

ALGORITMA DAN PEMROGRAMAN



Dr. Aris Puji Widodo, MT.

Kata Pengantar

Puji syukur kita panjatkan kepada Allah SWT atas rahmat dan hidayahnya, sehingga buku Algoritma dan Pemrograman telah dapat diselesaikan. Buku Algoritma Pemrograman ini disusun untuk mendukung proses pembelajaran mengenai mata kuliah Algoritma dan Pemrograman pada khususnya. Buku ini lebih difokuskan pada penggunaan algoritma dibandingkan dengan bahasa pemrograman itu sendiri. Oleh karena itu untuk menyelesaikan sebuah persoalan lebih pada pemberian solusi dalam bentuk sketsa solusi. Dari sketsa solusi untuk dapat memberikan solusi secara utuh diperlukan untuk menterjemahkan sketsa solusi dalam bentuk algoritma ke dalam sebuah bahasa pemrograman. Representasi algoritma yang digunakan pada buku ini adalah menggunakan Notasi Algoritmik yang dikembangkan oleh Ingriani Liem dosen Institut Teknologi Bandung (ITB) pada Diktat kuliah Algoritma dan Pemrograman. Bahasa pemrograman yang digunakan pada buku ini adalah menggunakan bahasa pemrograman C. Paradigma pemrograman yang difokuskan pada buku ini adalah pemrograman terstruktur atau modular yang direalisasikan dalam bentuk Abstrak Data Type (ADT).

Secara keseluruhan setiap bahasan materi yang disampaikan pada buku ini selalu disertai dengan contoh, penjelasan contoh, dan dilengkapi dengan soal-soal latihan untuk dilakukan penugasan dan diskusi.

Akhir kata, semoga buku Algoritma dan Pemrograman ini dapat memberikan manfaat dan untuk kesempurnaan buku ini, sangat terbuka bagi semua pihak untuk memberikan saran, kritik, dan masukannya.

Terima Kasih.

Semarang, 02-02-2020

Penulis

Dr. Aris Puji Widodo, MT.

DAFTAR ISI

BAB I	Pendahuluan
BAB II	Nama, Type Data, Konstanta, dan Variabel
BAB III	Notasi Algoritmik
BAB IV	Sequence
BAB V	Kondisional (Analisa Kasus)
BAB VI	Perulangan
BAB VII	Tabel (Array/Larik)
BAB VIII	Sub Program
BAB IX	Searching
BAB X	Sorting
BAB XI	Rekursif
BAB XII	Studi Kasus

Halaman ini sengaja dikosongkan

BAB I

Pendahuluan

Kata Algoritma memiliki sejarah yang agak unik, dimana orang hanya akan menemukan kata algorism yang berarti proses menghitung dengan angka arab. Para ahli bahasa berusaha menemukan asal kata Algoritma namun hasilnya masih belum memuaskan. Kemudian pada akhirnya para ahli sejarah matematika menemukan asal kata Algoritma yang berasal dari nama penulis buku arab yang terkenal yaitu Abu Ja'far Muhammad Ibnu Musa Al-Khuwarizmi. Kata Algoritma pada buku Al-Khuwarizmi dibaca oleh orang barat menjadi Algorism. Al-Khuwarizmi menulis buku yang berjudul Kitab Al Jabar Wal-Muqabala yang artinya “Buku pemugaran dan pengurangan” (The book of restoration and reduction). Perubahan kata dari algorism menjadi algorithm muncul karena kata algorism sering disalahartikan dengan kata arithmetic, sehingga akhiran –sm berubah menjadi –thm. Oleh karena perhitungan dengan angka Arab sudah menjadi hal biasa, maka secara perlahan-lahan kata algorithm secara bertahap digunakan sebagai metode perhitungan/komputasi secara umum, sehingga mengakibatkan timbulnya pergeseran makna dari kata aslinya. Untuk bahasa Indonesia, kata algorithm diserap menjadi kata algoritma.

1.1 Definisi Algoritma

Algoritma adalah urutan langkah yang sistematis/terstruktur dan logis untuk menyelesaikan persoalan. Dalam konteks yang berbeda, bahwa algoritma adalah spesifikasi urutan langkah untuk melakukan aktifitas pekerjaan tertentu yang sudah terdefinisi. Kunci utama dalam sebuah algoritma adalah difokuskan pada kata logis, artinya bahwa sebuah algoritma harus dapat di nilai kebenarannya.

Sebagai contoh diberikan algoritma untuk **“melakukan pemanggilan telpon dengan menggunakan handphone ke nomor handphone (phone) yang belum terdaftar pada phone book handphone si pemilik handphone”**.

Algoritma 1 dilakukan oleh Aurel:

1. Menekan tombol nomor-nomor handphone (phone) tujuan. **(kondisi awal)**
2. Menekan tombol memanggil pada handphone
3. Mendengarkan nada sambung
4. Melakukan percakapan dengan si penerima phone
5. Mengakhiri percakapan dengan menekan tombol menutup permbicaraan telpon.
(kondisi akhir)

Algoritma 2 dilakukan oleh Ghani:

1. Mengambil handphone dari tempatnya. **(kondisi awal)**
2. Membuka tombol pengunci handphone
3. Masuk ke menu handphone pada bagian melakukan panggilan handphone(phone)
4. Menekan tombol nomor-nomor handphone (phone) tujuan
5. Menekan tombol memanggil pada handphone
6. Mendengarkan nada sambung
7. Melakukan percakapan dengan si penerima phone
8. Mengakhiri percakapan dengan menekan tombol menutup permbicaraan telpon.
9. Meletakkan kembali handphone pada tempat semula. **(kondisi akhir)**

Pada algoritma 1 dan 2 dimulai dan diakhiri dengan menggunakan kondisi yang berbeda-beda, dan mungkin jika ada orang lain lagi melakukan algoritma tersebut juga sangat dimungkinkan untuk menghasilkan perbedaan pada kondisi awal dan akhir yang mereka gunakan. Pernyataan di atas sangat berpotensi untuk menghasilkan multi interpretasi pada kondisi awal dan akhir, hal ini disebabkan karena pada pernyataan di atas tidak didefinisikan secara jelas. Ketidakjelasan terdapat pernyataan mengenai kondisi awal dan akhir yang tidak disebutkan pada pernyataan di atas. Agar pernyataan di atas tidak berpotensi untuk menimbulkan multi interpretasi, maka perlu diberikan sebuah asumsi untuk dinyatakan sebagai kondisi awal dan akhir sehingga setiap orang yang melakukan akan memiliki keseragaman interpretasi terhadap kondisi awal dan akhir. Sedangkan pada bagian antara kondisi awal dan akhir dapat dinyatakan berbeda-beda

tergantung dari kejelasan dalam memberikan solusi terhadap persoalan, sehingga dapat digunakan untuk mengukur efektifitas dari sebuah algoritma yang dikerjakan dengan banyak cara. Dalam konteks pemrograman, algoritma adalah merupakan sebuah sketsa solusi secara global dari sebuah persoalan dan tidak memiliki mesin kompilasi, sehingga kebenaran sangat bergantung pada logika yang dimiliki oleh seorang programmer.

1.2. Representasi Algoritma

Untuk merepresentasikan algoritma terdapat beberapa cara, diantaranya menggunakan bahasa natural, flowchart, atau pseudo code (ada juga yang menyebut dengan nama notasi algoritmik). Cara merepresentasikan algoritma tidaklah diatur harus menggunakan salah satu dari beberapa cara yang telah disebutkan diatas, akan tetapi diberikan kebebasan sesuai dengan gaya masing-masing programmer.

Bahasa Natural merupakan cara untuk merepresentasikan sebuah algoritma dengan menggunakan bahasa sehari-hari (misalnya bahasa Indonesia, bahasa Inggris, dan bahasa lain yang digunakan sebagai alat komunikasi). Representasi algoritma dengan menggunakan bahasa natural tidak memiliki aturan yang baku, akan tetapi dalam menyajikannya harus tetap mengikuti kaedah baku pada bahasa yang digunakan. Meskipun dalam merepresentasikan sebuah algoritma tidak ada aturan yang baku, hanya sebagai kebiasaan dalam menuliskan algoritma diawali dengan menuliskan kata **Mulai** dan diakhiri dengan kata **Selesai**. Kemudian selain penggunaan istilah kata **Mulia** dan **Selesai**, sebaiknya untuk setiap langkah yang dituliskan harus menggunakan **penomoran** (1, 2, ..., dll) sebagai label. Setiap langkah yang disajikan dengan menggunakan penomoran sebaiknya menunjukan sebuah **urutan**, sehingga akan lebih mudah untuk dilakukan penelusuran dari langkah-langkahnya. Untuk lebih memberikan pemahaman mengenai representasi sebuah algoritma menggunakan bahasa natural, maka diberikan contoh persoalan dalam kehidupan sehari-hari sebagai berikut:

Contoh 1.1

Menukarkan isi gelas yang berisi kopi dengan isi gelas yang berisi susu.

Representasi algoritma menggunakan bahasa natural adalah sebagai berikut:

Untuk persoalan diatas, dimisalkan bahwa gelas yang berisi kopi diberikan nama A, dan gelas yang berikan susu diberikan nama B.

1. **Mulai** {*gelas A berisi kopi, dan B berisi susu*}
2. Sediakan satu gelas kosong, misalkan gelas kosong tersebut diberikan nama C
3. Masukkanlah isi gelas A (gelas yang berisi kopi) ke dalam gelas C (gelas kosong)
4. Masukkanlah isi gelas B (gelas yang berisi susu) ke dalam gelas A (gelas A kosong, setelah isi gelas A dimasukan ke gelas C)
5. Masukkanlah isi gelas C (gelas yang sudah di isikan kopi) ke dalam gelas B (gelas kosong , karena isinya telah dipindahkan ke dalam gelas A)
6. **Selesai** {*gelas A berisi susu, dan B berisi kopi*}

Contoh 1.2

Menghitung dan menampilkan luas bujursangkar.

Representasi algoritma menggunakan bahasa natural adalah sebagai berikut:

1. **Mulai** {*sisi bujursangkar belum terisi*}
2. Siapkan variabel sisi bujur sangkar misalnya diberikan nama S, dan variabel luas yang diberikan nama Luas.
3. Masukkanlah nilai variabel sisi bujursangkar (S)
4. Menghitung luas bujursangkar $Luas = S \times S$
5. Menampilkan luas bujur sangkar (Luas)
6. **Selesai** {*luas bujursangkar ditampilkan*}

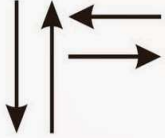
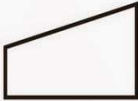

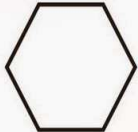

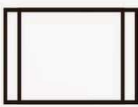

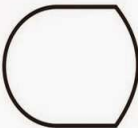



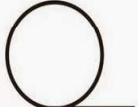
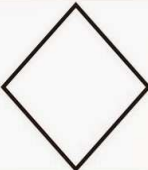
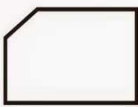


Berdasarkan contoh 1.1 dan 1.2 yang diberikan diatas, memang antara contoh satu dengan yang lainnya tidak memiliki kesamaan dalam menyajikannya, oleh karena itu sangat disarankan dalam membuat algoritma dengan menggunakan bahasa natural, sebaiknya disajikan dengan sederhana dan lebih mudah untuk dipahami. Hal ini perlu mengingat bahwa algoritma bukanlah produk akhir dari sebuah solusi permasalahan, melainkan harus dapat dikomunikasikan dengan bahasa pemrograman secara mudah.

Flowchart atau dapat juga disebut dengan nama **Diagram Alir** merupakan sebuah bagan dengan menggunakan simbol-simbol tertentu yang menggambarkan urutan langkah (proses) secara rinci dan rangkaian hubungan antara satu langkah (proses) dengan langkah (proses) lainnya pada sebuah algoritma. Untuk membuat flowchart tidak ada pedoman secara mutlak, hal ini dikarenakan bahwa flowchart merupakan hasil sintesa dari sebuah persoalan. Hasil sintesa akan memiliki variasi yang cukup banyak antara satu dengan yang lainnya. Meskipun bervariasi hasil dari sebuah sintesa permasalahan, akan tetapi secara konstruksi umum dalam membuat sintesa persoalan terdiri dari input, proses, dan output, seperti yang diberikan pada Gambar 1.1. Input merupakan sesuatu yang dimasukkan ke dalam sebuah proses, input dapat berupa masukan dari perangkat masukan (seperti keyboard, barcode, scanner, ... dll), berupa sebuah file atau dari sistem (proses) lainnya. Proses merupakan langkah atau cara untuk melakukan pemrosesan/pengolahan data yang diberikan dari sebuah masukan. Kemudian output merupakan keluaran dari hasil pemrosesan/pengolahan yang dilakukan proses untuk dikeluarkan pada perangkat keluaran (seperti printer, ..., dll), file, atau sistem (proses) lainnya. Oleh karena itu berdasarkan penjelasan mengenai input, proses dan output diatas, maka untuk membuat algoritma dengan menggunakan flowchart selalu diarahkan ke dalam 3 hal tersebut. Tujuannya adalah untuk mempermudah dalam melakukan konstruksi algoritma dan algoritma yang dibuat dapat dengan mudah dilakukan penelusuran dari setiap langkah satu ke langkah berikutnya. Untuk memberikan standarisasi dalam membuat algoritma menggunakan flowchart, maka diberikan simbol-simbol yang digunakan untuk membuat flowchart secara rinci yang meliputi notasi dan kegunaannya diberikan pada Tabel 1.1.

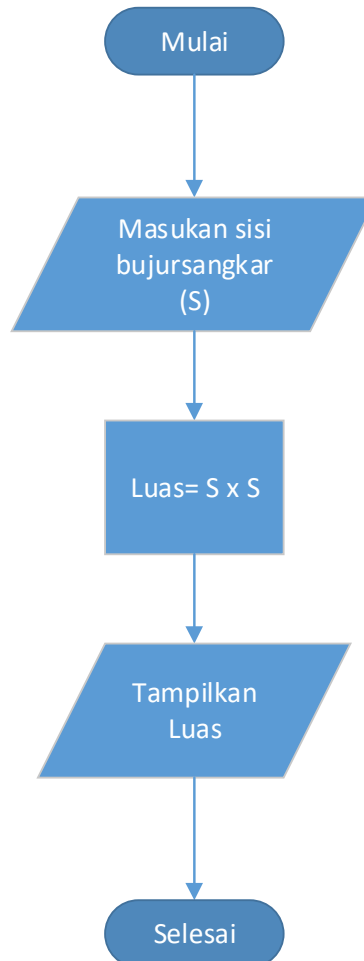


Gambar 1.1. Konstruksi Umum Sintesa Permasalahan

Tabel 1.1. Simbol Flowchart

	Flow Direction symbol Yaitu simbol yang digunakan untuk menghubungkan antara simbol yang satu dengan simbol yang lain. Simbol ini disebut juga connecting line.		Simbol Manual Input Simbol untuk pemasukan data secara manual on-line keyboard
	Terminator Symbol Yaitu simbol untuk permulaan (start) atau akhir (stop) dari suatu kegiatan		Simbol Preparation Simbol untuk mempersiapkan penyimpanan yang akan digunakan sebagai tempat pengolahan di dalam storage.
	Connector Symbol Yaitu simbol untuk keluar - masuk atau penyambungan proses dalam lembar / halaman yang sama.		Simbol Predefine Proses Simbol untuk pelaksanaan suatu bagian (sub-program)/prosedure
	Connector Symbol Yaitu simbol untuk keluar - masuk atau penyambungan proses pada lembar / halaman yang berbeda.		Simbol Display Simbol yang menyatakan peralatan output yang digunakan yaitu layar, plotter, printer dan sebagainya.
	Processing Symbol Simbol yang menunjukkan pengolahan yang dilakukan oleh komputer		Simbol disk and On-line Storage Simbol yang menyatakan input yang berasal dari disk atau disimpan ke disk.
	Simbol Manual Operation Simbol yang menunjukkan pengolahan yang tidak dilakukan oleh computer		Simbol magnetik tape Unit Simbol yang menyatakan input berasal dari pita magnetik atau output disimpan ke pita magnetik.
	Simbol Decision Simbol pemilihan proses berdasarkan kondisi yang ada.		Simbol Punch Card Simbol yang menyatakan bahwa input berasal dari kartu atau output ditulis ke kartu
	Simbol Input-Output Simbol yang menyatakan proses input dan output tanpa tergantung dengan jenis peralatannya		Simbol Dokumen Simbol yang menyatakan input berasal dari dokumen dalam bentuk kertas atau output dicetak ke kertas.

Algoritma menggunakan flowchart, kita menggunakan contoh 1.2., yaitu mengenai persoalan menghitung luas bujursangkar dan menampilkan hasil luas bujursangkar tersebut. Adapun representasi algoritma dengan menggunakan flowchart diberikan pada Gambar 1.2.



Gambar 1.2. Flowchart Menghitung Luas Bujursangkar (contoh 1.2.)

Pseudo code merupakan sebuah notasi yang digunakan untuk membuat algoritma, notasi yang digunakan tidak dalam bentuk diagram, namun lebih difokuskan pada sebuah bentuk notasi yang dipergunakan pada sebuah bahasa pemrograman. Pseudo code tidak menunjuk pada salah satu bahasa pemrograman tertentu, akan tetapi bebas dari bahasa pemrograman apapun. Pseudo code bukan merupakan sebuah bahasa pemrograman, sehingga pseudo code tidak memiliki mesin kompilasi. Kompilasi pseudo code dilakukan oleh programmer sendiri. Untuk merepresentasikan notasi-notasi pseudo code dalam membuat sebuah algoritma tidak ada

aturan yang baku dan variasinya sangat banyak. Pada buku ini difokuskan secara garis besar pada penggunaan notasi algoritmik yang dikembangkan oleh Inggriani Liem (2001). Akan tetapi sangat dimungkinkan dalam penggunaan notasi algoritmik pada buku ini, dapat juga dilakukan modifikasi notasi algoritmik yang digunakan, seperti yang diberikan pada bagian sub bab berikut ini. Representasi algoritma menggunakan notasi algoritmik, menggunakan contoh 1.2., yaitu mengenai persoalan menghitung luas bujursangkar dan menampilkan hasil luas bujursangkar tersebut. Adapun representasi algoritma dengan menggunakan notasi algoritmik diberikan pada Kode Sumber 1.1.

```
Program luasbjks
{ menghitung dan menampilkan luas bujur sangkar }

Kamus
    S      : integer { Sisi bujursangkar }
    Luas : integer { Luas bujursangkar }

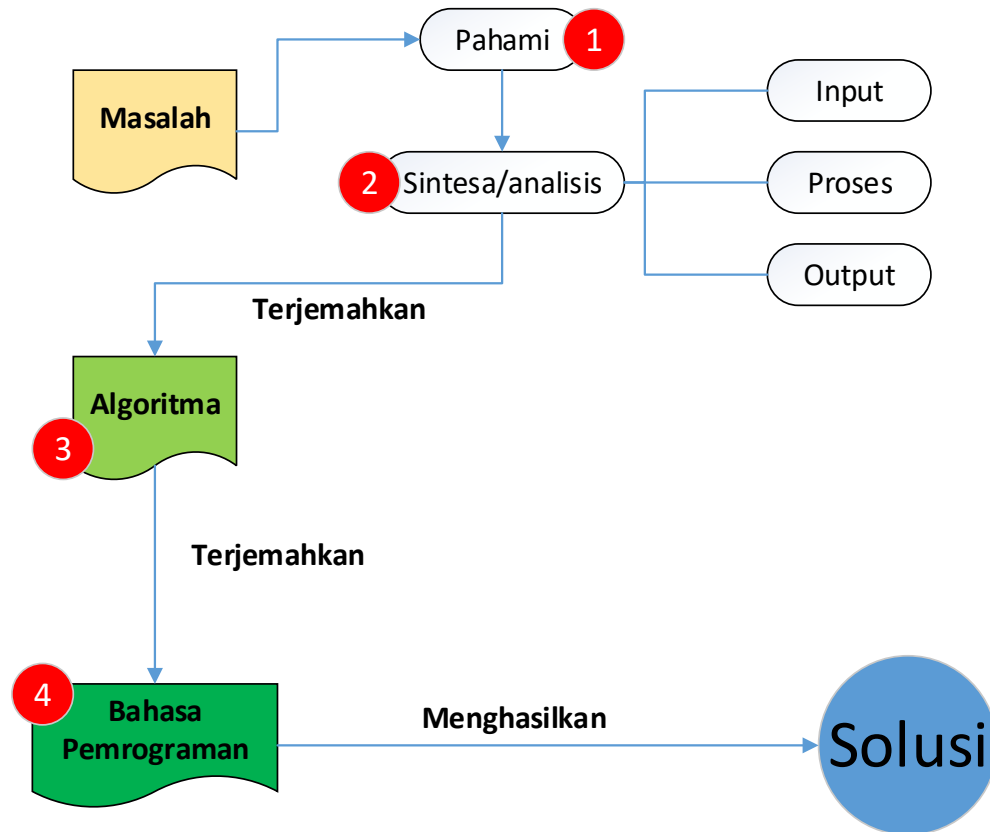
Algoritma
    input (S)
    Luas ← S * S { (proses) hitung luas bujursangkar }
    output (Luas) {menampilkan luas bujur sangkar}
```

Kode Sumber 1.1. Notasi Algoritmik Menghitung dan Menampilkan Luas Bujursangkar

Notasi input, ← (assignment), dan ouput merupakan salah satu bentuk notasi algoritmik yang digunakan untuk menuliskan algoritma menggunakan pseudo code, penjelasan secara rinci mengenai notasi algoritmik yang digunakan akan diberikan pada sub bab berikutnya.

Secara umum untuk menyelesaikan permasalahan secara algoritma diberikan dengan kerangka seperti yang diberikan pada Gambar 1.3. Untuk menyelesaikan permasalahan (persoalan) secara algoritma dapat dibagi menjadi 4 bagian tahapan-tahapan aktifitas utama yang terdiri dari sebagai berikut :

1. Pahami permasalahan
2. Lakukasn sintesa/analisis permasalahan (petakan ke dalam input, proses, dan output)
3. Buatlah sketsa solusi dalam bentuk algoritma menggunakan notasi algoritmik
4. Terjemahkan algoritma ke dalam bahasa pemrograman



Gambar 1.3. Konstruksi Penyelesaian Permasalahan Secara Algoritma

Untuk menyelesaikan permasalahan guna mendapatkan sebuah solusi, dimulai dengan melakukan pemahaman terhadap persoalan tersebut. Kemudian setelah memahami dilakukan sintesa/analisis dengan cara melakukan identifikasi dan klasifikasi terhadap bagian sebagai input, proses, dan output. Selanjutnya dari hasil sintesa dilanjutkan dengan menuangkan ke dalam sebuah algoritma yang merupakan sketsa solusi secara global. Algoritma yang sudah dibuat harus dilakukan proses pengujian untuk mengetahui kebenaran dari algoritmanya. Kemudian dari algoritma yang telah dilakukan pengujian, agar dapat dijalankan pada sebuah sistem komputer, maka perlu dilakukan untuk menterjemahkan ke dalam sebuah bahasa pemrograman untuk menghasilkan sebuah solusi secara menyeluruh (utuh).

Pada contoh 1.2 untuk menghitung luas bujursangkar dan menampilkan luasnya, maka yang pertama-tama harus perlu dipahami adalah bagaimana konstruksi untuk menghitung luas bujursangkar untuk dilakukan sintesa dengan cara dipetakan menjadi bagian input, proses, dan output. Kemudian berdasarkan hasil sintesa tuliskan kedalam algoritma dengan menggunakan

notasi algoritmik, seperti pada Kode Sumber 1.1. Algoritma yang telah dibuat, lakukanlah pengujian kebenaran terhadap semua kemungkinan logika dengan beberapa kasus uji yang merepresentasikan persoalan. Kemudian setelah diyakini bahwa algoritma sudah benar dari semua kemungkinan logika, maka selanjutnya terjemahkan ke dalam salah satu Bahasa pemrograman (buku ini menggunakan bahasa C), seperti yang diberikan pada Kode Sumber 1.2.

```
/*Nama File: luasbjsk.c*/  
/*Deskripsi: menghitung luas bujur sangkar dan menampilkan*/  
  
#include <stdio.h> /*Header file*/  
  
int main(){ /*Program Utama*/  
/*Kamus*/  
    int S; /*Sisi bujursangkar*/  
    int Luas; /*Luas bujursangkar*/  
/*Algoritma*/  
    printf("====Menghitung Luas Bujursangkar====\n");  
    printf("=====\n");  
    printf("\nMasukan Sisi (S) = ");  
    scanf("%d",&S); /*input*/  
    Luas=S*S; /*(proses) hitung luas bujursangkar*/  
    /*output*/  
    printf("\n=====\n");  
    printf("Luas bujursangkar (S x S)==> %d X %d =  
%d\n",S,S,Luas);  
    printf("=====\n");  
    return 0;  
}
```

Kode Sumber 1.2. Bahasa C Menghitung dan Menampilkan Luas Bujursangkar

Kode Sumber 1.2. adalah kumpulan perintah-perintah dari bahasa pemrograman C yang merupakan hasil terjemahan notasi algoritmik pada Kode Sumber 1.1. Penulisan perintah hasil terjemahan algoritma menggunakan bahasa pemrograman C (atau Bahasa pemrogram lain) menjadi lebih detil (tambahan variasi input dan output) agar program yang dibuat dapat dikomunikasikan ke pengguna. Kode sumber 1.2. pada saat dilakukan kompilasi menggunakan mesin kompiler untuk mengetahui kebenaran secara sintak penulisan yang diimplementasikan ke dalam bahasa pemrograman, jika secara sintak tidak ada yang salah maka akan dihasilkan seperti yang diberikan pada Gambar 1.4. Untuk mengetahui kebenaran logika program, perlu

dilakukan pengujian dengan cara mengamati hasil keluaran program terhadap masukan tertentu yang merepresentasikan persoalan.

```
=====Menghitung Luas Bujursangkar=====
Masukan Sisi (S) = 5
Luas bujursangkar (S x S)==> 5 X 5 = 25
```

Gambar 1.4. Hasil Kompilasi Program Luas Bujursangkar

Jika diberikan masukan sisi (S) sebesar 5, maka secara perhitungan luas bujursangkar diperoleh sebesar 25 yang diperoleh dari formula bahwa luas bujur sangkar $S \times S$, sehingga secara logika program yang telah dibuat sudah benar. Untuk memberikan keyakinan dalam melakukan pengecekan kebenaran logika program, perlu dilakukan pengujian beberapa kali dengan variasi masukan guna melakukan pengecekan semua kemungkinan logika program. Sangat tidak disarankan melakukan pengujian hanya satu kali kemudian memberikan kesimpulan bahwa program sudah benar, karena mungkin dengan satu kali pengujian tidak dapat memberikan gambaran secara utuh semua kemungkinan logika program dapat dijalankan secara benar.

Dalam melakukan penterjemahan algoritma ke dalam bahasa pemrograman, fokuskan pada algoritma utama, jangan terjebak focus pada tampilan. Tampilan memang diperlukan untuk dapat dikomunikasikan dengan pengguna, akan tetapi yang utama adalah kebenaran logika utama algoritma. Setelah logika utama algoritma sudah diyakini benar secara keseluruhan, maka dapat dikembangkan dengan mempercantik tampilan agar komunikasi dengan pengguna dapat memberikan kemudahan dan kenyamanan.

BAB II

Nama, Tipe Data, Konstanta, dan Variabel

Sebelum memulai membuat sebuah algoritma, hal utama yang dibutuhkan adalah harus terlebih dahulu memahami 4 komponen dasar, yang meliputi: nama, tipe data, konstanta, dan variable. Oleh karena itu pada bagian ini dibahas secara detail mengenai 4 komponen dasar di atas yang dilengkapi dengan beberapa contoh kasus penerapannya dalam membuat sebuah algoritma sebagai sketsa solusi.

2.1. Nama

Nama adalah merupakan pemberian label pada sesuatu agar dapat dikenali pada sebuah lingkungan yang digunakan (dalam hal ini pada sebuah algoritma). Nama memiliki sifat unik pada satu body algoritma, sehingga nama hanya dapat sekali digunakan pada satu body algoritma agar tidak terjadi konflik nama. Pemberian nama ada yang bersifat case sensitive atau tidak tergantung dari mekanisme bahasa pemrograman yang digunakan. Namun untuk mempermudah dalam melakukan organisasi sebuah nama, sangat disarankan untuk selalu menggunakan prinsip bahwa nama adalah bersifat case sensitive. Pemberian nama sebaiknya dapat merepresentasikan sesuatu yang diberikan nama (jika mungkin), tetapi jika tidak memungkinkan dapat menggunakan sembarang dengan ketentuan diberikan informasi sebagai komentar agar dapat mewakili kondisi yang direpresentasikan. Pemberian nama tidak dapat menggunakan spasi, karakter pertama angka, karakter lainnya (% , ^ , ~ , @ , # , dan lain-lainnya), dan nama yang sudah digunakan oleh mekanisme bahasa pemrograman atau mesin yang digunakan. Jika dalam memberikan nama tetap menginginkan spasi, maka spasi dapat diganti dengan tanda under score (_). Nama dalam algoritma atau bahasa pemrograman digunakan untuk menyimpan sebuah nilai sesuai dengan representasi yang didefinisikan. Nilai yang digunakan pada sebuah nama adalah nilai yang terakhir kali diberikan pada sebuah nama.

Contoh 2.1.

Berikut diberikan contoh-contoh pembuatan nama

Tinggi	(diperbolehkan)
BeratBadan	(diperbolehkan)
Kepemilikan12	(diperbolehkan)
Rata_rata	(diperbolehkan)
Rata2Val	(diperbolehkan)
Nilai Rata	(tidak diperbolehkan)
1jumlah	(tidak diperbolehkan)
Jumlah%	(tidak diperbolehkan)

Berdasarkan contoh 2.1, dalam membuat atau memberikan nama pada sebuah body algoritma atau program, baik meliputi nama konstanta, variable, program, fungsi, dan prosedur ikutilah kaedah dalam pemberian nama seperti yang diberikan pada contoh-contoh di atas.

2.2. Tipe Data

Tipe data adalah merupakan klasifikasi data agar dapat direpresentasikan pada sebuah sistem komputer. Dengan kata lain bahwa tipe data digunakan untuk menentukan jenis nilai yang akan disimpan di memori komputer dan akan diproses oleh sebuah algoritma atau program. Tipe data terdiri dari beberapa jenis tipe data tergantung dari mekanisme yang disediakan pada bahasa pemrograman. Namun pada buku algoritma ini, bahwa tipe data yang disediakan hanya tipe data dasar dan utama saja, dan untuk yang diluar tipe data dasar diserahkan pada mekanisme bahasa pemrograman yang digunakan. Tipe data secara garis besar terdiri dari 2, yaitu tipe data dasar dan tipe data bentukan.

Tipe Data Dasar pada buku algoritma ini yang didefinisikan terdiri dari Integer, Real, Character, String, dan Boolean.

Integer (simbol pada matematika adalah **Z**) adalah merupakan tipe data berbentuk numerik yang terdiri dari bilangan bulat positif (1, 4, 400, 1002, ...), nol (0), dan bulat negatif (-3, -34, -403,

-1003, ...). Operasi aritmatika yang dapat dijalankan pada tipe data integer terdiri dari +, -, *, / (pembagian), ^ (pangkat), % (modulus/sisa hasil bagi). Operasi perbandingan yang dapat dijalankan pada tipe data integer terdiri dari =, >, <, >=, <=, <> (tidak sama dengan).

Real (simbol pada matematika adalah **R**) adalah merupakan tipe data berbentuk numerik yang nilai bilangannya dapat dituliskan dengan decimal (0.0, 2.0, 3.2, 100.67, 2009,222, ...). Operasi aritmatika yang dapat dijalankan pada tipe data real terdiri dari +, -, *, / (pembagian), ^ (pangkat). Operasi perbandingan yang dapat dijalankan pada tipe data real terdiri dari =, >, <, >=, <=, <> (tidak sama dengan).

Character merupakan tipe data dalam bentuk karakter atau huruf tunggal yang ditulis dengan diawali dan diakhiri tanda petik tunggal ('a', 'b', 'c', 'x', 'Y', 'Z', '1', '34', '?', ...). Operasi yang dapat dijalankan pada tipe data character adalah operasi + (penggabungan). Operasi perbandingan yang dapat dijalankan pada tipe data character terdiri dari =, >, <, >=, <=, ≠ (tidak sama dengan). String merupakan tipe data kumpulan dari karakter yang dituliskan dengan cara dimulai dan di akhiri menggunakan tanda petik tunggal ('nama', '22223', 'saya tak tahu', 'nomor234ok', ...).

Boolean merupakan tipe data untuk menyatakan kebenaran logika yang nilainya hanya terdiri dari True (1) dan False (0). Operasi logika yang dapat dijalankan pada tipe data Boolean terdiri dari =, >, <, >=, <=, ≠, NOT, AND, OR, dan XOR.

Tipe Bentukan merupakan sebuah tipe data yang dibentuk dari kumpulan tipe-tipe dasar. Pembentukan tipe bentukan ini didasarkan pada sifat sesuatu yang akan dibentuk menjadi sebuah tipe bentukan. Ide dasar pada tipe bentukan adalah terinspirasi dari pada saat membuat program ada kebutuhan untuk menggunakan tipe, tetapi tipe ini tidak disediakan oleh mekanisme bahasa pemrograman. Misalnya tipe Titik, tipe ini tidak disediakan oleh mekanisme bahasa pemrograman. Untuk membuat tipe Titik harus merepresentasikan sifat-sifat dari titik, yaitu titik terdiri dari (memiliki) absis dan ordinat. Absis sebagai tipe dasar dengan tipe integer, dan ordinat sebagai tipe dasar dengan tipe integer juga. **Tipe Titik** sebagai tipe bentukan memiliki 2 komponen tipe dasar sebagai absis dan ordinat. Misalkan contoh lain adalah **tipe Jam** dengan versi tanpa am|pm, jam berdasarkan sifatnya terdiri dari komponen Jam (integer), Menit

(integer), dan Detik (integer). **Tipe Jam** dengan versi am|pm dapat dibuat menjadi sebuah tipe bentukan dengan komponen terdiri dari Jam (integer), Menit (integer), dan Detik (integer), ampm (string).

Dengan sebuah tipe bentukan dapat pula dibentuk menjadi tipe bentukan lainnya, misalkan dari tipe Titik dapat dibuat tipe Garis, hal ini dilandasi pada definisi dari sebuah garis dapat dibentuk dari 2 titik. Oleh karena itu tipe Garis memiliki 2 elemen (komponen) yang bertipe bentukan Titik. Contoh lain dari tipe Titik dapat dikembangkan menjadi tipe Segitiga, dimana untuk setiap segitiga dapat dibentuk dari 3 buah titik.

Berdasarkan contoh-contoh tipe bentukan yang diberikan di atas, maka sangat disarankan dalam membuat tipe bentukan sebaiknya dapat dipandang menjadi sebuah tipe yang sifatnya sangat generic, sehingga dapat digunakan pada algoritma atau program yang lainnya. Dalam membuat tipe bentukan tidak cukup didasari pada sebuah kebutuhan, akan tetapi esensinya adalah pada sebuah bentuk yang generic dalam implementasinya.

2.3. Konstanta

Konstanta merupakan sebuah besaran yang nilainya tetap (tidak pernah berubah) pada pemrosesan algoritma. Dalam matematika sebuah besaran yang merupakan konstanta diantaranya: $\Phi = 3,14159265358979323846$, Konstanta Hubble = 70 (km/s)/Mpc, kecepatan cahaya = 299.792.458 meter per detik, konstanta Planck = 6.626×10^{-34} Joule detik, dan lain-lainnya.

Dalam menyelesaikan (membuat) sebuah algoritma konstanta yang digunakan tidak hanya seperti konstanta yang sudah ditetapkan secara baku (seperti pada konstanta matematika di atas), akan tetapi dapat dibuat dengan cara user define sesuai dengan kebutuhan dan sifat persoalan yang akan diselesaikan. Konstanta harus direpresentasikan menggunakan sebuah tipe data sesuai dengan nilai konstanta yang akan diberikan untuk disimpan di memori komputer.

2.4. Variabel

Variabel merupakan sebuah besaran yang nilainya dapat berubah-ubah selama pemrosesan sebuah algoritma atau program. Variabel harus direpresentasikan menggunakan sebuah tipe data sesuai dengan nilai variabel yang akan diberikan untuk disimpan di memori komputer. Nilai sebuah variabel yang dapat berubah-ubah selama pemrosesan, akan tetapi untuk nilai yang digunakan adalah nilai paling terakhir pada perintah yang dilakukan eksekusi. Misalkan diberikan sederetan perintah terhadap variabel iApw: $iApw=0$, $iApw= iApw+10$, $iApw= iApw-2$, $iApw= iApw*10$, $iApw=4$, maka pada akhir eksekusi nilai variabel iApw adalah 4. Nilai 4 adalah nilai yang terakhir kali dilakukan eksekusi, meskipun sebelum perintah terakhir variabel iApw dilakukan beberapa operasi tetap tidak mempengaruhi nilai variabel iApw.

Sebagai contoh untuk menyimpan nilai tinggi badan seseorang, diperlukan untuk membuat sebuah variabel dengan nama TinggiBadan. Kemudian agar variabel TinggiBadan dapat direpresentasikan ke dalam sistem komputer, maka harus diberikan tipe data. Untuk memberikan jenis tipe data pada sebuah variabel, harus diamati mengenai sifat-sifat datanya secara generik. Faktanya tinggi badan seseorang besarnya nilai tidak selalu bulat, akan tetapi dapat juga dalam bentuk decimal, sehingga tipe data yang diberikan pada variabel TinggiBadan adalah Real. Akibat yang ditimbulkan jika kurang tepat dalam memberikan sebuah tipe data pada sebuah variabel akan berakibat informasinya menjadi tidak tepat atau kehilangan informasi yang sebenarnya. Sebagai contoh jika variabel TinggiBadan diberikan tipe data Boolean, sehingga semua orang tinggi badan yang disimpan di memori komputer hanya berisi True atau False. Dari sini sangatlah jelas bahwa informasi tinggi badan menjadi hilang atau tidak tepat mewakili kondisi yang sebenarnya.

Secara umum penggunaan pada bab ini terdapat pada bagian pendefinisian konstanta, dan variabel yang ditempatkan pada kamus sebagai bagain yang dikenal dengan nama deklarasi pada body algoritma, seperti yang diberikan pada Kode Sumber 2.1. Untuk mendefinisikan nama sangat disarankan jika nama yang tidak mudah untuk dikenali harus disertakan komentar yang digunakan sebagai deskripsi singkat dari nama konstanta atau variabel, akan tetapi sangat disarankan dalam membuat nama sebaiknya disertai dengan deskripsi sebagai komentar agar algoritma yang dibuat dapat dengan mudah untuk dikomunikasikan. Tujuan dari pemberian

deskripsi pada komentar adalah untuk mempermudah dalam memahami nama yang dimaksud untuk digunakan pada body algoritma.

```
{Deklarasi kamus yang terdiri nama, tipe data, konstanta dan variabel}
```

Kamus

```
S      : integer { Variabel, Sisi bujursangkar }  
Luas: integer { Variabel, Luas bujursangkar }  
constant Phi: integer = 3.14159265358979323846  
                        {Konstanta Phi}
```

Kode Sumber 2.1. Deklarasi pada Kamus

BAB III

Notasi Algoritmik

Untuk membuat sebuah algoritma seperti yang telah diberikan pada bab sebelumnya dapat dilakukan dengan menggunakan 3 cara, yang meliputi: bahasa natural, flowchart, dan pseudo code. Dalam merepresentasikan algoritma dari 3 cara yang disebutkan di atas tidak ada sebuah keharusan untuk menggunakan salah satu cara tersebut, artinya diberikan sebuah kebebasan berdasarkan cara yang paling familiar dan mudah bagi masing-masing orang. Akan tetapi pada buku ini lebih menekankan pada keseragaman dalam penggunaan cara untuk menulis algoritma, yaitu menggunakan cara dalam bentuk pseudo code. Pseudo code yang digunakan pada buku ini adalah menggunakan Notasi Algoritmik yang dikembangkan oleh Liem (2008). Notasi algoritmik yang dikembangkan oleh Liem (2008) tidak sepenuhnya diadopsi pada buku ini, akan tetapi ada beberapa notasi yang tidak digunakan atau dilakukan modifikasi serta dilakukan penambahan untuk memenuhi kebutuhan dalam merepresentasikan penulisan algoritma. Tujuan menggunakan notasi algoritmik adalah untuk lebih mempermudah dalam proses untuk melakukan penterjemahan dari algoritma ke dalam bahasa pemrograman, hal ini dikarenakan bahwa notasi algoritmik yang digunakan lebih mendekati dengan sintak-sintak pada bahasa pemrograman (perlu melakukan penyesuaian dengan sintak dari masing-masing bahasa pemrograman yang digunakan, seperti bahasa C, turbo pascal, lisp, python, ada, .Net, dan lain sebagainya). Perbedaan notasi algoritmik dengan bahasa pemrograman adalah lebih pada kepemilikan mesin compiler, untuk notasi algoritmik tidak memiliki mesin compiler (sangat tergantung pada logika pembuat), sedangkan bahasa pemrograman memiliki mesin compiler. Pada bagian ini selain diberikan notasi algoritmik juga dilengkapi dengan terjemahan ke dalam bahasa pemrograman, yaitu bahasa C. Adapun secara umum beberapa notasi algoritmik yang digunakan pada buku ini diberikan pada tabel 3.1.

Tabel 3.1. Notasi Algoritmik dan Terjemah Bahasa C

No	Notasi Algoritmik	Bahasa C
1.	Assignment $\langle \text{nama} \rangle \leftarrow \text{nilai}$	<code>nama = nilai;</code>
2.	Input/Output Piranti Masukan <code>input (nama)</code> Pembacaan File <code>read (nama)</code> Piranti Keluaran <code>output (nama)</code> Penulisan File <code>write (nama)</code>	<code>scanf (format, &nama);</code> <code>fscanf (stream, format, &nama)</code> <code>printf (format, nama);</code> <code>fprintf (stream, format, &nama)</code>
3.	Kondisional (Analisa kasus) Satu Kasus <u>if</u> $\langle \text{kondisi} \rangle$ <u>then</u> Aksi_1	<code>if (kondisi){</code> Aksi_1; <code>}</code>
	Dua Kasus <u>if</u> $\langle \text{kondisi} \rangle$ <u>then</u> Aksi_1 <u>else</u> Aksi_2	<code>if (kondisi){</code> Aksi_1; <code>}</code> <code>else{</code> Aksi_2; <code>}</code>
	Lebih Dari Dua Kasus <u>depend on</u> $\langle \text{nama} \rangle$ kondisi_1 : Aksi_1 kondisi_2 : Aksi_2 kondisi_3 : Aksi_3 . . . Kondisi_N : Aksi_N {Kondisi_N sama dengan NOT kondisi_1 AND NOT Kondisi_2 AND NOT Kondisi_3 AND ... NOT Kondisi_N-1 atau dapat menggunakan else }	<code>if (kondisi_1){</code> Aksi_1; <code>}</code> <code>else if (Kondisi_2){</code> Aksi_2; <code>}</code> <code>else if (Kondisi_3){</code> Aksi_3; <code>}</code> . . . <code>else{</code> Aksi_N; <code>}</code>
	Untuk <u>depend on</u> dengan nilai Kondisi_1 s/d Kondisi_N berbentuk: $\langle \text{namaVar} \rangle = \langle \text{ekspresi konstan} \rangle$	<code>switch (namaVar){</code> case eksp_konstan_1 : Aksi_1; break;

No	Notasi Algoritmik	Bahasa C
		<pre> case eksp_konstan_2 : Aksi_2; break; case eksp_konstan_3 : Aksi_3; break; . . . case eksp_konstan_N : Aksi_N; break; } </pre>
4.	Perulangan While <u>while</u> <kondisiUlang> <u>do</u> Aksi {endWhile}	<pre> while (kondisiUlang){ Aksi; } </pre>
	Repeat - Until <u>repeat</u> Aksi <u>until</u> <kondisiStop>	<pre> do{ Aksi; } while (kondisiUlang) </pre>
	Traversal <u>i traversal</u> [Awal..Akhir] Aksi Untuk lompatan lebih dari satu tidak dapat menggunakan Traversal, gunakanlah while atau repeat-until.	<pre> /*jika Awal < Akhir*/ for (i=Awal;i<=Akhir;i++){ Aksi; } /*jika Awal > Akhir*/ for (i=Awal;i>Akhir;i--){ Aksi; } </pre>
5.	Program Utama Program NamaProgram {Deskripsi singkat program} Kamus Algoritma	<pre> int main(){ /*Nama File: NamaProgram.c*/ /*Deskripsi: -----*/ /*Kamus*/ /*Algoritma*/ return 0; /*termination*/ } </pre>
6.	Sub Program Fungsi Function NamaFungsi(<listParameterIN>) → <typeHasil>	<pre> <typeHasil> NamaFungsi(<listParameterIN>){ /*Deskripsi singkat fungsi*/ } </pre>

No	Notasi Algoritmik	Bahasa C
	{Deskripsi singkat fungsi} Kamus Lokal Algoritma → Hasil	/*Kamus Lokal*/ /*Algoritma*/ return Hasil; }
	Prosedur <u>Procedure</u> NamaProsedur(<listParameter>) {Deskripsi singkat prosedur} {IS:-----} {FS:-----} {Proses:-----} Kamus Lokal Algoritma	void NamaProsedur(<listParameter>) /*Deskripsi singkat prosedur*/ /*IS:-----*/ /*FS:-----*/ /*Proses:-----*/ /*Kamus Lokal*/ /*Algoritma*/
7.	Tipe Bentuk <u>type</u> NamaTypeBentukan : < elemen_1 : tipeDasar_1, elemen_2 : tipeDasar_2, ... elemen_N : tipeDasar_N >	typedef struct { tipeDasar_1 elemen_1, tipeDasar_2 elemen_2, ... tipeDasar_N elemen_N } NamaTypeBentukan;

Untuk menuliskan sebuah teks algoritma dengan menggunakan notasi algoritmik secara umum digunakan acuan nomor 5 pada Tabel 3.1. Adapun secara lebih rinci untuk menuliskan sebuah teks algoritma dibagi menjadi 3 bagian yang terdiri dari bagian-bagian sebagai berikut:

1. Judul (header)
2. Kamus,
3. Algoritma

Untuk secara keseluruhan dalam menuliskan sebuah teks body algoritma dapat disajikan dengan bentuk sebagai berikut:

Judul NamaProgramFungsiProsedur
{Spesifikasi teks algoritma secara umum }

Kamus

{bagian ini digunakan untuk mendefinikan tipe, nama konstanta, nama variabel, nama dan spesifikasi fungsi dan prosedur}

Algoritma

{bagian ini digunakan untuk menuliskan teks body algoritma yang dapat berupa sequence, Analisa kasus, perulangan sesuai dengan persoalan yang dibuat algoritmanya}

{semua teks yang tidak dituliskan diantara tanda kurung kurawal {-----} dianggap sebagai notasi algoritmik. Bagian yang dituliskan diantara kurung kurawal {-----} dianggap sebagai komentar, yang tidak ikut dieksekusi pada pemrosesan algoritma}

Judul (Header) bagian dari teks algoritma untuk menspesifikasikan bahwa teks algoritma ini merupakan sebuah program utama, sub program dalam bentuk fungsi, atau sub program dalam bentuk prosedur. Setelah menuliskan judul (header) sangat disarankan untuk menuliskan deskripsi singkat mengenai spesifikasi dari teks algoritma sebagai bagian dari dokumentasi algoritma. Adapun sebagai contoh bagian judul (header) diberikan pada Kode Sumber 3.1, 3.2, dan 3.3.

Program HitungLuasBjisk
{Menghitung dan menampilkan luas bujur sangkar}

Kode Sumber 3.1. Bagian Judul (Header) Program

Function HitungLuasBjisk (Sisi: integer) → integer
{Mengirimkan luas bujur sangkar}

Kode Sumber 3.2. Bagian Judul (Header) Fungsi

Procedure CetakTabel(input: T : Tabel)
{IS: sembarang}
{FS: semua elemen tabel T ditampilkan di layar}
{Proses: Menampilkan semua elemen Tabel T}

Kode Sumber 3.3. Bagian Judul (Header) Prosedur

Kamus bagian dari teks algoritma untuk mendefinisikan (deklarasi) tipe, nama konstanta beserta tipe dan nilainya, nama variabel beserta tipenya, nama fungsi beserta domain fungsi, parameter input dan spesifikasinya, dan nama prosedur beserta list parameter formal (jika ada) dan spesifikasi prosedur yang meliputi Initial State (IS), Final State (FS) dan proses yang dilakukan oleh prosedur tersebut. Semua nama yang akan digunakan pada body teks algoritma harus didefinisikan terlebih dahulu (sebelum digunakan harus didefinisikan) pada bagian kamus, meliputi konstanta, variabel, fungsi, dan prosedur. Nilai variabel pada saat didefinisikan belum terdefinisi harganya, akan tetapi untuk mendefinisikan nilai variabel dapat dilakukan dengan cara inisialisasi (pemberian nilai awal) menggunakan assignment. Adapun sebagai contoh bagian kamus diberikan pada Kode Sumber 3.4.

Kamus**{Nama Type hanya untuk type yang bukan type dasar}****type** Titik : <X:integer, Y:integer> {Koordinat kartesian}**{Nama Konstanta harus menyebutkan type dan nilainya}****constant** Phi : real = 3.14159265358979323846**{Nama Variabel harus menyebutkan type}**S : integer {Sisi bujursangkar}Luas: integer {Luas bujursangkar}Ketemu: boolean {hasil pencarian, jika ketemu nilai 1}Total: real {total jumlahan}Pilih: character {pilihan menu}

T: Titik {Koordinat kartesian}

{Spesifikasi fungsi harus menyebutkan nama fungsi, domain dan range}**Function** HitungLuasBjks (Sisi: integer) → integer

{Mengirimkan luas bujur sangkar}

{Spesifikasi prosedur harus menyebutkan nama prosedur, parameter (jika ada), IS, FS dan proses}**Procedure** CetakTabel(input: T : Tabel)

{IS: sembarang}

{FS: semua elemen tabel T ditampilkan di layar}

{Proses: Menampilkan semua elemen Tabel T}

Kode Sumber 3.4. Bagian Kamus

Algoritma bagian dari teks algoritma yang berisi mengenai semua instruksi atau pemanggilan aksi yang sudah terdefinisi. Komponen teks algoritma pada pemrograman procedural (terstruktur/modular) dapat berbentuk instruksi dasar (input, output, atau assignment), sequence, analisa kasus, dan atau perulangan. Adapun untuk contoh pada bagian algoritma diberikan pada Kode sumber 3.5.

```
Algoritma  
  j ← 1 {inisialisasi}  
  input(i)  
  k ← j + i {proses}  
  output(k)
```

Kode Sumber 3.5. Bagian Algoritma

Secara keseluruhan untuk menuliskan sebuah teks algoritma dari sebuah persoalan dituliskan dengan cara seperti pada Kode Sumber 3.6. Kode Sumber 3.6 merupakan persoalan yang diketahui sebuah sisi bujur sangkar (S) integer sembarang yang dimasukan melalui piranti masukan (keyboard), maka hitung dan tampilkan di piranti keluaran (layer) luas bujur sangkar (Luas) tersebut.

```
Program luasbjsk  
{menghitung dan menampilkan luas bujur sangkar}  
  
Kamus  
  S      : integer {Sisi bujursangkar}  
  Luas: integer {Luas bujursangkar}  
  
Algoritma  
  input(S)  
  Luas ← S * S {(proses) hitung luas bujursangkar}  
  output(Luas) {menampilkan luas bujur sangkar}
```

Kode Sumber 3.6. Teks Algoritma Menghitung dan Menampilkan Luas Bujur Sangkar

BAB IV

Sequences

Sequences merupakan urutan eksekusi instruksi dan atau aksi oleh sistem komputer berdasarkan urutan penulisan. Sistem komputer akan melakukan pembacaan instruksi di mulai dari urutan instruksi pertama ke urutan instruksi berikutnya sampai dengan urutan instruksi yang paling terakhir. Penyajian urutan instruksi dapat dilakukan dengan menyusun instruksi dari atas ke bawah, atau dapat juga dengan cara menyamping sebelah kanan dengan cara memberikan pemisah tanda ; (titik koma). Untuk menuliskan urutan instruksi dengan cara menyamping sebelah kanan, sebaiknya hanya dilakukan pada instruksi-instruksi yang secara logika pada saat di tukar urutannya tidak mempengaruhi nilai. Akan tetapi jika urutan instruksi yang ditukar akan mempengaruhi logika sangat disarankan untuk tetap menuliskan dari atas ke bawah. Pada Kode Sumber 3.5. instruksi $j \leftarrow 1$ dan input(i) merupakan instruksi yang ditukar urutannya tidak mempengaruhi nilai, sehingga dapat dituliskan dengan menggunakan tanda titik koma disamping kanan. Perhatikan pada instruksi $k \leftarrow j + i$ dan output(k), dua instruksi ini jika ditukarkan urutannya dengan urutan instruksi lain, maka akan mempengaruhi nilai, sehingga dua instruksi ini tidak dapat dituliskan dengan tanda titik koma harus dituliskan dari atas ke bawah, sehingga pada Kode Sumber 3.5. dapat dituliskan dengan penulisan seperti pada Kode Sumber 4.1.

Algoritma

```
j ← 1; {inisialisasi} input(i)
k ← j + i {proses}
output(k)
```

Kode Sumber 4.1. Sequences Dapat Menggunakan Tanda Titik Koma

Misalkan pada Kode Sumber 3.5. dilakukan perubahan urutan instruksinya menjadi seperti yang dituliskan pada Kode Sumber 4.2. dengan cara merubah urutan instruksi output(k) ditempatkan sebelum instruksi $k \leftarrow j + i$.

Algoritma

```
j ← 1 {inisialisasi}
input(i)
output(k)
k ← j + i {proses}
```

Kode Sumber 4.2. Sequences Tidak Dapat Menggunakan Tanda Titik Koma

Pada Kode Sumber 3.5. nilai k yang ditampilkan dilayar sebesar nilai i yang dimasukan melalui keyboard ditambah dengan 1, misalkan nilai $i=4$, maka nilai k akan diperoleh $k = 1 + 4$, yaitu $k = 5$. Akan tetapi nilai k pada Kode Sumber 4.2. berapapun nilai i yang dimasukan melalui keyboard, nilai k selalu sama dengan nilai alamat memori (misalkan pada alamat, $k = 14618580$). Hal ini diakibatkan karena nilai k di outputkan pada layar terlebih dahulu sebelum dioperasikan dengan instruksi $k \leftarrow j + i$, sehingga nilai k yang ditampilkan pada layar adalah mengambil nilai default tipe integer, yaitu nilai 0. Pada kasus ini disebut dengan kasus merubah urutan instruksi akan memberikan pengaruh pada sebuah nilai, jadi perlu kehati-hatian dalam menuliskan urutan instruksi agar nilai yang dikeluarkan oleh algoritma tidak menjadi salah. Untuk lebih dapat memahami mengenai sequences, maka selanjutnya diberikan contoh-contoh yang disertai dengan cara untuk menyelesaikan sebuah algoritma dari sebuah persoalan yang diberikan.

Contoh 4.1.

Dibaca 2 buah nilai F (gaya, newton), dan S (jarak, meter) integer sembarang melalui masukan sebuah keyboard yang mewakili besarnya usaha dan energi W (newton meter atau joule). Hitung dan tampilkan di layar besarnya usaha dan energi (W) yang dihasilkan oleh 2 besaran gaya (F) dan jarak (S).

Untuk dapat membuat algoritma dari persoalan yang diberikan pada contoh 4.1., langkah pertama yang harus dilakukan adalah memahami persolan. Persolan ini mengenai persoalan untuk menghitung dan menampilkan besarnya usaha dan energi (W) dengan diketahui gaya (F) dan jarak (S). untuk menghitung besarnya usaha dan energi diketahui dengan menggunakan rumus W (newton meter atau joule) = F (newton) x S (meter).

Langkah berikutnya setelah dapat memahami persoalan, maka lakukanlah sintesa (analisis) persoalan dengan dengan cara melakukan pemetaan pada bagian input, proses dan output. Sintesa permasalahan pada contoh 3.1. diberikan sebagai berikut:

Input : F (gaya, newton), dan S (jarak, meter)

Proses : untuk menghitung usaha dan energi diketahui dengan menggunakan formula
$$W \text{ (newton meter atau joule)} = F \text{ (newton)} \times S \text{ (meter)}$$

Output : menampilkan besarnya usaha dan energi **W** (newton meter atau joule) di layar

Berdasarkan sintesa (analisis) permasalahan yang diberikan diatas, yang perlu diperhatikan adalah semua variabel, dan konstanta (jika ada) yang terdapat pada bagian input, proses, dan output harus didefinisikan pada kamus yang disertai dengan tipenya agar dapat direpresentasikan pada sistem komputer. Untuk melakukan sintesa (analisa) tidak harus seperti yang diberikan pada contoh di atas, dapat dengan cara lain akan tetapi selalu difokuskan pada pemahaman terhadap persoalan serta dapat melakukan pemetaan pada bagian input, proses, dan output. Cara lain yang dimaksud adalah dapat dilakukan dengan membuat sketsa secara corat-coret menggunakan kertas sesuai dengan kebiasaan atau kesukaan masing-masing dalam memahami terhadap persoalan dan melakukan sintesa persoalan.

Adapun teks algoritma yang dapat dituliskan dengan menggunakan notasi algoritmik yang telah diberikan pada bagian sebelum diberikan sebagai berikut:

```
Program usahal
{Menghitung dan menampilkan usaha W dilayar,  $W=F \times S$ }

Kamus
  F: integer {gaya, newton}
  S: integer {jarak, meter}
  W: integer {usaha, newton meter atau joule}

Algoritma
  input (F) {input}
  input (S)
   $W \leftarrow F * S$  {proses, dengan memorisasi}
  output (W) {output}
```

Kode Sumber 4.3. Hitung dan Tampilkan Usaha Versi 1

Program usaha1 pada Kode Sumber 4.3, untuk memberikan inputan terhadap variabel F dan S dilakukan secara terpisah, kemudian untuk menghitung usaha dilakukan dengan cara memorisasi, yaitu nilai $F \times S$ di simpan terlebih dahulu pada variabel W. Kemudian variabel W yang menyimpan nilai usaha ($F \times S$) dikeluarkan sebagai nilai output. Selanjutnya untuk menguji kebenaran algoritma, maka perlu dilakukan dengan mencoba memberikan masukan nilai F dan S (dilakukan terhadap semua kemungkinan logika) dan kemudian lakukan pengamatan terhadap nilai W apakah sudah sesuai hasil keluarannya berdasarkan proses operasi yang dijalankan, yaitu $W = F \times S$. Algoritma tidak memiliki mesin kompilasi, sehingga untuk mengetahui kebenaran logikanya harus dilakukan oleh pembuat algoritma (programmer) sendiri. Jika algoritma yang dituliskan sudah diyakini kebenaran logikanya, kemudian teks algoritma tersebut harus dapat dikomunikasikan dengan sistem komputer, yaitu dengan cara melakukan penterjemahan ke dalam sebuah bahasa pemrograman (pada buku ini yang digunakan adalah bahasa C). Langkah terakhir setelah diterjemahkan ke dalam bahasa pemrograman, untuk melakukan pengecekan kebenaran program, haruslah dilakukan kompilasi menggunakan kompiler bahasa pemrograman yang digunakan. Jika belum lolos kompilasi oleh kompiler harus di trace pada baris-baris perintah yang masih belum sesuai dengan tata cara penulisan (sintak) dari bahasa pemrograman untuk dilakukan perbaikan sampai semua memenuhi kaedah sintak pada bahasa pemrograman. Kemudian setelah semua sintak lolos kompilasi, selanjutnya lakukan pengecekan kebenaran logika terhadap semua kemungkinan logika dengan cara mencoba memberikan sebuah masukan dan mengamati proses dan outputnya sebanyak beberapa kali (jangan hanya satu kali saja, atau bahkan tidak sama sekali). Jika semua kemungkinan logika yang diamati sudah benar, maka program dapat diberikan kesimpulan sudah benar.

Satu hal yang perlu diperhatikan, untuk menterjemahkan teks algoritma ke dalam sebuah bahasa pemrograman, fokus utama adalah pada logika utama algoritma. Abaikan terlebih dahulu mengenai tampilan sebagai aksesoris yang dikomunikasikan ke pengguna. Tampilan sebagai aksesoris mulai diberikan jika logika utama algoritma sudah dipastikan kebenarannya melalui proses testing (tested).

Demikian proses yang disebutkan di atas adalah merupakan sebuah prinsip dasar yang harus selalu digunakan sebagai pedoman untuk menyelesaikan semua persoalan secara algoritma, yang harus diterapkan pada setiap aktifitas dalam melakukan coding.

```
Program usaha2
{Menghitung dan menampilkan usaha W dilayar,  $W=F \times S$ }

Kamus
F: integer {gaya, newton}
S: integer {jarak, meter}
W: integer {usaha, newton meter atau joule}

Algoritma
input (F,S) {input}
 $W \leftarrow F * S$  {proses, dengan memorisasi}
output (W) {output}
```

Kode Sumber 4.4. Hitung dan Tampilkan Usaha Versi 2

Program usaha2 pada Kode Sumber 4.4. dalam menuliskan teks algoritma secara prinsip adalah sama seperti pada program usaha1, hanya perbedaannya terdapat pada cara memberikan inputan variabel F dan S dilakukan secara Bersama-sama. Pemberian nilai variabel inputan secara Bersama-sama harus hati-hati. Hal ini dikarenakan urutan pembacaan mengikuti urutan yang dituliskan pada algoritma (urutan harus sesuai). Jika tidak sesuai, maka hasil pembacaan akan menjadi tidak tepat yang berakibat nilai yang dikeluarkan menjadi tidak tepat, jadi harus hati-hati jangan sampai tertukar urutannya dan selalu perhatikan urutannya seperti yang dituliskan (didefisikan). Penulisan variabel inputan secara Bersama-sama atau sendiri-sendiri ini hanya merupakan sebuah gaya penulisan saja, tidak ada kaitannya dengan esensi tertentu. Program usaha2 masih sama seperti pada program usaha1, yaitu masih menggunakan variabel W untuk proses memorisasi yang akan digunakan sebagai output dari algoritma tersebut, dengan proses $W = F \times S$.

```
Program usaha3
{Menghitung dan menampilkan usaha W dilayar,  $W=F \times S$ }

Kamus
F: integer {gaya, newton}
S: integer {jarak, meter}
```

Algoritma

```
input (F, S) {input}  
output (F*S) {output, tanpa memorisasi}
```

Kode Sumber 4.5. Hitung dan Tampilkan Usaha Versi 3

Program usaha3 pada Kode Sumber 4.5. dalam menuliskan teks algoritma menggunakan cara penulisan untuk memberikan nilai inputan terhadap variabel F dan S dilakukan secara Bersamaan seperti pada program usah2. Kemudian program usaha3 tidak menggunakan proses memorisasi untuk menyimpan nilai usaha W, akan tetapi nilai usaha langsung diproses pada output dengan cara menuliskan formula usaha adalah $F \times S$. Sebuah variabel yang akan diproses atau dioutputkan dapat dilakukan dengan cara memorisasi atau tanpa memorisasi. Memorisasi akan lebih menguntungkan jika variabel tersebut akan digunakan secara berulang-ulang pada sebuah body teks algoritma, karena pada saat menggunakan cukup menuliskan secara singkat nama variabel memorisasinya tanpa harus mengulang-ulang ekspresi yang diberikan pada variabel memorisasi tersebut. Misalkan variabel memorisasinya menggunakan ekspresi yang banyak variabel yang digunakan dan tingkat kekomplekan nya tinggi, sehingga dengan cara memorisasi akan lebih cepat dapat menggunakannya cukup dengan cara memanggil nama variabel, dan cara ini lebih menjamin tingkat kesalahan penulisan ekspresi menjadi kecil akibat tanpa menulis kembali. Tanpa memorisasi akan lebih menguntungkan pada saat ekspresi nya pendek dan atau variabel tersebut hanya satu kali digunakan pada satu body teks algoritma. Penggunaan variabel memorisasi atau tanpa memorisasi harus dilakukan secara bijak tergantung dengan solusi algoritma yang akan diberikan berdasarkan pertimbangan yang diberikan di atas.

Program usaha4

```
{Menghitung dan menampilkan usaha W dilayar,  $W = F \times S$ }
```

Kamus

```
F: integer {gaya, newton}  
S: integer {jarak, meter}  
W: integer {usaha, newton meter atau joule}
```

Algoritma

```
output ('Masukan nilai Gaya (F)=') {input}  
input (F)
```

```
output('Masukan nilai Jarak (S)=')  
input(S)  
W <-- F * S {proses}  
output('Usaha (W) adalah sebesar=',W) {output}
```

Kode Sumber 4.6. Hitung dan Tampilkan Usaha Versi 4

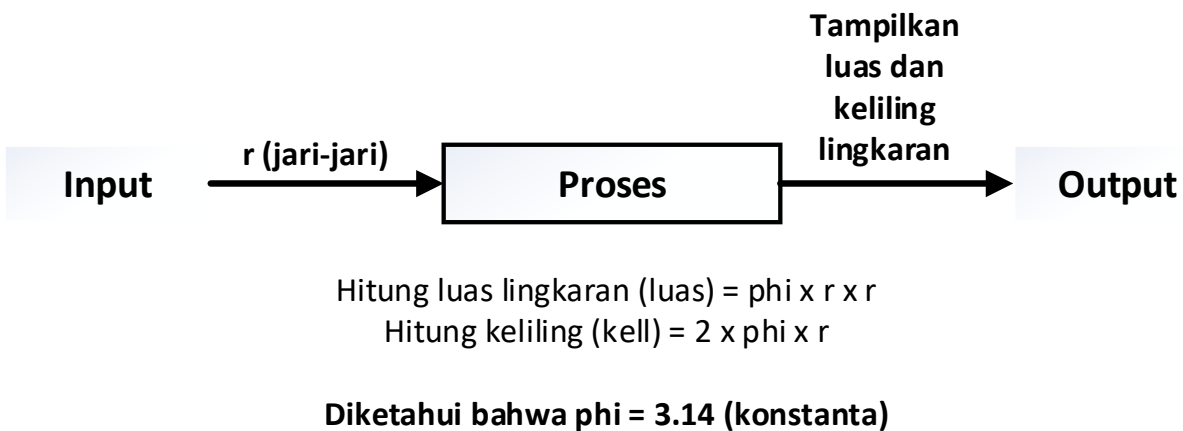
Program usaha4 pada Kode Sumber 4.6. dalam menuliskan teks algoritma secara konstruksi penulisan sama seperti pada program usaha1, akan tetapi pada program usaha4 cara penulisan dibuat menjadi lebih rinci. Cara penulisan yang rinci pada program usaha4 adalah **sangat tidak disarankan**, karena esensi membuat teks algoritma adalah merupakan sketsa solusi sehingga bentuknya lebih global yang fokus pada logika utama dari algoritma dari sebuah persoalan yang akan dijadikan sebagai solusi. Cara penulisan pada program usaha4 ini akan dilakukan pada proses menterjemahkan algoritma ke dalam bahasa pemrograman, karena hasil program harus dapat dikomunikasikan dengan pengguna. Algoritma merupakan sketsa global solusi terhadap persoalan, dan untuk detil solusinya adalah dilakukan dengan cara diterjemahkan ke dalam bahasa pemrograman. Perhatikan hal ini dan ingat selalu pergunakan sebagai pedoman dalam setiap menuliskan teks algoritma dan membuat program dengan bahasa pemrograman. Program usaha1, usaha2, dan usaha3 secara umum digunakan sebagai standart untuk membuat algoritma. Kemudian program usaha4 menjadi hal yang dilarang tidak diperbolehkan untuk digunakan dalam membuat sebuah teks algoritma.

Contoh 4.2.

Dibaca sebuah nilai r (jari-jari, meter) integer sembarang melalui masukan sebuah keyboard yang mewakili jari-jari pada sebuah lingkaran dalam meter. Hitung dan tampilkan di layar nilai luas dan keliling lingkaran tersebut dengan jari-jari sebesar r meter.

Dalam menyelesaikan contoh 4.2. diberikan cara lain untuk melakukan sintesa permasalahan, yaitu dengan cara membuat sebuah sketsa dalam bentuk coretan sebagai gambaran dari persoalan. Cara seperti ini tidak ada panduan sebagai standart yang digunakan, akan tetapi dapat dilakukan sebarang mungkin tergantung dari gaya dari setiap individu dalam melakukan abstraksi

terhadap sintesa yang akan dilakukan. Adapun sintesa persolan pada contoh 4.2. diberikan pada Gambar 4.1.



Gambar 4.1. Sintesa Permasalahan Luas dan Keliling Lingkaran

Berdasarkan sintesa permasalahan pada Gambar 4.1, maka teks algoritma yang dituliskan diberikan pada Kode Sumber 4.7. Semua variabel dan konstanta yang terdapat pada bagian input, proses, dan output dari proses sintesa permasalahan harus didefinisikan pada kamus yang disertai dengan definisi tipe.

```
Program luasKellLingk  
{Menghitung dan menampilkan di layar luas dan keliling lingkaran}  
{luas=Phi x r x r, dan keliling=2 x Phi x r}  
  
Kamus  
  r: integer {jari-jari, meter}  
  Luas: real {luas lingkaran, meter persegi}  
  Kell: real {keliling lingkaran, meter}  
  constant Phi: real = 3.14 {konstanta phi}  
  
Algoritma  
  input (r)  
  Luas  $\leftarrow$  Phi * r * r  
  Kell  $\leftarrow$  2 * Phi * r  
  output (Luas, Kell)
```

Kode Sumber 4.7. Hitung dan Tampilkan Luas dan Keliling Lingkaran

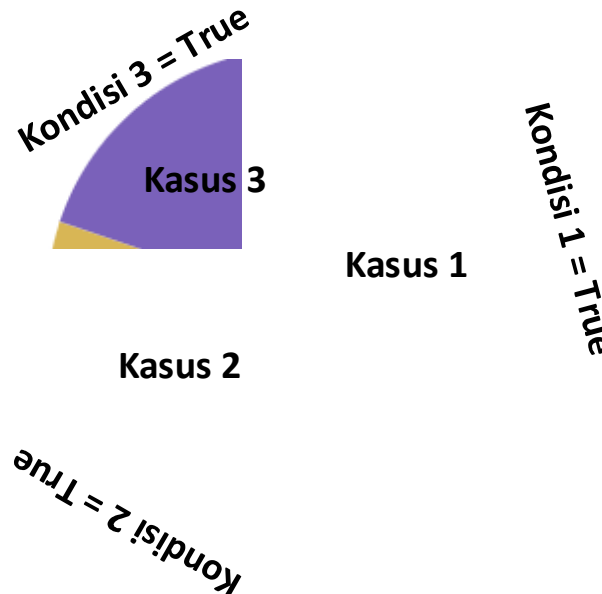
Perintah $Luas \leftarrow \Phi * r * r$, dan $Kell \leftarrow 2 * \Phi * r$ dapat dituliskan dengan menggunakan tanda koma, karena pada dua perintah tersebut dengan urutan dibolak-balik tidak akan memberikan pengaruh terhadap hasil logika pada algoritma yang dituliskan.

BAB V

Kondisional (Analisa Kasus)

Kondisional (analisa kasus) adalah merupakan sebuah bentuk penguraian terhadap satu masalah menjadi beberapa sub-sub masalah yang saling lepas (asing). Penguraian satu masalah menjadi sub-sub masalah harus merepresentasi enumerasi semua kemungkinan kasus-kasusnya. Penguraian menjadi beberapa sub-sub masalah dengan melakukan enumerasi semua kemungkinan kasus, hal ini terkait dengan adanya sebuah kondisional (persyaratan) yang harus dipenuhi (bernilai true pada operasi logika) untuk dapat membagi ke dalam sub-sub masalah yang saling asing, Istilah saling asing adalah setiap keanggotaan yang dimiliki oleh sesuatu hal hanya menjadi anggota bagian pada salah satu sub masalah (mengacu pada teori himpunan). Misalkan sebagai contoh diberikan persoalan mengenai jenis kelamin pada manusia. Persoalan utamanya adalah teletak pada jenis kelamin manusia, kemudian bagaimana membagi persoalan jenis kelamin menjadi beberapa sub-sub masalah yang saling asing dan merepresentasikan semua kemungkinan kasus. Persoalan jenis kelamin terdiri 2 kemungkinan kasus, yaitu laki-laki, dan perempuan (tidak ada jenis kelamin yang lainnya). Untuk dapat membagi seseorang itu termasuk laki-laki atau perempuan dibutuhkan sebuah kondisi yang sangat kuat untuk membedakannya. Misal kondisi yang digunakan adalah menggunakan anting untuk menilai seseorang adalah termasuk jenis kelamin laki-laki atau perempuan. Jika menggunakan anting adalah termasuk perempuan dan jika tidak menggunakan anting adalah laki-laki. Pemilihan kondisi menggunakan anting adalah kurang kuat karena ada mungkin dapat dijumpai ada seorang laki-laki yang menggunakan anting, sehingga kebenaran algoritma menjadi tidak tepat (bocor). Jika kondisi yang digunakan adalah rambut panjang, untuk seseorang yang berambut panjang adalah perempuan, dan yang berambut tidak panjang adalah laki-laki. Penggunaan kondisi berambut panjang juga kurang kuat karena sangat dimungkinkan dijumpai laki-laki dengan berambut panjang. Dari pemilihan kondisi seperti menggunakan anting dan berambut panjang

didasar adalah merupakan kondisi yang tidak kuat, karena sangat berpotensi untuk menghasilkan logika yang kurang tepat (bocor). Oleh karena itu, agar kondisi yang digunakan memiliki sifat sangat kuat dan tidak berakibat bocor pada algoritma, maka harus dipertimbangkan pemilihan kondisi yang benar-benar dapat memisahkan antara semua enumerasi kemungkinan kasus. Pada persoalan jenis kelamin, kondisi yang tidak akan memungkinkan dimiliki secara bersama oleh laki-laki atau perempuan adalah pada domain kelamin. Domain kelamin kondisinya adalah jika seseorang yang memiliki kelamin ♂ termasuk laki-laki. Jika seseorang yang memiliki jenis kelamin ♀ termasuk perempuan. Secara umum untuk persoalan kondisional (analisa kasus) dapat diberikan dengan ilustrasi pada Gambar 5.1.



Gambar 5.1. Abstraksi Kondisional (analisa kasus)

Kasus 1, kasus 2, dan kasus 3 pada Gambar 5.1. merupakan enumerasi semua kemungkinan kasus yang saling asing pada setiap persoalan. Setiap kasus tersebut memiliki spesifikasi dan karakteristik yang berbeda-beda antara satu kasus dengan kasus lainnya. Kondisi 1, kondisi 2, dan kondisi 3 harus bernilai True agar memenuhi spesifikasi dan karakteristik pada setiap kasus yang terdefinisi. Kondisi yang digunakan untuk membagi menjadi beberapa sub-sub permasalahan dapat terdiri 1 atau lebih dengan dikombinasikan menggunakan operasi logika. Secara notasi

algoritmik pada permasalahan kondisional terdiri dari beberapa bentuk yang diberikan dibawah ini.

Satu Kasus merupakan bagian dari ekspresi kondisional yang hanya memiliki 1 kondisi, atau hanya dapat diuraikan menjadi 1 sub masalah. Sebagai contoh pada permasalahan aktifitas mendengarkan musik yang dijalankan menggunakan media player. Aktifitas mendengarkan musik ada sebuah keinginan agar musiknya dapat didengarkan orang lain atau tidak. Pada persoalan ini hanya memiliki 1 kondisi yaitu agar tidak dapat didengarkan oleh orang lain, maka gunakan headset. Esensi aksi yang dijalankan jika kondisi terpenuhi (bernilai True) pada persoalan mendengarkan musik ini adalah terletak pada perlunya penggunaan headset atau tidak.

Adapun bentuk umum kondisional dengan 1 kasus diberikan sebagai berikut:

```
if <kondisi> then  
    Aksi_1
```

Kode Sumber 5.1. Notasi Algoritmik 1 Kasus

Kondisi berupa ekspresi logika dengan satu atau lebih ekspresi logika, sedangkan Aksi_1 merupakan perintah atau dapat terdiri dari kumpulan perintah yang akan dilakukan eksekusi, jika ekspresi kondisinya bernilai true. Ekspresi logika jika lebih dari satu dikombinasi dengan operator logika (AND, OR, NOT, XOR), sehingga nilai kebenaran ekspresi logika menggunakan kaedah nilai tabel kebenaran pada operator logika tersebut. Jika Aksi_1 terdiri dari lebih satu perintah maka sebaiknya diberikan tanda sebagai bagian dalam satu blok menggunakan komentar agar lebih mempermudah dalam membaca algoritma. Untuk setiap blok harus dituliskan dengan menggunakan tab agar lebih mempermudah pembacaan perintah yang dituliskan pada teks algoritma.

Contoh 5.1.

Dibaca *i* yang merupakan sebuah nilai integer sembarang yang dimasukan melalui keyboard. Jika nilai *i* tersebut memiliki nilai lebih besar dari 20, maka nilai *i* ditambahkan 100. Tampilkanlah nilai *i* tersebut sebagai nilai akhir eksekusi.


```
Program kasus1
{representasi kondisional dengan satu kasus dengan 1 ekspresi}

Kamus
  i: integer {nilai yang dibaca}

Algoritma
  input(i)
  if (i>20) then
    i ← i + 100
  output(i)
```

Kode Sumber 5.2. Notasi Algoritmik 1 Kasus

Kode sumber 5.2. jika nilai i diberikan nilai 5, maka pada ekspresi kondisi $i > 20$ bernilai false, sehingga nilai i pada akhir eksekusi bernilai 5. Jika nilai i diberikan nilai 35, maka pada ekspresi kondisi $i > 20$ bernilai true, sehingga i akan diproses dengan menambahkan 100 ($i \leftarrow i + 100$), sehingga nilai i pada akhir eksekusi bernilai 135.

Contoh 5.2.

Dibaca i yang merupakan sebuah nilai integer sembarang yang dimasukan melalui keyboard. Jika nilai i tersebut memiliki nilai lebih besar dari 20 dan kurang dari sama dengan 40, maka nilai ditambahkan 100. Tampilkanlah nilai i tersebut sebagai nilai akhir eksekusi.

```
Program kasus1
{representasi kondisional dengan satu kasus dengan 2 ekspresi}

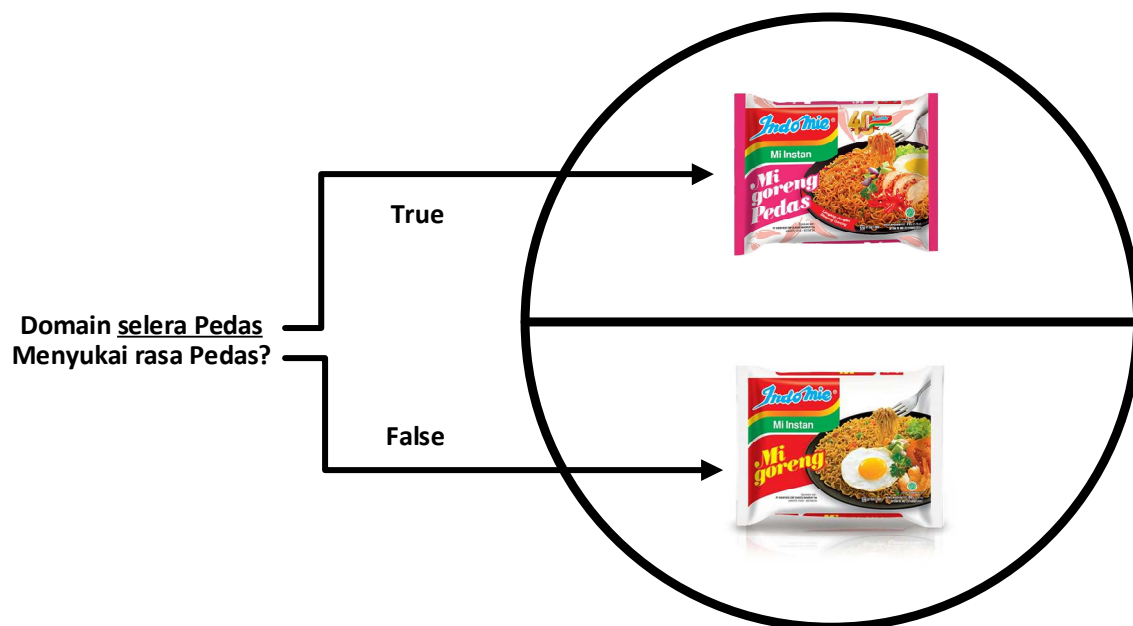
Kamus
  i: integer {nilai yang dibaca}

Algoritma
  input(i)
  if (i>20 AND i<=40) then
    i ← i + 100
  output(i)
```

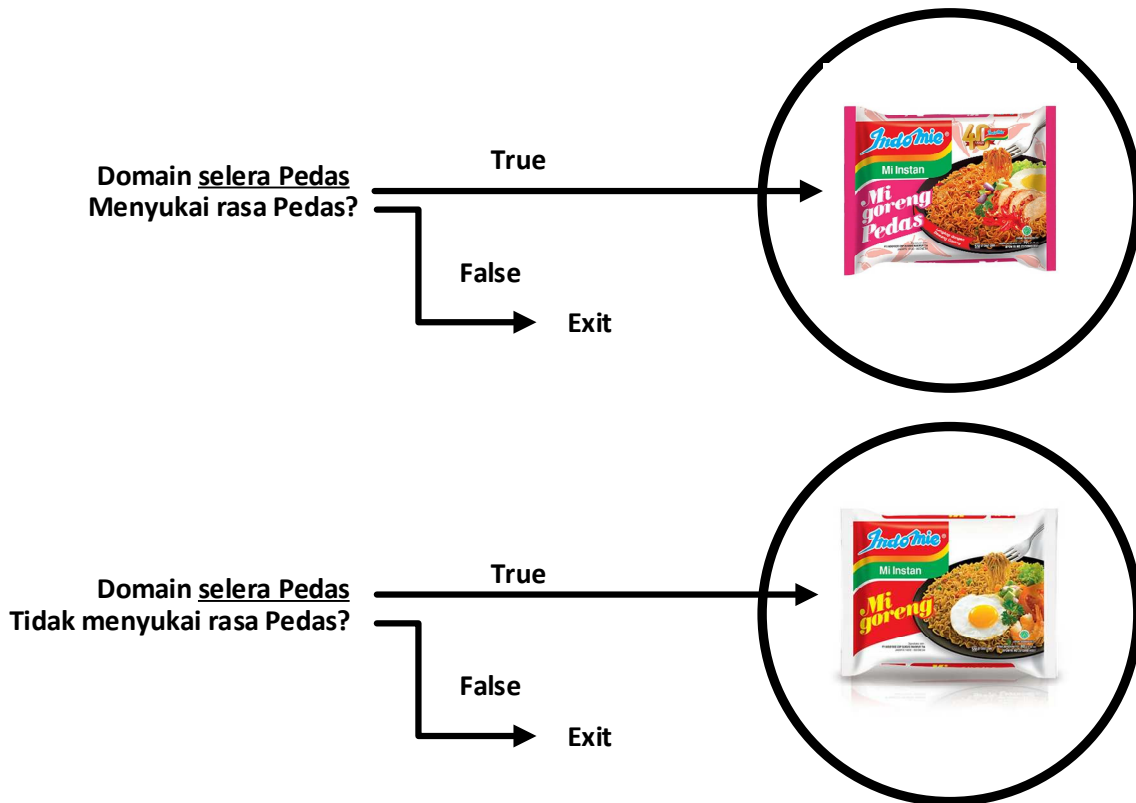
Kode Sumber 5.3. Notasi Algoritmik 1 Kasus

Kode sumber 5.3. jika nilai i diberikan nilai 5, maka pada ekspresi kondisi $(i > 20 \text{ AND } i \leq 40)$ bernilai false, sehingga nilai i pada akhir eksekusi bernilai 5. Jika nilai i diberikan nilai 55, maka pada ekspresi kondisi $(i > 20 \text{ AND } i \leq 40)$ bernilai false, sehingga nilai i pada akhir eksekusi bernilai 55. Jika nilai i diberikan nilai 35, maka pada ekspresi kondisi $(i > 20 \text{ AND } i \leq 40)$ bernilai true, sehingga i akan diproses dengan menambahkan 100 ($i \leftarrow i + 100$), sehingga nilai i pada akhir eksekusi bernilai 135.

Dua Kasus merupakan bagian dari ekspresi kondisional yang hanya memiliki 2 kondisi, atau dapat diuraikan menjadi 2 sub masalah. Sebagai contoh pada permasalahan mengenai selera menu makanan indomie goreng. Domain selera ini difokuskan pada rasa pedas atau tidak. Jika menyukai rasa pedas, maka menu indomie goreng yang dipesan adalah indomie goreng pedas. Akan tetapi, jika tidak menyukai rasa pedas, maka indomie goreng yang dipesan adalah indomie goreng. Pada persoalan ini memiliki 2 kondisi yaitu selera pedas dan tidak pedas. Esensi aksi yang dijalankan jika kondisi terpenuhi (bernilai True) pada persoalan domain selera pedas akan dipenuhi (bernilai true), maka akan menjalankan aksi memesan indomie goreng pedas. Jika kondisi tidak dipenuhi (bernilai false), maka akan menjalankan aksi memesan indomie goreng. Abstraksi persolan domain selera rasa pedas pada indomie goreng diberikan pada Gambar 5.2.



Gambar 5.2. Abstraksi Domain Selera Pedas (2 kasus)



Gambar 5.3. Abstraksi Domain Selera Pedas (2 kondisional)

Perhatikan Gambar 5.3. persoalan selera pedas dipandang menjadi 2 kondisional, yaitu kondisi pedas dan tidak pedas sehingga pada eksekusinya kedua kondisional nya akan dilakukan eksekusi sebanyak 2 kali. Jika kondisi menyukai rasa pedas bernilai true, maka yang dipesan adalah indomie goreng pedas, akan tetapi jika bernilai false, maka exit. Kemudian berikutnya dilakukan pengecekan pada kondisi tidak menyukai rasa pedas. Jika kondisi tidak menyukai rasa pedas bernilai true, maka yang dipesan adalah indomie goreng, akan tetapi jika bernilai false, maka exit. Dalam hal ini cara solusi yang digunakan adalah dengan menggunakan pendekatan bahwa terdapat 2 permasalahan dengan masing-masing hanya memiliki satu sub masalah (kasus). Solusi algoritma pada Gambar 5.2 dengan menggunakan pendekatan bahwa satu permasalahan dibagi menjadi sub-sub masalah (2 sub masalah/kasus). Kedua model solusi algoritma di atas secara keluaran sama-sama benar, hanya berbeda pada sisi efektifitas algoritma. Hal ini terjadi pada Gambar 5.2. eksekusi kondisi cukup satu kali. Pada saat kondisi menyukai rasa pedas bernilai true, maka eksekusi terminate. Sedangkan pada Gambar 5.3. eksekusi kondisi dilakukan

sebanyak 2 kali. Pada saat kondisi menyukai selera pedas bernilai true, harus menjalankan lagi kondisi tidak menyukai rasa pedas.

Oleh karena itu, standart penulisan teks algoritma yang digunakan adalah mengacu cara solusi yang diberikan pada Gambar 5.2.

Adapun bentuk umum kondisional dengan 2 kasus diberikan sebagai berikut:

```
if <kondisi> then  
    Aksi_1  
else  
    Aksi_2
```

Kode Sumber 5.4. Notasi Algoritmik 2 Kasus

Jika kondisi bernilai true, maka akan menjalankan perintah Aksi_1, dan jika kondisi bernilai false maka akan menjalankan Aksi_2. Aksi_1 dan Aksi_2 dapat berupa satu perintah algoritma atau lebih, jika terdiri lebih satu perintah sebaiknya diberikan batas penanda dalam satu blok perintah algoritma dengan menggunakan komentar. Akan tetapi untuk kemudahan dalam pembacaan 1 atau lebih perintah algoritma dalam penulisannya sebaiknya diberikan batas penanda dalam satu blok kumpulan perintah algoritma.

Contoh 5.3.

Dibaca *i* yang merupakan sebuah nilai integer sembarang yang dimasukan melalui keyboard. Lakukan klasifikasi dan tampilkan pada layar apakah nilai *i* tersebut merupakan bilangan Genap atau Ganjil.

Dari soal Contoh 5.3. diingatkan kembali bahwa sebelum membuat sebuah teks algoritma selalu dimulai dari memahami persoalan, lakukan sintesa (analisis) persoalan dengan melakukan mapping ke dalam input, proses, dan output. Hal utama yang harus dapat dipahami terlebih dahulu, pada Contoh 5.3. ini adalah mengetahui bagaimana aturan (cara) untuk melakukan klasifikasi bilangan integer ke dalam bilangan ganjil atau genap. Untuk melakukan klasifikasi ganjil atau genap diketahui dengan menggunakan operasi modulo. Jika hasil modulo 2 sama dengan 0,

maka bilangan integer tersebut sebagai bilangan genap, akan tetapi jika nilai modulo 2 sama dengan 1, maka bilangan integer tersebut sebagai bilangan ganjil. Teks algoritma nya untuk soal Contoh 5.3. diberikan pada Kode Sumber 5.5.

Program GanjilGenap1
{klasifikasi bilangan ganjil atau genap}

Kamus

i: integer {nilai yang dibaca}

Algoritma

```
input(i)
if (i mod 2)=1 then {cek ganjil}
    output('Bilangan Ganjil')
else {i mod 3 = 0}
    output('Bilangan Genap')
```

Kode Sumber 5.5. Klasifikasi Bilangan Ganjil atau Genap

Kode sumber 5.5., jika i diberikan nilai 5, maka dilakukan pengecekan pada kondisi $5 \bmod 2 = 1$, sehingga kondisinya bernilai true dan akan melakukan eksekusi perintah output('Bilangan Ganjil'). 5 adalah termasuk bilangan ganjil. Jika i diberikan nilai 100, maka dilakukan pengecekan pada kondisi $100 \bmod 2 = 0$, sehingga kondisinya bernilai false dan akan keluar dari bagian perintah if, kemudian masuk pada bagian perintah else, sehingga akan melakukan eksekusi perintah output('Bilangan Genap'). 100 adalah termasuk bilangan genap.

Program GanjilGenap2
{klasifikasi bilangan ganjil atau genap}

Kamus

i: integer {nilai yang dibaca}

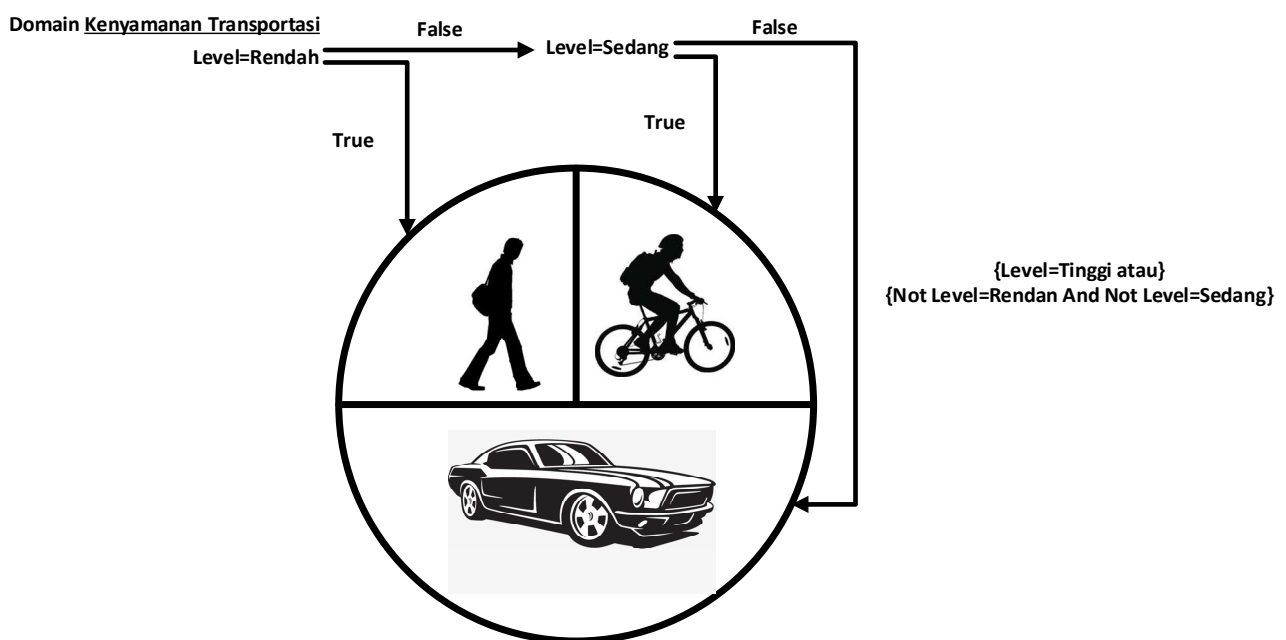
Algoritma

```
input(i)
if (i mod 2)=1 then {cek ganjil}
    output('Bilangan Ganjil')
if (i mod 2)=1 then {cek genap}
    output('Bilangan Genap')
```

Kode Sumber 5.5. Klasifikasi Bilangan Ganjil atau Genap

Bandingkan teks algoritma pada program GanjilGenap1 dengan GanjilGenap2, seperti yang diberikan penjelasan di atas pada persolan selera rasa pedas, bahwa program GanjilGenap2 tidak efektif, karena solusi yang diberikan dalam bentuk 2 kondisional. Perintah algoritma 2 `if` akan dieksekusi walau pada `if` pertama sudah bernilai true. Oleh karena itu teks algoritma pada program GanjilGenap2 sangat tidak disarankan, akan tetapi yang digunakan acuan dalam membuat sketsa solusi secara algoritma adalah pada program GanjilGenap1.

Lebih dari Dua Kasus merupakan bagian dari ekspresi kondisional yang memiliki lebih dari 2 kondisi, atau dapat diuraikan menjadi lebih dari 2 sub masalah. Sebagai contoh pada permasalahan mengenai kenyamanan sarana transportasi untuk bepergian dari tembalang ke simpang lima. Domain kenyamanan sarana transportasi terdiri dari level rendah, sedang, dan tinggi. Jika menginginkan level rendah, artinya kondisinya bernilai true, maka sarana transportasi yang digunakan jalan kaki. Akan tetapi jika bernilai false, maka dilakukan pengecekan pada kondisi level sedang. Jika menginginkan level sedang, artinya kondisinya bernilai true, maka sarana transportasi yang digunakan sepeda motor. Akan tetapi jika bernilai false, maka sarana transportasi yang digunakan mobil. Abstraksi persolan domain kenyamanan sarana transportasi diberikan pada Gambar 5.4.



Gambar 5.4. Abstraksi Domain Kenyamanan Sarana Transportasi (3 kasus)

Adapun bentuk umum ekspresi kondisional (analisa kasus) lebih dari 2 kasus diberikan pada Kode Sumber 5.6., 5.7., dan 5.8.

```
depend on <nama>
    kondisi_1 : Aksi_1
    kondisi_2 : Aksi_2
    kondisi_3 : Aksi_3
    .
    .
    .
    Kondisi_N : Aksi_N

{Kondisi_N sama dengan NOT kondisi_1 AND NOT Kondisi_2 AND NOT
Kondisi_3 AND ... NOT Kondisi_N-1 atau dapat menggunakan else}
```

Kode Sumber 5.6. Notasi Algoritmik Lebih dari 2 Kasus (a)

```
depend on <nama>
    kondisi_1 : Aksi_1
    kondisi_2 : Aksi_2
    kondisi_3 : Aksi_3
    .
    .
    .
    NOT kondisi_1 AND NOT Kondisi_2 AND NOT Kondisi_3 AND
    ... NOT Kondisi_N-1: Aksi_N
```

Kode Sumber 5.7. Notasi Algoritmik Lebih dari 2 Kasus (b)

```
depend on <nama>
    kondisi_1 : Aksi_1
    kondisi_2 : Aksi_2
    kondisi_3 : Aksi_3
    .
    .
    .
    else      : Aksi_N
```

Kode Sumber 5.8. Notasi Algoritmik Lebih dari 2 Kasus (c)

Kode sumber 5.6., 5.7., dan 5.8. secara logika adalah sama, hanya berbeda pada cara penulisan algoritmanya, sehingga bentuk manapun dapat digunakan bebas sesuai selera masing-masing.

Contoh 5.4.

Dibaca i yang merupakan sebuah nilai integer sembarang yang dimasukan melalui keyboard. Lakukan konversi dari nilai angka menjadi nilai huruf, yang ditampilkan pada layar. Aturan konversi adalah diberikan sebagai berikut:

Angka	Huruf
$i \geq 80$	A
$70 \leq i \leq 79$	B
$60 \leq i \leq 69$	C
$50 \leq i \leq 59$	D
$i \leq 49$	E

Program KonvAngkaHuruf1
{Konversi nilai angka ke huruf dan tampilkan dilayar}

Kamus

i : integer {nilai yang dibaca}

Algoritma

```
input ( $i$ )  
depend on  $i$   
     $i \geq 80$  : output ('A')  
     $70 \leq i \leq 79$ : output ('B')  
     $60 \leq i \leq 69$ : output ('C')  
     $50 \leq i \leq 59$ : output ('D')  
     $i \leq 49$  : output ('E')
```

Kode Sumber 5.9. Konversi Nilai Angka ke Nilai Huruf

Program KonvAngkaHuruf1 tanpa menggunakan memorisasi untuk menampilkan nilai huruf, akan tetapi nilai huruf langsung dilakukan output pada setiap kasus. Jika $i \geq 80$ bernilai true maka output nilai huruf adalah A dan terminate. Jika $i \geq 80$ bernilai false, maka dilakukan pengecekan $70 \leq i \leq 79$. Jika $70 \leq i \leq 79$ bernilai true maka output nilai huruf adalah B dan terminate. Jika $70 \leq i \leq 79$ bernilai false, maka dilakukan pengecekan $60 \leq i \leq 69$. Jika $60 \leq i \leq 69$ bernilai true maka output nilai huruf adalah C dan terminate. Jika $60 \leq i \leq 69$ bernilai false maka dilakukan pengecekan $50 \leq i \leq 59$. Jika $50 \leq i \leq 59$ bernilai true maka output nilai huruf adalah D dan terminate. Jika $50 \leq i \leq 59$ bernilai false, maka output nilai huruf adalah E dan terminate. Jika i diberikan nilai 65, maka dilayar akan

ditampilkan nilai huruf C. Demikian untuk nilai A, B, D, E dapat dicoba untuk diberikan nilai i yang sesuai dengan aturannya yang telah didefinisikan. Program KonvAngkaHuruf2 secara logika sama dengan program KonvAngkaHuruf, hanya berbeda cara penulisan teks algoritmanya, yaitu dengan menggunakan perintah else untuk menggantikan kondisi ≤ 59 (terakhir).

```
Program KonvAngkaHuruf2
{Konversi nilai angka ke huruf dan tampilkan dilayar}

Kamus
  i: integer {nilai yang dibaca}

Algoritma
  input(i)
  depend on i
    i >= 80      : output('A')
    70 <= i <= 79: output('B')
    60 <= i <= 69: output('C')
    50 <= i <= 59: output('D')
    else        : output('E')
```

Kode Sumber 5.10. Konversi Nilai Angka ke Nilai Huruf

```
Program KonvAngkaHuruf3
{Konversi nilai angka ke huruf dan tampilkan dilayar}

Kamus
  i: integer {nilai angka, nilai yang dibaca}
  huruf: character {nilai huruf}

Algoritma
  input(i)
  depend on i
    i >= 80      : huruf ← 'A'
    70 <= i <= 79: huruf ← 'B'
    60 <= i <= 69: huruf ← 'C'
    50 <= i <= 59: huruf ← 'D'
    i <= 59      : huruf ← 'E'
  output(huruf)
```

Kode Sumber 5.11. Konversi Nilai Angka ke Nilai Huruf

Program KonvAngkaHuruf3 teks algoritma yang ditulis menggunakan memorisasi untuk menampung nilai huruf dengan variabel `huruf` di assign sebuah nilai huruf. Penulisan dengan memorisasi atau tanpa memorisasi secara output sama, akan tetapi perbedaannya pada kebutuhan pada saat akan digunakan untuk diproses lebih lanjut oleh bagian atau proses lain. Jika akan digunakan untuk atau oleh proses lain, harus digunakan memorisasi, tetapi hanya satu kali saja penggunaannya, maka dapat dibuat dengan tanpa memorisasi.

BAB VI

Perulangan

Perulangan merupakan sebuah proses aktifitas terhadap sebuah pernyataan (perintah pada teks algoritmik) yang sama dilakukan secara berulang-ulang. Dalam kehidupan sehari-hari aktifitas sama yang dilakukan secara berulang-ulang, diantaranya seperti aktifitas makan, minum, mandi, tidur, pergi ke kampus, pergi ke mall, dan lain-lainnya. Perhatikan aktifitas yang disebutkan di atas, pada setiap periode waktu tertentu akan terus dilakukan kembali seperti pada periode sebelumnya. Dalam konteks algoritma (pemrograman) perulangan sangat dibutuhkan, agar dalam penulisan teks algoritma menjadi lebih efisien. Efisiensi penulisan dapat diperoleh dengan cara cukup menuliskan perintah yang sama sekali saja, kemudian dengan mekanisme perulangan akan dijalankan berulang-ulang sesuai dengan jumlah kebutuhannya banyaknya perulangan. Dapat kita bayangkan, seandainya tidak ada mekanisme perulangan dalam menuliskan sebuah teks algoritma, maka untuk perintah yang sama akan dituliskan sebanyak jumlah perulangan yang dibutuhkan. Misalnya dilakukan perulangan 1.000 kali, maka perintah yang sama ini harus dituliskan sebanyak 1000 kali, sehingga cukup membutuhkan waktu. Bagaimana jika perintah yang sama dilakukan sebanyak 1.000.000.000? Tentu dapat dibayangkan sendiri, betapa kerepotan yang terjadi dan berapa lama waktu yang dibutuhkan untuk menuliskan perintah yang sama sebanyak 1.000.000.000. Terkait berapa banyak jumlah perulangan yang akan dijalankan, hal ini terkait dengan sebuah kondisi untuk menjalankan perulangan atau kondisi untuk menghentikan perulangan (istilahnya kondisi ulang atau kondisi stop). Jika kondisi ulang bernilai true, maka perintah yang sama akan dijalankan dan akan berhenti sampai kondisinya bernilai false. Demikian juga dengan kondisi stop menjadi berbanding terbalik (sebaliknya) dengan kondisi ulang. Pada algoritma terdapat beberapa skema perulangan yang dapat digunakan. Secara prinsip adalah sama, yaitu merupakan perintah yang digunakan untuk melakukan proses perulangan, akan tetapi dari setiap skema memiliki makna yang berbeda. Oleh karena itu dalam

menggunakan skema harus tepat. Walaupun sebenarnya ketidaktepatan dalam penggunaan skema tidak akan memberikan pengaruh pada output algoritma, akan tetapi akan memberikan pengaruh pada efisiensi teks algoritma yang dibuat. Adapun skema perulangan yang digunakan pada buku ini terdiri dari tiga skema perulangan. Skema-skema yang diberikan pada buku ini dibagi menjadi dua. Pertama untuk skema perulangan yang pasti, yaitu sudah diketahui berapa banyak perulangan yang akan dijalankan. Kedua untuk skema perulangan yang belum pasti, yaitu belum diketahui berapa banyak perulangan yang akan dijalankan. Secara lengkap ketiga skema perulangan yang diberikan pada buku ini diberikan sebagai berikut:

Traversal merupakan sebuah skema perulangan dengan banyaknya jumlah perulangan sudah diketahui secara pasti. Misalkan dalam kehidupan sehari-hari mengenai aktifitas sholat 5 waktu sebagai perintah Allah SWT dalam ajaran Islam. Sholat 5 waktu adalah sebuah aktifitas sholat yang dilakukan secara berulang-ulang sebanyak 5 kali dalam sehari. Melaksanakan sholat 5 waktu banyaknya aktifitas sholat yang dilakukan sudah diketahui dengan pasti, yaitu sebanyak 5 kali. Bentuk umum perulangan traversal diberikan pada Kode Sumber 6.1.

```
i traversal [Awal..Akhir]
  Aksi
```

Kode Sumber 6.1. Notasi Algoritmik Perulangan Transversal

i sebagai counter yang digunakan pada proses perulangan sehingga selalu bilangan integer. Awal merupakan batas bawah sebagai counter untuk memulai proses perulangan, sedangkan Akhir merupakan batas atas sebagai counter untuk menghentikan (mengakhiri) proses perulangan. Bagian Aksi merupakan bagian yang berisi mengenai satu atau lebih perintah (sebagai satu blok perintah algoritma) yang akan dijalankan secara berulang-ulang. Penulisan bagian Aksi baik yang terdiri hanya 1 atau lebih perintah dituliskannya dengan cara menggunakan tab. Dalam penulisan teks algoritma, jika setiap ketemu blok perintah harus dituliskan dengan menggunakan tab, untuk menandai bahwa perintah-perintah ini adalah dalam satu blok perintah. Perintah yang termasuk dalam kriteria blok adalah meliputi perintah: program, function, procedure, kondisional (analisa kasus), perulangan. Kembali pada bentuk perulangan traversal, Lompatan atau jeda perulangan

traversal dilakukan dengan beda satu pada saat menjalankan perintah satu dengan perintah lainnya yang berurutan. Jika nilai Awal < Akhir, maka perulangan traversal dilakukan secara menaik dimulai dari batas bawah (Awal) secara berturut-turut ditambahkan satu, sampai berhenti pada batas atas (Akhir). Akan tetapi jika Awal > Akhir, maka perulangan traversal dilakukan secara menurun dimulai dari batas bawah (Awal) secara berturut-turut dikurangkan satu, sampai berhenti pada batas atas (Akhir).

Contoh 6.1.

Tampilkanlah pada layar secara ke bawah (mulai dari kecil ke besar) bilangan bulat positif yang dapat dibentuk dari bilangan integer positif sembarang N ($N > 0$) yang dimasukan melalui keyboard.

```
Program CetakBil1
{menampilkan bilangan integer positif pada layar secara menaik}

Kamus
  N: integer {banyaknya bilangan}
  i: integer {counter}

Algoritma
  input (N)
  i traversal [1..N]
  output{i}
```

Kode Sumber 6.2. Mencetak Bilangan Integer Positif Sebanyak N (Traversal)

Program CetakBil1 akan menampilkan bilangan integer yang di mulai dari 1 sampai dengan N secara menurun. Perintah `output (i)` akan di eksekusi sebanyak N kali. Perhatikan jika nilai N diberikan nilai ≤ 0 , maka algoritma tidak pernah masuk ke dalam body perulangan `output (i)` atau langsung keluar perulangan. Jika dijalankan pada bahasa pemrograman, maka pada layar monitor tidak ada hasil eksekusi. Hal ini disebabkan karena batas akhirnya lebih kecil dari batas bawah (kondisinya bernilai false atau tidak dipenuhi). Secara logika kondisi seperti ini sudah benar, hanya permasalahannya karena pada layar monitor tidak mengeluarkan apa-apa, akan tampak seperti error eksekusi, sehingga kurang memberikan informatif terhadap pengguna. Agar lebih informatif, teks algoritma yang dibuat diberikan penanganan kasus kosong atau kondisi

exception, sehingga algoritma yang dibuat lebih bersifat komunikatif. Perhatikan program CetakBil2, diberikan sebuah penanganan kasus kosong, yaitu jika nilai $N \leq 0$, maka pada layar ditampilkan pesan “**N harus positif**”. Dengan memberikan penanganan kasus kosong, maka algoritma menjadi lebih komunikatif dan tidak menimbulkan multi tafsir oleh pengguna. Penanganan kasus kosong atau kondisi exception adalah sebuah upaya untuk meminimalkan kesalahan input baik sengaja atau tidak yang dilakukan oleh pengguna untuk menjalankan teks algoritma.

```
Program CetakBil2
{menampilkan bilangan integer positif pada layar secara menaik}

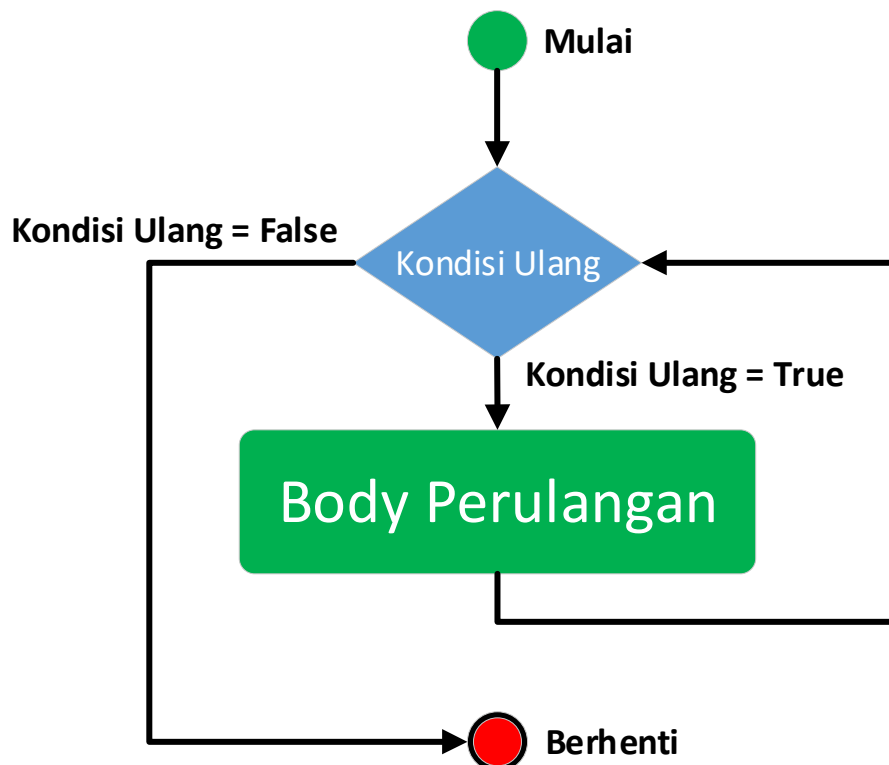
Kamus
  N: integer {banyaknya bilangan}
  i: integer {counter}

Algoritma
  input (N)
  if (N<=0) then
    output('N harus positif')
  else {N>0}
    i traversal [1..N]
    output{i}
```

Kode Sumber 6.3. Mencetak Bilangan Integer Positif Sebanyak N (Traversal)

While - do merupakan sebuah skema perulangan dengan banyaknya jumlah perulangan belum diketahui secara pasti. Misalkan dalam kehidupan sehari-hari mengenai aktifitas sholat 5 waktu, jika sebelumnya dipandang sebagai perintah Allah SWT, maka aktifitas sholat 5 waktu ada sebuah perulangan dengan jumlah perulangan sudah pasti, yaitu sebanyak 5 kali. Akan tetapi jika dipandang dari banyaknya setiap hamba Allah SWT menjalankan sholat 5 waktu adalah merupakan sebuah skema perulangan yang tidak pasti. Hal ini dikarenakan pada setiap hamba Allah SWT, ada yang menjalankan sholat 5 waktu penuh (5 kali), ada yang hanya 4 kali, 3 kali, 2 kali, 1 kali, dan bahkan ada yang tidak menjalankan sama sekali (0 kali). Ketidakpastian adalah ditekankan pada banyaknya aktifitas sholat 5 waktu pada setiap hamba Allah SWT. Sebagai contoh lain sebuah aktifitas perulangan dengan menggunakan skema tidak pasti While - do, adalah mengenai aktifitas menghitung kehadiran mahasiswa yang hadir pada perkuliahan Algoritma dan

Pemrograman. Dalam menghitung banyaknya mahasiswa yang hadir, dapat dilakukan sebanyak jumlah peserta perkuliahan, atau hanya 30 kali, atau 10 kali, atau 2 kali atau 1 kali, atau tidak pernah melakukan aktifitas menghitung (0 kali). Aktifitas tidak pernah melakukan proses menghitung (0 kali), diakibatkan semua peserta kuliah kompak untuk tidak hadir mengikuti perkuliahan (mungkin terjadi karena mahasiswa memiliki hak sebanyak 25% untuk tidak hadir diperkuliahan). Banyaknya jumlah perulangan yang dijalankan pada skema perulangan While - do adalah minimal sebanyak 0 kali, seperti aktifitas menjalankan sholat 5 waktu oleh setiap hamba Allah SWT. Oleh karena itu pada skema perulangan While - do kondisinya ditempatkan sebelum menjalankan body perintah yang akan dilakukan perulangan. Secara garis besar alur logika dalam skema perulangan While - do dapat diberikan pada Gambar 6.1 di bawah ini.



Gambar 6.1. Abstraksi Alur Logika Skema Perulangan While - do

Alur logika skema perulangan While - do pada Gambar 6.1. dimulai dengan melakukan pengecekan Kondisi Ulang. Kondisi Ulang merupakan ekspresi logika yang memiliki nilai true dan false. Jika Kondisi Ulang bernilai true, maka masuk ke perulangan untuk menjalankan Body

Perulangan, dan kembali dilakukan pengecekan Kondisi Ulang dan selama Kondisi Ulang bernilai true akan dilakukan proses ini secara terus-menerus. Jika Kondisi Ulang bernilai false, maka keluar dari perulangan untuk dihentikan proses perulangannya (Berhenti).

Bentuk umum perulangan While - do diberikan pada Kode Sumber 6.1.

```
while <kondisiUlang> do
    Aksi
{endWhile}
```

Kode Sumber 6.4. Notasi Algoritmik Perulangan While - do

KondisiUlang merupakan ekspresi logika yang dapat bernilai true atau false. Ekspresi logika dapat terdiri dari 1 ekspresi atau lebih, jika lebih 1 ekspresi maka ekspresi logika ini dikombinasikan dengan menggunakan operator logika (AND, OR, NOT, XOR, ... dll). Pada saat menggunakan lebih dari 1 ekspresi logika harus memperhatikan nilai tabel kebenaran yang berlaku. Aksi dapat berbentuk 1 atau lebih perintah teks algoritma yang sering disebut dengan nama body perulangan. Body perulangan adalah merupakan bagian dalam 1 blok perulangan, sangat disarankan untuk dituliskan dengan tab masuk, agar lebih mudah dalam pembacaan teks algoritma. Dalam 1 blok harus dapat dengan mudah dibaca, oleh karena itu perlu ditambahkan komentar {endWhile} sebagai penanda batas blok. Secara garis besar alur logika skema perulangan While - do, jika KondisiUlang bernilai true maka Aksi dijalankan, dan kembali ke KondisiUlang. Proses ini dilakukan secara terus-menerus selama KondisiUlang bernilai true. Jika KondisiUlang bernilai false, maka proses perulangan dihentikan.

Contoh 6.1. jika skema perulangan Traversal dirubah menjadi menggunakan skema While - do, maka teks algoritmanya diberikan pada Kode Sumber 6.5. Perhatikan perbedaan antara Kode Sumber 6.3. dengan 6.5, yaitu pada penggunaan skema perulangannya. Skema perulangan pada Kode Sumber 6.5. menggunakan While - do sehingga perlu dilakukan perubahan dengan memberikan inisialisasi nilai $i=1$ (sebagai nilai batas Bawah), dan pada body perulangan ditambahkan increment $i \leftarrow i+1$ yang digunakan sebagai counter dengan penambahan satu agar nilai i dapat mencapai nilai N (batas Akhir). Nilai i pada akhir eksekusi teks algoritma adalah sebesar $N + 1$. Pada nilai $i = N + 1$ mengakibatkan nilai kondisi perulangan $i \leq N$ bernilai false, sehingga proses perulangan akan diberhentikan. Kode Sumber 6.3. dan 6.5. secara output

algoritma memiliki output yang sama, akan tetapi jika dipandang pada ketepatan penggunaan skema tidak tepat. Ketidaktepatan penggunaan skema dapat ditunjukkan bahwa berapapun nilai N yang dimasukan body perulangan selalu dilakukan eksekusi. Hal ini ditunjukkan dengan diberikannya penanganan kasus kosong, sehingga jika tidak kasus kosong maka dipastikan masuk ke perulangan dan menjalankan body perulangan. Pada skema perulangan Kode Sumber 6.5 tidak akan pernah tidak masuk ke dalam proses perulangan untuk menjalankan body perulangan.

Program CetakBil3

```
{menampilkan bilangan integer positif pada layar secara menaik}
```

Kamus

N: integer {banyaknya bilangan}

i: integer {counter}

Algoritma

```
input (N)
```

```
if (N<=0) then
```

```
    output('N harus positif')
```

```
else {N>0}
```

```
    i ← 1 {inisialisasi}
```

```
    while (i<=N) do
```

```
        output{i}           {body perulangan}
```

```
        i ← i + 1 {increment}
```

```
{endWhile} {i = N + 1, nilai i terakhir}
```

Kode Sumber 6.5. Mencetak Bilangan Integer Positif Sebanyak N (While - do)

Program CetakBil4

```
{menampilkan bilangan integer positif pada layar secara menaik}
```

Kamus

N: integer {banyaknya bilangan}

i: integer {counter}

Algoritma

```
input (N)
```

```
i ← 1 {inisialisasi}
```

```
while (i>=0 AND i<=N) do
```

```
    output{i}           {body perulangan}
```

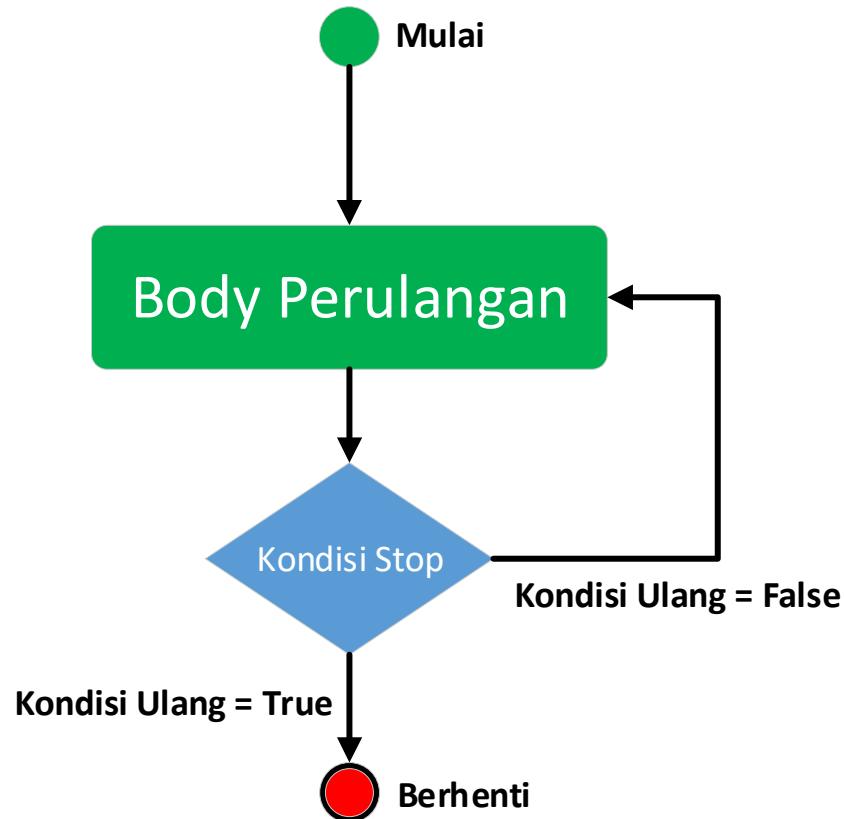
```
    i ← i + 1 {increment}
```

```
{endWhile} {i = N + 1, nilai i terakhir}
```

Kode Sumber 6.6. Mencetak Bilangan Integer Positif Sebanyak N (While - do)

Program CetakBil3 dengan menggunakan skema perulangan While - do menjadi kurang tepat, karena minimal satu kali body perulangan dilakukan eksekusi. Minimal satu kali akan terjadi eksekusi body perulangan disebabkan dengan pemberian penanganan kasus kosong. Agar penggunaan skema perulangan While - do menjadi tepat, maka Kode Sumber 6.5. perlu dilakukan perubahan seperti yang diberikan pada Kode Sumber 6.6. Pada saat nilai $N \leq 0$, maka kondisi bernilai false AND true sama dengan false, sehingga tidak pernah masuk ke proses perulangan untuk menjalankan body perulangan (minimal 0 kali).

Repeat - until merupakan sebuah skema perulangan dengan banyaknya jumlah perulangan belum diketahui secara pasti. Misalkan dalam kehidupan sehari-hari mengenai aktifitas sholat 5 waktu yang dipandang dari banyaknya menjalankan sholat bagi setiap hamba Allah SWT yang memiliki tingkat keimanan (pada contoh skema While - do domain yang digunakan adalah semua hamba Allah SWT). Bagi hamba Allah SWT yang memiliki tingkat keimanan, minimal akan menjalankan sholat 5 waktu minimal 1 kali. Jumlah sholat 5 waktu yang dijalankan oleh hamba Allah SWT yang memiliki tingkat keimanan bervariasi tergantung dari tingkat keimanan masing-masing. Ada yang menjalankan (5 kali) penuh, ada yang hanya 4 kali, 3 kali, 2 kali, dan 1 kali. Ketidakpastian adalah ditekankan pada banyaknya aktifitas sholat 5 waktu pada setiap hamba Allah SWT. Sebagai contoh lain sebuah aktifitas perulangan dengan menggunakan skema tidak pasti Repeat - until, adalah mengenai aktifitas memasukan Personal Identification Number (PIN) pada saat menggunakan Automated Teller Machine (ATM). Aktifitas memasukan PIN pada mesin ATM minimal dilakukan 1 kali, dan maksimal sebanyak 3 kali. Ketidakpastian banyaknya aktifitas memasukan PIN untuk setiap orang mungkin akan berbeda-beda, ada yang cukup 1 kali, ada yang 2 kali, dan bahkan ada sampai 3 kali. Banyaknya jumlah perulangan yang dijalankan pada skema perulangan Repeat - until adalah minimal sebanyak 1 kali, seperti aktifitas menjalankan sholat 5 waktu oleh setiap hamba Allah SWT yang memiliki tingkat keimanan atau seperti aktifitas memasukan PIN pada mesin ATM. Oleh karena itu pada skema perulangan Repeat - until kondisinya ditempatkan setelah menjalankan body perintah yang akan dilakukan perulangan. Secara garis besar alur logika dalam skema perulangan Repeat - until dapat diberikan pada Gambar 6.2 di bawah ini.



Gambar 6.2. Abstraksi Alur Logika Skema Perulangan Repeat - until

Alur logika skema perulangan Repeat - until pada Gambar 6.2. dimulai terlebih dahulu masuk ke dalam proses perulangan untuk menjalankan body perulangan. Kemudian dilanjutkan untuk melakukan pengecekan Kondisi Stop. Kondisi Stop merupakan ekspresi logika yang memiliki nilai true dan false. Jika Kondisi Stop bernilai false, maka masuk ke perulangan untuk menjalankan Body Perulangan, dan kembali dilakukan pengecekan Kondisi Stop dan selama Kondisi Stop bernilai false akan dilakukan proses ini secara terus-menerus. Jika Kondisi Stop bernilai true, maka keluar dari perulangan untuk dihentikan proses perulangannya (Berhenti).

Bentuk umum perulangan Repeat - until diberikan pada Kode Sumber 6.7.

```
repeat
    Aksi
until <kondisiStop>
```

Kode Sumber 6.7. Notasi Algoritmik Perulangan Repeat - until

Program CetakBil3 pada Kode Sumber 6.5. skema perulangan yang digunakan dilakukan perubahan menjadi menggunakan skema perulangan Repeat - until. Secara konsep adalah sama, hanya perlu melakukan perubahan penggunaan kondisi, jika skema While - do menggunakan kondisi ulang, sedangkan skema Repeat - until menggunakan kondisi stop, seperti yang diberikan pada Kode Sumber 6.8.

```
Program CetakBil5
{menampilkan bilangan integer positif pada layar secara menaik}

Kamus
  N: integer {banyaknya bilangan}
  i: integer {counter}

Algoritma
  input (N)
  if (N<=0) then
    output('N harus positif')
  else {N>0}
    i ← 1 {inisialisasi}
    repeat
      output{i}          {body perulangan}
      i ← i + 1 {increment}
    until (i>N){i = N + 1, nilai i terakhir}
```

Kode Sumber 6.8. Mencetak Bilangan Integer Positif Sebanyak N (Repeat - until)

Perubahan skema perulangan dari Kode Sumber 6.3. menjadi menggunakan skema perulangan Repeat - until pada Kode Sumber 6.8. sudah sangat tepat penggunaan skemanya, karena skema Repeat - until minimal 1 kali dilakukan eksekusi pada body perulangan. Pada saat nilai N>0, maka dipastikan minimal sebanyak 1 kali body perulangan dilakukan eksekusi.

Ketiga skema perulangan yang diberikan di atas secara output adalah memberikan output yang sama, akan tetapi dalam membuat sebuah teks algoritma selain kebenaran juga harus ditekankan pada ketepatan penggunaan skema, agar tingkat efektifitas algoritmanya lebih baik.

Dalam membuat teks algoritma yang digunakan sebagai representasi sketsa solusi secara global, solusi yang digunakan tidak cukup dengan satu perintah saja, akan tetapi solusi yang digunakan adalah dapat hanya berbentuk perintah tunggal, atau gabungan dari beberapa perintah. Secara konkretnya adalah demikian, sketsa solusi yang digunakan dapat hanya menggunakan

sequences, sequences dikombinasi kondisional (analisa kasus), kondisional dikombinasi dengan kondisional lainnya (komposit), perulangan dikombinasi dengan perulangan lainnya (komposit), kondisional dikombinasi dengan perulangan dan seterusnya. Perintah yang digunakan sangat bervariasi meliputi sequences, kondisional, dan perulangan menyesuaikan dengan persoalan yang akan diselesaikan. Penggunaan perintah algoritma, sangat tergantung dari tingkat kompleksitas persoalan, jika persoalan sangat kompleks penggunaan kombinasi perintah juga semakin kompleks, dan sebaliknya. Misalkan diberikan contoh sebagai berikut:

Contoh 6.2.

Tampilkanlah pada layar secara ke bawah (mulai dari kecil ke besar) bilangan genap yang dapat dibentuk dari bilangan integer positif sembarang N ($N > 0$) yang dimasukan melalui keyboard.

Persoalan pada Contoh 6.2. sketsa solusinya tidak cukup diselesaikan menggunakan sequences atau kondisional atau perulangan saja, akan tetapi dibutuhkan kombinasi dari ketiga perintah tersebut. Perintah sequences cukup jelas, kemudian perintah kondisional digunakan untuk menyelesaikan pemisahan bilangan genap atau ganjil. Perintah perulangan digunakan untuk mengulang proses mengecek bilangan genap dan menampilkan di layar. Teks algoritma untuk Contoh 6.2. diberikan pada Kode Sumber 6.9. dan 6.10. Skema perulangan Kode Sumber 6.9 menggunakan Traversal, sedangkan pada Kode Sumber 6.10 menggunakan Repeat - until. Kode Sumber 6.9. menggunakan skema Traversal dipandang banyaknya perulangan pada body perulangan adalah sudah pasti sebanyak N . Akan tetapi pada Kode Sumber 6.10. dipandang dari sisi minimal body perulangan dilakukan eksekusi 1 kali, akan tetapi jumlahnya perulangan dikesampingkan. Oleh karena itu untuk solusi Contoh 6.2 yang paling tepat adalah menggunakan skema perulangan Traversal.

Program CetakGenap1

```
{menampilkan bilangan genap integer positif pada layar secara menaik}
```

Kamus

```
N: integer {banyaknya bilangan}  
i: integer {counter}
```

Algoritma

```
input (N)
if (N<=0) then
    output('N harus positif')
else {N>0}
    i traversal [1..N]
    if (i mod 2=0) then {cek genap}
        output{i}          {body perulangan}
```

Kode Sumber 6.9. Mencetak Bilangan Integer Genap Positif Sebanyak N (Traversal)**Program** CetakGenap2

{menampilkan bilangan genap integer positif pada layar secara menaik}

Kamus

N: integer {banyaknya bilangan}
i: integer {counter}

Algoritma

```
input (N)
if (N<=0) then
    output('N harus positif')
else {N>0}
    i ← 1 {inisialisasi}
    repeat
        if (i mod 2=0) then {cek genap}
            output{i}          {body perulangan}
        i ← i + 1 {increment}
    until (i>N){i = N + 1, nilai i terakhir}
```

Kode Sumber 6.10. Mencetak Bilangan Integer Genap Positif Sebanyak N (Repeat - until)

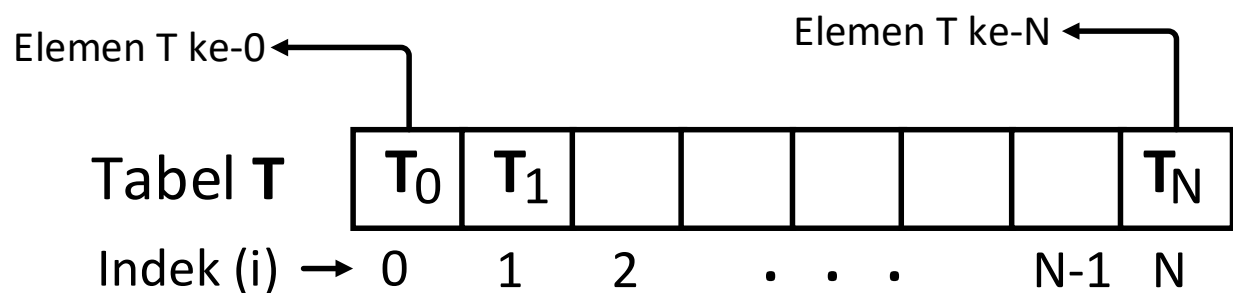
Jika $N \leq 0$, maka pada layar akan ditampilkan pesan “N harus positif”. Jika $N > 0$, maka akan dilakukan proses perulangan sebanyak N, dimana pada setiap perulangan (pada skema Traversal atau Repeat - until) dilakukan pengecekan bilangan genap atau tidak (pada kondisi if (i mod $2=0$) then). Jika kondisi bernilai true, maka nilai i dicetak dilayar sebagai bilangan genap. Proses ini diulang-ulang sebanyak N dengan cara menambahkan counter sebanyak 1. Pada skema traversal i terakhir bernilai N, sedangkan pada skema Repeat - until bernilai $N + 1$.

BAB VII

Tabel (Array/Larik)

Tabel atau disebut juga dengan array atau larik adalah merupakan kumpulan elemen data yang memiliki tipe data sama. Tipe data pada sebuah tabel dapat berupa tipe data dasar atau dapat juga tipe data bentukan, seperti yang telah diberikan pada bagian sebelumnya mengenai bahasan tipe data. Elemen pada sebuah tabel dapat diakses menggunakan indeks (alamat) yang dimiliki oleh tabel, dimana untuk indeks pada sebuah tabel selalu dimulai dengan nilai 0. Elemen sebuah tabel selain dapat diakses menggunakan indeks, dapat juga diakses menggunakan nilai. Akan tetapi untuk akses elemen berdasarkan nilai harus hati-hati karena nilai elemen pada sebuah tabel dapat memiliki nilai sama, sehingga yang dikuatirkan terjadi ketidaktepatan nilai yang diharapkan. Tabel dapat berbentuk dimensi satu (1 baris dengan N kolom) atau multi dimensi (N baris dengan N kolom). Tabel dimensi satu dalam kehidupan sehari-hari dapat diperlihatkan pada sebuah antrian di bank atau antrian yang lainnya atau layar display mesin antrian atau mesin pom bensin atau phone book pada handphone dan lain sebagainya. Sedangkan contoh tabel multi dimensi dapat diperlihatkan pada susunan kursi tempat duduk nonton bioskop atau stadion sepak bola atau susunan kursi pada bus atau dalam matematika dapat diperlihatkan pada sebuah matrik dan lain sebagainya.

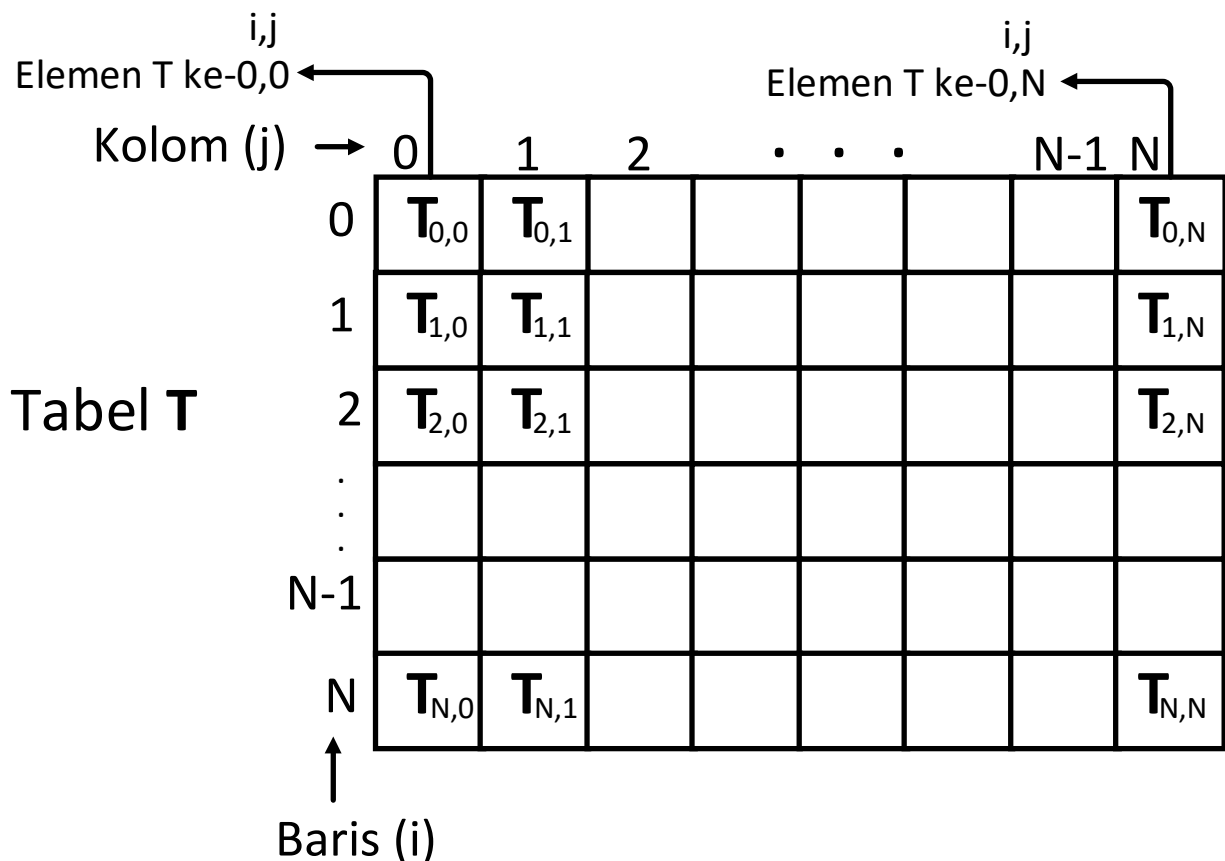
Abstraksi representasi tabel dimensi satu dapat diberikan pada Gambar 7.1. dibawah ini.



Gambar 7.1. Abstraksi Representasi Tabel Dimensi Satu

Tabel T terdiri dari kumpulan data yang memiliki tipe sama, dengan tipe dapat berbentuk tipe dasar atau tipe bentukan. Indeks (i) pada tabel T lazimnya bernilai numerik yang dimulai dari 0, 1, 2, ..., N (indeks dapat juga bernilai karakter). T_i adalah merupakan elemen ke-i pada urutan elemen-elemen tabel T. Elemen-elemen pada tabel T dapat dibaca secara sequences atau lompat-lompat sesuai dengan kebutuhan.

Demikian untuk penjelasan abstraksi tabel dimensi satu, kemudian abstraksi tabel T multi dimensi diberikan pada Gambar 7.2. Tabel T multi dimensi secara definisi sama seperti pada dimensi satu, akan tetapi yang berbeda adalah pada indeks yang digunakan. Tabel T multi dimensi menggunakan 2 indeks, yaitu baris (horizontal (i)) dan kolom (vertikal (j)). Elemen tabel multi dimensi diakses dengan cara baris kolom seperti $T_{i,j}$ (dibaca elemen ke-i,j). Akses elemen-elemen pada tabel T multi dimensi dilakukan dengan cara dimulai dari baris (i) kemudian dipasangkan dengan semua kolom (j), dan dilanjutkan pada baris(i) berikutnya secara terus-menerus sampai baris(i) yang paling terakhir.



Gambar 7.2. Abstraksi Representasi Tabel Multi Dimensi

Bentuk umum notasi algoritmik tabel (array/larik) adalah diberikan pada Kode Sumber 7.1.

```
{Tabel Dimensi Satu}
NamaTabel : array [Awal .. Akhir] of TipeData

{Tabel Multi Dimensi}
NamaTabel : array [Awal .. Akhir][Awal .. Akhir] of TipeData
```

Kode Sumber 7.1. Notasi Algoritmik Tabel (Array/Larik)

NamaTabel adalah digunakan untuk mendefinisikan nama variabel yang bertipe tabel. [Awal .. Akhir] merupakan ukuran tabel yang didefinisikan sebagai variabel NamaTabel. TipeData merupakan tipe data yang didefinisikan untuk NamaTabel, dapat berbentuk tipe dasar atau tipe bentukan. Tabel dimensi satu didefinisikan dengan cara menyebutkan ukuran [Awal .. Akhir] sebanyak 1 kali, sedangkan tabel multi dimensi menyebutkan ukuran [Awal .. Akhir] sebanyak 2 kali. Adapun cara mengacu (akses) elemen tabel dilakukan menggunakan indek (jika indek terdefinisi), seperti yang diberikan sebagai berikut: NamaTabel₄, atau NamaTabel_{1,2}, dan lain sebagainya. Pendefinisian (deklarasi) tabel dilakukan seperti deklarasi variabel, konstanta, dan lain sebagainya, yaitu ditempatkan pada bagian kamus pada teks algoritma.

Misalkan diberikan bahwa T1 adalah sebuah tabel dimensi satu yang bernilai integer dengan ukuran sebesar 8 dan T2 adalah sebuah tabel multi dimensi yang bernilai integer dengan ukuran 8x8, maka secara algoritmik menggunakan notasi algoritmik didefinisikan pada Kode Sumber 7.2.

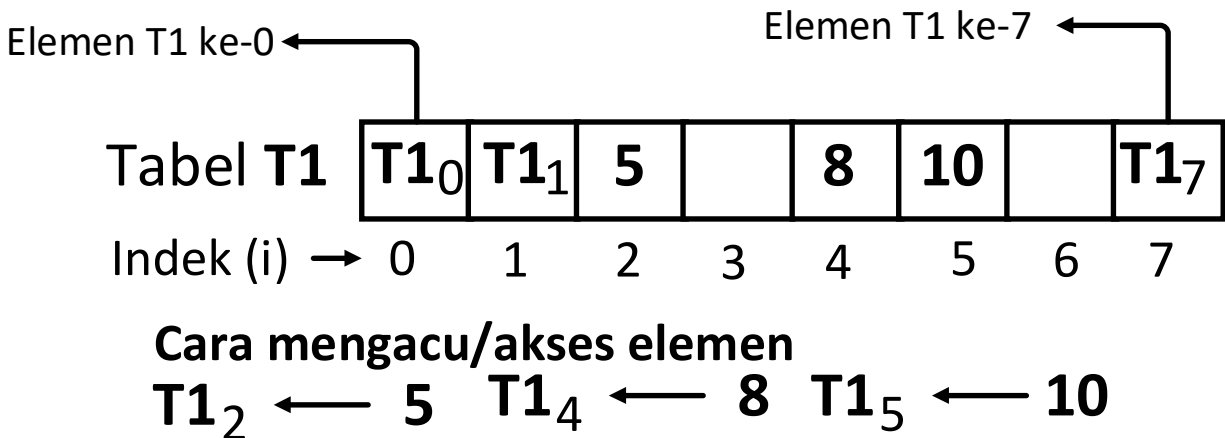
```
{Tabel T1 Dimensi Satu}
T1 : array [1 .. 8] of integer

{Tabel T2 Multi Dimensi}
T2 : array [1 .. 8][1 .. 8] of integer
```

Kode Sumber 7.2. Notasi Algoritmik Tabel (Array/Larik)

Pada saat deklarasi (pendefinisian) T1 sebagai tabel dengan ukuran sebanyak 8, maka pada memori sistem komputer akan dialokasikan sebesar 8 ukuran memori yang masih kosong seperti yang diberikan pada Gambar 7.3. Kemudian pada tabel T1 diberikan nilai dengan cara assignment pada elemen ke-2 ($T1_2 \leftarrow 5$), maka pada indek ke-2 akan terisi elemennya dengan nilai 5.

Selanjutnya $T_{14} \leftarrow 8$, dilanjutkan $T_{15} \leftarrow 10$, maka diakhir eksekusi tabel T terisi nilai 5 pada elemen ke-2, terisi nilai 8 pada elemen ke-4, dan terisi nilai 10 pada elemen ke-5. Demikian juga pada tabel T2 seperti yang dijelaskan pada tabel T1, akan tetapi perbedaannya terdapat pada penggunaan indeks (sebanyak 2, baris dan kolom).



Gamabr 7.3. Representasi dan Cara Mengacu/Akses Tabel Dimensi Satu

Contoh 6.1.

Diketahui sebuah tabel T yang bertipe integer, buatlah teks algoritma untuk mengisi tabel T dengan bilangan integer sembarang X yang dimasukan melalui keyboard. Proses pengisian tabel T akan berhenti jika ukuran tabel T sudah penuh atau nilai yang dimasukan adalah 9999, dan kemudian tampilkan pada layar isi pada tabel T tersebut.

Persoalan pada Contoh 6.1. membutuhkan sebuah tabel (array/larik) T yang akan digunakan untuk menyimpan nilai-nilai yang dimasukan melalui keyboard (X). Ukuran tabel yang akan digunakan mengikuti kebutuhan berapa banyak yang akan dibutuhkan (kira-kira cukup untuk setiap persoalan yang akan disolusikan). [1..5] pada Kode Sumber 7.3 artinya ukuran tabel T sebanyak 5 elemen yang akan dialokasikan pada memory. Setelah tabel T terdefinisi, maka proses berikutnya adalah mengisi elemen tabel T dengan nilai integer sembarang selama nilai yang dimasukan tidak sama dengan 9999 atau tabel masih belum penuh. Jika tabel T telah penuh, maka akan ditampilkan pesan Tabel Telah Penuh. Proses terakhir adalah dilakukan proses cetak pada layar komputer untuk setiap elemen tabel T secara sequeces.

Program Tabel1

{Mengisi dan menampilkan tabel T}

Kamus

T : array [1..5] of integer {array berukuran 5}
X : integer {nilai yang dimasukan}
i, j : integer {counter}

Algoritma

```
input (X)
i ← 1
while (x <> 9999 and i ≤ 5) {isi tabel}
    Ti ← x
    i ← i + 1
    input (X)
{endWhile}

if (i > 5) then {cek penuh}
    output ('Tabel Sudah Penuh')

j traversal [1..i] {menampilkan isi tabel T}
    output Tj
```

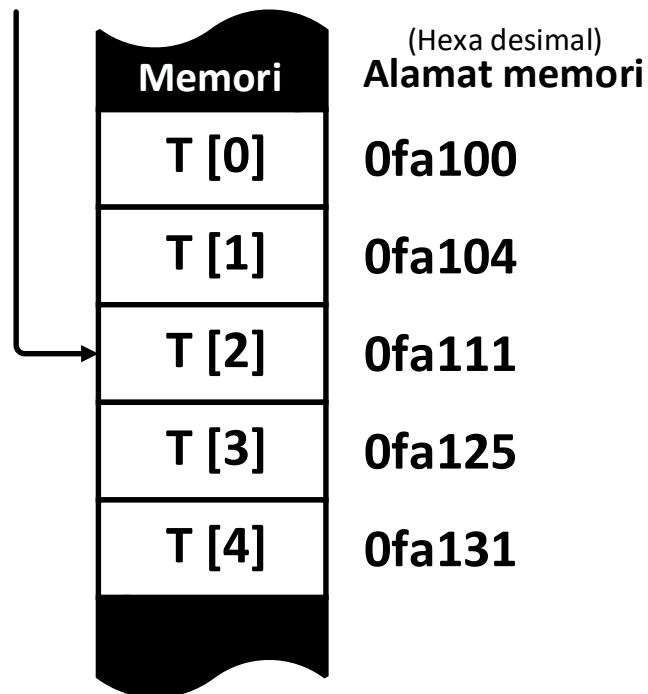
Kode Sumber 7.3. Mengisi dan Menampilkan Tabel T

Representasi sebuah tabel pada sistem komputer adalah dikaitkan dengan lokasi penyimpanan yang di alokasi pada memori sistem komputer tersebut. Alokasi memori untuk sebuah tabel pada sistem komputer dapat dilakukan dengan 2 cara, yaitu secara statik atau dinamik. Untuk memberikan penjelasan mengenai alokasi statik dan dinamik dilakukan dengan menggunakan penjelasan dalam bahasa pemrograman C.

Alokasi Statik merupakan cara melakukan alokasi variabel dalam bentuk tabel pada sebuah memori sistem komputer secara statik. Alokasi statik merupakan sebuah alokasi pada memori sistem komputer yang dilakukan dengan cara fixed sesuai ukuran yang dipesankan, baik digunakan. Misalnya sebuah variabel didefinisikan sebagai tabel dengan ukuran sebesar 10, maka sebanyak 10 ukuran pada memori sistem komputer akan di alokasikan untuk tabel tersebut, baik hanya dipakai 1 ruang memori, atau 2 atau 3, ..., 10, bahkan ekstrem tidak digunakanpun maka alokasi pada ruang memori tetap sebanyak 10 ukuran pada ruang memori. Ruang memori yang

sudah dialokasikan pada sebuah variabel hanya dapat digunakan oleh variabel tersebut saja, dan tidak dapat digunakan oleh variabel lain. Alokasi statik seperti yang telah diberikan di atas dapat dilakukan dengan menggunakan `array` (seperti pada bahasa pemrograman C). Adapun susunan lokasi penyimpanan pada sebuah alokasi memori secara statik pada sebuah array dapat ditunjukkan pada Gambar 7.4.

```
int T [5]; /*Deklarasi tabel T sebanyak 5 ukuran*/
```

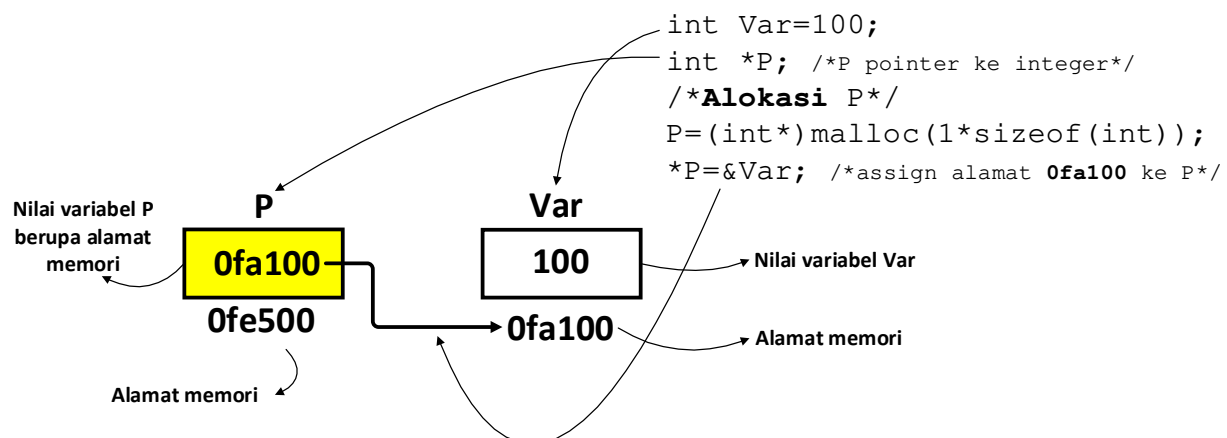


Gambar 7.4. Representasi Lokasi Penyimpanan Alokasi Statik

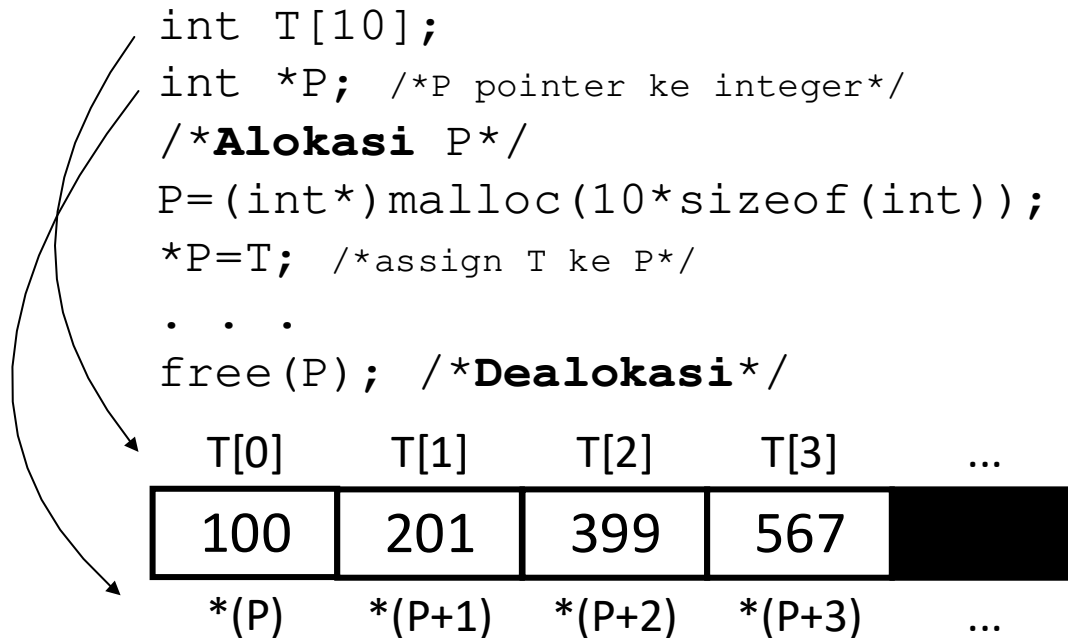
Pada saat dilakukan deklarasi pada sebuah kamus seperti `int T[5]` (pada bahasa pemrograman C), maka pada oleh sistem komputer akan diberikan alokasi sebanyak 5 ruang memori (0fa100, 0fa104, 0fa111, 0fa125, dan 0fa131) yang akan digunakan untuk menyimpan elemen-elemen tabel yang bertipe integer. Kelima ruang memori yang sudah dialokasikan untuk tabel T pada memori, tidak dapat digunakan oleh variabel yang lain walaupun mungkin hanya terisi 1 atau 2 atau belum penuh atau bahkan tidak pernah digunakan untuk mengisi nilai tabel T. Alokasi memori secara statik memiliki konsekuensi meski ada sisa yang belum digunakan, akan tetapi tidak dapat digunakan untuk alokasi yang lain, atau dengan kata lain memori akan terus dipegang

selama belum dilakukan release kembali. Mekanisme ini akan memiliki konsekuensi boros dalam penggunaan memori, jika ukuran yang akan digunakan dengan ukuran yang dipesan tidak tepat.

Alokasi Dinamik merupakan cara melakukan alokasi variabel dalam bentuk tabel pada sebuah memori sistem komputer sebesar ukuran yang sudah diketahui besarnya. Perbedaan utama antara alokasi dinamik dengan alokasi statik adalah pada teknik pemesanannya. Alokasi statik mekanisme yang dilakukan menyiapkan tempat sebanyak yang kira-kira akan digunakan, akan tetapi jika yang digunakan tidak sebesar yang digunakan oleh sistem komputer tetap akan mengalokasi sebesar ukuran yang telah dipesan. Alokasi statik mekanisme pemesanan, setelah diketahui berapa besarnya ukuran yang akan digunakan. Alokasi dinamik dapat direalisasikan dengan menggunakan Pointer pada bahasa pemrograman C. Pointer adalah merupakan sebuah variabel yang berisi alamat dari variabel lain. Pointer dalam bahasa sehari-hari dapat disebut sebagai penunjuk atau dapat disebut sebagai penentu. Pointer secara sederhana dapat diartikan sebagai sebuah tipe data yang nilainya menunjuk pada nilai yang terdapat pada sebuah alamat memori. Namun dalam bahasa pemrograman C, pointer dapat berfungsi sebagai variabel `array`, sehingga pointer dapat sebagai penunjuk pada elemen `array` ke-0 dalam variabel bahasa pemrograman C. Abstraksi pointer sebagai alokasi dinamik dapat diberikan pada Gambar 7.5. dan 7.6.



Gambar 7.5. Representasi Lokasi Penyimpanan Alokasi Dinamik



Gambar 7.6. Representasi Akses Array Alokasi Dinamik (Pointer)

Pointer dalam bahasa pemrograman C memungkinkan untuk melakukan alokasi dinamik, memori dapat di **alokasi** berdasarkan kontrol yang dikendalikan oleh pemrogram jika dibutuhkan. Akan tetapi setelah tidak dibutuhkan, maka oleh pemrogram memori dapat di dealokasi. Kontrol penggunaan memori sangat tergantung oleh pemrogram, maka diperlukan kehati-hatian pada saat alokasi dan dealokasi. Secara umum pointer dalam bahasa pemrograman C dapat diberikan seperti pada Tabel 7.1.

Tabel 7.1. Pointer dalam Bahasa Pemrograman C

Deklarasi	<TypeData> *<NamaVar>
Alokasi	NamaVar = (TypeData*) malloc (sizeof(TypeData))
Assignment	*<NamaVar>=Nilai
Dealokasi	free (<NamaVar>)

Pada saat menggunakan perintah `malloc`, terkadang terdapat kompiler tertentu, diperlukan untuk menyebutkan standart file header `<stdlib.h>`.

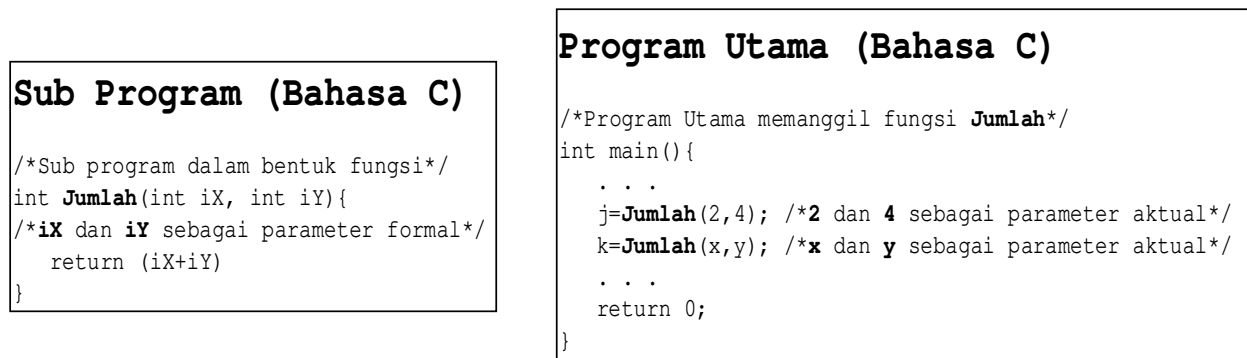
BAB VIII

Sub Program

Sub program adalah merupakan bagian dari program (sering disebut dengan nama program utama). Program atau program utama dapat ditunjukan pada bab-bab sebelumnya, yaitu semua bahasan contoh-contohnya direalisasikan ke dalam program utama. Sub program berdasarkan definisi yang diberikan, dapat diartikan bahwa sub program berada di dalam program utama atau cara penggunaannya (pemanggilannya) selalu melalui program utama. Tujuan membuat sebuah sub program adalah lebih difokuskan pada efisiensi dalam penulisan algoritma. Esensi sub program adalah terletak pada penggunaan sebuah fungsional (proses) secara berkali-kali pada sebuah body program utama. Oleh karena itu dengan sub program, membuatnya cukup satu kali kemudian dapat dipanggil berkali-kali. Jika pada sebuah fungsional (proses) akan digunakan berkali-kali pada sebuah bodi program utama, akan tetapi fungsional (proses) ini tidak direalisasikan dengan menggunakan sub program, maka konsekuensinya fungsional (proses) ini akan dituliskan sebanyak jumlah penggunaannya. Kondisi seperti inilah yang disebut tidak efisien, karena harus berkali-kali menuliskan perintah yang sama. Sebenarnya dalam membuat sebuah algoritma baik menggunakan sub program atau tidak, hal ini secara output tidak akan memberikan perbedaan, hanya saja yang berbeda pada sisi efisiensi penulisan perintah. Sub program terdiri dari 2 bentuk, yaitu dapat dalam bentuk fungsi dan prosedur. Sebelum membahas mengenai bentuk sub program (fungsi dan prosedur), terlebih dahulu diberikan mengenai terminology parameter formal dan parameter aktual yang dapat dimiliki oleh sub program.

Parameter formal adalah merupakan sebuah variabel yang terdapat pada list daftar parameter yang digunakan untuk mendefinisikan fungsi atau prosedur. Kemudian yang disebut dengan parameter aktual adalah sebuah variabel atau nilai yang digunakan pada saat memanggil

sebuah fungsi atau prosedur. Sebagai ilustrasi mengenai parameter formal dan parameter aktual diberikan pada Gambar 8.1.



Gambar 8.1. Parameter Formal dan Parameter Aktual

Parameter formal dan aktual yang diberikan pada Gambar 8.1. menggunakan penjelasan dalam bahasa pemrograman C. Variabel *iX* dan variabel *iY* pada pendefinisian sub program dalam bentuk fungsi disebut sebagai parameter formal. Nilai 2, nilai 4, variabel *x*, dan variabel *y* pada program utama parameter yang digunakan untuk memanggil fungsi *Jumlah*, sehingga Nilai 2, nilai 4, variabel *x*, dan variabel *y* disebut sebagai parameter aktual. Selain dapat menggunakan nilai yang digunakan pada parameter aktual, dapat juga menggunakan variabel dan sebuah ekspresi matematika. Ekspresi matematika yang dimaksud adalah seperti pada pemanggilan fungsi *Jumlah* dengan cara *Jumlah* ($2+3, 5-y$), $2+3$ dan $5-y$ adalah sebuah ekspresi matematika. Nama variabel pada parameter aktual dapat tidak sama namanya dengan variabel formal, akan tetapi harus memiliki tipe data sama. Parameter formal dan aktual terdiri dari 3 jenis yang meliputi: input, output, dan input/output.

Parameter input adalah sebuah variabel sebagai parameter pada sub program yang memiliki sifat sebagai input. Sebagai contoh yang dapat digunakan parameter input adalah misalnya pada persoalan menghitung nilai rata-rata dari sekumpulan data integer (tabel T) sebanyak *N* jumlah data. Dari persoalan tersebut dapat diberikan analisis bahwa yang merupakan variabel sebagai parameter input adalah kumpulan data integer (tabel T), dan banyaknya jumlah

data N. Tabel T dan N memiliki peran sebagai inputan untuk menghitung nilai rata-rata, sehingga kedua variabel ini dinyatakan sebagai parameter input.

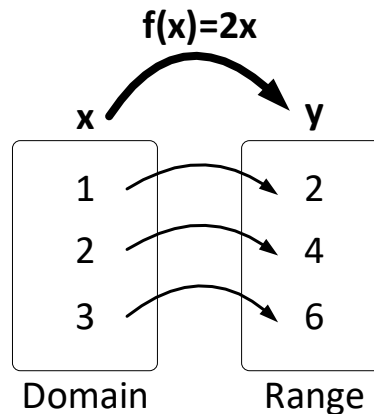
Parameter output adalah sebuah variabel sebagai parameter pada sub program yang memiliki sifat sebagai output. Dengan menggunakan contoh sebelumnya pada parameter input untuk persoalan menghitung nilai rata-rata, maka variabel nilai rata-rata hanya akan dapat diperoleh setelah tabel T dan N sebagai input di proses. Variabel nilai rata-rata berperan sebagai parameter output yang dihasilkan dari sebuah proses terhadap parameter-parameter input.

Parameter input/output adalah sebuah variabel sebagai parameter pada sub program yang memiliki sifat sebagai input dan sekaligus sebagai output. Sebagai contoh pada persoalan menukar 2 buah bilangan integer sembarang j dan k . Misalkan $j=2$ dan $k=4$, maka setelah dilakukan proses menukar 2 buah bilangan integer tersebut akan diperoleh nilai $j=4$ dan $k=2$. Dalam hal ini bilangan integer j dan k berperan sebagai input dan output dari proses menukar adalah bilangan integer j dan k dengan nilai yang dipertukarkan. Oleh karena itu variabel j dan k berperan sebagai parameter input sekaligus output.

Untuk memperdalam mengenai pemahaman jenis-jenis parameter yang dapat dimiliki oleh sebuah sub program, maka sangat disarankan untuk melakukan eksplorasi terhadap contoh-contoh persoalan lainnya. Selanjutnya kembali pada bagian sub program yang dapat berbentuk sebuah fungsi atau prosedur.

Fungsi berdasarkan definisi pada matematika, adalah merupakan sebuah transformasi pemetaan dari satu nilai ke nilai lainnya (pemetaan satu ke satu) seperti yang diberikan pada Gambar 8.2. Setiap x (domain) akan dipetakan tepat satu ke y (range), range atau lebih dikenal dengan nama daerah hasil. Misalkan untuk $x=1$, maka oleh fungsi $f(x)=2x$ akan diperoleh hasil $y=2$, dan seterusnya sehingga setiap nilai x akan memiliki satu nilai (berpasangan dengan satu nilai) y . Secara algoritmik, fungsi sebagai sebuah sub program akan menerima suatu nilai yang

diberikan melalui parameter formal yang bertipe tertentu (jika ada) dan menghasilkan suatu nilai sesuai dengan domain yang didefinisikan pada spesifikasinya. Parameter formal yang dimiliki oleh fungsi hanya parameter input. Nilai yang dihasilkan oleh fungsi akan dikirimkan sebagai return value dari sebuah fungsi.



Gambar 8.2. Abstraksi Fungsi pada Matematika

Secara penulisan teks algoritma, fungsi diberikan nama dan parameter formal, serta harus didefinisikan pada kamus. Fungsi yang didefinisikan dapat dipanggil untuk dilakukan eksekusi melalui namanya yang diikuti dengan parameter aktualnya.

Bentuk umum notasi algoritmik fungsi diberikan pada Kode Sumber 8.1.

```
Function NamaFungsi(<ListParameterInput>) → TipeDataHasil  
{Spesifikasi fungsi}  
  
Kamus Lokal  
{semua Nama yang dipakai pada body algoritma}  
  
Algoritma  
{deretan instruksi input, output, sequeces, analisa kasus, dan  
atau perulangan}  
{nilai yang dikirimkan fungsi harus sesuai dengan TipeDataHasil}  
→ Hasil {return value}
```

Kode Sumber 8.1. Notasi Algoritmik Fungsi

Cara pemanggilan fungsi secara lebih rinci diberikan pada Kode Sumber 8.2. dengan mengacu pada behavior dari fungsi.

Program DriverMainPersoalan

{Spesifikasi program: input, proses, dan output}

Kamus

{semua Nama yang dipakai pada body algoritma}

Function NamaFungsi(<ListParameterInput>) → TipeDataHasil
{Spesifikasi fungsi}**Algoritma**

{deretan instruksi input, output, sequeces, analisa kasus, dan atau perulangan}

{nilai yang dikirimkan fungsi harus sesuai dengan TipeDataHasil}

Nama ← NamaFungsi(<ListParameterInput>)

output (NamaFungsi(<ListParameterInput>))

{Nilai yang dihasilkan fungsi dapat dikombinasikan menggunakan operasi aritmatika untuk menghasilkan sebuah ekspresi}

Kode Sumber 8.2. Notasi Algoritmik Cara Pemanggilan Fungsi

Sebagai contoh akan diberikan realisasi fungsi yang diberikan pada $f(x)=2x$ pada Gambar 8.1. dengan nama fungsi Linier. Fungsi $f(x)=2x$ memiliki 1 parameter input x (sebagai parameter formal) yang bertipe integer. Oleh karena parameter input x bertipe integer, maka tipe hasil yang akan diberikan adalah bertipe integer. Hal ini dapat ditunjukkan bahwa 2 (bilangan integer) dikalikan dengan x bilangan integer akan selalu menghasilkan bilangan integer juga. Teks algoritma pada fungsi Linier diberikan pada Kode Sumber 8.3.

Function Linier(x : integer) → integer{Diberikan nilai x integer dan menghitung $f(x)=2x$ }**Kamus Lokal****Algoritma**→ ($2 * x$) {return value}**Kode Sumber 8.3.** Notasi Algoritmik Fungsi Linier $f(x)=2x$

Esensi sebuah sub program adalah membuat sebuah modul yang bersifat generik, sehingga sangat disarankan untuk tidak menggunakan perintah output atau sesuatu yang terkait dengan tampilan (aksesori). Tampilan (aksesoris) sebaiknya dilakukan pada program utama, sehingga

dapat dikomunikasikan dengan pengguna sesuai kebutuhan. Sub program pada saat dilakukan eksekusi hanya digunakan untuk melakukan pengecekan secara sintak, akan tetapi untuk mengetahui kebenaran logika dari sub program haruslah dilakukan pemanggilan pada program utama dengan cara menyebutkan pada kamus. Oleh karena itu untuk melakukan pengecekan kebenaran logika pada fungsi Linier(x) harus dilakukan dengan memanggil fungsi tersebut pada program utama, seperti yang diberikan pada Kode Sumber 8.4.

Untuk dapat menggunakan sebuah sub program, terlebih dahulu harus disebutkan pada kamus. Cara menyebutkan pada kamus cukup dilakukan dengan menyebutkan spesifikasi pada sub program tersebut. Cara memanggil fungsi dapat dilakukan dengan beberapa cara, yaitu dilakukan assignment ke variabel, atau langsung dilakukan pada perintah output. Pemanggilan pada sebuah fungsi, karena fungsi memiliki nilai, maka pemanggilan fungsi dapat diekspresikan menggunakan ekspresi aritmatika. Kemudian parameter aktual yang digunakan tidak harus memiliki nama yang sama pada nama parameter formal, akan tetapi yang harus memiliki tipe sama. Parameter aktual yang digunakan selain variabel dapat juga menggunakan nilai secara langsung dari variabel yang didefinisikan sebagai parameter aktual.

Program DriverMain

```
{memanggil fungsi Linier,  $f(x)=2x$ }
```

Kamus

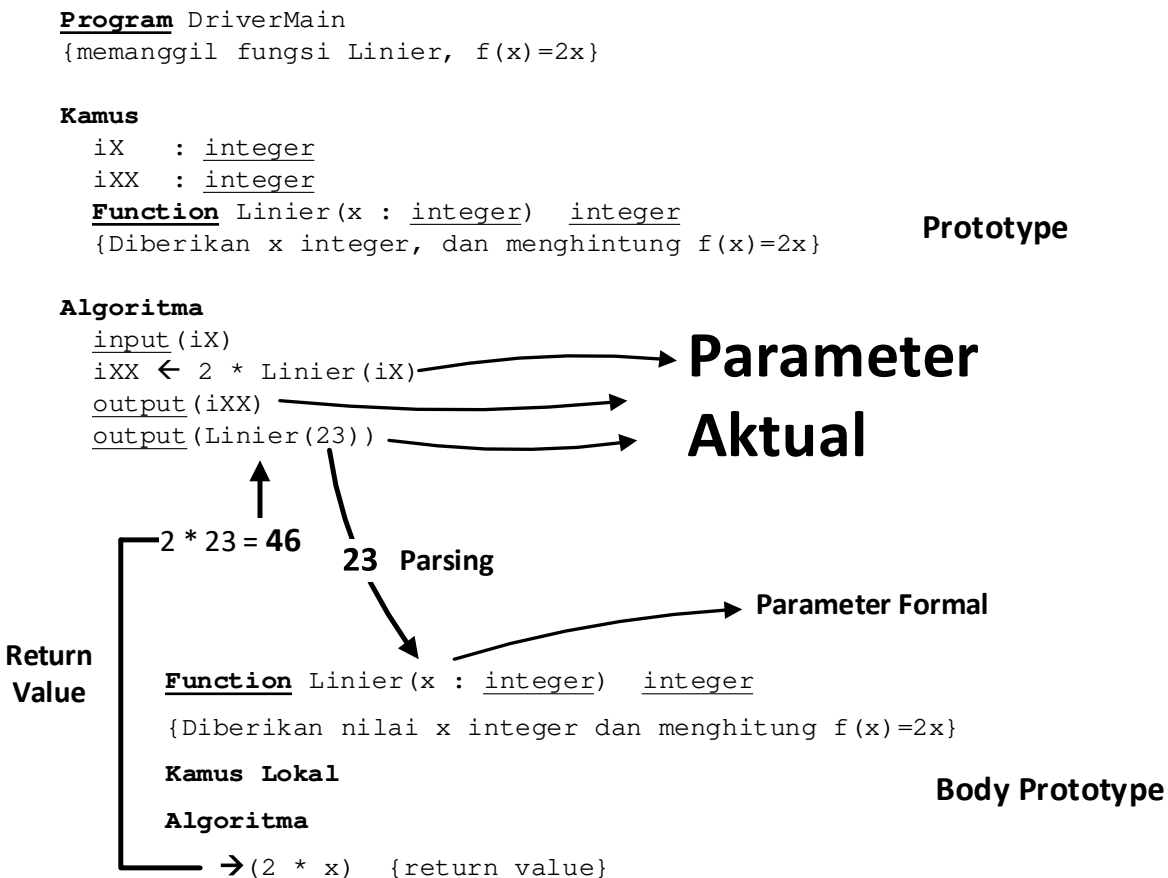
```
iX   : integer  
{integer sembarang yang dimasukan melalui keyboard}  
iXX  : integer  
{variabel memorisasi untuk menyimpan nilai fungsi}  
{PROTOTYPE}
```

```
Function Linier(x : integer) → integer  
{Diberikan x integer, dan menghitung  $f(x)=2x$ }
```

Algoritma

```
input(iX)  
{fungsi di kombinasikan dengan operasi aritmatika menghasilkan}  
{sebuah ekspresi}  
iXX ← 2 * Linier(iX) {memorisasi}  
output(iXX)  
{fungsi dioutputkan secara langsung}  
output(Linier(23))
```

Kode Sumber 8.4. Notasi Algoritmik Pemanggilan (Penggunaan) Fungsi Linier $f(x)=2x$

**Gambar 8.3.** Parsing Parameter

Prototype adalah merupakan definisi dan spesifikasi dari fungsi yang disebutkan pada kamus untuk digunakan (dipanggil) pada program utama. Body Prototype adalah merupakan realisasi dari prototype dalam bentuk teks algoritma. Parsing parameter seperti yang diberikan pada Gambar 8.3. terjadi pada saat pemanggilan sebuah fungsi menggunakan nama beserta parameter-nya. Pemanggilan fungsi `Linier(23)` pada program utama akan terjadi mekanisme nilai 23 (parameter aktual) akan di parsing ke body prototype fungsi `linier`. Nilai 23 akan menggantikan parameter formal x dan selanjutnya diproses $2 \times 23 = 46$, nilai 46 sebagai nilai return value yang akan dikirimkan kembali ke program utama sehingga berdasarkan perintah yang diberikan, maka nilai 46 akan ditampilkan di layar. Demikian mekanisme parsing parameter yang terjadi pada `Linier(iX)`. Fungsi memiliki sebuah nilai, sehingga dengan fungsi dapat melakukan operasi aritmatika dengan membuat sebuah ekspresi matematika, seperti pada perintah

$i_{XX} \leftarrow 2 * \text{Linier}(i_X)$. Selain dikombinasikan dengan operasi aritmatika sebagai sebuah ekspresi, fungsi dapat juga dioutputkan secara langsung karena fungsi memiliki nilai.

Prosedur adalah merupakan sederetan instruksi yang diberikan nama dan menghasilkan sebuah efek yang terdefinisi. Secara algoritmik mendefinisikan sebuah prosedur berarti menentukan nama prosedur tersebut dan parameter formalnya (jika ada) serta sekaligus mendefinisikan **Initial State (IS)** dan **Final State (FS)**. IS adalah kondisi awal sebelum prosedur dijalankan, sedangkan FS adalah kondisi akhir setelah prosedur dijalankan. Prosedur dan parameter formal (jika ada) harus didefinisikan pada kamus sebelum digunakan pada program utama. Prosedur dapat didefinisikan dengan menggunakan parameter formal atau tanpa parameter formal. Prosedur tanpa parameter formal, yaitu dengan memanfaatkan nama pada kamus global, harus berhati-hati untuk program yang sangat besar dan implementasinya sudah banyak file. Kamus global adalah sebuah variabel atau konstanta yang didefinisikan diluar program utama dan berlaku pada semua body program.

Prosedur secara prinsip sama dengan fungsi, akan tetapi perbedaannya terletak pada kepemilikan parameter formalnya yang dapat berupa parameter input, output dan input/output. Perbedaan yang lainnya bahwa pada prosedur tidak memiliki return value, sehingga prosedur tidak dapat dikombinasikan dengan operasi aritmatika atau langsung di outputkan pada devices keluaran. Cara penggunaan (pemanggilan) prosedur juga berbeda dengan fungsi, perhatikan pada Kode Sumber 8.6.

Bentuk umum notasi algoritmik prosedur diberikan pada Kode Sumber 8.5.

```
Procedure NamaProsedur(input/output:<ListParameterFormal>)  
{Spesifikasi prosedur, IS, dan FS}  
  
Kamus Lokal  
{semua Nama yang dipakai pada body algoritma}  
  
Algoritma  
{deretan instruksi input, output, sequeces, analisa kasus, dan  
atau perulangan}
```

Kode Sumber 8.5. Notasi Algoritmik Prosedur

Program DriverMainPersoalan

{Spesifikasi program: input, proses, dan output}

Kamus

{semua Nama yang dipakai pada body algoritma}

Procedure NamaProsedur(input/output:<ListParameterFormal>)

{Spesifikasi prosedur, IS, dan FS}

Algoritma

{deretan instruksi input, output, sequeces, analisa kasus, dan atau perulangan}

{nilai yang dikirimkan fungsi harus sesuai dengan TipeDataHasil}

NamaProsedur (<ListParameterAktual>)

{Nilai yang dihasilkan prosedur tidak dapat digunakan pada sebuah ekspresi atau dioutputkan secara langsung}

Kode Sumber 8.6. Notasi Algoritmik Cara Pemanggilan Prosedur**Contoh 8.1.**

Buatlah sebuah sub program (Tukarkan) yang digunakan untuk menukarkan 2 buah bilangan integer sembarang iX dan iY.

Procedure Tukarkan(input/output: iX,iY: integer)

{Menukarkan 2 bilangan integer yang disimpan pada iX dan iY}

{IS: iX=i dan iY=j}

{FS: iX=j dan iY=i}

Kamus LokalTemp: integer {memorisari}**Algoritma**

Temp ← iX {Temp=i, iX=i, iY=j}

iX ← iY {Temp=i, iX=j, iY=j}

iY ← Temp {Temp=i, iX=j, iY=i}

Kode Sumber 8.7. Teks Algoritmik Prosedur Tukarkan

Berdasarkan persoalan yang diberikan pada Contoh 8.1., variabel iX dan iY sebagai parameter formal adalah merupakan input sekaligus output (input/output) sehingga berdasarkan

kepemilikan parameter formal, maka sub program Contoh 8.1. paling tepat direalisasikan sebagai prosedur.

Kemudian cara penggunaan (pemanggilan) prosedur `Tukarkan(iX, iY)` diberikan pada Kode Sumber 8.8. Mekanisme parsing parameter yang berlaku pada prosedur, sama seperti pada fungsi, hanya pada prosedur tidak ada nilai yang dikirimkan kembali sebagai return value.

Program DriverMainPersoalan

```
{cara memanggil prosedur Tukarkan(iX, iY) }
```

Kamus

`X, y: integer {nilai yang dipertukarkan}`

Procedure `Tukarkan(input/output: iX, iY: integer)`

`{Menukarkan 2 bilangan integer yang disimpan pada iX dan iY}`

`{IS: iX=i dan iY=j}`

`{FS: iX=j dan iY=i}`

Algoritma

`input(x, y)`

`Tukarkan(x, y) {x diparsing menggantikan iX, dan y gantikan iY}`

`output(x, y)`

Kode Sumber 8.8. Pemanggilan Prosedur `Tukarkan(iX, iY)`

Seperti yang telah diberikan pada bagian sebelumnya, bahwa sub program dapat berbentuk fungsi atau prosedur. Kemudian pertanyaan yang timbul adalah kapan sebuah persoalan sub program harus diselesaikan sebagai fungsi atau prosedur. Solusi sebagai fungsi atau prosedur sangat tergantung analisis yang dilakukan terhadap parameter global yang didefinisikan sebagai spesifikasi pada sub program. Jika parameter hanya input maka dijadikan sebagai fungsi, dan selain itu dijadikan sebagai prosedur. Selain justifikasi berdasarkan jenis parameter formal, dapat juga digunakan justifikasi berdasarkan skala kebutuhan penggunaan. Jika output (nilai) dari sebuah sub program akan digunakan untuk dapat di proses lebih lanjut pada sebuah program pemanggil, maka pilihannya adalah dijadikan sebagai fungsi. Hal yang perlu dipertimbangkan adalah pada variabel selain input dijadikan sebagai variabel yang didefinisikan pada kamus lokal (bukan sebagai parameter) sedangkan variabel yang bersifat input didefinisikan sebagai parameter formalnya. Jika tidak terdapat kebutuhan untuk di proses lebih lanjut, maka sub program dapat dijadikan sebagai prosedur.

Untuk lebih memberikan pemahaman terhadap sub program yang dapat berbentuk fungsi atau prosedur, maka diberikan contoh mengenai pemrosesan tabel secara umum dengan spesifikasi kamus seperti yang diberikan pada Kode Sumber 8.9.

Kamus

{Kamus umum yang digunakan untuk pemrosesan tabel}

constant IdxMax : integer = 100

constant IdxMin : integer = 1 {batas bawah dan atas tabel}

type ElmtType : integer {type yang terdefinisi}

type IdxType : integer

type TabInt: {tabel yang terdefinisi dengan index efektif}

<

TI : array [IdxMin..IdxMax] of ElmtType, /*Tabel*/

NEff : integer /*Indek efektif Tabel*/

>

{KONSTRUCTOR}

Procedure CreateEmpty(output : T : TabInt)

{Proses : Mengeset nilai T.NEff dengan nilai 0.}

{I.S : Sembarang.}

{F.S : Terbentuk tabel T kosong dengan kapasitas Nmax-Nmin+1.}

{OPERASI TERHADAP INDEKS TABEL}

Function IsIdxValid(T : TabInt, i : IdxType)→boolean

{Mengirimkan true apabila i merupakan index yang valid bagi}

{TabInt T, yaitu IdxMin <= i <= IdxMax.}

Function IsIdxEff(T : TabInt, i : IdxType)→boolean

{Mengirimkan true apabila i merupakan indeks yang efektif}

{berisi suatu nilai, yaitu idxmin <= i <= NEff.}

{PREDIKAT TABEL}

Function IsEmpty(T : TabInt)→ boolean

{Mengirimkan true apabila T merupakan TabInt yang kosong,}

{yaitu apabila T.NEff == 0.}

Function IsFull(T : TabInt)→ boolean

{Mengirimkan true apabila T merupakan TabInt yang penuh,}

{yaitu apabila T.NEff == IdxMax.}

{ISI TULIS, HUBUNGAN DENGAN I/O DEVICE}**Procedure** PrintTabel(input : T : TabInt)

```
{Proses : Melakukan traversal sebanyak banyaknya elemen}
{      efektif, dan menuliskan nilainya ke layar.}
{I.S : T boleh kosong.}
{F.S : Elemen tabel ditulis ke layar, jika tidak kosong.}
{      jika tabel kosong menampilkan "Tabel Kosong."}
```

Procedure IsiTabel(output : T : TabInt, input : N : ElmtType)

```
{Proses : Melakukan traversal sebanyak N untuk mengisi nilai}
{      elemen-elemen Tabel T dengan 2 kali indexnya}
{I.S : Sembarang.}
{F.S : Tabel T terdefinisi.}
```

{NILAI EKSTRIM}**Function** ValMax(T : TabInt)→ElmtType

```
{Mengirimkan nilai maksimum TabInt T.}
{Pre Condition : Tabel tidak kosong.}
```

Function ValMin(T : TabInt)→ElmtType

```
{Mengirimkan nilai minimum TabInt T.}
{Pre Condition : Tabel tidak kosong.}
```

{INDEX NILAI EKSTRIM}**Function** IdxValMax(T : TabInt)→ElmtType

```
{Mengirimkan index nilai maksimum TabInt T.}
{Pre Condition : Tabel tidak kosong.}
```

Function IdxValMin(T : TabInt)→ElmtType

```
{Mengirimkan index nilai minimum TabInt T.}
{Pre Condition : Tabel tidak kosong.}
```

Kode Sumber 8.9. Kamus Umum Pemrosesan Tabel

Primitif-primitif sub program yang diberikan pada Kode Sumber 8.9. untuk melakukan realisasi ke dalam teks algoritma harus memperhatikan (berdasarkan) definisi dan spesifikasi yang diberikan pada setiap primitif sub program. Adapun secara keseluruhan teks algoritma untuk setiap primitif diberikan pembahasan di bawah ini.

Procedure CreateEmpty(output : T : TabInt)
{Proses : Mengeset nilai T.NEff dengan nilai 0.}
{I.S : Sembarang.}
{F.S : Terbentuk tabel T kosong dengan kapasitas Nmax-Nmin+1.}

Kamus Lokal

Algoritma

T.NEff \leftarrow 0

Kode Sumber 8.10. Realisasi Algoritma CreateEmpty

Function IsIdxValid(T : TabInt, i : IdxType) \rightarrow boolean
{Mengirimkan true apabila i merupakan index yang valid bagi}
{TabInt T, yaitu IdxMin \leq i \leq IdxMax.}

Kamus Lokal

Algoritma

$\rightarrow ((i \geq \text{IdxMin}) \ \&\& \ (i \leq \text{IdxMax}))$

Kode Sumber 8.11. Realisasi Algoritma IsIdxValid

Function IsIdxEff(T : TabInt, i : IdxType) \rightarrow boolean
{Mengirimkan true apabila i merupakan indeks yang efektif}
{berisi suatu nilai, yaitu idxmin \leq i \leq NEff.}

Kamus Lokal

Algoritma

$\rightarrow ((i \geq \text{IdxMin}) \ \&\& \ (i \leq \text{T.NEff}))$

Kode Sumber 8.12. Realisasi Algoritma IsIdxEff

Function IsEmpty(T : TabInt) \rightarrow boolean
{Mengirimkan true apabila T merupakan TabInt yang kosong,}
{yaitu apabila T.NEff == 0.}

Kamus Lokal

Algoritma

$\rightarrow (\text{T.NEff} == 0)$

Kode Sumber 8.13. Realisasi Algoritma IsEmpty

Function IsFull(T : TabInt) → boolean
{Mengirimkan true apabila T merupakan TabInt yang penuh,}
{yaitu apabila T.NEff == IdxMax.}

Kamus Lokal

Algoritma

→ (T.NEff == IdxMax)

Kode Sumber 8.14. Realisasi Algoritma IsFull

Procedure PrintTabel(input : T : TabInt)
{Proses : Melakukan traversal sebanyak banyaknya elemen}
{ efektif, dan menuliskan nilainya ke layar.}
{I.S : T boleh kosong.}
{F.S : Elemen tabel ditulis ke layar, jika tidak kosong.}
{ jika tabel kosong menampilkan "Tabel Kosong."}

Kamus Lokal

i : integer {counter}

Algoritma

if (T.NEff!=0) then
 i traversal [IdxMin..T.NEff]
 output (i,T.TI[i])
else
 output ('Tabel Kosong')

Kode Sumber 8.15. Realisasi Algoritma PrintTabel

Procedure IsiTabel(output : T : TabInt, input : N : ElmtType)
{Proses : Melakukan traversal sebanyak N untuk mengisi nilai}
{ elemen-elemen Tabel T dengan 2 kali indexnya}
{I.S : Sembarang.}
{F.S : Tabel T terdefinisi.}

Kamus Lokal

i : integer {counter}

Algoritma

CreateEmpty(T)
if (N>=IdxMin and N<=IdxMax) then
 i traversal [1..N]
 T.TI_i ← 2*i

```
T.NEff ← i  
else  
  output('Harus N>=IdxMin && N<=IdxMax')
```

Kode Sumber 8.16. Realisasi Algoritma IsiTabel

Function ValMax(T : TabInt) → ElmtType
{Mengirimkan nilai maksimum TabInt T.}
{Pre Condition : Tabel tidak kosong.}

Kamus Lokal

max : ElmtType {nilai maksimum}
i : integer {counter}

Algoritma

```
max ← T.TI[1]  
i traversal [IdxMin+1..T.NEff]  
  if(T.TI[i]>max) then  
    max ← T.TIi  
→ max
```

Kode Sumber 8.17. Realisasi Algoritma ValMax

Function ValMin(T : TabInt) → ElmtType
{Mengirimkan nilai minimum TabInt T.}
{Pre Condition : Tabel tidak kosong.}

Kamus Lokal

min : ElmtType {nilai minimum}
i : integer {counter}

Algoritma

```
min ← T.TI[1]  
i traversal [IdxMin+1..T.NEff]  
  if(T.TI[i]<min) then  
    min ← T.TIi  
→ min
```

Kode Sumber 8.18. Realisasi Algoritma ValMin

Function IdxValMax(T : TabInt) → ElmtType
{Mengirimkan index nilai maksimum TabInt T.}
{Pre Condition : Tabel tidak kosong.}

Kamus Lokal

indexMax : ElmtType {index nilai maksimum}
i : integer {counter}

Algoritma

IndexMax ← IdxMin
 i traversal [IdxMin+1..T.NEff]
 if(T.TI[i]>indexMax)then
 indexMax ← i
→ indexMax

Kode Sumber 8.19. Realisasi Algoritma IdxValMax

Function IdxValMin(T : TabInt) → ElmtType
{Mengirimkan index nilai minimum TabInt T.}
{Pre Condition : Tabel tidak kosong.}

Kamus Lokal

indexMin : ElmtType {index nilai minimum}
i : integer {counter}

Algoritma

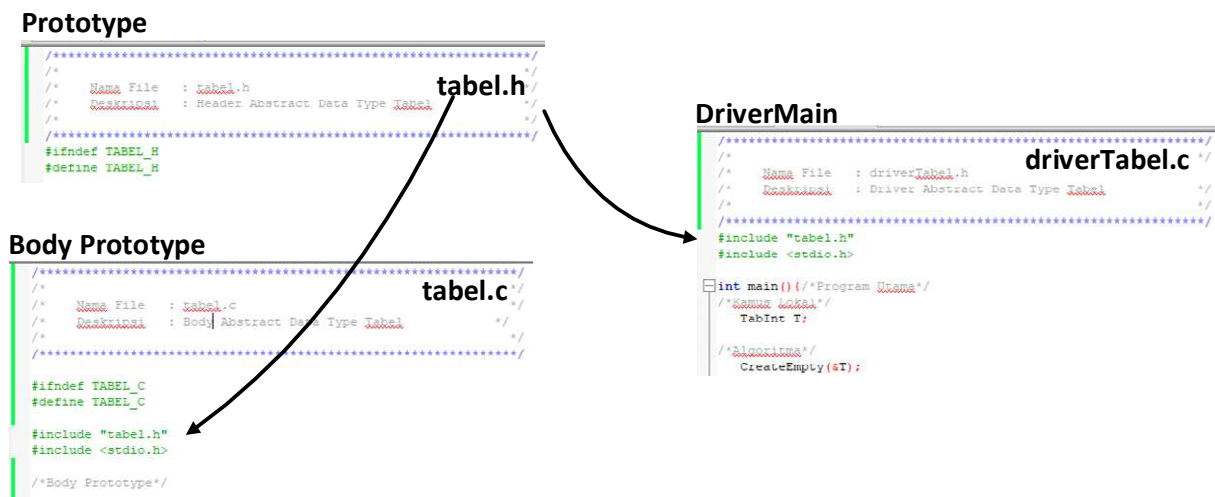
IndexMin ← IdxMin
 i traversal [IdxMin+1..T.NEff]
 if(T.TI[i]<indexMin)then
 indexMin ← i
→ indexMin

Kode Sumber 8.19. Realisasi Algoritma IdxValMin

Pendekatan paradigma yang digunakan pada buku ini adalah menggunakan pendekatan paradigma pemrograman terstruktur (modular). Paradigma pemrograman modular adalah sebuah teknik pemrograman yang dilakukan dengan cara membagi sebuah program menjadi modul kecil-kecil (sub program) dengan tujuan agar lebih mudah dipahami dan digunakan kembali pada program itu sendiri atau program lain yang memiliki proses sama. Oleh karena itu untuk merealisasikan teks algoritma dalam bentuk sub program pada buku ini menggunakan

teknik Abstract Data Type (ADT). ADT adalah koleksi data dan operasi yang dapat digunakan untuk melakukan manipulasi data, tipe data tertentu yang didefinisikan oleh programmer untuk kemudahan pemrograman serta untuk mengakomodasi tipe-tipe data yang tidak secara spesifik diakomodasi oleh bahasa pemrograman. ADT memiliki fungsi untuk memisahkan struktur penyimpanan (data dan tipe) pada lokasi memori dari perilaku (fungsi atau prosedur). Secara umum bentuk ADT diberikan pada Gambar 8.4.

Abstract Data Type (ADT)



Gambar 8.4. Bentuk Umum ADT

Bagian prototype digunakan untuk mendefinisikan struktur data dan spesifikasi sub program (fungsi atau prosedur) yang diberikan sebagai file header. Bagian body prototype digunakan untuk merealisasikan sub program yang telah didefinisikan pada file header. Aturan pemberian nama pada bagian prototype dan body prototype harus sama hanya dibedakan pada ekstensinya. File header diberikan ekstensi *.h, sedangkan bagian realisasi diberikan ekstensi *.c, seperti yang diberikan pada Gambar 8.4. yaitu tabel.h dan tabel.c. tabel.c merupakan realisasi dari file header, maka perlu menyebutkan file header tabel.h pada body programnya dengan cara `#include "tabel.h"`. Kemudian untuk menguji kebenaran logika yang direalisasikan pada body prototype (tabel.c), maka dibutuhkan program utama (driverTabel.c) agar hasilnya dapat dikomunikasikan dengan pengguna dengan cara menyebutkan file header tabel.h.

Fungsi driver main hanya sebagai pengujian kebenaran logika pada realisasi sub program, oleh karena itu untuk realisasinya tidak perlu mengedepankan tampilan (presentasi) akan tetapi fokuskan pada kebenaran logika. Untuk mempermudah pemahaman, maka diberikan contoh realisasi ADT tabel menggunakan bahasa pemrograman C pada Kode Sumber 8.20., 8.21., dan 8.22.

```
/* **** */
/*
/*      Nama File      : tabel.h
/*      Deskripsi      : Header Abstract Data Type Tabel
/*
/* **** */

#ifndef TABEL_H
#define TABEL_H

#include "boolean.h"

#define IdxMax 100
#define IdxMin 1

typedef int ElmtType;
typedef int IdxType;
typedef struct {
    ElmtType TI[IdxMax-IdxMin+1]; /*Tabel*/
    int NEff;                     /*Indek efektif Tabel*/
} TabInt;

/*Prototype*/

/* **** */
/*
/*      KONSTRUCTOR
/*
/* **** */

void CreateEmpty(TabInt* T);
/* Proses : Mengeset nilai T.Neff dengan nilai 0.
/* I.S : Sembarang.
/* F.S : Terbentuk tabel T kosong dengan kapasitas Nmax-Nmin+1.*/

/* **** */
/*
/*      OPERASI TERHADAP INDEKS TABEL
/*
/* **** */
```



```
boolean IsIdxValid(TabInt T, IdxType i);
/* Mengirimkan true apabila i merupakan index yang valid bagi */
/* TabInt T, yaitu IdxMin <= i <= IdxMax. */

boolean IsIdxEff(TabInt T, IdxType i);
/* Mengirimkan true apabila i merupakan indeks yang efektif */
/* berisi suatu nilai, yaitu idxmin <= i <= NEff. */

/*****
/*
/*          PREDIKAT TABEL
/*
*****/

boolean IsEmpty(TabInt T);
/* Mengirimkan true apabila T merupakan TabInt yang kosong, */
/* yaitu apabila T.NEff == 0. */

boolean IsFull(TabInt T);
/* Mengirimkan true apabila T merupakan TabInt yang penuh, */
/* yaitu apabila T.NEff == IdxMax.

/*****
/*
/*          ISI TULIS, HUBUNGAN DENGAN I/O DEVICE
/*
*****/

void PrintTabel(TabInt T);
/* Proses : Melakukan traversal sebanyak banyaknya elemen */
/* efektif, dan menuliskan nilainya ke layar. */
/* I.S : T boleh kosong. */
/* F.S : Elemen tabel ditulis ke layar, jika tidak kosong. */
/* jika tabel kosong menampilkan "Tabel Kosong." */

void IsiTabel(TabInt* T, ElmtType N);
/* Proses : Melakukan traversal sebanyak N untuk mengisi nilai */
/* elemen-elemen Tabel T dengan 2 kali indexnya */
/* I.S : Sembarang. */
/* F.S : Tabel T terdefinisi.

/*****
/*
/*          NILAI EKSTRIM
/*
*****/

ElmtType ValMax(TabInt T);
/* Mengirimkan nilai maksimum TabInt T. */
/* Pre Condition : Tabel tidak kosong.
```

```
ElmtType ValMin(TabInt T);
/* Mengirimkan nilai minimum TabInt T. */
/* Pre Condition : Tabel tidak kosong. */

/*****
/*
/*          INDEX NILAI EKSTRIM
/*
*****/

ElmtType IdxValMax(TabInt T);
/* Mengirimkan index nilai maksimum TabInt T. */
/* Pre Condition : Tabel tidak kosong. */

ElmtType IdxValMin(TabInt T);
/* Mengirimkan index nilai minimum TabInt T. */
/* Pre Condition : Tabel tidak kosong. */

#endif
```

Kode Sumber 8.20. Realisasi Prototype (Fie Header) tabel.h

```
/*****
/*
/*      Nama File      : tabel.c
/*      Deskripsi      : Body Abstract Data Type Tabel
/*
*****/

#ifndef TABEL_C
#define TABEL_C

#include "tabel.h"
#include <stdio.h>

/*Body Prototype*/

/*****
/*
/*          KONSTRUCTOR
/*
*****/

void CreateEmpty(TabInt* T){
/* Proses : Mengeset nilai T.Neff dengan nilai 0. */
/* I.S : Sembarang. */
/* F.S : Terbentuk tabel T kosong dengan kapasitas Nmax-Nmin+1.*/

/*Kamus Lokal*/
```

```
/*Algoritma*/
(*T).NEff=0; /*dapat dengan cara T->Neff=0 karena pointer*/
}

/*****
/*
/*          OPERASI TERHADAP INDEKS TABEL          */
/*
/*
/*****

boolean IsIdxValid(TabInt T, IdxType i){
/* Mengirimkan true apabila i merupakan index yang valid bagi */
/* TabInt T, yaitu IdxMin <= i <= IdxMax.                      */

/*Kamus Lokal*/

/*Algoritma*/
    if (i>=IdxMin && i<=IdxMax){
        return(true);
    }
    else{
        return(false);
    }
    /*atau dipersingkat return ((i >= IdxMin) && (i <= IdxMax));*/
}

boolean IsIdxEff(TabInt T, IdxType i){
/* Mengirimkan true apabila i merupakan indeks yang efektif */
/* berisi suatu nilai, yaitu idxmin <= i <= NEff.          */
/*Kamus Lokal*/

/*Algoritma*/
    if (i>=IdxMin && i<=T.NEff){
        return(true);
    }
    else{
        return(false);
    }
    /*atau dipersingkat return ( ( i >= IdxMin ) && ( i <= T.NEff )
);*/
}

/*****
/*
/*          PREDIKAT TABEL          */
/*
/*
/*****

boolean IsEmpty(TabInt T){
/* Mengirimkan true apabila T merupakan TabInt yang kosong, */
/* yaitu apabila T.NEff == 0.                                */
```

```
/*Kamus Lokal*/

/*Algoritma*/
    return (T.NEff == 0);
}

boolean IsFull(TabInt T){
/* Mengirimkan true apabila T merupakan TabInt yang penuh, */
/* yaitu apabila T.NEff == IdxMax. */

/*Kamus Lokal*/

/*Algoritma*/
    return (T.NEff == IdxMax);
}

/*****
/*
/*          BACA TULIS, HUBUNGAN DENGAN I/O DEVICE
/*
/*
*****/

void PrintTabel(TabInt T){
/* Proses : Melakukan traversal sebanyak banyaknya elemen */
/*          efektif, dan menuliskan nilainya ke layar. */
/* I.S : T boleh kosong. */
/* F.S : Elemen tabel ditulis ke layar, jika tidak kosong. */
/*          jika tabel kosong menampilkan "Tabel Kosong." */

/*Kamus Lokal*/
    int i; /*counter*/

/*Algoritma*/
    if (T.NEff!=0){
        for (i = IdxMin; i <= T.NEff; i++) {
            printf("Data ke-%d : %d\n",i,T.TI[i]);
        }
        /* i > T.NEff */
    }
    else{
        printf("Tabel Kosong\n");
    }
}

void IsiTabel(TabInt* T, ElmtType N){
/* Proses : Melakukan traversal sebanyak N untuk mengisi nilai */
/*          elemen-elemen Tabel T dengan 2 kali indexnya */
/* I.S : Sembarang. */
/* F.S : Tabel T terdefinisi. */
```

```
/*Kamus Lokal*/
int i; /*counter*/

/*Algoritma*/
CreateEmpty(&T); /*karena pointer*/
if (N>=IdxMin && N<=IdxMax) {
    for(i=1;i<=N;i++) {
        (*T).TI[i]=2*i;
        (*T).NEff=i;
    }
}
else{
    printf("Harus N>=IdxMin && N<=IdxMax\n");
}
}

/*****
/*
/*                               NILAI EKSTRIM
/*
/*
/*****/

ElmtType ValMax(TabInt T){
/* Mengirimkan nilai maksimum TabInt T.
/* Pre Condition : Tabel tidak kosong.

/*Kamus Lokal*/
ElmtType max; /*nilai maksimum*/
int i; /*counter*/

/*Algoritma*/
max=T.TI[1];
for(i=IdxMin+1;i<=T.NEff;i++) {
    if(T.TI[i]>max){
        max=T.TI[i];
    }
}
return max;
}

ElmtType ValMin(TabInt T){
/* Mengirimkan nilai minimum TabInt T.
/* Pre Condition : Tabel tidak kosong.

/*Kamus Lokal*/
ElmtType min; /*nilai minimum*/
int i; /*counter*/

/*Algoritma*/
min=T.TI[1];
for(i=IdxMin+1;i<=T.NEff;i++) {
```

```
        if (T.TI[i]<min) {
            min=T.TI[i];
        }
    }
    return min;
}

/*****
/*
/*          INDEX NILAI EKSTRIM          */
/*
/*          */
*****/

ElmtType IdxValMax(TabInt T){
/* Mengirimkan index nilai maksimum TabInt T.          */
/* Pre Condition : Tabel tidak kosong.                  */

/*Kamus Lokal*/
    ElmtType indexMax; /*index nilai maksimum*/
    int i; /*counter*/

/*Algoritma*/
    indexMax=IdxMin;
    for(i=IdxMin+1;i<=T.NEff;i++){
        if (T.TI[i]>indexMax){
            indexMax=i;
        }
    }
    return indexMax;
}

ElmtType IdxValMin(TabInt T){
/* Mengirimkan index nilai minimum TabInt T.          */
/* Pre Condition : Tabel tidak kosong.                  */

/*Kamus Lokal*/
    ElmtType indexMin; /*nilai minimum*/
    int i; /*counter*/

/*Algoritma*/
    indexMin=IdxMin;
    for(i=IdxMin+1;i<=T.NEff;i++){
        if (T.TI[i]<indexMin){
            indexMin=i;
        }
    }
    return indexMin;
}

#endif
```

Kode Sumber 8.21. Realisasi Body Prototype tabel.c

```
/* **** */
/*
/*      Nama File      : driverTabel.h
/*      Deskripsi      : Driver Abstract Data Type Tabel
/*
/* **** */

#include "tabel.h"
#include <stdio.h>

int main(){/*Program Utama*/
/*Kamus Lokal*/
    TabInt T;

/*Algoritma*/
    CreateEmpty(&T);
    printf("Indek Efektif CreateEmpty(TabInt* T)=%d\n", T.NEff);
    printf("IsIdxValid(TabInt T, IdxType i)=%d\n", IsIdxValid(T,2000));
    printf("IsIdxEff(TabInt T, IdxType i)=%d\n", IsIdxEff(T,2));
    printf("IsEmpty(TabInt T)=%d\n", IsEmpty(T));
    printf("IsFull(TabInt T)=%d\n", IsFull(T));
    PrintTabel(T);
    IsiTabel(&T,10);
    PrintTabel(T);
    printf("ValMax(TabInt T)=%d\n", ValMax(T));
    printf("ValMin(TabInt T)=%d\n", ValMin(T));
    printf("IdxValMax(TabInt T)=%d\n", IdxValMax(T));
    printf("IdxValMin(TabInt T)=%d\n", IdxValMin(T));
    return 0;
}
```

Kode Sumber 8.22. Realisasi Program Utama driverTabel.c