordinal (integer,char atau boolean).
- ☑ Klausa OTHERWISE pada struktur CASE bersifat opsional, seperti halnya ELSE pada struktur CASE.
- ☑ Struktur pemilihan bersarang dan struktur CASE dapat menyederhanakan program yang menyelesaikan kasus dengan nilai kondisi berjumlah lebih dari dua.

Kuis Benar Salah

1. Klausa ELSE dalam struktur pemilihan IF... ELSE... berarti ‘jika tidak’.
2. Klausa THEN dalam struktur pemilihan IF... 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... ELSE..., jika kondisi yang disebutkan setelah klausa IF bernilai benar maka program akan menjalankan aksi setelah klausa ELSE.

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

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

Jika inputan nilai = 60, maka outputnya adalah 'Nilai mutu = B'.

7. Jika inputan nilai = 101, maka outputnya adalah 'Nilai mutu = E'
8. 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')
```

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

```
Input (nilai)
IF nilai = 'A'
    Output ('Sangat memuaskan')
ELSE
    IF nilai = 'B'
        Output ('Memuaskan')
    ELSE
        IF nilai = 'C'
            Output ('Cukup')
        ELSE
            IF nilai = 'D'
                Output ('Kurang')
            ELSE
                IF nilai = 'E'
                    Output ('Sangat kurang')
                ELSE
                    Output ('Inputkan A,B,C,D atau E')
```

Pada struktur pemilihan IF... THEN..., pemrogram tidak perlu menentukan parameter kondisi yang digunakan.

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

    Input (x)
    IF *x mod 2 = 0*
        Output ('sisa bagi adalah nol')
    ELSE
        Output ('sisa bagi bukan nol')

    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
   B. alternatif aksi dr nilai kondisi
   C. aksi
   D. reaksi
   E. parameter kondisi

2. Perhatikan potongan algoritma berikut ini:
   IF *badan belum bersih*
   teruskan mandi
   ELSE
   mandi selesai
   Menurut potongan algoritma tersebut, mandi selesai jika … .
   A. tangan capek
   B. sudah terlalu lama
   C. badan sudah bersih
   D. mandi tidak diteruskan
   E. pilihan A,B,C,dan D salah

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 mod 2 >= 0*
   Output ('x adalah bilangan genap')
   ELSE
   IF *x mod 2 < 0*

Output ('x adalah bilangan ganjil')

b. *Input (x)*
IF *x div 2 >= 0*
Output ('x adalah bilangan genap')
ELSE
IF *x div 2 < 0*
Output ('x adalah bilangan ganjil')

c. *Input (x)*
IF *x mod 2 = 0*
Output ('x adalah bilangan genap')
ELSE
Output ('x adalah bilangan ganjil')

d. *Input (x)*
IF *x div 2 = 0* THEN
Output ('x adalah bilangan genap')
ELSE
Output ('x adalah bilangan ganjil')

e. Pilihan A,B,C,D salah

5. Berikut ini adalah contoh kondisi yang tidak dapat digunakan dalam pemilihan, yaitu … .
   A. angka1 > angka2
   B. gaji = 1000000
   C. baju = baru
   D. a * b <= 0
   E. Pilihan A,B,C,D salah

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*
    Beli

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… 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*
    Output (p/2)
ELSE
    *Output (p*2)*

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
```

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

A. 'Pengajar = Ana'
B. 'Pengajar = Ani'
C. 'Pengajar = Ina'
D. Program error
E. Pilihan A,B,C dan D salah

Latihan

1. Buatlah minimal 5 program dengan menggunakan if dan case dengan aturan:
    a. If Sederhana
    b. If Jamak (if..else)
    c. If Bersarang
    d. Case
    e. Case Bersarang
2. Setelah program dibuat, konversikan program tersebut dari If ke Case dan sebaliknya
3.

```
#include "conio.h"
#include "stdio.h"
#include "iostream.h"
#include "stdlib.h"


void main(){

int d,m,zodiak;
char a;

lagi:
clrscr();
cout<<"-----------------------------------------"<<endl;
cout<<"---------.:  Program Zodiak :.----------"<<endl;
cout<<" Sekedar Hiburan jangan percaya.. musyrik "<<endl;
cout<<"-----------------------------------------"<<endl;

cout<<"Masukan Tanggal Lahir Anda? :";
cin>>d;

cout<<"Masukan Bulan Lahir Anda? :";
cin>>m;

//filter zodiak berdasarkan tanggal dan bulan lahir lahir
if(((d>29 && m==2) || (d>31 && m==1) || (d>31 && m==1) || (d>31 && m==3) ||
(d>30 && m==4) || (d>31 && m==5) || (d>30 && m==6) || (d>31 && m==7) || (d>31
&& m==8) || (d>30 && m==9) || (d>31 && m==10) || (d>30 && m==11) || (d>31 &&
m==12) )){
      zodiak =0;
```

```
}else if((d>=21 && m==3) || (d <= 19 && m==4)){
        zodiak = 1;
}else if(((d>=20 && m==4) || (d <=20 && m==5))){
        zodiak = 2;
}else if(((d>=21 && m==5) || (d<=20 && m==6))){
        zodiak = 3;
}else if(((d>=21 && m==6) || (d<=22 && m==7))){
        zodiak = 4;
}else if(((d>=23 && m==7) || (d<=22 && m==8))){
        zodiak = 5;
}else if(((d>=23 && m==8) || (d<=22 && m==9))){
        zodiak = 6;
}else if(((d>=23 && m==9) || (d<=22 && m==10))){
        zodiak = 7;
}else if(((d>=23 && m==10) || (d<=21 && m==11))){
        zodiak = 8;
}else if(((d>=22 && m==11) || (d<=21 && m==12))){
        zodiak = 9;
}else if(((d>=22 && m==12) || (d<=19 && m==1))){
        zodiak = 10;
}else if(((d>=20 && m==1) || (d<=18 && m==2))){
        zodiak = 11;
}else if(((d>=19 && m==2) || (d<=20 && m==3))){
        zodiak = 12;
}else{
        zodiak = 0;
}

//tampilkan ramalan berdasarkan filter zodiak
clrscr();
switch(zodiak){
case 1:
        printf(".:: Zodiak Anda Aries ::. \n \n");
        printf("Kesehatan       : \n Kurangi makan daging, karena anda gk mampu beli
\n\n");
        printf("Keuangan : \n Seret kaya becak kempes \n\n");
        printf("Karir : \n genjot terus becak anda sampai ujung dunia \n\n");
        printf("Asmara : \n Istri orang memang menggoda, tahan diri anda masih \n banyak
janda di luar sana \n\n");
        goto codea;
 //break;
case 2:
        printf(".:: Zodiak Anda Taurus ::. \n \n");
        printf("Kesehatan       : \n Kurangi makanan yang berlemak terutama lemak babi,
haram\n\n");
        printf("Keuangan : \n Anda akan mendapatkan rejeki tidak nomplok\n\n");
        printf("Karir : \n Anda sebaiknya tidak bekerja di dalam air, tidak cocok\n\n");
        printf("Asmara : \n Mantan anda susah menikah, trus anda belum... kasian\n\n");
        goto codea;
 //     break;
case 3:
        printf(".:: Zodiak Anda Gemini ::. \n \n");
        printf("Kesehatan       : \n Minumlah vitamin terutama vitamin otak biar tidak
pikun\n\n");
        printf("Keuangan : \n Uang anda habis buat beli vitamin\n\n");
        printf("Karir : \n Anda melupakan karir anda\n\n");
```

```
        printf("Asmara : \n Kekasih anda melupakan anda, dan pergi bersama bang
toyib\n\n");
        goto codea;
 //     break;
case 4:
        printf(".:: Zodiak Anda Cancer ::. \n \n");
        printf("Kesehatan       : \n Jaga Kesehatan Anda, selalu pakai jaket di malam hari.\n
jangan lupa minum tolak angin, tolak miskin dan tolak mantan \n\n");
        printf("Keuangan : \n Sepertinya Mulai seret maka mintalah kepada yang punya,\n
duduk lah di trotoar dan tadahkan tangan anda \n\n");
        printf("Karir : \n Nikmati saja, suatu saat anda akan Melejit seperti roket\n ke
angkasa..! \n\n");
        printf("Asmara : \n Jika sudah nyaman kenapa gk dicoba, dekati terus duda itu\n
siapa tau dia tertarik pada anda. yang lalu biarlah berlalu.. \n Move On Coy..!! \n\n");
        goto codea;
 //     break;
case 5:
        printf(".:: Zodiak Anda Leo ::. \n \n");
        printf("Kesehatan       :\n Sesuai dengan zodiak anda waspada dengan penyakit
singa \n\n");
        printf("Keuangan :\n Sebaiknya berhemat karena dompet anda sudah kering \n\n");
        printf("Karir :\n Jadi pawang singa cocok untuk anda \n\n");
        printf("Asmara :\n Kekasih anda sedang Galak Seperti Singa \n\n ");

        goto codea;

 //     break;
case 6:
        printf(".:: Zodiak Anda Virgo ::. \n\n");
        printf("Kesehatan       : \nJangn terlalu sering ke rumah sakit, apalagi cuman
ngeceng suster disana \n\n");
        printf("Keuangan : \nNebeng itu tidak dosa, selama yang di tebeng mau \n\n");
        printf("Karir : \nAnda tidak cocok kerja di bulan \n\n");
        printf("Asmara : \nSeperti Punguk merindukan bulan \n\n");
        goto codea;
 //     break;
case 7:
        printf(".:: Zodiak Anda Libra ::. \n \n");
        printf("Kesehatan       : \nJangan terlalu bnyak begadang, karena dilarang Bang Haji
Roma Irama \n\n");
        printf("Keuangan : \nMasih seret aja... Ngamen gih sana \n\n");
        printf("Karir : \nKarir anda stabil, trus berusaha lagi \n\n");
        printf("Asmara : \nAnda pasti sedang G.A.L.A.U , cari kesibukan saja dengan istri
orang \n\n");
        goto codea ;
 //     break;
case 8:
        printf(".:: Zodiak Anda Scorpio ::. \n \n");
        printf("Kesehatan       :\nHindari racun kalajengking siapa tau anda berubah jadi
scorpio king\n\n ");
        printf("Keuangan :\nAman terkendali, masih bisa makan di warung padang\n\n ");
        printf("Karir :\nSetelah makan jangan lupa cuci piringnya\n\n ");
        printf("Asmara :\nUni sepertinya suka anda (sebagai karyawan) ");
        goto codea;
 //     break;
case 9:
```

```
        printf(".:: Zodiak Anda Sagitarius ::. \n \n");
        printf("Kesehatan        :\nCiyus mau tau ?\n\n");
        printf("Keuangan :\nMiapah ?\n\n");
        printf("Karir :\nEnelan ? \n\n");
        printf("Asmara :\nCungguh ?\n\n");
        goto codea;
//      break;
case 10:
        printf(".:: Zodiak Anda Capricons ::. \n \n");
        printf("Kesehatan        :\n Baygon tidak baik untuk kesehatan, kurangi meminumnya
\n\n");
        printf("Keuangan :\n Harta warisan anda akan segera habis, segera isi ulang  \n\n");
        printf("Karir :\n Karir anda menanjak naik-naik ke puncak gunung... \n\n");
        printf("Asmara :\n Pakai HIT jika anda Galau ");
        goto codea;
 //     break;
case 11:
        printf(".:: Zodiak Anda Aquarius ::. \n \n");
        printf("Kesehatan        :\n Batuk pak Haji....\n\n");
        printf("Keuangan :\n Siap2 mendapatkan bonus \n\n");
        printf("Karir :\n Bisnis isi ulang air kemasan tampak menjanjikan \n\n");
        printf("Asmara :\n Janda sebelah rumah anda memperhatikan anda.\n\n");
        goto codea;
 //     break;
case 12:
        printf(".:: Zodiak Anda Pisces ::. \n \n");
        printf("Kesehatan        : \n Kurangi makanan laut terutama ikan, tau kenapa ?\n\n");
        printf("Keuangan : \n rejeki yang terduka tidak akan datang\n\n");
        printf("Karir : \n masih betah mengngangur ?\n\n");
        printf("Asmara : \n masih betah menjomblo ?\n\n");
        goto codea;
//      break;
default:
        printf(" Error \nSomething wrong..!! \n\n Gunakan Tanggal Yang Valid \n\n");
        goto codea;
 //     break;
}

codea:
//ulangi  aplikasi
        cout<<"Created Agus Perdana Windarto, M.Kom"<<endl;
        cout<<"Mau Ulangi Lagi...(Y/T)? : ";
        cin>>a;
        if(a == 'y' || a == 'Y')
        {
        goto lagi;
        }
        else{
        //return(0);
         exit(0);
        }
}
```

Apakah output dari program di atas? Asumsikan data inputan bebas

```
#include <iostream.h>
#include <iomanip.h>
#include <string.h>


main (int)
{
            char nama[35];
            int tgLahir,  bLahir, tahunLahir;
 int tgSekarang, bSekarang, tahunSekarang;
 int uBulan=0, uTahun=0, uHari=0;

            cout<<"-----------------------------------"<<endl;
            cout<<" Isikan Sesuai dengan Petunjuk "<<endl;
            cout<<" Program Hitung Usia "<<endl;
            cout<<"-----------------------------------"<<endl;

            cout << "Nama Anda : ";
            cin>> nama;
            //Memberikan data kelahiran
            cout <<"Tanggal Kelahiran         [1-31] : ";
            cin >> tgLahir;

            cout <<"Bulan Lahir [1-12]                    : ";
            cin >> bLahir;

            cout <<"Tahun Kelahiran                       : ";
            cin >> tahunLahir;
 cout<<endl;



            //Memberikan data sekarang
            cout <<"Tanggal Sekarang [1-31]: ";
            cin >> tgSekarang;

            cout <<"Bulan Sekarang [1-12] : ";
            cin >> bSekarang;

            cout <<"Tahun Sekarang                        : ";
            cin >> tahunSekarang;

            //Membandingkan data
            if(bLahir  > bSekarang && tgLahir > tgSekarang)
            {
                        uHari = tgLahir-tgSekarang;
                        uBulan = 12-(bLahir-bSekarang);
                        uTahun = (tahunSekarang-tahunLahir)-1;
            }

            else if(bLahir  > bSekarang && tgLahir < tgSekarang)
            {
                        uHari = tgSekarang-tgLahir;
                        uBulan = 12-(bLahir-bSekarang);
                        uTahun = (tahunSekarang-tahunLahir)-1;
            }
```

```
        if(bLahir  < bSekarang && tgLahir > tgSekarang)
        {
                uHari = tgLahir-tgSekarang;
                uBulan = bSekarang-bLahir;
                uTahun = tahunSekarang-tahunLahir;
        }

        else if(bLahir  < bSekarang && tgLahir < tgSekarang)
        {
                uHari = tgSekarang-tgLahir;
                uBulan = bSekarang-bLahir;
                uTahun = tahunSekarang-tahunLahir;
        }

        //Menampilkan usia

        cout<<"-----------------------------------"<<endl;
        cout<<" Hasil Perhitungan Komputer "<<endl;
        cout<<" Program Hitung Usia "<<endl;
        cout<<"-----------------------------------"<<endl;
        cout<<"Hai " << nama;
        cout<<"!....."<<endl;
        cout<<"Usia Anda Sekarang : "<< uTahun;
        cout <<" tahun "<< uBulan << " bulan " << uHari<< " hari" <<endl;

        return 0;
}
```

Apakah output dari program di atas? Asumsikan data inputan bebas

PENGULANGAN (LOOPING)

8

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* memeiliki keunikan tersendiri. Di dalam bab ini akan dijelaskan tentang struktur pengulangan dalam algoritma serta implementasi struktur perulangan dengan perintah *loop* yang ada di dalam C.

Tujuan

- ☑ Memahami konsep dasar dan struktur pengulangan
- ☑ Memahami perintah pengulangan dalam C
- ☑ Menerapkan sintaks-sintaks pengulangan dalam menyelesaikan persoalan

8.1 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

8.1.1 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:

```
while (ungkapan);
 {
 pernyataan1;
 pernyataan2;
 ……
 pernyataanN;
 }
```

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
{inisialisasi jumlah kentang yang sudah dikupas} 1 kentang ← 0 2 WHILE (kentang < 10){kondisi iterasi dilakukan} 3 Ambil sebuah kentang 4 Kupas kulit kentang 5 kentang←kentang+1 {pencapaian kondisi berhenti}

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 (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 (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:

- Algoritma untuk menampilkan karakter ‘ * ‘ sebanyak 5 kali.

```
ALGORITMA Tampil_Bintang
IS : -
FS : Jumlah bintang yg tampil = 5
KAMUS DATA
 i : integer
1       i ← 0 {inisialisasi pencacah i}
2       WHILE (i < 5) {jumlah iterasi}
3           Output (‘*’) {aksi}
4           i ← i + 1 {pencapaian kondisi berhenti}
```

Output dari algoritma ini:

```
*
*
*
*
*
```

↳ 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         i ← 1 {inisialisasi pencacah i}
2         WHILE (i <= 5)
3             Output (‘Ini adalah iterasi ke-’,i)
4             i ← i + 1
5
```

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
```

- 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       i ← 1
2       WHILE (i <= 3)
3         Output ('Nilai mahasiswa ke-',i,'adalah:')
4         Input (nilai)
5         i ← i + 1
```

Output dari algoritma ini:

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

- 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     jumlah ← 0        {variabel jumlah bilangan}
2     bilangan ← 3      {inisialisasi variabel pencacah}
3     WHILE (bilangan > 0)
4      Output('Masukkan angka : ')
5        Input (x)      {masukkan sebuah bilangan}
6        jumlah ← jumlah + x {tambahkan bilangan}
7        bilangan← bilangan – 1 {pencacah mundur}
```

8	rerata ← jumlah/ 3 {menghitung rerata}
9	Output (‘Rerata : ‘,rerata)

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

WHILE (*kondisi*)
aksi
ubah pencacah
	while (kondisi)
	{
	//aksi
	//ubah pencacah
	}

...
i ← 0
WHILE (i < 5)
Output(‘*’)
i ← i + 1
...
	...
	i = 0;
	while (i < 5)
	{
	printf(“*”);
	i = i + 1;
	}
	...

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 tebakan salah, dan menampilkan “Anda Benar” jika tebakan benar
KAMUS DATA huruf : character
1 {inisialisasi huruf tebakan pertama } 2 Output(‘Masukkan tebakan : ‘) 3 Input(huruf) 4 WHILE (huruf != ‘q’) {pengecekan kondisi} 5 Output(‘Maaf Anda salah ‘) 6 Output(‘Masukkan tebakan : ‘) 7 Input(huruf) {input huruf berikutnya} 8 Output (‘Anda Benar’)

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  }
```

Bagian pernyataan yang mengikuti **while** akan dieksekusi selama *ungkapan* pada **while** bernilai benar (tidak sama dengan nol). Pengujian terhadap ungkapan **while** dilakukan sebelum bagian pernyataan. Perhatikan contoh-contoh berikut :

Contoh 24	Hasil

//*---------- Contoh 24 : pemakaian while untuk menampilkan tulisan C++ sebanyak 10 kali ----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int i; // Sebagai variabel pencacah yang menyatakan
// jumlah tulisan C++ yang harus ditampilkan
clrscr(); // Hapus layar
i = 0; // Mula-mula diisi sama dengan nol
while (i < 10)
{
cout << " C++ " << endl;
i ++ ; // Menaikkan pencacah sebesar 1
}
}
	C++
	C++
	C++
	C++
	C++
	C++
	C++
	C++
	C++
	C++

Pada program diatas, variabel i bertindak sebagai pencacah yang gunanya untuk mengingat jumlah tulisa C++ yang telah ditampilkan. Itulah sebabnya mula-mula didisi dengan nol. Kemudian untuk setiap putaran, isi variabel ini dinaikkan. Oleh karena variabel i dijadikan sebagai kondisi pada **while**, suatu ketika ketika kondisi i<10 akan bernilai salah, maka **while** berakhir.

8.1.2 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 perulangan 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:

```
do
{
 pernyataan1;
 pernyataan2;
 ….
 pernyataanN;
} while (ungkapan)
```

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. Dengan kata lain Bagian *pernyataan1* hingga *pernyataanN* dijalankan secara berulang sampai *ungkapan* bernilai salah (sama dengan nol). Namun berbeda

dengan **while**, pengujian *ungkapan* dilakukan dibelakang (setelah bagian *peryataan*).

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

↳ Algoritma ibu mengupas kentang

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

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.

- 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       i ← 0 {inisialisasi pencacah i}
2       DO
3           Output ('*') {aksi}
4           i ← i + 1 {pencapaian kondisi berhenti}
5       WHILE (i < 5) {jumlah iterasi}
6
```

Output dari algoritma ini:

```
*
*
*
*
*
```

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

- 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     i ← 1 {inisialisasi pencacah i}
2     DO
3         Output ('Ini adalah iterasi ke-',i){aksi}
4         i ← i + 1
5     WHILE (i <= 5) {kondisi berhenti}
```

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.

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

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<= 3 salah.

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

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 tebakan salah, dan menampilkan “Anda Benar” jika tebakan benar

KAMUS DATA
huruf : character

```
1       Output(‘Masukkan tebakan : ‘)
2    Input(huruf)
3      WHILE (huruf != ‘q’)
4       Output(‘Maaf Anda salah ‘)
5           Output(‘Masukkan tebakan : ‘)
6           Input(huruf)
7      Output (‘Anda Benar’)
```

DO...WHILE

ALGORITMA Tebak_Huruf

IS : -
FS : Menampilkan pesan “Maaf Anda salah” jika tebakan salah, dan menampilkan “Anda Benar” jika tebakan benar

```
KAMUS DATA
huruf : character
```

```
1       Output(‘Masukkan tebakan : ‘)
2     Input(huruf)
3       DO
4        Output(‘Maaf Anda salah ‘)
5            Output(‘Masukkan tebakan : ‘)
6            Input(huruf)
7       WHILE (huruf != ‘q’)
8       Output (‘Anda Benar’)
```

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       DO
2            Output(‘MENU : ‘)
```

```
3           Output(‘1. Ulang’)
4           Output(‘2. Keluar’)
5           Output(‘Pilihan : ‘)
6           Input(pilihan)
7       WHILE (pilihan != 2) {pengecekan kondisi}
8       Output (‘Anda Pilih Keluar’)
```

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
```

Bagian pernyataan yang mengikuti **while** akan dieksekusi selama *ungkapan* pada **while** bernilai salah (tidak sama dengan nol). Pengujian terhadap ungkapan **while** dilakukan sebelum bagian pernyataan. Perhatikan contoh-contoh berikut :

Contoh 25	Hasil

//*----------*
//* Contoh 25 : pemakaian do-while untuk *
//* menampilkan tulisan C++ *
//* sebanyak 10 kali *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int i; // Sebagai variabel pencacah yang menyatakan
// jumlah tulisan C++ yang harus ditampilkan
clrscr(); // Hapus layar
i = 0; // Mula-mula diisi sama dengan nol
do
{
cout << " C++ " << endl;
i ++ ; // Menaikkan pencacah sebesar 1
} while (i < 10);
}
	C++
	C++
	C++
	C++
	C++
	C++
	C++
	C++
	C++
	C++

Pada program diatas, variabel i bertindak sebagai pencacah yang gunanya untuk mengingat jumlah tulisa C++ yang telah ditampilkan. Itulah sebabnya mula-mula didisi dengan nol. Kemudian untuk setiap putaran, isi variabel ini dinaikkan. Oleh karena variabel i dijadikan sebagai kondisi pada **while**, suatu ketika

ketika kondisi i<10 akan bernilai benar, maka **while** berakhir. Dengan kata lain do..while kebalikan dari while. Dimana do..while akan mengulang saat kondisi bernilai salah, dan akan berhenti jika kondisi bernilai benar.

8.1.3 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 kentang>=1 dan kentang<=10 terpenuhi, aksi pengulangan akan dilakukan.

Pernyataan **for** berguna untuk menggulang pengeksekusian terhadap satu atau sejumlah pernyataan. Bentuk format :

```
for (ungkapan1; ungkapan2; ungkapan3)
  pernyataan;
```

Dimana :

- Ungkapan1 : untuk memberikan nilai awal untuk variabel pencacah.
- Ungkapan2 : kondisi pengulangan akan berhenti atau tidak.

- Ungkapan3 : 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 :

- Algoritma ibu mengupas kentang

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

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.

↳ 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       FOR(i ← 0; i < 5; i++)
2           Output ('*') {aksi}
```

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.

↳ 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    FOR(i ← 1; i <= 5;i++)
2        Output ('Ini adalah iterasi ke-',i){aksi}
```

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<=5 bernilai salah.

- 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    FOR(i ← 1; i <= 3;i++)
2        Output ('Nilai mahasiswa ke-',i,'adalah:')
3        Input (nilai)
```

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<=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 :

- 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 jumlah ← 0 {variabel jumlah bilangan} 2 {inisialisasi variabel pencacah} 3 FOR(bilangan ← 3; bilangan > 0;bilangan--) 4 Output('Masukkan angka : ') 5 Input (x) 6 jumlah ← jumlah + x 7 rerata ← jumlah/ 3 {menghitung rerata} 8 Output ('Rerata : ',rerata)

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.

- 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 FOR(hitung ← 5; hitung > 0;bilangan--)

```
2       Output(hitung)
3       Output ('DOR!!!')
```

Output dari algoritma ini:

```
5
4
3
2
1
DOR!!!
```

- 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       //inisialisasi,kondisi,  dan pengubah pencacah
2       //dideklarasikan dalam sintaks for
3       FOR(kentang ← 10; kentang > 0; kentang--)
4           Ambil sebuah kentang
5           Kupas kulit kentang
```

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);
}
```

Dalam perulangan mekanismenya saja yang berbeda. Pelaksanaan atau outpun yang dihasilkan tetap sama. Dalam arti apabila kita menbuat suatu program perulangan dengan menggunakan while, maka dapat dikonversi ke bentuk do while dan for. Perhatikan contoh for berikut :

Contoh 26	Hasil

//*----------*
//* Contoh 26 : Menampilkan bilangan genap *
//* yang nilainya kurang atau sama *
//* dengan n dan ditampilkan dari *
//* terbesar sampai nol *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int n;
clrscr();
cout << "Menampilkan bilangan genap yang nilainya "<< endl;
cout <<" kurang atau sama dengan n " << endl;
 	
cout << "Masukkan nilai n = " ;
cin >> n;
// Jika n ganjil, maka dikurangi 1
if (n % 2)
n --;
// tampilkan deret bilangan genap dari besar ke kecil
for (; n >= 0; n -= 2)
cout << n << ' ';
}
	Menampilkan bilangan genap yang nilainya
	kurang atau sama dengan n
	Masukkan nilai n = 11 ↵
	10 8 6 4 2 0

Pada program diatas terdapat :

```
n --; ungkapan kosong
for ( ; n >= 0; n -= 2 )
```

sama artinya dengan :

```
for (n -- ; n >= 0 ; n - = 2 )
```

Contoh 27	Hasil

//*----------*
//* Contoh 27 : Memebentuk segitiga yang berisi *
//* karakter ' * ' dengan menggunakan *
//* for didalam for *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int tinggi, // Menyatakan tinggi segi tiga
baris, // Pencacah untuk baris
kolom; // Pencacah untuk kolom
clrscr();
cout << " Tinggi segitiga = " ;
cin >> tinggi;
cout << endl; //Membuat baris kosong
for (baris = 1; kolom <= baris; kolom ++)
{
for (klom = 1; kolom <= baris ; klom ++)
cout << ' * ' ;
cout << endl ; // Pindah baris
}
}
	Tinggi segitiga = 5 ↵
	*
	**
	***
	****
	*****

Rangkuman

- ☑ Struktur pengulangan memungkinkan program melakukan satu atau lebih aksi beberapa kali.
- ☑ Tiga komponen pengendali aksi adalah inisialisasi, jumlah iterasi, dan kondisi berhenti.
- ☑ Tiga struktur pengulangan yang dapat digunakan adalah struktur pengulangan dengan sintaks WHILE, DO...WHILE, dan FOR.
- ☑ Struktur pengulangan dapat dibuat bersarang dengan sintaks pengulangan yang sama atau berbeda, bahkan dapat digabungkan dengan struktur pemilihan.
- ☑ Untuk keluar dari struktur pengulangan sebelum kondisi berhenti, kita dapat menggunakan sintaks BREAK
- ☑ 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')
```

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       a ← a + 1
9    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 pembedaan 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

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

A. a ← 0
B. a ← 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 ← 0
3   IF (jmlTiket < 1)
4       Output ('Jumlah tiket minimal 1')
5   ELSE
6       FOR (i ← 1; i<= jmlTiket; i++)
7           IF (jmlTiket > 3)
8               jmlTiket ← jmlTiket – 3
9               jmlBayar ← jmlBayar + 3
10          ELSE
11              jmlBayar ← jmlBayar + jmlTiket
12      harga ← jmlBayar * 100000
13      Output (harga)
```

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       ELSE
9           Output (i)
```

Apakah output dari algoritma di atas?

A. I
   xoxo
B. xoxo
   1
C. 1
   Oxox
D. oxox
   1
E. 1
   1

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?

ARRAY

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

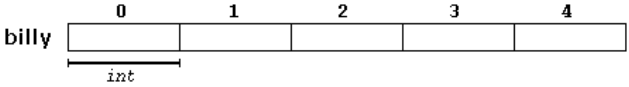
- ☑ Memahami tipe data *array* dan keuntungan yang dapat diberikan
- ☑ Memahami *array* yang memiliki dimensi lebih dari satu
- ☑ Dapat meng-implementasikan tipe data array dalam program
- ☑ Memahami cara menentukan tipe data bentukan dan menggunakannya dalam program

9.1 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.

Dengan kata lain Array adalah himpunan elemen (variable) dengan tipe yang sama dan disimpan secara berurutan dalam memory yang ditandai dengan memberikan index pada suatu nama variable. Contohnya, kita dapat menyimpan 5 nilai dengan tipe **int** tanpa harus mendeklarasikan 5 identifier variabel yang berbeda. Perhatikan contoh dibawah ini :



Bagian kosong diatas merepresentasikan *elemen* array, dalam kasus ini adalah nilai integer. Angka **0 - 4** merupakan index dan selalu dimulai dari **0**. Seperti penggunaan variable pada umumnya, array harus dideklarasikan terlebih dahulu, dengan format sbb :

type name [*elements*];

Maka contoh array diatas dideklarasikan sbb :

```
int billy [5];
```

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

9.1.1 *Array* Satu Dimensi

Sebelum digunakan, variabel array perlu dideklarasikan terlebih dahulu. Cara mendeklarasikan variabel array sama seperti deklarasi variabel yang lainnya, hanya saja diikuti oleh suatu indek yang menunjukan jumlah maksimum data yang disediakan.

Deklarasi Array Bentuk Umum pendeklarasian array :

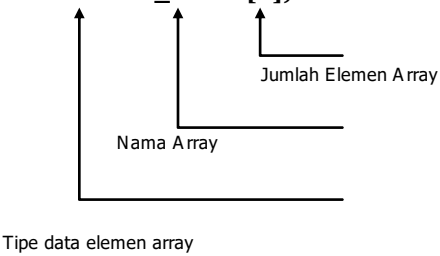
```
Tipe-Data Nama_Variabel[Ukuran]
```

Keterangan :

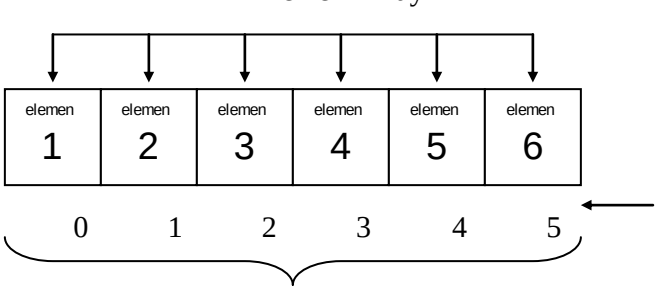
- Type Data: Untuk menyatakan type data yang digunakan.
- Ukuran : Untuk menyatakan jumlah maksimum elemen array.

Contoh Pendeklarasian Array

```
float Nil_Akhir[6];
```



Suatu array dapat digambarkan sebagai kotak panjang yang berisi kotak-kotak kecil didalam kotak panjang tersebut.



Suatu array, dapat diakses dengan menggunakan subscript atau index nya. Bentuk umum pengaksesan adalah :

```
Nama_Array[Subscript/Index]
```

Contoh

```
Nil_Akhir[3];
Nil_Akhir[1];
Nil_Akhir[0];
```

Contoh 28	Hasil

//*----------*
//* Contoh 28 : program menggunakan *
//* array *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
float suhu[5]; // Array dengan elemen 5 bertipe float
clrscr();
// Membaca data dari keyboard dan meletakkan array
cout << "Masukkan 5 buah data suhu" << endl;
for (int i = 0; i < 5; i ++)
{
cout << i + 1 << " . ";
cin >> suhu[i];
}
// Menampilkan isi array ke layar
cout << "Data suhu yang anda masukkan : " << endl;
for (i = 0; i < 5; i ++)
cout << suhu[i] << endl;
}
	Masukkan 5 buah data suhu
	1 : 27.5 ↵
	2 : 28 ↵
	3 : 27.5 ↵
	4 : 30.5 ↵
	5 : 27 ↵
	Data suhu yang anda masukkan
	27.5
	28
	27.5
	30.5
	27

Tampak diatas terdapat pernyataan :

- **float suhu[5];** menyatakan array **suhu** dapat menyimpan 5 (lima) buah data bertipe **float**.
- **suhu[i]** menyatakan elemen suhu dengan *subscript* sama dengan **i**.
- **cin >> suhu[i];** : membaca data dari keyboard dan meletakkan ke elemen nomor **i** pada array **suhu**.
- **cout << suhu[i];** : akan menampilkan elemen bernomor i pada array **suhu**.

Apa output dari program berikut ?

Contoh 29

```
   /* ------------------------------  */
/* Program Array Satu Dimensi */
   /* ------------------------------  */
   #include<conio.h>
   #include<stdio.h>
   #include<iostream.h>
   #include<iomanip.h>

   main()
   {
    int i;
    char nama[5][20];
    float nilai1[5];
    float nilai2[5];
    float hasil[5];

    clrscr();
    for(i=1;i<=2;i++)
    {
                                        cout<<"Data Ke - "<<i<<endl;
                                        cout<<"Nama Siswa : "; gets(nama[i]);
                                        cout<<"Nilai Teori : "; cin>>nilai1[i];
                                        cout<<"Nilai Praktek : "; cin>>nilai2[i];
                                        hasil[i] = (nilai1[i] * 0.40)+ (nilai2[i] * 0.60);
                                        cout<<endl;
    }

    cout<<"-------------------------------------------------";
    cout<<"-------"<<endl;
    cout<<"No. Nama Siswa Nilai Nilai ";
    cout<<"Hasil"<<endl;
    cout<<" Mid Tes FInal ";
    cout<<"Ujian"<<endl;
    cout<<"-------------------------------------------------";
    cout<<"-------"<<endl;

    for(i=1;i<=2;i++)
    {
                                        cout<<setiosflags(ios::left)<<setw(4)<<i;

   cout<<setiosflags(ios::left)<<setw(20)<<nama[i];
                                        cout<<setprecision(2)<<"  "<<nilai1[i];
                                        cout<<setprecision(2)<<"  "<<nilai2[i];
                                        cout<<setprecision(2)<<"  "<<hasil[i]<<endl;
    }

    cout<<"-------------------------------------";
    cout<<"-------"<<endl;
    getch();
   }
```

9.1.2 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.

Array ini dapat digunakan untuk berbagai keperluan. Sebagai gambaran, data kelulusan dari jurusan Teknik Tnformatika, Manajemen Informatika dan Teknik Komputer pada sekolah tinggi Komputer dari tahun 1992 hingga 1995 dapat dinyatakan dengan array berdimensi dua.

Sebelum membahas cara pendefinisian array berdimensi dua, perhatikan tabel berikut :

Jurusan	1992	1993	1994	1995
1. Teknik Informatika	35	45	80	120
2. Manajemen Informatika	100	110	70	101
3. Teknik Komputer	10	15	20	17

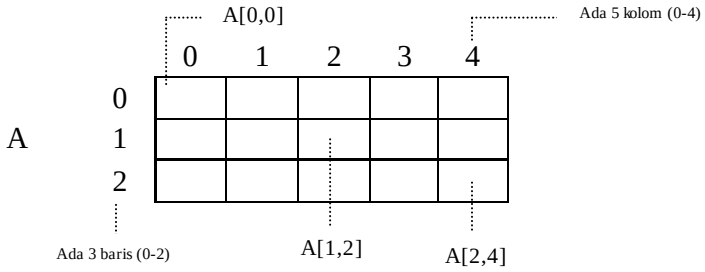
Bentuk seperti tabel diatas dapat dituangkan kedalam array berdimensi dua. Pendefinisiannya :

```
int data_lulus[3][4];
 pada pendefiniasian diatas :
    3 menyatakan jumlah baris (mewakili jurusan)
    4 menyatakan jumlah kolom (mewakili tahun kelulusan)
```

Array berdimensi dua dapat diakses dengan bentuk :

nama_array[*subscript_baris, subscript_kolom*]

Representasi *array* dua dimensi:

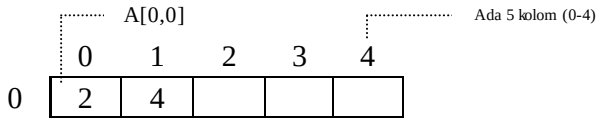


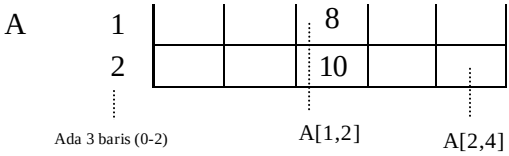
Gambar di atas merepresentasikan *array* yang terdiri dari 3 baris dan 5 kolom, dan jumlah elemennya = 3 x 5 = 15 elemen. Karena terdiri dari baris (*row*) dan kolom (*column*), maka *array* dua dimensi sering juga disebut **matrix**.

Mengakses data *array* dua dimensi:
Seperti a*rray* 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:





Contoh 30	Hasil

//*----------*
//* Contoh 30 : Pemakaian array berdimensi *
//* dua *
//*----------*
#include <iostream.h>
#include <conio.h>
void main()
{
int data_lulus[3][4]; // Array berdimensi dua
int tahun, jurusan;
clrscr();
// Memberikan data ke elemen array data_lulus
data_lulus[0][0] = 35; // data TI – 1992
data_lulus[0][1] = 45; // data TI - 1993
data_lulus[0][2] = 90; // data TI - 1994
data_lulus[0][3] = 120; // data TI - 1995
 	
data_lulus[1][0] = 100; // data MI – 1992
data_lulus[1][1] = 110; // data MI - 1993
data_lulus[1][2] = 70; // data MI – 1994
data_lulus[1][3] = 101; // data MI - 1995
data_lulus[2][0] = 10; // data TK – 1992
data_lulus[2][1] = 15; // data TK- 1993
data_lulus[2][2] = 20; // data TK - 1994
data_lulus[2][3] = 17; // data TK – 1995
// Proses untuk memperoleh informasi kelulusan
while (1)
{
cout << "Jurusan (0 = TI, 1 = MI, 2 = TK) : " ;
cin >> jurusan;
if ((jurusan = = 0) || (jurusan = = 1) || (jurusan = = 3))
break; // Keluar dari while
}
 	
while (1)
{
cout << "Tahun (1992 – 1995) : " ;
cin >> tahun;
if ((tahun >= 1992) && (tahun <= 1995)
{
tahun - = 1992; // Konversi ke 0, 1, 2 atau 3
break; // Keluar dari while
	Jurusan (0 = TI, 1 = MI, 2 = TK) : 1 ↵
	Tahun (1992 – 1195) : 1992 ↵
	Jumlah yang lulus = 100

```
}
}
cout << "Jumlah yang lulus = "
 << data_lulus[jurusan][tahun]  << endl;
}
```

Mula-mula program mengisi data ke array data_lulus. Kemudian program meminta data jurusan dan tahun dari keyboard. Kedua data masukkan dipastikan tidak akan berada diluar jangkauan kedua subscript-nya.

Contoh 31

```
/* ---------------  */
/* Array Dimensi 2 */
/* ---------------  */
#include<conio.h>
#include<stdio.h>
#include<iostream.h>
#include<iomanip.h>

main()
{
 int i, j;
 int data_jual[4][4];

 clrscr();

 for(i=1;i<=3;i++)
 {
                                        for(j=1;j<=3;j++)
                                        {
                                                cout<<"Data Ke - "<<i<<" "<<j<<endl;
                                                cout<<"Jumlah Penjulan : ";
 cin>>data_jual[i][j];
                                        }
                                       }

 cout<<"Data Penjualan Pertahun"<<endl;
 cout<<"----------------------"<<endl;
 cout<<"NO 2001 2002 2003"<<endl;
 cout<<"----------------------"<<endl;

 for(i=1;i<=3;i++)

 {
                                        cout<<setiosflags(ios::left)<<setw(5)<<i;
                                        for(j=1;j<=3;j++)
                                        {
                                                cout<<setiosflags(ios::right)<<setw(4);
                                                cout<<data_jual[i][j];
                                                cout<<" ";
```

```
                                        }
                                        cout<<endl;
                                       }

    cout<<"----------"<<endl;

    getch();
  }
```

Apa output dari program berikut ?

Rangkuman

- ☑ Tipe data *array* digunakan untuk menampung/menyimpan banyak nilai pada satu variabel.
- ☑ Setiap elemen pada tipe data *array* ditandai dengan indeks.
- ☑ Indeks penanda elemen pada *array* menggunakan tipe data yang memiliki keterurutan.
- ☑ Tipe data *array* memiliki dimensi minimal satu hingga n-dimensi.
- ☑ 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.
- ☑ Tipe data bentukan adalah tipe data yang dirancang/dibentuk (dan diberi nama) dari beberapa elemen bertipe tertentu.
- ☑ Tipe data bentukan dapat disimpan dalam variabel bertipe *array*.
- ☑ Elemen dalam tipe data bentukan dapat menggunakan variabel bertipe *array*.
- ☑ Tipe data bentukan yang di dalamnya terdapat elemen bertipe *array*, dapat disimpan dalam variabel bertipe *array*.

Pilihan Ganda

Petunjuk: Pilihlah jawaban yang paling tepat!

1. Manakah deklarasi array yang benar berikut ini dalam 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 terurut dan memiliki tipe data yang sama disebut:
   A. Rekursif
   B. Record
   C. Array
   D. File

3. struct siswa mahasiswa[500]. Dari array ini yang merupakan NAMA dari suatu ARRAY adalah:
   A. struct
   B. siswa
   C. mahasiswa
   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
   B. banyak elemen array
   C. record
   D. tipe data

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

A. 1
B. 2
C. 3
D. 4

Latihan

1. Buat algoritma untuk menghitung nilai total dan nilai rata-rata dari sejumlah nilai yang diinputkan dalam *array*. (Asumsikan *array* memiliki 5 elemen)

2. 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

3. 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)

PEMROGRAMAN MODULAR

10

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

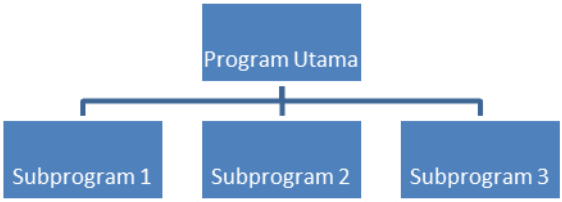
- ☑ Memahami konsep pemrograman modular
- ☑ Mengetahui dua cara pemrograman modular: fungsi dan prosedur
- ☑ Mengetahui cara mengimplementasikan fungsi dan prosedur dalam program
- ☑ Mengetahui dan dapat menerapkan pemanggilan subprogram dari program utama.

10.1 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 10.1 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

10.2 Variabel Lokal dan Variabel Global

10.2.1 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.

10.2.2 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.

10.3 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:

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

Pada intinya fungsi berguna untuk :

a. Mengurangi pengulangan penulisan program yang berulangan atau sama.
b. Program menjadi terstruktur, sehingga mudah dipahami dan dikembangkan.

Fungsi-fungsi yang sudah kita kenal sebelumnya adalah fungsi *main()*, yang bersifat mutlak, karena fungsi ini program akan dimulai, sebagai contoh yang lainnya fungsi *printf()* yang mempunyai tugas untuk menampilkan informasi atau data kelayar dan masih banyak lainnya.

10.3.1 Struktur Fungsi

Sebuah fungsi sederhana mempunyai bentuk penulisan sebagai berikut :

```
nama_fungsi(argumen)
            {
                ...     pernyataan      /
            perintah;
                ...     pernyataan      /
            perintah;
```

Keterangan:

- Nama fungsi, boleh dituliskan secara bebas dengan ketentuan, tidak menggunakan spasi dan nama-nama fungsi yang mempunyai arti sendiri.
- Argumen, diletakan diantara tanda kurung "( )" yang terletak dibelakang nama fungsi. Argumen boleh diisi dengan suatu data atau dibiarkan kosong.
- Pernyataan / perintah, diletakan diantara tanda kurung '{ }'.

Contoh :

Contoh definisi kuadrat() :

```
// Prototipe fungsi
long kuadrat (long 1);
-------------------------------------
// Definisi fungsi
long kuadrat(long 1)
{
 return(1 * 1);
}
```

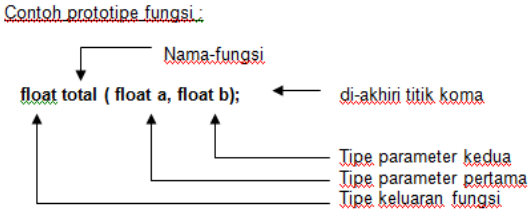
Pernyataan **return** di dalam fungsi digunakan untuk memeberikan nilai balik fungsi. Pada contoh diatas, fungsi kuadrat() memberikan nilai balik berupa nilai kuadrat dari argumen.

10.3.2 Prototipe Fungsi

Prototipe fungsi digunakan untuk menjelaskan kepada kompiler mengenai :

- Tipe keluaran fungsi.
- Jumlah parameter.
- Tipe dari masing-masing parameter.

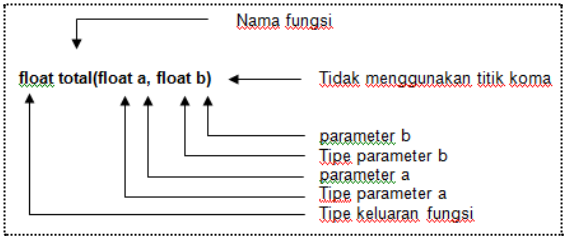
Salah satu keuntungan pemakai prototipe, kompiler akan melakukan konversi antara tipe parameter dalam definisi dan parameter saat pemanggilan fungsi tidak sama atau akan menunjukkan kesalahan jika jumlah parameter dalam definisi dan saat pemanggilan berbeda.



Jika dalam penggunaan fungsi yang dideklarasikan dengan menggunakan prototipe, maka bentuk definisi harus diubah. Sebagai contoh pada pendefinisian berikut :

```
float total(a, b)
float a, y;
```

Bentuk pendefinisian diatas harus diubah menjadi bentuk modern pendefinisian fungsi :

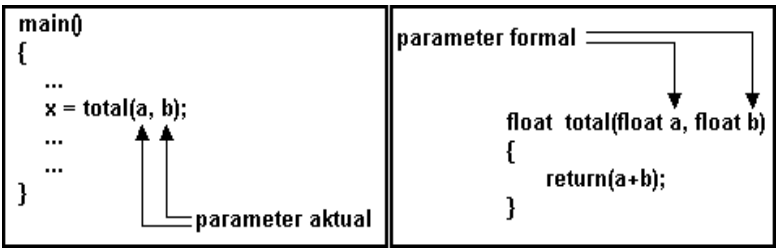


10.3.3 Parameter Fungsi

Terdapat dua macam para parameter fungsi, yaitu :

- **Parameter formal** adalah *variabel yang ada pada daftar parameter dalam definisi fungsi*.
- **Parameter Aktual** adalah *variabel yang dipakai dalam pemanggilan fungsi.*

Bentuk penulisan Parameter Formal dan Parameter Aktual.



Ada dua cara untuk melewatkan parameter ke dalam fungsi, yaitu berupa :

- Pemanggilan dengan nilai (*Call by Value*)
- Pemanggilan dengan Referensi (*Call by Reference*)

10.3.4 Pemanggilan dengan nilai (*Call by Value*)

Pemanggilan dengan nilai merupakan cara yang dipakai untuk seluruh fungsi buatan yang telah dibahas didepan. Pada pemanggilan dengan nilai, nilai dari parameter aktual akan ditulis keparameter formal. Dengan cara ini nilai parameter aktual tidak bisa berubah, walaupun nilai parameter formal berubah.

```
/* ------------------------  */
/* Penggunaan Call By Value */
/* Program Pertukaran Nilai */
/* ------------------------  */

#include<conio.h>
#include<stdio.h>
#include<iostream.h>

tukar(int x, int y);

main()
{
 int a, b;

 a = 88;
 b = 77;

 clrscr();

 cout<<"Nilai Sebelum Pemanggilan Fungsi";
 cout<<"\na = "<<a<<" b = "<<b;

 tukar(a,b);

 cout<<"\nNilai Setelah Pemanggilan Fungsi";
 cout<<"\na = "<<a<<" b = "<<b;

 getch();
}


tukar(int x, int y)
{
 int z;

 z = x;
 x = y;
 y = z;

 cout<<"\n\nNilai di dalam Fungsi Tukar()";
 cout<<"\nx = "<<x<<" y = "<<y;
 cout<<endl;
}
```

10.3.5 Pemanggilan dengan Referensi (*Call by Reference*)

Pemanggilan dengan reference merupakan upaya untuk melewatkan alamat dari suatu variabel kedalam fungsi. Cara ini dapat dipakai untuk mengubah isi suatu variabel diluar fungsi dengan melaksanakan pengubahan dilakukan didalam fungsi.

```
/* ---------------------------- */
/* Penggunaan Call By Reference */
/* Program Pertukaran Nilai */
/* ---------------------------- */

#include<conio.h>
#include<stdio.h>
#include<iostream.h>

tukar(int *x, int *y);

main()
{
 int a, b;
 a = 88;
 b = 77;

 clrscr();

 cout<<"Nilai Sebelum Pemanggilan Fungsi";
 cout<<"\na = "<<a<<" b = "<<b;

 tukar(&a,&b);

 cout<<endl;
 cout<<"\nNilai Setelah Pemanggilan Fungsi";
 cout<<"\na = "<<a<<" b = "<<b;

 getch();
}

tukar(int *x, int *y)
{
 int z;

 z = *x;
 *x = *y;
 *y = z;

 cout<<endl;
 cout<<"\nNilai di Akhir Fungsi Tukar()";
 cout<<"\nx = "<<*x<<" y = "<<*y;
}
```

Contoh 32	Hasil

//*----------*
//* Contoh 32 : Pembuatan fungsi dengan argumen *
//* bertipe long dan nilai balik berupa *
//* long *
//*----------*
 	
#include <iostream.h>
#include <iomanip.h>
#include <conio.h>
long kuadrat(long1); // prototipe fungsi
void main()
{
clrscr();
for (long bil = 200; bil < 2000; bil+= 200)
cout << setw(8) << bil
<< setw(8) << kuadrat(bil) << endl;
}
 	
// Definisi fungsi
 	
long kuadrat(long 1)
{
return(1 * 1);
}
	200 40000
	400 160000
	600 360000
	800 640000
	1000 1000000
	1200 1440000
	1400 1960000
	1600 2560000
	1800 3240000

Contoh 33	Hasil

//*----------*
//* Contoh 33 : Menggambarkan nilai bawaan dalam *
//* argumen fungsi *
//*----------*
#include <iostream.h>
#include <conio.h>
void tulis_cplus(int jum); // Prototipe fungsi
void main()
{
clrscr();
tulis_cplus(1); // Untuk menuliskan sebuah tulisan C++
}
 	
void tulis_cplus(int jum);
{
for (int i = 0; i < jum; i ++)
cout << " C++ " << endl;
cout << " Seleseai " << endl;
}
 	
}
	C++
	Selesai

Contoh 34	Hasil

//*----------*
//* Contoh 34 : Menggambarkan nilai bawaan *
//* Dalam argumen fungsi *
//*----- ----------*
#include <iostream.h>
#include <conio.h>
void tulis_cplus(int jum = 1); // Prototipe fungsi
// Dan menyetel nilai bawaan fungsi
void main()
{
clrscr();
tulis_cplus(); // Argumen tidak perlu disebutkan
}
 	
void tulis_cplus(int jum);
{
for (int i = 0; i < jum; i ++)
cout << " C++ " << endl;
cout << " Seleseai " << endl;
}
	C++
	Selesai

Pada contoh program 33 dan 34 mempunyai kesamaan hanya saja pada contoh program 34 dalam prototipe fungsi nilai bawaannya dikut sertakan sehingga pada saat argumen pemanggilan fungsi tidak perlu di tuliskan lagi.

10.4 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:

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

Bentuk umum :

```
Procedure Nama_Prosedur(param1:tipedata,param2:tipedata,)
```

Contoh:

- Procedure TambahKali; (procedure tanpa parameter).
- Procedure Hitung(a,b : integer); (procedure dengan parameter).

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 procedurenya kemudian diikuti daftar parameternya sebagai berikut:

```
nama_prosedure
```

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.

Untuk dapat menggunakan prosedur tersebut perlu adanya pemanggilan prosedur sama seperti pemanggilan fungsi. Untuk lebih jelasnya kita lihat potongan program berikut :

```
void hitungluaslingkaran()
 {
 float phi;
 float r;
 luas=phi*r*r;
 printf("luas lingkaran : %0.2f\n",luas);
 }
```

Pada prosedur di atas kita mencari luas lingkaran dengan menggunakan rumus phi*r*r dengan deklarasi lokal berupa phi dan r.

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

```
1.  #include <stdio.h>
2.  //Variabel Global
3.  float nilai1, nilai2, rataan;
4.
```

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

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.

10.5 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 *ceil* (untuk membulatkan keatas nilai pecahan).	var-int ← ceil(ekspresi float)
- Fungsi *min/ max* (menentukan nilai minimal atau maksimal dari dua bilangan)	var-int ← min(3,5) var-int ← max(3,5)
- Fungsi *random* (mendaparkan nilai secara acak dari rentang tertentu)	var-int ← random(10)
- Fungsi *sin* (memperoleh nilai sinus dari suatu bilangan)	Var-float ← sin(int)

Beberapa prosedur yang telah terdefinisi, antara lain:

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

10.6 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 Dati %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 arsir. 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,

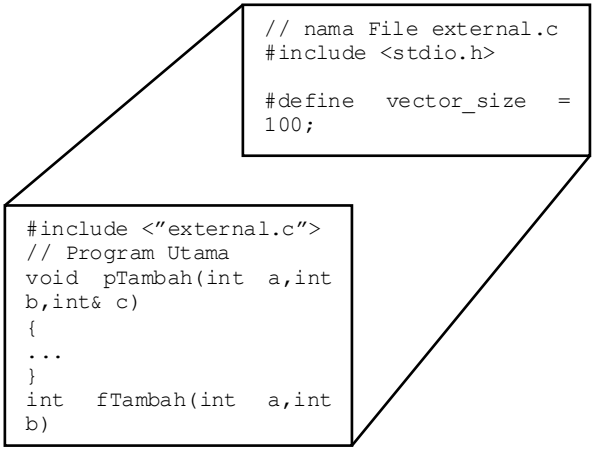
maka akan menghasilkan 1. Namun jika tidak, proses akan memanggil fungsi yaitu dirinya sendiri.

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
```

10.7 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.



Gambar 10.1. Contoh deklarasi unit dalam program utama

Rangkuman

- ☑ Pemrograman modular adalah upaya memecah program yang kompleks ke dalam sub-subprogram yang lebih kecil untuk menyederhanakan penyelesaian persoalan.
- ☑ Setelah didefinisikan, subprogram dapat dipanggil berkali-kali dengan parameter yang berbeda-beda.
- ☑ Dua tipe subprogram yang biasa digunakan adalah fungsi (*function*) dan prosedur (*procedure*).
- ☑ Sebuah subprogram dapat disimpan dalam file yang sama dengan program utama, ataupun dalam file terpisah.
- ☑ Sebuah fungsi memetakan sebuah nilai (dalam *domain*) menjadi nilai lain (dalam *range*) dengan operasi / proses tertentu.
- ☑ Dalam penggunaan prosedur, dikenal dua tipe parameter, yaitu parameter formal dan parameter aktual.
- ☑ Fungsi dan prosedur bekerja menurut mekanisme pemanggilan-pengembalian (*call-return mechanism*).
- ☑ Terdapat beberapa fungsi dan prosedur yang telah terdefinisi dan dapat langsung digunakan dalam program.
- ☑ 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 dilokalisir.
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. Progam 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)
{
    rata ← (var1+var2)/ 2
    return (rata)
}
```

   Bagian apakah yang kurang dari fungsi di atas?
   F. deklarasi variabel global
   G. deklarasi variabel lokal
   H. deklarasi parameter input
   I. deklarasi parameter output
   J. deklarasi output fungsi

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←10 dan iUmurOrtu←23?
3. Sebutkan variabel-variabel lokal dan global dalam potongan program di atas!
4. Sebutkan tiga macam parameter dalam subprogram!

PENGURUTAN (SORTING)

11

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

- ☑ Memahami konsep pengurutan
- ☑ Mengenal beberapa algoritma pengurutan
- ☑ Menerapkan algoritma pengurutan dalam program

11.1 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

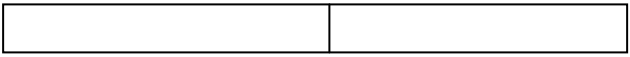
Dalam *courseware* ini hanya akan dibahas tiga metode *sort* yang pertama yang dianggap mudah, yaitu: ***Bubble Sort , Selection Sort dan Insertion Sort***

11.1.1 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,……., N-2 , lakukan :
Mulai dari elemen J = N-1, N-2,….., 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,…,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,…,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,…,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 minimun 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		**1**	8	9	7	6

Hasil akhir langkah 1 :

1	8	9	7	6
0	1	2	3	4

Pass 2 :

I = 1 ;J= N-1= 4	**1**	8	**9**	**6**	7	
J = 3		**1**	**8**	**6**	9	7
J = 2		**1**	**6**	8	9	7

Hasil akhir langkah 2 :

1	**6**	8	9	7
0	1	2	3	4

Pass 3 :

I = 2 ;J= N-1= 4	**1**	**6**	**8**	**7**	9	
J = 3		**1**	**6**	**7**	8	9

Hasil akhir langkah 3 :

1	**6**	**7**	8	9
0	1	2	3	4

Pass 4 :

I = 3 ;J= N-1= 4 **1** **6** **7** **8** **9**

Hasil akhir langkah 4 :

1	**6**	**7**	**8**	**9**
0	1	2	3	4

Selesai. Array L sudah terurut !!

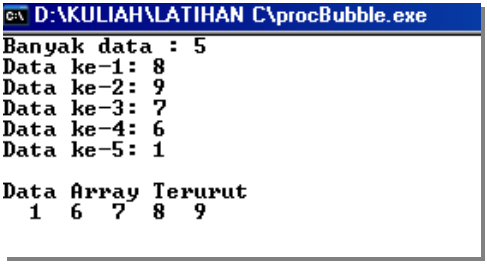
Pseudocode prosedur algoritma *Bubble Sort* secara *Ascending*

```
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[j-1]←A[j]
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. getche();
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:



11.1.2 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* :

- **Algoritma pengurutan maksimum** (*maximum selection sort*), yaitu memilih **elemen maksimum sebagai basis pengurutan**.
- **Algoritma pengurutan minimum** (*minimum selection sort*), yaitu memilih **elemen minimum sebagai basis pengurutan**.

11.1.2.1 Maximum Selection Sort Ascending

Untuk mendapatkan *array* yang terurut menaik (*ascending*), algoritma *maximum selection sort* dapat ditulis sebagai berikut :

- Jumlah Pass = N-1 (*jumlah pass*)
- Untuk setiap pass ke – I = 0,1,….., 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 :

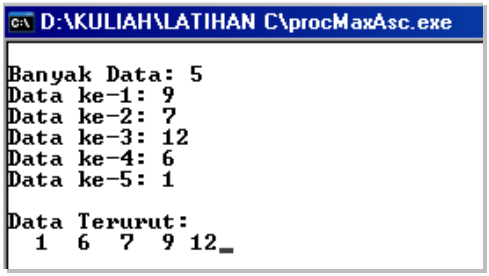
```
1. //prosedur algoritma Tukar Tempat
2. //I.S:nilai-nilai yang dikirimkan sudah terdefinisi sebelumnya
3. //F.S:nilai yang dikirimkan tertukar nilainya
4. procedure v_Tukar(input/output P:integer,
   input/output M:integer)
5. KAMUS:
6. temp:integer
7. ALGORITMA:
8. temp ← P
9. P ← M
10. M ← temp
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
 getche();
}

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:



11.1.2.2Maximum 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* :

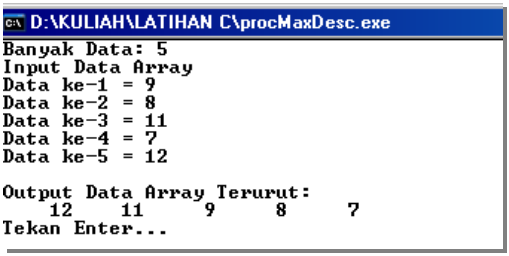
```
1. //prosedur algoritma Maximum Selection Sort secara Descending
2. //I.S:array sudah berisi nilai integer yang belum terurut
3. //F.S:nilai-nilai dalam array terurut secara Descending
4. procedure v_SelDesc(input/output A:array[0..4]of integer,
   input N:integer)
5. KAMUS:
6. k,maks,j,temp:integer
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. getche();
```

```
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:



11.1.2.3Minimum 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,….., 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 s*election 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
```

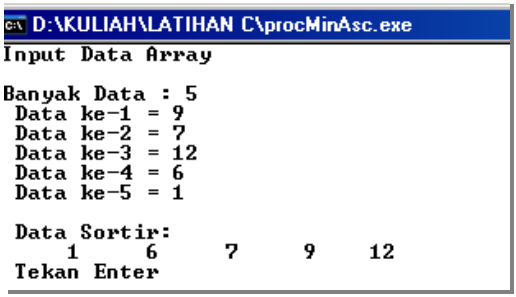
```
15. endfor
16. v_Tukar(A[k],A[min])
17.endfor
```

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. getche();
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; } //endloop 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:



11.1.2.4 Minimum Selection Sort Descending

Misal, terdapat *array* L dengan N = 5 buah elemen yang belum terururt. *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]. 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]. 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)

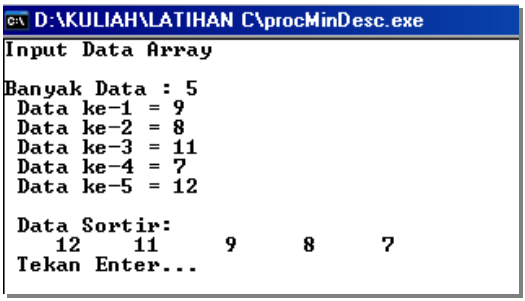
Pseudocode Algoritma *Minimum Selection Sort* secara *Descending* :

```
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. getche();
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:



11.1.3 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] : (**Pass-1**) Simpan nilai L[1] ke variabel X. 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]: (Pass-2) Simpan nilai L[2] ke variabel X 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)

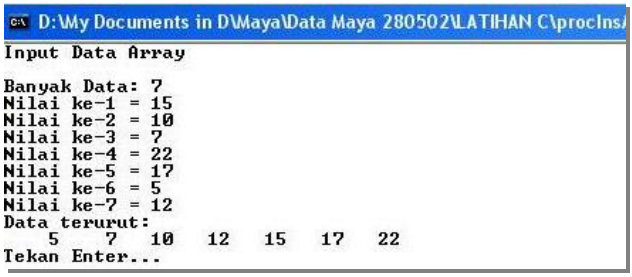
Pseudocode Algoritma *Insertion Sort* secara *Ascending* :

```
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");
 getche();
 }

 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:



Rangkuman

- ☑ ***Proses Sorting*** merupakan proses 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
- ☑ Terdapat dua macam proses pengurutan, yaitu pengurutan internal (*internal sort*) dan pengurutan eksternal (*external 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.
- ☑ Algoritma *Selection sort* memilih elemen maksimum/minimum *array*, lalu menempatkan elemen maksimum/minimum itu pada awal atau akhir *array* (tergantung pada urutannya *ascending*/*descending*).
- ☑ 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
   B. Divide
   C. Sorting
   D. Conquer

2. Berikut ini adalah metode yang digunakan pada teknik *sorting*, ***kecuali***:
   A. Bubble
   B. Heap
   C. Fibonacci
   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
   B. 0 4 8 2
   C. 0 8 2 4
   D. 0 2 4 8

4. Pada data 0 6 3 2 4, akan dilakukan *maximum selection sort* secara *ascending*, maka pada langkah pertama, urutan data yang terjadi adalah:
   A. 0 2 6 3 4
   B. 0 2 3 4 6
   C. 0 4 3 2 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**.

Daftar Referensi

[1] Munir, Rinaldi. 2011, *Algoritma dan Pemrograman : Dalam Pascal dan C*. Bandung : Informatika.

[2] Wesley, Addison. 1997, *Algorithm Data Structures and Problem Solving with C++*.

[3] Moh. Sjukani, *Algoritma dan Struktur Data*. Mitra Wacana Media

[4] Schaum, *Programming with C++ 2nd. 2000*. McGraw-Hill

[5] Schaum. *Teach yourself C++ in 21 days. 2007*. McGraw-Hill

[6] Inggriani Liem, Diktat Catatan Singkat Bahasa C Agustus 2003. ITB

[7] Inggriani Liem, Diktat Kuliah Dasar Pemrograman April 2007. ITB

[8] http://www.cs.aau.dk/~normark/prog3-03/html/notes/paradigms_themes-paradigms.html akses pada 30 Desember 2015 14.00

[9] http://encyclopedia.thefreedictionary.com/Programming%20paradigm akses pada 30 Desember 2015 14.00

[10] http://risca.staff.gunadarma.ac.id/Downloads/files/29968/bahasa+C%2B%2B.doc akses pada 02 Januari 2015 09.00

View publication stats