Diktat Kuliah
IK-310 Algoritma dan Pemrograman I

Oleh:
Rosa Ariani Sukamto

Program Ilmu Komputer
Universitas Pendidikan Indonesia
2010

Thanks to:

Allah, my husband, my daughter, and my students to become my inspiration sources. Thank you all.

- Rosa A. S.

Kata Pengantar

Puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat-Nya kepada penulis sehingga penulis dapat menyelesaikan buku yang berjudul **Diktat Algoritma dan Pemrograman I** untuk Ilmu Komputer Universitas Pendidikan Indonesia .

Algoritma dan Pemrograman adalah jantung dari pemahaman mengenai pemrograman dan merupakan fondasi awal bagi ilmu komputer. Buku ini dibuat dengan tulus bertujuan agar para mahasiswa dapat lebih mudah memahami bagaimana membuat algoritma dan memetakannya dalam bahasa pemrograman (Amin).

Seperti slogan dalam sebuah iklan mobil "*practice make perfect*". Jika pembaca ingin menjadi seorang yang berkecimpung di dunia teknologi informasi yang hebat maka pembaca harus banyak berlatih dan memahami konsep. Kesabaran sangat dibutuhkan untuk menjadi sukses. Karena kesuksesan adalah keberhasilan yang terencana di mana selalu mampu bangkit dari segala rintangan. Praktikum juga sangat memegang peranan penting dalam mata kuliah ini.

Penulis merasa masih banyak sekali kekurangan dalam penyusunan buku ini. Akhirnya penulis mengucapkan terima kasih kepada semua pihak yang tidak dapat penulis sebutkan satu persatu, yang telah membantu penyusunan buku ini.

Segala saran, kritik, dan pertanyaan dapat dikirimkan melalui email penulis atau disampaikan langsung kepada penulis.

Bandung, Februari 2010

Penulis

Daftar Isi

1 Pengenalan Algoritma

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

Algoritma harus dipikirkan secara logika di pikiran manusia dengan pemikiran yang lebih mudah dimengerti (menggunakan gambaran-gambaran tertentu di dalam pikiran) agar dapat lebih mudah dimengerti oleh manusia dan dapat dengan mudah dipetakan (diubah) menjadi bahasa pemrograman untuk dieksekusi oleh komputer.

Misal secara logika kita dapat membayangkan sebuah solusi harus diselesaikan pada sebuah tanah lapang yang kosong. Lalu kita harus melakukan mengupas kentang di tanah lapang itu, maka yang harus kita lakukan secara runut adalah sebagai berikut:

Membuat kotak kosong di tanah lapang untuk menyimpan kentang yang akan dikupas
Mengisi kotak kosong untuk kentang dengan kentang yang akan dikupas
Membuat kotak kosong di tanah lapang untuk menyimpan pisau yang akan digunakan untuk mengupas kentang
Mengisi kotak kosong untuk pisau dengan pisau yang akan digunakan untuk mengupas kentang
Setelah pisau dan kentang telah ada di tanah lapang, maka barulah kita bisa melakukan proses mengupas kentang

Belajar membuat algoritma seperti halnya belajar menyetir dimana harus belajar membuat algoritma dan memprogram sendiri (mengetik sendiri) untuk menjadi bisa.

Belajar algoritma harus dengan sabar dan menghargai tahap demi tahap pembelajaran

(menghargai proses) untuk dapat memahami algoritma

Belajar algoritma memerlukan banyak latihan (jam terbang dalam membuat algoritma) agar menjadi andal (*practice make perfect*).

Jika Anda mengabaikan ini, maka Anda memilih jalan hidup untuk tidak bisa mengerti algoritma (bisa jadi tidak lulus) karena hidup adalah pilihan yang hasilnya sesuai dengan jalan yang sudah kita jalani. *Keep Fight*.

1.1 Memetakan Algoritma ke dalam Bahasa Pemrograman

Sebagai manusia, kita pasti sudah mengerti jika langkah-langkah mengupas kentang adalah seperti langkah-langkah di atas, masalahnya adalah, kita harus membuat komputer mengerti langkah demi langkah yang kita inginkan sehingga menghasilkan hasil yang kita inginkan. Agar dapat dijalankan pada komputer, maka langkah-langkah solusi yang diinginkan harus menggunakan bahasa yang dimengerti oleh komputer yang dikemas dalam bentuk program komputer.

Bahasa algoritmik (sering juga disebut *pseudo-code*) adalah sebuah bahasa penengah antara manusia dan komputer, sebenarnya komputer tidak langsung dapat mengeksekusi bahasa algoritmik, oleh karena itu berikut adalah tahapan membuat sebuah program komputer yang dapat dieksekusi oleh komputer :

1. Membuat langkah-langkah yang dibutuhkan untuk menyelesaikan masalah dengan menggunakan bahasa manusia seperti contoh pembuatan mengupas kentang di atas secara runut mulai dari inisialisasi, proses penyelesaian dan finalisasi.
   Urutan membuat sebuah algoritma yang baik dan terstruktur adalah sebagai berikut :

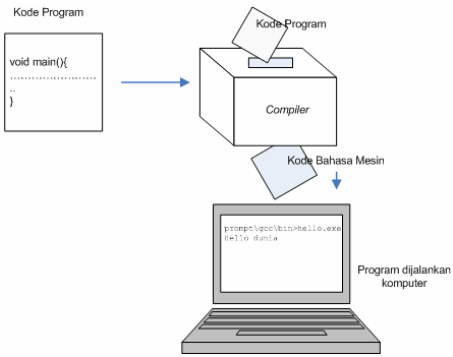
No.	Tahapan Algoritma	Keterangan
1.	Deklarasi	Mendeklarasikan kotak kosong yang dipakai sebagai tempat untuk

No.	Tahapan Algoritma	Keterangan
		menyimpan sesuatu Contoh kode pada tahap deklarasi adalah sebagai berikut : `kentang : integer` `pisau : integer`
2.	Inisialisasi	Merupakan tahapan mempersiapkan proses yang akan dikerjakan untuk menyelesaikan masalah misalnya mengisi kotak-kotak kosong yang akan dipergunakan untuk menyelesaikan permasalahan. Contoh kode pada tahap inisialisasi adalah sebagai berikut: `kentang <- 1` `pisau <- 1`
3.	Proses Penyelesaian Masalah	Merupakan tahapan penyelesaian masalah untuk memenuhi tujuan sebuah algoritma dibuat. Contoh kode tahap ini misalnya : `kentang <- kentang + pisau`
4.	Finalisasi	Merupakan tahapan bersih-bersih atau tahap akhir misalnya menghapus alokasi kotak yang tidak diperlukan lagi atau mengeset isi kotak yang butuh diset pada akhir algoritma. Contoh kode tahap finalisasi misalnya : `pisau <- 0`

2. Membuat langkah-langkah dari bahasa manusia menjadi bahasa algoritmik.

3. Mengubah bahasa algoritmik menjadi bahasa pemrograman yang dimengerti oleh komputer, misalnya dengan bahasa pemrograman pascal, C, C++, atau Java dan menyimpannya dalam sebuah file dengan ekstensi sesuai dengan ketentuan bahasa pemrograman yang digunakan misalnya .pas untuk bahasa pemrograman pascal, dan .c untuk bahasa pemrograman C.
4. Melakukan kompilasi untuk mengetahui apakah kode program mengandung kesalahan penulisan, jika ada kesalahan penulisan kode program maka harus diperbaiki dahulu lalu dikompilasi lagi sampai tidak ada kesalahan penulisan kode lagi; cara mengkompilasi bergantung pada *compiler* atau interpreter yang digunakan misalnya GPC (*GNU Pascal Compiler*) atau Free Pascal untuk Pascal dan GCC (*GNU C Compiler*) untuk bahasa pemrograman C (ketiga *compiler* merupakan *compiler freeware*).
5. Mengeksekusi program pada komputer, cara eksekusi juga bergantung pada *compiler* atau interpreter yang digunakan.

Sebenarnya tidak ada standar yang baku tentang bahasa algoritmik, asalkan bisa dipahami oleh manusia dan dapat dengan mudah diterjemahkan menjadi kode program, maka sudah dapat dianggap sebagai bahasa algoritmik, namun biasanya pada sebuah institusi pendidikan atau bahkan setiap pengajar bisa mempunyai standar sendiri mengenai bahasa algoritmik, dalam buku ini kode yang bergaris bawah pada bahasa algoritmik adalah kata kunci yang aplikasinya dalam pemrograman biasanya telah disediakan oleh *compiler* atau interpreter sehingga kita tidak perlu lagi membuatnya sendiri, hanya tinggal memakai. Mekanisme eksekusi program dapat dilihat pada Gambar 1.



Gambar 1 Mekanisme Eksekusi Program

Gunakanlah *compiler freeware* jika Anda tidak mampu atau tidak mau membeli yang berlisensi. Pembajakan perangkat lunak di Indonesia telah mencapai lebih dari 80%, itu berarti jika Anda membuat sebuah perangkat lunak untuk dijual, Anda hanya akan memperoleh 20 juta dari 100 juta yang seharusnya Anda dapatkan jika tidak ada pembajakan.

Jangan mencuri jika kita tidak memiliki kemampuan membeli atau tidak ingin membeli. Perangkat lunak adalah sebuah kerja keras. Dukung perangkat lunak *freeware* dalam penggunaan dan pembuatan untuk mengurangi pembajakan.

Dengan memakai perangkat lunak bajakan, berarti Anda sudah mencabut hak Anda untuk marah jika perangkat lunak Anda dibajak orang lain.

1.2 Sekilas Tentang Bahasa Pemrograman Pascal dan C

Bahasa pemrograman Pascal memiliki kelebihan lebih terstruktur dan membedakan blok-blok pendeklarasian komponen pemrograman. Pascal membedakan pendeklarasian tipe, variabel, dan penulisan kode program. Pascal memiliki keunggulan untuk dipelajari oleh pemula karena memiliki struktur yang jelas. Disamping itu bahasa pemrograman Pascal tidak *case sensitive* yang berarti tidak membedakan huruf besar dan huruf kecil jadi ketika mengetikkan `writeln` dengan `WriTeLn` maka *compiler* tidak membedakan jadi akan sangat memudahkan bagi pemula yang belum fasih mengetik. Kelemahan dari bahasa Pascal adalah bahwa Pascal tidak memiliki bahasa pemrograman lain yang cara menuliskan kodenya mirip dengan bahasa Pascal sehingga aturan penulisan pada bahasa Pascal hanya dapat dipergunakan untuk bahasa Pascal saja.

Bahasa pemrograman C memiliki kelebihan memiliki bahasa pemrograman lain yang cara menuliskan kode programnya mirip dengan bahasa C. Misalnya bahasa pemrograman Java, PHP, C# sehingga jika

mempelajari bahasa C akan lebih mudah untuk beradaptasi dengan cara berpikir dan penulisan kode program pada bahasa pemrograman Java, PHP, C# jadi jika belajar memprogram dengan bahasa C maka dapat dikatakan juga sekaligus mempelajari bahasa pemrograman Java, PHP, C#. Bahasa C bersifat *case sensitive* yang membedakan huruf besar dan huruf kecil. Jika menuliskan `printf` dan `Printf` pada bahasa C maka *compiler* C akan menganggap kedua tulisan itu berbeda. Hal ini dapat melatih kita untuk lebih teliti dalam mengetikkan kode program.

Apapun bahasa pemrograman yang digunakan tidak menjadi masalah, tapi harus dipastikan bahwa konsep algoritma dimengerti dengan baik agar dapat membuat kode program yang rapi dan terstruktur dengan baik apapun bahasa pemrograman yang digunakan.

Beda *programmer* yang belajar dengan konsep dan tidak adalah pada kerapian dan struktur program yang dibuat.

2 Komentar

Komentar merupakan bagian kode program yang tidak dieksekusi oleh saat program dijalankan. Komentar dianggap penting untuk memperjelas program agar lebih mudah dimengerti dan memberikan informasi-informasi dari kode program yang diperlukan. Pemrograman merupakan penuangan sebuah ide untuk menyelesaikan masalah dengan menggunakan perangkat yang bernama komputer, oleh karena itu sering kali ide hanya muncul pada saat-saat tertentu dan tidak jarang ide yang baru saja kita temukan kita lupakan, apalagi jika dalam kondisi stres. Untuk itulah diperlukan komentar.

Berikut adalah penulisan komentar dalam suatu kode program :

Keterangan	Bahasa Algoritmik	Bahasa Pascal	Bahasa C
penulisan komentar yang hanya satu baris dimana dalam bahasa pemrograman C penulisannya diawali dengan tanda // dan diakhiri dengan enter	{ *end while* }	{ end while }	// end while
penulisan komentar lebih dari satu baris	{ *Nama Program: .....* *Tanggal Pembuatan : 13 Januari 2007* *Fungsi : .....* }	{ Nama Program : ..... Tanggal Pembuatan : 13 Januari 2007 Fungsi : ..... }	/* Nama Program : ..... Tanggal Pembuatan : 13 Januari 2007 Fungsi : ..... */

Berikut adalah fungsi-fungsi utama perlunya komentar dalam sebuah kode program.

- Memberikan informasi pembuatan kode program misalnya sebagai berikut :

```
/*
 *  Nama Programmer : Gadiza Mutia Shalahuddin
 *  Nama Kelas : DataBuku
 *  Versi Kode Program : 1.0
 *  Tanggal Pembuatan :  28 Januari 2007
 *  Fungsi Kelas : Menyimpan data buku
*/
```

biasanya komentar ini diletakkan pada bagian paling atas sebuah kode program agar kita bisa tahu siapa yang bertanggung jawab pada pembuatan kode program, fungsi kode program dan versi pembuatannya karena program dapat mengalami banyak perubahan untuk menambah fungsi program, jika tidak diketahui versi program, bagaimana jika yang mengembangkan program adalah orang yang bukan pemrogram awalnya, pasti menjadi pusing tujuh keliling.

- Memberikan informasi tujuan dibuatnya sebuah prosedur, fungsi, atau metode, misalnya sebagai berikut :

```
/*
 *  Nama Metode :  simpanNama
 * Keterangan Parameter Masukan :
     nama  ->    merupakan  nama  mahasiswa  yang  akan
disimpan
 *  Keadaan Awal :  Nama belum disimpan ke basis data
 *  Keadaan Akhir : Nama telah dismpan ke basis data
*/
void cariNama(char* nama){
 ..............................
}
```

biasanya komentar ini diletakkan di atas sebuah kode prosedur atau fungsi.

- Memberikan informasi fungsi variabel atau konstanta, misalnya sebagai berikut :

```
int menit ; // variabel untuk menyimpan nilai menit
```

biasanya komentar ini digunakan untuk mengetahui fungsi variabel atau konstanta yang dibuat karena nama variabel atau konstanta sangat tergantung selera pembuat program, hal ini sangat penting agar orang lain dapat membaca program dan bila pembuat program telah lupa dengan pemikirannya saat membuat program, ada keterangan untuk mencoba mengingat kembali jalannya program.

- Memberikan informasi langkah-langkah jalannya program, misalnya sebagai berikut :

```
/* Melakukan proses pencarian nama */
while (ketemu == 0){
   ......................
}
```

biasanya komentar ini digunakan untuk mengetahui jalannya proses dalam sebuah program.

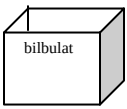
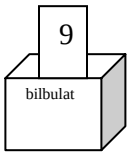
Jika kita mencoba untuk mencari berbagai kode program di internet kita akan banyak menemukan bahwa banyak sekali kode program yang sebagian besar kodenya hanya berisi komentar, dan kita bisa mencoba mengerti dan membaca kode program dengan ide orang lain melalui komentar yang ada. Oleh karena itu komentar memang tidak dijalankan sebagai instruksi oleh komputer, tapi sangat penting dan bermanfaat. Memang dengan menambah banyak komentar akan menambah jumlah baris pada kode program dan dapat memperbesar ukuran file program, tapi komentar sangatlah penting untuk kelangsungan pengembangan program.

Programmer yang handal dan profesional tidak malas menulis komentar ☺.

3 Tipe data

Tipe data adalah pengelompokan data berdasarkan isi dan sifatnya. Dalam logika kita tipe data adalah jenis dari sesuatu yang dapat dimasukkan ke dalam kotak kosong yang hanya khusus dibuat untuk jenis benda dengan jenis tertentu.

Variabel merupakan tempat untuk menyimpan data dengan tipe tertentu yang isinya bisa diubah-ubah sesuai dengan tipenya. Dalam logika kita, kita dapat membayangkan sebuah variabel sebagai sebuah kotak kosong yang dapat diisi dengan sesuatu dengan jenis tertentu, misal kita membuat sebuah variabel berupa bilangan bulat, maka dalam logika, kita sedang membuat kotak kosong yang hanya dapat diisi dengan kertas bertuliskan bilangan bulat, tidak boleh jenis bilangan selain bilangan bulat.

Dalam Logika Pemikiran	Dalam Bahasa Algoritmik
Membuat kotak kosong yang memiliki jenis bilangan bulat, dan kotak itu kita beri nama bilbulat 	`bilbulat : integer`
Mengisi kotak kosong bilbulat dengan kertas berisikan bilangan bulat 9. 	`bilbulat <- 9`

Setelah dibuat bahasa algoritmik maka dapat diubah menjadi bahasa pemrograman sebagai berikut:

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`bilbulat : integer`	`var` `bilbulat : integer;`	`int bilbulat;`
`bilbulat <- 9`	`bilbulat := 9;`	`bilbulat = 9;`

Rahasia belajar algoritma dan pemrograman adalah **sabar dan menghargai proses** tahap demi tahap dari logika manusia ke bahasa pemrograman

3.1 Bilangan Bulat (Integer)

Tipe bilangan bulat pada bahasa algoritmik biasa disebut sebagai integer walau dalam aplikasinya pada bahasa pemrograman, tipe bilangan bulat tidak hanya dinyatakan dengan integer, masih ada tipe lain seperti short dan long yang juga merupakan bilangan bulat, yang membedakan tipe-tipe bilangan bulat itu adalah jangkauan bilangan. Jangkauan tipe bilangan bulat adalah sebagai berikut :

Tipe Data Bilangan Bulat	Jangkauan Nilai pada Bahasa C
long	-2147483648, ..., +2147483647
integer / int	-32768, ..., +32768
short	-128, ..., +127

berikut adalah contoh penulisan variabel dengan menggunakan bahasa algoritmik, bahasa pemrograman Pascal, dan bahasa pemrograman C.

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`integer cangkir`	`var` `cangkir : integer;`	`int cangkir;`
`cangkir <- 1`	`cangkir := 1;`	`cangkir = 1;`

3.2 Bilangan Riil (Floating-Point)

Bilangan riil biasa digunakan untuk menyatakan bilangan yang membutuhkan ketelitian dengan adanya nilai di belakang koma. Berikut adalah cara mendeklarasikan bilangan riil dengan menggunakan bahasa algoritmik, bahasa pemrograman Pascal, dan bahasa pemrograman C :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`real kurs`	`var` `kurs : real;`	`float kurs;`
`kurs <- 1.02`	`kurs := 1.02;`	`kurs = 1.02;`

3.3 Karakter

Karakter atau biasa ditulis char pada pemrograman merupakan tipe data untuk menyimpan sebuah karakter atau gabungan karakter yang merepresentasikan sebuah karakter misalnya karakter-karakter sebagai berikut:

Karakter	Keterangan
`\0`	karakter null (kosong)
`\n`	karakter *newline* (pindah baris)
`\"`	karakter tanda petik dua agar tidak rancu dengan pernyataan string maka diberi tanda *escape* '\'
`\'`	karakter tanda petik satu agar tidak rancu dengan pernyataan karakter maka diberi tanda *escape* '\'
`\\`	karakter *backslash* agar tidak rancu dengan tanda *escape* '\'
`\?`	karakter tanda tanya (?)

Penulisan nilai sebuah karakter dinyatakan di dalam dua buah tanda petik satu (') sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`karakter : char`	`var` `karakter : char;`	`char karakter;`
`karakter <- 'A'`	`karakter := 'A';`	`karakter = 'A';`
`karakter <- '?'`	`karakter := '\?';`	`karakter = '\?';`

3.4 String

String adalah tipe data yang berupa kumpulan karakter (satu atau lebih) yang berada di dalam dua buah tanda petik dua (") dalam bahasa C dan dalam tanda petik satu (') dalam bahasa Pascal. Dalam aplikasinya dalam bahasa pemrograman biasanya tipe string hanya dapat memuat karakter sebanyak 1 sampai 256 karakter.

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`kata : string`	`var` `kata : string;`	`char kata[4];` untuk mengisi variabel kata adalah sebagai berikut : `kata[0] = 'C';` `kata[1] = '+';` `kata[2] = '+';` `kata[3] = '\0';`
`kata <- "ini adalah string"`	`kata :=` `'ini adalah string';`	`char kata[] = "ini adalah string";` banyak karakter tidak perlu ditulis karena variabel kata langsung diisi nilanya sehingga tempat yang disediakan langsung disesuaikan dengan panjang karakter di dalam kumpulan karakter yang diisikan pada variabel, hal ini hanya berlaku jika isi variabel langsung diisikan pada saat variabel di deklarasikan juga dapat ditulis dengan : `char *kata = "ini adalah string ";`

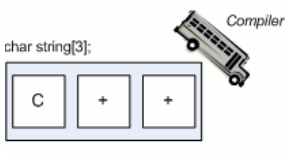
3.4.1 Representasi String pada Bahasa Pemrograman C

Pada bahasa pemrograman C tidak mengenal tipe string secara tersendiri, dalam bahasa pemrograman C string dianggap sebagai sebuah tabel 1 x banyaknya karakter yang dalam penyimpanannya diakhiri dengan karakter null (kosong ('\0')) sebagai tanda berakhirnya pembacaan string oleh *compiler*. Untuk lebih jelasnya tentang representasi string pada bahasa C mari kita lihat gambar berikut :

B	a	h	a	s	a		P	e	m	r	o	g	r	a	m	a	n		C	\0

Gambar 2 Representasi String pada Bahasa Pemrograman C

Isi dari representasi penyimpanan string pada gambar di atas adalah "Bahasa Pemrograman C" dimana pada saat *compiler* membaca kode tersebut, maka *compiler* akan berhenti membacanya sebagai string saat ditemukan karakter null (kosong ('\0')) pada tabel penyimpanan string. Jika karakter null (kosong ('\0')) tidak ditemukan oleh *compiler* sampai pada ruang tabel dengan indeks yang terakhir maka akan terjadi peringatan kesalahan (*error*) pada saat program dijalankan, karena *compiler* akan terus membaca walaupun ruang tabel telah habis dibaca, akhirnya terjadilah apa yang disebut *overflow* yaitu ruang tabel yang dialokasikan telah habis padahal *compiler* belum menemukan tanda berhenti melakukan pembacaan atau untuk secara lebih jelasnya, *compiler* membaca ruang atau tempat yang belum dipesan sehingga ruangan atau tempat yang dimaksud sebenarnya belum ada. Sebagai ilustrasi mari kita lihat gambar berikut :



Gambar 3 Ilustrasi Pembacaan String pada Bahasa Pemrograman C

Pada saat *compiler* membaca tabel karakter string dengan indeks nol (`string [0]`) (indeks tabel pada bahasa pemrograman C dimulai dari angka nol), *compiler* masih bisa melewati karena masih ada ruang, sampai pada indeks kedua *compiler* masih mempunyai ruang, namun karena *compiler* tidak menemukan tanda berhenti *compiler* akan terus berjalan hingga tak ada lagi ruang sehingga program mengalami *error*. *Compiler* akan berhenti melakukan pembacaan sebagai string jika menemukan tanda berhenti ('\0') walaupun ruang kosong tabel masih tersedia. Misalnya jika ruang tabel ada lima, tetapi tanda berhenti ('\0') ada di ruang ketiga, maka *compiler* akan membaca

sebagai string pada ruang ketiga. Hal ini berlaku untuk representasi string dengan menggunakan alokasi tempat yang telah didefinisikan misalnya :

```
char kata[4];
```

untuk representasi string yang menggunakan `char kata[]` dan `char *kata`, tanda berhenti sudah didefinisikan oleh *compiler*.

3.5 Boolean

Boolean merupakan sebuah tipe data untuk menyatakan pernyataan benar atau salah sehingga tipe data ini hanya dapat diisi dengan dua buah nilai yaitu `true` atau `false`. Tipe data boolean biasanya digunakan sebagai penanda apakah sebuah proses telah selesai dilakukan atau belum, misalnya sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik
kupas kentang belum dilakukan	`kupas_kentang : boolean`
kupas kentang sudah dilakukan	`kupas_kentang <- true`

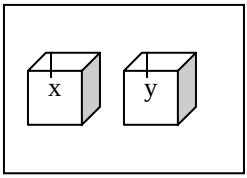
Berikut adalah cara pendeklarasian tipe data boolean :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`kupas_kentang : boolean`	`var` `kupas_kentang boolean;`	`int kupas_kentang;`
`kupas_kentang <- true`	`kupas_kentang := true;`	`kupas_kentang = 1;`
`kupas_kentang <- false`	`kupas_kentang := false;`	`kupas_kentang = 0;`

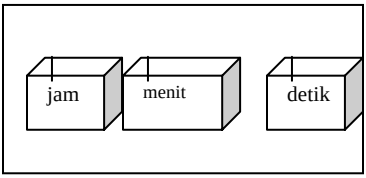
Bahasa pemrograman C tidak mengenal tipe data boolean namun ada beberapa *compiler* yang mengenali tipe bool sebagai pengganti boolean.

3.6 Tipe Terstruktur

Tipe terstruktur adalah tipe data yang isinya ditentukan sendiri oleh pembuatnya, misalnya tipe titik yang berisi koordinat x dan y seperti deklarasi berikut :

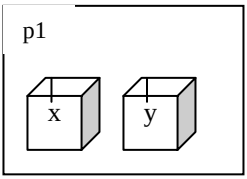
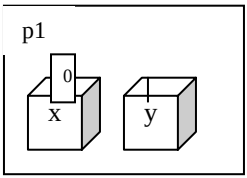
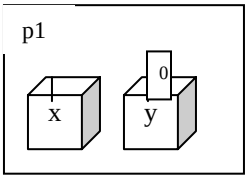
Dalam Logika Pemikiran	Bahasa Algoritmik
Jenis/tipe titik adalah sebuah bungkusan yang memiliki kotak x dan kotak y yang dapat diisi 	`type titik : < x : real, y : real>`

atau tipe data pukul yang terdiri dari jam, menit dan detik seperti deklarasi berikut :

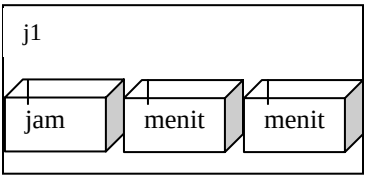
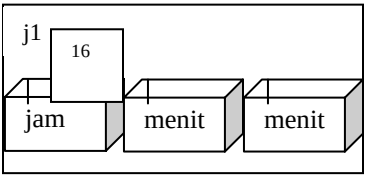
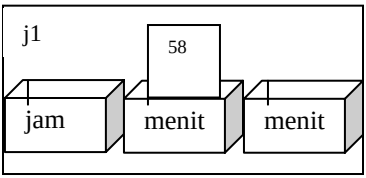
Dalam Logika Pemikiran	Bahasa Algoritmik
Jenis/tipe titik adalah sebuah bungkusan yang memiliki kotak x dan kotak y yang dapat diisi 	`type pukul : <jam : integer, menit : integer, detik : integer>`

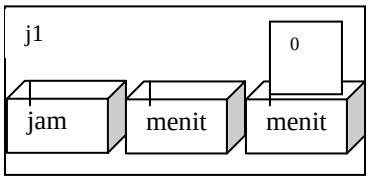
Pendefinisian tipe terstruktur dapat kita logikakan sebagai membuat sebuah jenis, misal jenis manusia, maka memiliki tangan, kaki, kepala, dan bagian lainnya. Jika jenis tipe adalah titik maka akan memiliki koordinat x dan koordinat y.

dalam pemakaiannya tipe titik adalah sebagai berikut :

Dalam Logika Pemikiran	Bahasa Algoritmik
Sebuah jenis benda perlu diberi nama seperti halnya sebuah benda manusia memiliki nama, dan kita memanggil manusia lain dengan menggunakan namanya. Begitu juga titik perlu dibuat bendanya dan diberi nama untuk diakses dalam kode program. Secara logika kita dapat menganalogikan pembuatan benda titik dengan namanya sebagai berikut: 	`p1 : titik`
	`p1.x <- 0`
	`p1.y <- 0`

untuk menyatakan titik x,y bernilai 0,0 sedangkan pemakaian tipe pukul adalah sebagai berikut :

Dalam Logika Pemikiran	Bahasa Algoritmik
Secara logika kita dapat menganalogikan pembuatan benda pukul dengan namanya sebagai berikut: 	`j1 : pukul`
	`j1.jam <- 16`
	`j1.menit <- 58`




```
j1.detik <- 0
```

untuk menyatakan pukul 16 lewat 58 menit.

Berikut adalah deklarasi tipe terstruktur dengan bahasa algoritmik, bahasa pemrograman Pascal, dan bahasa pemrograman C:

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
type nama_tipe : < nama_struktur1 : tipe_data, nama_struktur2 : tipe_data, ... > nama_variabel : nama_tipe nama_variabel.nama_struktur <- nilai	type nama_tipe : record nama_struktur1 : tipe_data; nama_struktur2 : tipe_data; ..... end; var nama_variabel : nama_tipe; nama_variabel.nama_struktur := nilai;	typedef struct{ tipe_data nama_struktur1; tipe_data nama_struktur2; ..... } nama_tipe; nama_tipe nama_variabel; nama_variabel.nama_struktur = nilai;
type titik : < x : real, y : real > p1 : titik p1.x <- 0 p1.y <- 0	type titik : record x : real; y : real; end; var p1 : titik; p1.x := 0; p1.y := 0;	typedef struct{ float x; float y; }titik; titik p1; p1.x = 0; p1.y = 0;

4 Input dan Output

4.1 Menampilkan Nilai Variabel

Nilai dari sebuah variabel dapat ditampilkan ke layar dengan menggunakan fungsi yang telah ada pada pustaka (*library*) dari *compiler* yang digunakan sesuai dengan aturan bahasa pemrograman yang ada. Berikut adalah cara menampilkan nilai variabel pada layar:

<table>
<tr><th>Bahasa Algoritmik</th><th>Bahasa Pascal</th><th>Bahasa C</th></tr>
<tr>
<td>

<u>output</u>(string, nama_variabel_yang_ditampilkan)

misal nilai variabel integer bilBulat adalah 9 maka untuk menampilkan ke layar:

<u>output</u>("bilangan bulat : ", bilBulat)

menghasilkan keluaran :

```
bilangan bulat : 9
```

</td>
<td>

```
write(string,
nama_variabel_yang_ditampi
lkan,              string,
nama_variabel_yang_ditampi
lkan,......);
```

misal:

```
write('bilangan bulat: ',
bilBulat);
```

jika ingin menambahkan ganti baris (*newline*) pada akhir kalimat yang ditampilkan maka dapat menggunakan

```
writeln(string,
nama_variabel_yang_ditampi
lkan,              string,
nama_variabel_yang_ditampi
lkan,......);
```

misal:

```
writeln('bilangan bulat:
', bilBulat);
```

menghasilkan keluaran :

```
bilangan bulat : 9
```

</td>
<td>

```
printf(string,
nama_variabel_yang_dita
mpilkan1,
nama_variabel_yang_dita
mpilkan1, .....);
```

string terdiri dari kata-kata yang ingin ditampilkan dan tempat di dalam string yang merupakan tempat dikeluarkannya nilai variabel ditandai dengan tanda :

tanda	tipe data
%d	long int short
%f	float double long double
%c	char
%s	char nama_variabel []

misal jika bilBulat merupakan variabel bertipe integer bernilai 9 maka cara

</td>
</tr>
</table>

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
		menampilkan ke layar adalah : `printf("bilangan bulat : %d\n", bilBulat);` menghasilkan keluaran : `bilangan bulat : 9`

4.2 Menerima Masukan dari Keyboard

Nilai dari sebuah variabel dapat diisi dengan masukan dari *keyboard* menggunakan fungsi yang telah ada pada pustaka (*library*) dari *compiler* yang digunakan sesuai dengan aturan bahasa pemrograman yang ada. Berikut adalah cara menerima masukan dari *keyboard* :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`input(nama_variabel_yang_menangkap_masukan)` misal `bilBulat` adalah variabel integer yang menangkap masukan maka untuk menangkap masukan dari *keyboard* adalah : `input(bilBulat)`	`read(nama_variabel_yang_menangkap_masukan);` misal: `read(bilBulat);` jika ingin menambahkan ganti baris (*newline*) pada akhir kalimat yang ditampilkan maka dapat menggunakan `readln(nama_variabel_yang_menangkap_masukan);` misal: `readln(bilBulat);` maka variabel `bilBulat` akan menangkap	`scanf("tanda_tipe",&nama_variabel);` tanda tipe adalah sebagai berikut : `%d` — `long` `int` `short` `%f` — `float` `double` `long double` `%c` — `char` `%s` — `char nama_variabel[]` misal jika `bilBulat` merupakan variabel bertipe integer maka cara menangkap masukan integer adalah :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
	masukan dari *user*	`scanf("%d", &bilBulat);` maka variabel `bilBulat` akan menangkap masukan dari *user* (tanda `&` di depan nama varibel merujuk pada alamat di memori yang akan diisi dengan masukan dari *user*)

Contoh Kasus:

- Buat algoritma yang menerima dua buah masukan dan menampilkan hasil pertambahan dari kedua bilangan masukan!

Penyelesaian:

Bahasa Manusia secara Logika	Bahasa Algoritmik
Buat kotak untuk memasukkan dua buah masukan dan sebuah kotak untuk menyimpan hasil pertambahan	`a : integer` `b : integer` `c : integer`
Meminta masukan pemakai (*user*) untuk mengisi kedua buah kotak	`input(a)` `input(b)`
Memasukkan hasil pertambahan kedua masukan	`c <- a + b`
Menampilkan hasil pertambahan	`output(c)`

- Buat algoritma yang menerima sebuah masukan dan menampilkan hasil kuadrat dari bilangan masukan!

Penyelesaian:

Bahasa Manusia secara Logika	Bahasa Algoritmik
Buat kotak untuk memasukkan sebuah masukan dan sebuah kotak untuk menyimpan hasil kuadrat	`a : integer` `b : integer`

Bahasa Manusia secara Logika	Bahasa Algoritmik
Meminta masukan pemakai (*user*) untuk kotak yang akan dikuadratkan	`input(a)`
Memasukkan hasil kuadrat ke kotak penyimpanan hasil kuadrat	`b <- a * a`
Menampilkan hasil kuadrat	`output(b)`

Soal Latihan:

1. Buatlah algoritma dan program dengan bahasa Pascal atau C yang menerima tiga buah masukan dan menampilkan hasil pertambahan ketiga bilangan!
2. Buatlah algoritma dan program dengan bahasa Pascal atau C yang menerima tiga buah masukan dan menampilkan hasil perkalian ketiga bilangan!
3. Buat algoritma dan program dengan bahasa Pascal atau C yang menerima sebuah masukan berupa derajat celcius dan menampilkan hasil bilangan masukan itu menjadi derajat fahrenheit! (Rumus Celcius ke Fahrenheit => ((9 * celcius) / 5) + 32)
4. Buat algoritma dan program dengan bahasa Pascal atau C yang menerima sebuah masukan berupa derajat celcius dan menampilkan hasil bilangan masukan itu menjadi derajat Reamur! (Rumus Celcius ke Reamur => (4 * celcius) / 5)
5. Buatlah algoritma dan program dengan bahasa Pascal atau C yang menerima tiga buah masukan berupa sisi balok dan menampilkan hasil volume balok!
6. Buatlah algoritma dan program dengan bahasa Pascal atau C yang menerima dua buah masukan berupa sisi tegak dan sisi alas segitiga dan menampilkan hasil luas segitiga!

5 Operator

Operator adalah simbol atau tanda yang jika diletakkan pada dua buah operan dapat menghasilkan sebuah hasil, contohnya pada matematika dimana tanda tambah ('+') jika diletakkan diantara dua buah angka akan menghasilkan angka lain hasil pertambahan dua angka tersebut. Tanda tambah inilah yang disebut sebagai operator. Operator memiliki beberapa jenis sebagai berikut :

Jenis Operator	Keterangan	Contoh
Unary	operator yang hanya melibatkan satu operan	-9
Binary	operator yang melibatkan dua buah operan	5 + 8
Ternary	operator yang melibatkan tiga buah operan	(a > b) ? a : b

Operator yang dapat digunakan dalam pemrograman antara lain :

Operator	Keterangan	Contoh
Operator Aritmetik	operator yang biasa digunakan dalam matematika misalnya operator penjumlahan (+), operator pembagian (/) dan lain sebagainya	2 + 3
Operator Relasi	operator yang biasa digunakan untuk membandingkan dua buah nilai	a < b
Operator Logika Boolean	operator yang biasa digunakan untuk mengaitkan dua buah ungkapan kondisi menjadi sebuah kondisi	ungkapan_kondisi1 AND ungkapan_kondisi2

5.1 Operator Aritmetik

Berikut adalah beberapa operator aritmetik yang dapat digunakan pada operasi-operasi dalam pemrograman :

Keterangan Operator	Tipe Data Operan	Tipe Data Hasil	Bahasa Algoritmik	Bahasa Pascal	Bahasa C
Operator Perkalian (Operator untuk mengalikan dua buah operan)	integer real	integer real	* contoh : 2 * 3	* contoh : 2 * 3	* contoh : 2 * 3
Operator Pembagian (Operator untuk operasi pembagian dua buah operan, untuk div menghasilkan bilangan bulat hasil tanpa koma, misal 5 div 3 hasilnya 1)	integer real	integer real	/ contoh : 2 / 3	/ contoh : 2 / 3	/ contoh : 2 / 3
	integer	integer	div contoh : 5 div 3	div contoh: 5 div 3	menggunakan rumus matematika, misalnya x div y maka akan menjadi ((x/y) - ((x%y)/y))
Operator Sisa Pembagian (modulo) (Operator untuk dua	integer	integer	mod contoh : 5 mod 3	mod contoh : 5 mod 3	% contoh : 5 % 3

Keterangan Operator	Tipe Data Operan	Tipe Data Hasil	Bahasa Algoritmik	Bahasa Pascal	Bahasa C
buah operan yang menghasilkan sisa pembagian misal 5 mod 3 hasilnya 2)					
Operator Penjumlahan (Operator untuk penjumlahan dua buah operan)	integer real	integer real	+ contoh : 2 + 3	+ contoh : 2 + 3	+ contoh : 2 + 3
Operator Pengurangan (Operator untuk pengurangan dua buah operan)	integer real	integer real	- contoh : 8 - 4	- contoh : 8 - 4	- contoh : 8 - 4
Tanda minus	integer real	integer real	- contoh : -6	- contoh : -6	- contoh : -6

5.2 Operator Relasi

Operator-operator relasi yang dapat digunakan dalam pemrograman adalah sebagai berikut :

Keterangan Operator	Bahasa Algoritmik	Bahasa Pascal	Bahasa C
operator sama dengan (=) (operator yang menyatakan bahwa nilai yang dibandingkan sama)	= contoh : `a = b`	= contoh : `a = b`	== contoh : `a == b`
operator tidak sama dengan (operator yang menyatakan bahwa nilai yang dibandingkan tidak sama)	<> contoh : `a <> b`	<> contoh : `a <> b`	!= contoh : `a != b`
operator lebih dari (>) (operator yang menyatakan bahwa nilai pertama lebih besar dari nilai yang kedua)	> contoh : `a > b`	> contoh : `a > b`	> contoh : `a > b`
operator kurang dari (<) (operator yang menyatakan bahwa nilai pertama lebih kecil dari nilai kedua)	< contoh : `a < b`	< contoh : `a < b`	< contoh : `a < b`
operator lebih dari sama dengan (≥) (operator yang menyatakan bahwa nilai pertama lebih besar atau sama dengan nilai kedua)	≥ contoh : `a ≥ b`	>= contoh : `a >= b`	>= contoh : `a >= b`
operator kurang	≤	<=	<=

Keterangan Operator	Bahasa Algoritmik	Bahasa Pascal	Bahasa C
dari sama dengan (≤) (operator yang menyatakan bahwa nilai pertama kurang dari atau sama dengan nilai kedua)	contoh : `a ≤ b`	contoh : `a <= b`	contoh : `a <= b`

Operator relasi biasanya digunakan untuk menyatakan sebuah kondisi dan menghasilkan nilai benar (1) atau salah (0) misalkan kondisi dalam ungkapan jika seperti berikut :

Bahasa Manusia	Bahasa Algoritmik
Jika a = b maka tulis "a sama dengan b"	`if a = b then` `output("a sama dengan b")`

5.3 Operator Logika Boolean

Operator-operator logika boolean yang biasa digunakan dalam pemrograman adalah sebagai berikut :

Keterangan Operator	Bahasa Algoritmik	Bahasa Pascal	Bahasa C
operator dan (operator logika yang menyatakan logika dan) Keterangan : B : Benar S : Salah	`and` contoh : `(a > b) and (a <> 5)`	`and` contoh : `(a > b) and (a <> 5)`	`&&` contoh : `(a > b) && (a != 5)`

Ungkapan Kondisi 1	Ungkapan Kondisi 2	Hasil Operator Logika dan
B	B	B

<table>
<tr><th>Keterangan Operator</th><th>Bahasa Algoritmik</th><th>Bahasa Pascal</th><th>Bahasa C</th></tr>
<tr><td>
<table>
<tr><td>B</td><td>S</td><td>S</td></tr>
<tr><td>S</td><td>B</td><td>S</td></tr>
<tr><td>S</td><td>S</td><td>S</td></tr>
</table>
</td><td></td><td></td><td></td></tr>
<tr><td>operator atau

(operator logika yang menyatakan logika atau)

Keterangan :
B : Benar
S : Salah

<table>
<tr><th>Ungkapan Kondisi 1</th><th>Ungkapan Kondisi 2</th><th>Hasil Operator Logika atau</th></tr>
<tr><td>B</td><td>B</td><td>B</td></tr>
<tr><td>B</td><td>S</td><td>B</td></tr>
<tr><td>S</td><td>B</td><td>B</td></tr>
<tr><td>S</td><td>S</td><td>S</td></tr>
</table>
</td><td>or

contoh :

<code>(a > b) or
(a <> 5)</code></td><td>or

contoh :

<code>(a > b) or
(a <> 5)</code></td><td>||

contoh :

<code>(a > b)
||
(a != 5)</code></td></tr>
<tr><td>operator bukan (negasi)

(operator logika yang menyatakan logika bukan (negasi))

Keterangan :
B : Benar
S : Salah

<table>
<tr><th>Ungkapan Kondisi 1</th><th>Hasil Operator Logika bukan</th></tr>
<tr><td>B</td><td>S</td></tr>
<tr><td>S</td><td>B</td></tr>
</table>
</td><td>not

contoh :

<code>not(a = b)</code></td><td>not

contoh :

<code>not(a == b)</code></td><td>!

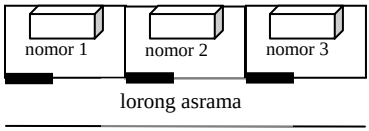
contoh :

<code>!(a == b)</code></td></tr>
</table>

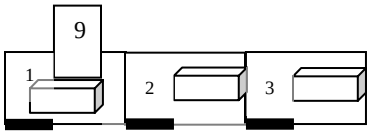
6 Array (Larik)

Array secara gambaran pada dunia nyata hampir sama dengan tabel, dimana tabel adalah sekumpulan elemen yang pada setiap elemennya dapat diakses dengan indeksnya. *Array* biasa digunakan untuk menyimpan banyak data dalam sebuah tabel yang terstruktur. *Array* merupakan bagian penting dalam penyimpanan data pada pemrograman, karena alokasi atau pemesanan tempat dalam sebuah *array* tergantung dari kebutuhan. *Array* sangat penting dalam penyimpanan data karena jika *array* tidak ada bayangkan saja jika dibutuhkan sepuluh tempat untuk menyimpan sepuluh nilai, apakah harus dibuat sepuluh buah variabel, bisa jadi dalam pengaksesannya nanti akan menjadi sangat rumit.

Dalam logika pemikiran manusia, kita dapat menganalogikan *array* sebagai sebuah lorong kamar asrama. Dimana setiap kamar asrama memiliki nomor kamar untuk mengakses kamar yang dituju.

Logika Pemikiran Manusia	Bahasa Algoritmik
Misal di sebuah lorong kamar asrama memiliki 3 kamar seperti gambar berikut:  kita asumsikan setiap kamar asrama memiliki kotak yang hanya dapat diisi oleh bilangan bulat (integer), dan pada awal dialokasikan maka setiap kamar memiliki sebuah kotak kosong. Kotak kosong hanya dapat diisi dengan bilangan bulat	deklarasi sebuah *array* dalam bahasa algoritmik adalah sebagai berikut: `nama_array : array [1…n] of tipe_data` misalnya: `asrama : array [1..3] of integer`

Ketika kita akan mengisi setiap kotak di dalam kamar pada lorong asrama maka kita akan memasukkan sebuah kertas di setiap kotak dalam setiap kamar



lorong asrama

maka kita akan menyebutkan bahwa isi kotak pada kamar nomor 1 adalah angka 9

jika akan mengisi sel kosong pada *array* maka cara mengaksesnya adalah:

```
asrama1 <- 9
```

yang artinya kotak pada kamar nomor 1 dari *array* asrama diisi dengan 9. Nomor kamar sering disebut sebagai nomor **indeks**.

Deklarasi array satu dimensi adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`nama_array : array [1…n] of tipe_data`	`type` `nama_tipe_array = array [1…n] of tipe_data;` `var` `nama_array : nama_tipe_array;`	`tipe_data nama_array[jumlah_indeks];`
`tabInt : array [1…20] of integer`	`type` `tabel = array [1…20] of integer;` `var` `tabInt : tabel;`	`int tabInt[20];`

sedangkan cara mengaksesnya adalah sebagai berikut :

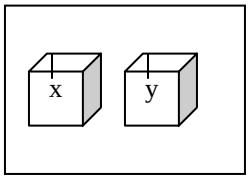
Keterangan	Bahasa Algoritmik	Bahasa Pascal	Bahasa C
mengisi nilai *array* dengan indeks tertentu	`nama_arrayindeks <- nilai`	`nama_array[indeks] := nilai;`	`nama_array[indeks] = nilai;`
	`tabInt1 <- 5`	`tabInt[1] := 5;`	`tabInt[0] = 5;`

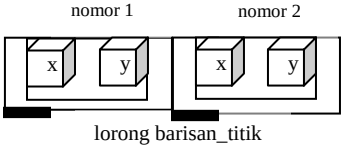
Keterangan	Bahasa Algoritmik	Bahasa Pascal	Bahasa C
mengakses nilai *array* dengan indeks tertentu	`nama_array`$_{indeks}$	`nama_array[indeks];`	`nama_array[indeks];`
	`tabInt`$_{1}$	`tabInt[1];`	`tabInt[0];`

Pada bahasa algoritmik dan Pascal, indeks *array* dimulai dari angka 1 sampai jumlah indeks sedangkan pada bahasa pemrograman C indeks *array* dimulai dari angka 0 sampai jumlah indeks dikurangi 1.

Array hanya bisa diisi sesuai kapasitasnya, jika indeks yang diakses lebih dari yang ada maka akan terjadi kesalahan (*error*) pada program karena mengakses kamar yang belum dialokasikan.

Jika membuat *array* untuk tipe terstruktur maka di dalam logika manusia isi di dalam setiap kamar yang ada di lorong adalah tipe terstruktur itu, misalkan sebagai berikut:

Logika Pemikiran Manusia	Bahasa Algoritmik
Membuat jenis dari suatu benda (dalam logika sebelumnya dianggap sebagai bungkusan), dimana benda itu adalah titik, maka secara logika dapat kita logikakan sebagai berikut:  	`type titik : < x : real, y : real>`
Misal di sebuah lorong kamar asrama memiliki 3 kamar seperti gambar berikut:	deklarasi sebuah *array* titik dalam bahasa algoritmik adalah sebagai berikut:

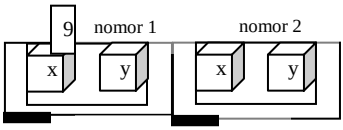


lorong barisan_titik

```
barisan_titik : array [1..2] of titik
```

kita asumsikan setiap kamar berisi titik yang memiliki koordinat x dan y, sehingga kotak x dan y untuk setiap kamar dapat diisi.

Ketika kita akan mengisi setiap titik pada kotak x dan y nya maka dapat kita logikakan sebagai berikut:



lorong barisantitik

```
barisantitik1.X <- 9
```

maka kita akan menyebutkan bahwa isi titik pertama pada kotak x pada kamar nomor 1 adalah angka 9

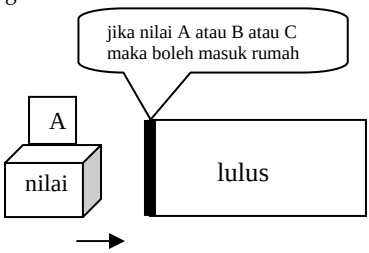
sedangkan cara mengaksesnya adalah sebagai berikut :

Keterangan	Bahasa Algoritmik	Bahasa Pascal	Bahasa C
mengisi kotak x dari titik pertama dengan angka 9	`barisantitik1.X <- 9`	`barisantitik[1].x := 9;`	`barisantitik[0].x = 9;`

7 Percabangan / Pemilihan If

Percabangan `if` merupakan sebuah blok program yang menyatakan bahwa sebuah aksi akan dijalankan jika kondisi percabangan dipenuhi jika tidak dipenuhi maka aksi tidak akan dijalankan. Percabangan `if` biasa digunakan untuk mengerjakan aksi yang memiliki syarat tertentu untuk menjalankannya.

Dalam logika manusia sebuah percabangan/pemilihan `if` dapat dianalogikan sebagai sebuah rumah yang memiliki pintu dimana hanya kondisi-kondisi tertentu yang mengijinkan kotak berisi sesuatu masuk melalui pintu itu, jika kondisi tidak dipenuhi maka kotak tidak dapat masuk melalui pintu itu.

Logika Pemikiran Manusia	Bahasa Algoritmik
Misal ada sebuah rumah yang hanya mau menerima kotak yang berisi nilai A, B, dan C untuk menyimpulkan bahwa nilai itu merupakan nilai lulus, maka kita dapat menganalogikan seperti gambar berikut:  maka jika isi dari kotak nilai tidak memenuhi syarat maka kotak nilai tidak akan diijinkan masuk dan tidak dinyatakan lulus	`nilai : char` `input(nilai)` `if nilai = 'A' or nilai = 'B' or nilai = 'C' then` `output("lulus")` `{end if}`

Percabangan merupakan salah satu inti dari analisis kasus pada pembuatan algoritma, sebuah kasus harus dipikirkan penyelesaiannya dengan pemikiran ada sebuah syarat dan proses atau aksi yang harus dikerjakan jika syarat tidak terpenuhi dan jika syarat terpenuhi, misalkan membuat sebuah penyelesaian kasus menyatakan apakah sebuah bilangan ganjil atau genap, maka dapat dibuat sebuah penyelesaian sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik
jika sebuah bilangan dibagi dengan 2 (dua) masih sisa 1 (satu) maka merupakan bilangan ganjil	`if (bilangan mod 2) = 1 then` `  output("bilangan ganjil")` `{end if}`
jika sebuah bilangan dibagi dengan 2 (dua) sisanya adalah 0 (nol) maka merupakan bilangan genap	`if (bilangan mod 2) = 0 then` `  output("bilangan genap")` `{end if}`
jika sebuah bilangan dibagi dengan 2 (dua) masih sisa 1 (satu) maka merupakan bilangan ganjil, tapi jika syarat tidak dipenuhi maka merupakan bilangan genap	`if (bilangan mod 2) = 1 then` `  output("bilangan ganjil")` `{end if}` `else` `  output("bilangan genap")` `{end else}`

Jika kondisi percabangan lebih dari 1 maka sebaiknya dikelompokkan dalam tanda kurung, misal:

```
if (a = 9) and (b = 9) then
    output('a dan b bernilai 9')
{end if}
```

7.1 Satu Kondisi

Blok program `if` untuk satu kondisi berarti hanya ada sebuah blok aksi yang akan dikerjakan jika syarat kondisi terpenuhi. Deklarasi percabangan `if` satu kondisi adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`if kondisi_percabangan then` `  {proses}` `  .....` `{end if}`	`if kondisi_percabangan then` `  begin` `  { proses }` `  .....` `end;`	`if(kondisi_percabangan){` `  // proses` `  .....` `}`
`ketemu : boolean` `ketemu <- true` `if  ketemu = true then` `  {proses}` `  output("nilai variabel ketemu : ", ketemu)` `{end if}`	`var` `ketemu : boolean;` `begin` `ketemu := true;` `if ketemu = true then` `begin` `  {proses}` `  write('nilai variabel ketemu : true\n');` `end;` `end.`	`int ketemu = 1;` `if(ketemu == 1){` `  // proses` `  printf("nilai variabel ketemu : true\n");` `}`

Deklarasi percabangan berikut :

```
if  ketemu = true then

  {proses}
   output("nilai variabel ketemu : ", ketemu)

{end if}
```

berarti ketika eksekusi program sampai pada blok percabangan akan dilakukan pengecekan nilai variabel `ketemu`, jika variabel ketemu bernilai true maka aksi menuliskan nilai variabel `ketemu` ke layar akan dikerjakan, tapi jika nilai `ketemu` adalah false maka aksi menuliskan nilai variabel `ketemu` ke layar tidak dikerjakan.

7.2 If-else (dua kondisi)

Blok program `if-else` dipergunakan untuk menyatakan percabangan dua kondisi dimana ada dua blok aksi yang dipilih untuk dikerjakan jika syarat kondisi aksi terpenuhi. Saat pembacaan program sampai pada blok `if-else` maka akan dilakukan pengecekan terhadap syarat kondisi percabangan yang ada pada deklarasi `if`, jika syarat dipenuhi maka yang akan dijalankan adalah aksi yang ada di dalam blok `if`, tapi jika syarat tidak dipenuhi maka aksi yang dikerjakan adalah yang ada di dalam blok `else`.

Deklarasi percabangan `if-else` adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
if kondisi_percabangan then {*proses*} ..... {*end if*} else {*proses*} ..... {*end else*}	if kondisi_percabangan then begin { proses } ..... end else{ begin { proses } ..... end;	if(kondisi_percabangan){ // proses ..... } else{ // proses ..... }
ketemu : boolean ketemu <- false if ketemu = true then {*proses*} output("(if) nilai variabel ketemu : ", ketemu) {*end if*} else {*proses*} output("(else) nilai variabel ketemu : ", ketemu) {*end else*}	var ketemu : boolean; begin ketemu = false; if ketemu = true then begin { proses } write('(if) nilai variabel ketemu : true'); end else begin { proses } write('(else) nilai variabel ketemu : false'); end; end.	int ketemu; ketemu = 0; if(ketemu == 1){ // proses printf("(if) nilai variabel ketemu : true\n"); } else{ // proses printf("(else) nilai variabel ketemu : false\n"); }

Deklarasi percabangan berikut :

```
if  ketemu = true then
```

```
    {proses}
     output("(if) nilai variabel ketemu : ", ketemu)

  {end if}
  else

    {proses}
     output("(else) nilai variabel ketemu : ", ketemu)

  {end else}
```

berarti saat eksekusi program sampai pada blok percabangan akan dilakukan pengecekan pada variabel `ketemu`, jika variabel `ketemu` bernilai true maka proses di dalam blok `if` akan dikerjakan, tapi jika variabel `ketemu` bernilai false maka yang dikerjakan adalah proses yang ada di dalam blok `else`.

Contoh Kasus:

- Buat algoritma yang menerima masukan tiga buah sisi, dimana jika semua sisi sama maka tampilkan "Termasuk Kubus", jika ada sisi yang tidak sama maka tampilkan "Bukan Kubus"!

Penyelesaian:

Bahasa Manusia secara Logika	Bahasa Algoritmik
Buat kotak untuk memasukkan tiga buah masukan	`s1 : integer` `s2 : integer` `s3 : integer`
Meminta masukan pemakai (*user*) untuk mengisi ketiga buah kotak	`input(s1)` `input(s2)` `input(s3)`
Memeriksa isi kotak apakah isi ketiga kotak sama atau tidak, secara logika adalah sebagai berikut: jika s1=s2 dan s2=s3 maka termasuk kubus jika tidak memenuhi syarat di atas maka bukan kubus	`if s1=s2 and s2=s3 then` `output("Termasuk Kubus")` `{end if}` `else` `output("Bukan Kubus")` `{end else}`

- Buat algoritma yang menerima tiga buah angka masukan dan menampilkan nilai yang paling besar dari ketiga masukan!

Penyelesaian:

Bahasa Manusia secara Logika	Bahasa Algoritmik
Buat kotak untuk memasukkan tiga buah masukan	`a : integer` `b : integer` `c : integer`
Meminta masukan pemakai (*user*) untuk mengisi ketiga buah kotak	`input(a)` `input(b)` `input(c)`
Memeriksa isi kotak mencari yang paling besar, secara logika adalah sebagai berikut: jika a > b dan a > c maka a paling besar jika b > a dan b > c maka b paling besar jika c > a dan c > b maka c paling besar lainnya ada dua atau tiga masukan memiliki nilai sama	`if a > b and a > c then` `output("a paling besar")` `{end if}` `else if b > a and b > c then` `output("b paling besar")` `{end if}` `else if c > a and c > b then` `output("c paling besar")` `{end if}` `else` `output("ada dua atau tiga masukan memiliki nilai sama")` `{end else}`

7.3 Banyak if Digunakan untuk Memilih Salah Satu Blok atau Banyak Blok

Misalkan dibutuhkan pemilihan kondisi percabangan untuk banyak syarat, dan hanya ingin agar program hanya memilih salah satu maka dapat ditulis seperti contoh berikut (dalam bahasa algoritmik):

```
if  kondisi_percabangan_1 then

     { proses jika kondisi percabangan 1 terpenuhi }

{end if}
else if kondisi_percabangan_2 then

     { proses jika kondisi percabangan 1 tidak terpenuhi dan kondisi
     percabangan 2 terpenuhi }

{end if}
else if kondisi_percabangan_3 then

     { proses jika kondisi percabangan 1 dan 2 tidak terpenuhi dan
     kondisi percabangan 3 terpenuhi }

{end if}
else

     { proses jika kondisi percabangan 1, 2, 3  tidak terpenuhi }

{end else}
```

Jika kita ingin melakukan pemeriksaan secara terurut untuk banyak syarat dimana semua syarat harus dilalui oleh semua yang memenuhi syarat atau tidak maka kita dapat membuat percabangan sebagai berikut:

```
if  kondisi_percabangan_1 then

     { proses jika kondisi percabangan 1 terpenuhi }

{end if}

if kondisi_percabangan_2 then

     { proses jika kondisi percabangan 1 terpenuhi atau tidak terpenuhi
     dan kondisi percabangan 2 terpenuhi }

{end if}

if kondisi_percabangan_3 then

     { proses jika kondisi percabangan 1 dan 2 terpenuhi atau tidak
     terpenuhi dan kondisi percabangan 3 terpenuhi }

{end if}
```

7.4 If di dalam if

Sebuah program mengijinkan blok percabangan `if` di dalam blok percabangan lainnya, dan tidak membatasi jenis percabangan apa yang boleh berada di dalam percabangan lainnya misalnya dalam bahasa algoritmik berikut:

```
if  kondisi_percabangan_1 then

     { proses jika kondisi percabangan 1 terpenuhi }

     if  kondisi_percabangan_1_1 then

               { proses jika kondisi percabangan 1_1 terpenuhi }

     {end if}

     if kondisi_percabangan_1_2 then

               { proses jika kondisi percabangan 1_1 terpenuhi atau
               tidak terpenuhi dan kondisi percabangan 1_2 terpenuhi }

     {end if}

{end if}
else

     { proses jika kondisi percabangan 1 tidak terpenuhi }

{end else}
```

7.5 Break

`Break` biasanya digunakan untuk keluar dari sebuah blok program tanpa mengerjakan semua aksi yang ada setelah `break`. Deklarasi `break` adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
break	break;	break;

7.6 Depend On (dua kondisi atau lebih)

Percabangan `depend on` biasa digunakan untuk dua kondisi atau lebih bergantung pada nilai sebuah variabel, syarat kondisi pada percabangan `depend on` biasanya hanya sebuah nilai. Deklarasi percabangan `depend on` adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
depend on (nama_variabel) nilai_variabel_1 : aksi_1 break nilai_variabel_2 : aksi_2 break ..... nilai_variabel_n : aksi_n break default : aksi_standar break {*end depend on*}	case nama_variabel of nilai_variabel_1 : begin aksi_1 end; nilai_variabel_2 : begin aksi_2 end; ..... nilai_variabel_n : begin aksi_n end; else begin aksi_standar end; end;	switch(nama_variabel){ case nilai_variabel_1 : aksi_1 break; case nilai_variabel_2 : aksi_2 break; ..... case nilai_variabel_n : aksi_n break; default : aksi_default break; }
depend on (hari) 1 : output("hari senin") break 2 : output("hari selasa") break 3 : output("hari rabu") break ..... 7 : output("hari minggu") break default : output("tidak ada hari ke : ", hari) break	case hari of 1 : begin write('hari senin'); end; 2 : begin write('hari selasa'); end; 3 : begin write('hari rabu'); end; ..... 7 : begin write('hari minggu'); end; else begin write('tidak ada hari ke : ', hari); end;	switch(hari){ case 1 : printf("hari senin"); break; case 2 : printf("hari selasa"); break; case 3 : printf("hari rabu"); break; ..... case 7 : printf("hari minggu"); break; default : printf("tidak ada hari ke : %d\n", hari); break;

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
{*end depend on*}	end;	}

Deklarasi depend on berikut :

```
depend on (hari)
  1 : output("hari senin")
      break

  2 : output("hari selasa")
      break

  3 : output("hari rabu")
      break
  ...................

  7 : output("hari minggu")
      break

  default : output("tidak ada hari ke : ", hari)
            break

{end depend on}
```

berarti saat eksekusi program sampai pada blok `depend on`, nilai variabel hari akan dicek, misalkan jika nilai variabel `hari` adalah 3 (tiga), maka eksekusi akan langsung menuju ke blok aksi untuk nilai 3 (tiga) dan mengerjakan aksi menuliskan "hari rabu" ke layar. Jika kode `break` tidak dituliskan maka eksekusi akan terus membaca ke bawah sehingga bisa banyak aksi akan dikerjakan. Jika semua nilai tidak ada yang dipenuhi, maka eksekusi akan langsung menuju ke aksi `default` dan mengerjakan aksi `default`.

Soal Latihan:

1. Buatlah algoritma dan program dengan bahasa Pascal atau C yang menerima sebuah masukan berupa jam lembur, jika jam lembur kurang dari 12 jam maka akan menampilkan gaji lebur sebesar Rp. 100.000, jika jam lembur sama dengan 12 jam maka akan menampilkan gaji lebur sebesar Rp. 200.000, jika jam lembur lebih dari 12 jam maka akan menampilkan gaji lebur sebesar Rp. 300.000!
2. Buatlah algoritma dan program dengan bahasa Pascal atau C yang menerima menu yang dapat berisi 1, 2, dan 3 dan dua buah masukan bilangan, jika menu sama dengan 1 maka akan menampilkan hasil dari bilangan 1 ditambah dengan bilangan 2, jika menu sama dengan 2 maka akan menampilkan hasil dari bilangan 1 dikali dengan bilangan 2, jika menu sama dengan 3 maka akan menampilkan hasil dari bilangan 1 dikurangi dengan bilangan 2!

3. Buatlah algoritma dan program dengan bahasa Pascal atau C yang menerima masukan berupa tiga buah sisi segitiga, periksa apakah segitiga masukan merupakan segitiga siku-siku atau bukan!

8 Pengulangan

Pengulangan (*looping*) adalah suatu bagian yang bertugas melakukan kegiatan mengulang suatu proses sesuai dengan yang diinginkan. Banyak dari aplikasi perangkat lunak yang melakukan pekerjaan berulang-ulang sampai sebuah kondisi yang diinginkan, oleh karena itu pengulangan merupakan bagian yang penting dalam pemrograman karena dengan adanya pengulangan pembuat program tidak perlu menulis kode program sebanyak pengulangan yang diinginkan.

Pengulangan mempunyai beberapa bagian yang harus dipenuhi antara lain:

- **Inisialisasi**

  Inisialisasi adalah tahap persiapan membuat kondisi awal sebelum melakukan pengulangan, misalnya mengisi variabel dengan nilai awal. Tahap ini dilakukan sebelum memasuki bagian pengulangan.
- **Proses**

  Tahap proses terjadi di dalam bagian pengulangan dimana berisi semua proses yang perlu dilakukan secara berulang-ulang.
- **Iterasi**

  Iterasi terjadi di dalam pengulangan dimana merupakan kondisi pertambahan agar pengulangan dapat terus berjalan
- **Terminasi/Kondisi Pengulangan**

  Terminasi adalah kondisi berhenti dari pengulangan, kondisi berhenti sangat penting dalam pengulangan agar pengulangan dapat berhenti, tidak menjadi pengulangan yang tanpa henti. Kondisi pengulangan adalah kondisi yang dipenuhi oleh kondisi jalannya algoritma untuk masuk ke dalam blok pengulangan.

Pengulangan merupakan salah satu inti dari analisis kasus pada pembuatan algoritma, sebuah kasus harus dipikirkan penyelesaiannya dengan pemikiran ada proses atau aksi yang harus dikerjakan secara berulang-ulang agar sebuah kasus terselesaikan, misalkan membuat sebuah penyelesaian kasus-kasus sebagai berikut:

Bahasa Manusia	Bahasa Algoritmik
kasus menuliskan sesuatu ke layar sebanyak 10 kali, maka harus diselesaikan dengan membuat kode pengulangan yang akan dijalankan oleh *compiler* atau interpreter sebanyak 10 kali dengan aksi yang harus dijalankan adalah menuliskan sesuatu ke layar	`i : integer` `for i=0 to 10 do` `  output("sesuatu")` `{end for}`
kasus mencari rata-rata dari sepuluh bilangan positif pertama, maka harus diselesaikan dengan membuat kode pengulangan yang menambahkan bilangan 1 sampai sepuluh dengan menggunakan pengulangan yang dijalankan 10 kali, baru kemudian hasilnya dibagi dengan banyaknya bilangan, misalkan membuat sebuah variabel yang menyimpan nilai awal 0 kemudian setiap dijalankan blok pengulangan dirinya ditambahkan dengan nilai berapa kali pengulangan telah dijalankan, secara logika pada akhir pengulangan variabel ini akan bernilai 1 + 2 + 3 + ... + 10 baru kemudian dicari rata-ratanya	`i : integer` `hasil : integer` `hasil <- 0` `for i=1 to 10 do` `  hasil <- hasil + i` `{end for}` `hasil <- hasil / 10`

8.1 For

Pengulangan menggunakan `for` biasanya digunakan untuk pengulangan yang **sudah jelas perlu dilakukan berapa kali**, dengan kata lain jumlah pengulangan yang dibutuhkan sudah diketahui oleh pembuat program.

Deklarasi penggunaan pengulangan `for` adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`{pengulangan for positif}`	`/* pengulangan for positif */`	`/* pengulangan for positif */`

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
for nama_variabel <- nilai_awal to nilai_berhenti do {proses} ..... {end for}	for nama_variabel:=nilai_awal to nilai_berhenti do begin // proses ..... end;	for(nama_variabel=nilai_awal;nama_variabel operator_relasi;nama_variabel++){ // proses ..... }
{pengulangan for positif} for i <- 1 to 10 do {proses} ..... {end for}	/* pengulangan for positif */ for i:=1 to 10 do begin // proses ..... end;	/* pengulangan for positif */ for(i=1; i <= 10; i++){ // proses ..... }
{pengulangan for negatif} for nama_variabel <- nilai_awal downto nilai_berhenti do {proses} ..... {end for}	/* pengulangan for negatif */ for nama_variabel:=nilai_awal downto nilai_berhenti do begin // proses ..... end;	/* pengulangan for negatif */ for(nama_variabel=nilai_awal; nama_variabel operator relasi nilai_berhenti; nama_variabel --){ // proses ..... }
{pengulangan for negatif} for i <- 10 downto 1 do {proses} ..... {end for}	/* pengulangan for negatif */ for i:=10 downto 1 do begin // proses ..... end;	/* pengulangan for negatif */ for(i=10; i >= 1; i--){ // proses ..... }

Deklarasi pengulangan `for` berikut :

```
for i <- 1 to 10 do
  {proses}
  .....
{end for}
```

berarti proses yang ada di dalam pengulangan `for` dilakukan mulai dari 1 sampai 10 (pengulangan dilakukan 10 kali).

Tahapan pengulangan `for` adalah sebagai berikut :

Tahapan	Deklarasi dalam Pengulangan For	Keterangan
Inisialisasi	`i <- 1`	tahapan mengeset nilai awal dari pengulangan dimana pengulangan

Tahapan	Deklarasi dalam Pengulangan For	Keterangan
		dilakukan mulai dari angka 1
Iterasi	`to`	tahapan pertambahan agar pengulangan dapat berjalan mulai dari nilai awal sampai nilai berhenti
Terminasi	`nilai_berhenti`	merupakan nilai berhenti dari pengulangan sehingga ada saat dimana pengulangan harus berhenti yaitu ketika kondisi berhenti telah dipenuhi yang dalam hal ini berarti pengulangan terlah dilakukan 10 kali dan pengulangan dilakukan selama nilai variabel i lebih kecil atau sama dengan 10
Proses	proses dalam pengulangan `for`	proses yang perlu dilakukan dalam pengulangan

Untuk mempelajari pengulangan `for`, dapat dibuat algoritma untuk menjumlahkan bilangan dari angka 1 sampai bilangan masukan yang dikehendaki sebagai berikut:

Logika Pemikiran Manusia	Bahasa Algoritmik
Dibutuhkan sebuah kotak untuk menyimpan masukan *user* (pemakai program) sampai bilangan ke berapa akan dijumlahkan, sebuah kotak untuk menyimpan hasil penjumlahan, dan sebuah kotak untuk menjadi *counter* /penghitung sedang	`berapa_kali : integer` `hasil_jumlah : integer` `penghitung : integer`

Logika Pemikiran Manusia	Bahasa Algoritmik
melakukan pengulangan keberapa untuk menjumlahkan bilangan	
kotak penyimpan penjumlahan diberi nilai awal 0 karena dianggap belum melakukan penjumlahan (jika tidak diberi nilai awal, maka saat penjumlahan pertama kali jumlah sudah dianggap tidak sama dengan 0)	`hasil_jumlah <- 0`
Menerima masukan dari *user* / pemakai program untuk dimasukkan ke dalam kotak yang menyimpan nilai sampai berapa bilangan akan dihitung	`input(berapa_kali)`
Melakukan pengulangan untuk menambahkan bilangan satu per satu dalam setiap pengulangan, secara logika adalah sebagai berikut: melakukan pengulangan sebanyak nilai yang dimasukkan oleh *user* / pemakai program untuk setiap pengulangan maka tambahkan bilangan penghitung dengan jumlah sebelumnya	`for penghitung <- 1 to berapa_kali do` `hasil_jumlah <- hasil_jumlah + penghitung` `{end for}`
tampilkan hasil penjumlahan	`output(hasil_jumlah)`

Berikut adalah algoritma yang menuliskan kalimat ke layar hasil perhitungan nilai faktorial sebanyak nilai berhenti masukan dari *user* (pemakai program) menggunakan pengulangan `for`:

Logika Pemikiran Manusia	Bahasa Algoritmik
Membuat sebuah kotak untuk menyimpan berapa faktorial yang akan dihitung (masukan dari *user* / pemakai program), sebuah kotak sebagai penghitung (*counter*) pengulangan, sebuah kotak untuk menyimpan hasil faktorial yang dihitung.	`fak_berapa : integer` `penghitung : integer` `hasil_faktorial : integer`
Menerima masukan dari *user* / pemakai program untuk nilai faktorial yang akan dihitung dan dimasukkan ke	`input(fak_berapa)`

Logika Pemikiran Manusia	Bahasa Algoritmik
kotak `faktorial_berapa`	
Isi kotak `hasil_faktorial` dengan 1 (sebagai nilai awal pengali faktorial)	`hasil_faktorial <- 1`
Melakukan pengulangan untuk menghitung faktorial dengan logika sebagai berikut: melakukan pengulangan sebanyak `fak_berapa` kali sesuai dengan masukan dari *user* / pemakai program dikurangi 1 karena dimulai dari 2 menghitung faktorial dengan mengalikan hasil pengalian sebelumnya dengan pehitung pada saat pengulangan	`for penghitung <- 2 to fak_berapa do` `hasil_faktorial <- hasil_faktorial * penghitung` `{end for}`
Menampilkan hasil faktorial	`output("hasil faktorial : ", hasil_faktorial)`

Pengulangan `for` dipilih jika jumlah pengulangan diketahui dengan pasti, tipe pengulangan yang telah diketahui jumlah pengulangan yang harus dilakukan biasanya disebut dengan traversal.

Contoh berikut adalah algoritma untuk mengisi *array* `integer` dengan bilangan berurut dari 1 sampai panjang *array*:

Logika Pemikiran Manusia	Bahasa Algoritmik
Membuat *array* dengan tipe `integer` sebanyak 5 kamar dengan nama `tabInt`	`tabInt : array [1..5] of integer`
Dibutuhkan sebuah kotak untuk menjadi *counter* /penghitung sedang melakukan pengulangan keberapa untuk mengisi *array*	`penghitung : integer`
Melakukan pengulangan untuk mengisi *array* dengan bilangan penghitung sebagai berikut: melakukan pengulangan sebanyak	

Logika Pemikiran Manusia	Bahasa Algoritmik
jumlah kamar *array* untuk setiap pengulangan maka isikan penghitung ke dalam *array* dengan nomor indeks/nomor kamar penghitung	for penghitung <- 1 to 5 do tabInt$_{penghitung}$ <- penghitung {*end for*}

8.2 While

Pengulangan while biasa digunakan jika jumlah pengulangan tidak diketahui. Pengulangan while akan melakukan pengulangan selama kondisi pengulangan terpenuhi. Deklarasi pengulangan while adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
{inisialisasi} ..... while kondisi_pengulangan do {*proses*} ..... {*iterasi*} {*end while*}	// inisialisasi ..... while kondisi_pengulangan do begin // proses ..... // iterasi end;	// inisialisasi ..... while(kondisi_pengulanga n){ // proses ..... // iterasi }
i : integer {*inisialisasi*} i <- 1 while i ≤ 9 do {*proses*} ..... {*iterasi*} i <- i + 1 {*end while*}	var i : integer; begin // inisialisasi i := 1; while i <= 9 do begin // proses ..... // iterasi i := i + 1; end; end.	int i; // inisialisasi i = 1; while(i <= 9){ // proses ..... // iterasi i = i + 1; }

Deklarasi pengulangan while berikut :

```
i : integer

{inisialisasi}
i <- 1
```

```
while i ≤ 9 do

   {proses}
   ..............
   {iterasi}
   i <- i + 1

{end while}
```

berarti pengulangan akan terus dilakukan selama nilai variabel `i` lebih kecil atau sama dengan 9, dimana di dalam pengulangan nilai `i` terus ditambah dengan 1. Jika kondisi berhenti pengulangan tidak pernah tercapai, maka pengulangan akan dilakukan tanpa henti. Agar lebih jelas dapat ditelusuri pengulangan `while` di atas sebagai berikut

- **pengulangan ke-1**
  nilai variabel `i` adalah 1, ketika masuk pengulangan, nilai `i` memenuhi kondisi pengulangan sehingga jalannya program masuk ke blok pengulangan, proses dijalankan. Saat memasuki iterasi, nilai `i` berubah menjadi 2, dan masuk ke pengulangan ke-2.
- **pengulangan ke-2**
  nilai variabel `i` adalah 2, ketika masuk pengulangan, nilai `i` memenuhi kondisi pengulangan sehingga jalannya program masuk ke blok pengulangan, proses dijalankan. Saat memasuki iterasi, nilai `i` berubah menjadi 3, dan masuk ke pengulangan ke-3

- **pengulangan ke-8**
  nilai variabel `i` adalah 8, ketika masuk pengulangan, nilai `i` memenuhi kondisi pengulangan sehingga jalannya program masuk ke blok pengulangan, proses dijalankan. Saat memasuki iterasi, nilai `i` berubah menjadi 9, dan masuk ke pengulangan ke-9
- **pengulangan ke-9**
  nilai variabel `i` adalah 9, ketika masuk pengulangan, nilai `i` memenuhi kondisi pengulangan sehingga jalannya program masuk ke blok pengulangan, proses dijalankan. Saat memasuki iterasi, nilai `i` berubah menjadi 10, nilai `i` sudah tidak memenuhi kondisi pengulangan, maka blok pengulangan tidak dijalankan lagi (berhenti).

Jika pengulangan berjalan terus tanpa henti karena kondisi berhenti tidak pernah tercapai, dapat ditekan tombol ctrl + C untuk menghentikan jalannya program (jika memakai *compiler* dengan mode *console* / terminal / *command prompt* (cmd)).

Tahapan pengulangan `while` adalah sebagai berikut :

Tahapan	Deklarasi dalam Pengulangan While	Keterangan
Inisialisasi	`integer i <- 1`	tahapan mempersiapkan nilai awal dari variabel yang berpengaruh pada kondisi pengulangan dan kondisi terminasi
Kondisi pengulangan	`while i ≤ 9 do`	kondisi persyaratan apakah blok pengulangan dieksekusi atau tidak
Proses	proses dalam pengulangan `while`	proses yang perlu dilakukan dalam pengulangan
Iterasi	`i <- i + 1`	tahapan pertambahan agar pengulangan dapat berjalan disertai dengan pertambahan nilai variabel yang berpengaruh pada kondisi pengulangan dan terminasi

Berikut adalah algoritma yang menuliskan kalimat ke layar dengan dasar rangka program seperti contoh pengulangan `while` di atas sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik
Mendeklarasikan proses inisialisasi	{*inisialisasi*} `i <- 1`
Mendeklarasikan pengulangan `while`	`while i ≤ 9 do`

Bahasa Manusia	Bahasa Algoritmik
Menuliskan isi variabel i dan ketemu sebagai blok proses di dalam pengulangan	{*proses*} output("nilai variabel i : ", i)
Mendeklarasikan tahap iterasi	{*iterasi*} i <- i + 1 {*end while*}

Pengulangan `while` memiliki pemeriksaan kondisi pengulangan di bagian awal blok pengulangan, hal ini dapat dianalogikan bahwa blok pengulangan `while` memiliki sebuah "pos satpam" di awal blok (pada pintu masuk) yang bertugas memeriksa apakah kondisi yang dicapai program sudah sesuai dengan dengan kondisi yang disyaratkan di kondisi pengulangan agar program dapat masuk ke blok pengulangan atau tidak.

Pengulangan `while` dipilih jika jumlah pengulangan tidak diketahui dengan pasti dan diperlukan pengecekan kondisi pengulangan di awal blok, sehingga jika tidak memenuhi kondisi pengulangan ada kemungkinan pengulangan `while` tidak dikerjakan sama sekali.

8.3 Repeat

Pengulangan `repeat` biasa digunakan jika jumlah pengulangan tidak diketahui, namun berbeda dengan `while` karena kondisi pengulangan ada di bagian bawah blok pengulangan. Pengulangan `repeat` minimal selalu dilakukan sekali karena kondisi pengulangan ada di bagian bawah, berbeda dengan pengulangan `while` yang saat pertama kali masuk blok pengulangan dilakukan pengecekan kondisi pengulangan. Hal tersebut dapat dianalogikan dengan pada `repeat` letak dari "pos satpam" yang bertugas memeriksa apakah kondisi program memenuhi syarat pengulangan atau tidak berada di akhir blok pengulangan (dapat dianalogikan berada pada pintu keluar blok).

Deklarasi pengulangan `repeat` adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
{inisialisasi} ..... repeat {*proses*} ..... {*iterasi*} until kondisi_terminasi	{ inisialisasi } ..... repeat begin { proses } ..... { iterasi } end; until(kondisi_pengulangan);	// inisialisasi ..... do{ // proses ..... // iterasi } while(kondisi_pengulangan);
i : integer {inisialisasi} i <- 1 repeat {*proses*} ..... {*iterasi*} i <- i + 1 until (i = 2)	var i : integer; begin { inisialisasi } i := 1; repeat begin { proses } ..... { iterasi } i := i + 1; end; until(i = 2); end.	int i; // inisialisasi i = 1; do{ // proses ..... // iterasi i = i + 1; }while(i < 2);

Pada deklarasi pengulangan `repeat` terdapat perbedaan antara bahasa algoritmik dan bahasa pemrograman C yaitu pada kondisi pengulangan ("pos satpam"). Pada bahasa algoritmik deklarasinya adalah `repeat-until` yang berarti lakukan pengulangan sampai pada kondisi setelah deklarasi `until`, sedangkan pada bahasa pemrograman C menggunakan kata kunci `do-while` yang berarti lakukan pengulangan selama kondisi setelah deklarasi `while`, sehingga pada bahasa algoritmik menggunakan kondisi berhenti dan pada bahasa pemrograman C menggunakan kondisi diperbolehkan pengulangan dijalankan, tapi pada dasarnya keduanya bertujuan sama yaitu menentukan kondisi berhenti pengulangan. Agar lebih jelas dapat ditelusuri pengulangan `repeat` di atas sebagai berikut :

- **pengulangan ke-1**

  nilai variabel `i` adalah 1, ketika masuk pengulangan, tidak ada pengecekan sehingga pada kali pertama kondisi apapun blok `repeat` akan dijalankan,. Saat memasuki iterasi, nilai `i` berubah menjadi 2, nilai ini tidak memenuhi kondisi berhenti maka pengulangan berhenti dan tidak dijalankan lagi.

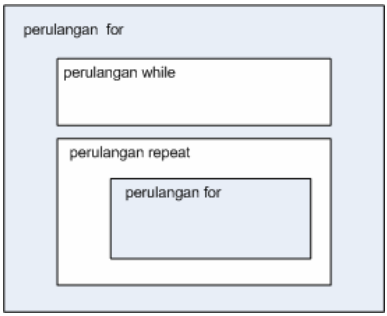
Tahapan pengulangan `repeat` adalah sebagai berikut :

Tahapan	Deklarasi dalam Pengulangan Repeat	Keterangan
Inisialisasi	{*inisialisasi*} i <- 1	tahapan mempersiapkan nilai awal dari variabel yang berpengaruh pada kondisi pengulangan atau kondisi terminasi pengulangan
Proses	proses dalam pengulangan `repeat`	proses yang perlu dilakukan dalam pengulangan
Iterasi	i <- i + 1	tahapan pertambahan agar pengulangan dapat berjalan disertai dengan pertambahan nilai variabel yang berpengaruh pada kondisi pengulangan atau terminasi
Terminasi	(i = 2)	merupakan tahapan dimana merupakan pernyataan kapan pengulangan harus tetap berjalan dan berhenti

Pengulangan `repeat` dipilih jika jumlah pengulangan tidak diketahui dengan pasti dan tidak diperlukan pengecekan kondisi pengulangan di awal blok, sehingga minimal blok pengulangan dikerjakan sekali.

8.4 Pengulangan di dalam Pengulangan

Sebuah program mengijinkan blok pengulangan di dalam blok pengulangan lainnya, dan tidak membatasi jenis pengulangan apa yang boleh berada di dalam pengulangan lainnya, misalnya di dalam blok pengulangan `for` terdapat pengulangan `while`, atau di dalam pengulangan `while` terdapat pengulangan `repeat` dan lain sebagainya, jenis pengulangan tergantung dengan kebutuhan misalnya seperti pada gambar 4.



Gambar 4 Ilustrasi Pengulangan di dalam Pengulangan

Contoh Kasus:

- Buat algoritma yang menerima masukan sebuah angka masukan yang akan menampilkan tampilan sebagai berikut;

  misalnya masukan adalah 5, maka hasil tampilannya adalah:

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

Penyelesaian:

Bahasa Manusia secara Logika	Bahasa Algoritmik
Membuat sebuah kotak untuk menyimpan masukan dari *user*/pemakai program akan digambar pada bilangan keberapa pola gambar pada kasus di atas, sebuah kotak untuk menyimpan angka penghitung pengulangan arah	`jumlah_bintang : integer` `penghitung_vertikal : integer` `penghitung_samping : integer`

vertikal, dan sebuah kotak penghitung arah samping	
Meminta masukan pemakai (*user*) untuk mengisi kotak `jumlah_bintang`	`input(jumlah_bintang)`
Pada pola gambar, gambar dapat dipisahkan sebagai dua karakter pada setiap barisnya yaitu spasi (' ') dan bintang (`'*'`) maka secara logika pola gambar pada kasus di atas dapat diselesaikan dengan cara berikut: melakukan pengulangan sebanyak isi dari `jumlah_bintang` kali (untuk memproses pola ke arah bawah) melakukan pengulangan untuk menggambarkan spasi ke arah samping untuk setiap baris dengan kondisi berhenti tertentu sesuai dengan pola, jumlah spasi adalah deret menaik sebagai berikut: baris 1 – jumlah spasi 0 baris 2 – jumlah spasi 1 baris 3 – jumlah spasi 2 ..... maka dapat disimpulkan rumus dari jumlah spasi setiap baris adalah sama dengan `penghitung_vertikal` dikurangi dengan 1 (karena `penghitung_vertikal` berjalan mulai dari 1 sampai `jumlah_bintang`) pada pengulangan tampilkan spasi ke layar melakukan pengulangan untuk menggambarkan bintang ke arah samping	`for penghitung_vertikal <- 1 to jumlah_bintang do` `for penghitung_samping <- 1 to (penghitung_vertikal - 1) do` `output(' ')` `{end for}` `for penghitung_samping <- 1 to jumlah_bintang do` `output('*')`

sebanyak `jumlah_bintang` kali tampilkan kode ganti baris	{*end for*} output("\n") {*end for*}

- Buat algoritma yang menerima masukan sebuah angka masukan yang akan menampilkan tampilan sebagai berikut;

  misalnya masukan adalah 5, maka hasil tampilannya adalah:

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

  misalnya masukan adalah 4, maka hasil tampilannya adalah:

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

Penyelesaian:

Bahasa Manusia secara Logika	Bahasa Algoritmik
Membuat sebuah kotak untuk menyimpan masukan dari *user*/pemakai program akan digambar pada bilangan keberapa pola gambar pada kasus di atas, sebuah kotak untuk menyimpan angka penghitung pengulangan arah vertikal, dan sebuah kotak penghitung arah samping	`jumlah_bintang : integer` `penghitung_vertikal : integer` `penghitung_samping : integer`
Meminta masukan pemakai (*user*) untuk mengisi kotak `jumlah_bintang`	`input(jumlah_bintang)`
Pada pola gambar, gambar dapat dipisahkan sebagai dua karakter pada setiap barisnya yaitu spasi (' ') dan bintang ('*')	

Bahasa Manusia secara Logika	Bahasa Algoritmik
maka secara logika pola gambar pada kasus di atas dapat diselesaikan dengan cara berikut:	
melakukan pengulangan sebanyak isi dari `jumlah_bintang div 2` kali (untuk memproses pola ke arah bawah)	`for penghitung_vertikal <- 1 to (jumlah_bintang div 2) do`
melakukan pengulangan untuk menggambarkan spasi ke arah samping untuk setiap baris dengan kondisi berhenti tertentu sesuai dengan pola, jumlah spasi adalah deret menaik sebagai berikut: baris 1 – jumlah spasi 0 baris 2 – jumlah spasi 1 baris 3 – jumlah spasi 2 ..... maka dapat disimpulkan rumus dari jumlah spasi setiap baris adalah sama dengan `penghitung_vertikal` dikurangi dengan 1 (karena `penghitung_vertikal` berjalan mulai dari 1 sampai `jumlah_bintang`) pada pengulangan tampilkan spasi ke layar	`for penghitung_samping <- 1 to (penghitung_vertikal – 1) do` `output(' ')` `{end for}`
melakukan pengulangan untuk menggambarkan bintang ke arah samping sebanyak `jumlah_bintang` kali	`for penghitung_samping <- 1 to jumlah_bintang do` `output('*')` `{end for}`
tampilkan kode ganti baris	`output("\n")` `{end for}`
menggambar pola ditengah jika isi `jumlah_bintang` ganjil maka dilakukan pemeriksaan sebagai berikut:	`if (jumlah_bintang div 2) = 1 then`

Bahasa Manusia secara Logika	Bahasa Algoritmik
jika berisi bilangan ganjil maka	
buat gambar spasi sebanyak `jumlah_bintang div 2`	`for penghitung_samping <- 1 to (jumlah_bintang div 2) do` `output(' ')` `{end for}`
menggambar pola bintang sebanyak `jumlah_bintang`	`for penghitung_samping <- 1 to jumlah_bintang do` `output('*')` `{end for}`
tampilkan kode ganti baris	`output("\n")` `{end if}`
melakukan pengulangan sebanyak isi dari `jumlah_bintang div 2` kali (untuk memproses pola ke arah bawah setelah pola tengah)	`for penghitung_vertikal <- 1 to (jumlah_bintang div 2) do`
melakukan pengulangan untuk menggambarkan spasi ke arah samping untuk setiap baris dengan kondisi berhenti tertentu sesuai dengan pola, jumlah spasi adalah deret menurun dimulai dari (`jumlah_bintang div 2`) dikurangi 1 menggunakan pengulangan turun	`for penghitung_samping <- ((jumlah_bintang div 2)-1) downto 1 do`
pada pengulangan tampilkan spasi ke layar	`output(' ')` `{end for}`
melakukan pengulangan untuk menggambarkan bintang ke arah samping sebanyak `jumlah_bintang` kali	`for penghitung_samping <- 1 to jumlah_bintang do` `output('*')` `{end for}`
tampilkan kode ganti baris	`output("\n")` `{end for}`

- Buat algoritma yang mengisi sebuah *array integer* dan menampilkan isi *array* yang merupakan bilangan ganjil

Penyelesaian:

Bahasa Manusia secara Logika	Bahasa Algoritmik
Membuat sebuah *array* bertipe integer	`tabInt : array [1..10] of integer`
Membuat kotak penghitung untuk memasukkan isi *array* dan memproses isi *array*	`penghitung : integer`
Meminta masukan dari *user* /pemakai program untuk mengisi *array* dengan logika berikut; melakukan pengulangan sebanyak kamar *array* isi *array*	`for penghitung <- 1 to 10 do` `input(tabInt`$_{penghitung}$`)` `{end for}`
Melakukan pengulangan untuk memeriksa apakah isi *array* satu per satu berisi ganjil atau tidak dengan logika sebagai berikut; melakukan pengulangan untuk menelusuri semua isi *array* jika elemen *array* ke penghitung mod 2 adalah 1 maka tampilkan isi elemen *array* ke penghitung	`for penghitung <- 1 to 10 do` `if (tabInt`$_{penghitung}$ `mod 2) = 1 then` `output(tabInt`$_{penghitung}$`)` `{end if}` `{end for}`

- Buat algoritma yang mengisi sebuah *array integer* dan menampilkan isi *array*, tapi berhenti menampilkan jika ditemukan angka 999

Penyelesaian:

Bahasa Manusia secara Logika	Bahasa Algoritmik

Bahasa Manusia secara Logika	Bahasa Algoritmik
Membuat sebuah *array* bertipe integer	`tabInt : array [1..10] of integer`
Membuat kotak penghitung untuk memasukkan isi *array* dan memproses isi *array*	`penghitung : integer`
Meminta masukan dari *user* /pemakai program untuk mengisi *array* dengan logika berikut; melakukan pengulangan sebanyak kamar *array* isi *array*	`for penghitung <- 1 to 10 do` `input(tabInt`$_{penghitung}$`)` `{end for}`
Melakukan pengulangan untuk memeriksa apakah isi *array* satu per satu berisi 999 atau tidak dengan logika sebagai berikut; inisialisasi nilai awal penghitung sebagai pengakses indeks *array* diisi dengan 1 melakukan pengulangan untuk menelusuri isi *array* dimana kondisi pengulangan berjalan jika bertemu isi elemen *array* bukan 999 dan penghitung tidak lebih dari 10 tampilkan isi elemen *array* ke penghitung iterasi penambahan penghitung dengan 1 untuk mengakses elemen *array* berikutnya	`penghitung <- 1` `while (tabInt`$_{penghitung}$ `<> 999) and (penghitung ≤ 10) do` `output(tabInt`$_{penghitung}$`)` `penghitung <- penghitung + 1` `{end while}`

- Buat algoritma yang mengisi sebuah *array integer* dan mencari elemen *array* yang paling kecil kemudian ditampilkan

Penyelesaian:

Bahasa Manusia secara Logika	Bahasa Algoritmik
Membuat sebuah *array* bertipe integer	`tabInt : array [1..10] of integer`
Membuat kotak penghitung untuk memasukkan isi *array* dan memproses isi *array*, dan sebuah kotak untuk menyimpan nilai minimal	`penghitung : integer` `nilai_minimal : integer`
Meminta masukan dari *user* /pemakai program untuk mengisi *array* dengan logika berikut; melakukan pengulangan sebanyak kamar *array* isi *array*	`for penghitung <- 1 to 10 do` `input(tabInt`$_{penghitung}$`)` `{end for}`
Melakukan inisialisasi terhadap `nilai_minimal` karena akan dibandingkan, untuk mencari nilai minimal maka `nilai_minimal` harus diset dengan nilai yang besar (secara logika jika kita ingin tahu hal yang buruk maka kita harus tahu yang baik dulu agar tahu kalau yang buruk itu adalah buruk)	`nilai_minimal <- 999999999`
Melakukan pengulangan untuk memeriksa apakah isi *array* merupakan isi minimal dengan logika sebagai berikut; melakukan pengulangan untuk menelusuri semua isi *array* jika elemen *array* ke `penghitung` lebih kecil dari `nilai_minimal` maka ganti `nilai_minimal` dengan isi *array* ke `penghitung`	`for penghitung <- 1 to 10 do` `if tabInt`$_{penghitung}$ `< nilai_minimal then` `nilai_minimal <- tabInt`$_{penghitung}$ `{end if}` `{end for}`
tampilkan nilai_minimal	`output(nilai_minimal)`

- Buat algoritma yang mengisi sebuah *array string* dan menampilkan isi *array* yang diawali dengan huruf 'A' atau 'a'

Penyelesaian:

Bahasa Manusia secara Logika	Bahasa Algoritmik
Membuat sebuah *array* bertipe *string*	`tabStr : array [1..10] of string`
Membuat kotak penghitung untuk memasukkan isi *array* dan memproses isi *array*	`penghitung : integer`
Meminta masukan dari *user* /pemakai program untuk mengisi *array* dengan logika berikut; melakukan pengulangan sebanyak kamar *array* isi *array*	`for penghitung <- 1 to 10 do` `input(tabStr`$_{penghitung}$`)` `{end for}`
Melakukan pengulangan untuk memeriksa apakah isi *array* memiliki huruf depan 'A' atau 'a' dengan logika sebagai berikut; melakukan pengulangan untuk menelusuri semua isi *array* jika elemen *array* ke `penghitung` huruf pertamanya adalah 'A' atau 'a' maka tampilkan isi elemen *array* ke `penghitung`	`for penghitung <- 1 to 10 do` `if (tabStr`$_{penghitung,1}$ `= 'A') or (tabStr`$_{penghitung,1}$ `= 'a') then` `output(tabStr`$_{penghitung}$`)` `{end if}` `{end for}`

Cara penulisan kode program untuk mengakses huruf pada *string* adalah sebagai berikut:

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`tabStr`$_{penghitung,1}$	`tabStr[penghitung, 1]`	`tabStr[penghitung][0];`

Soal Latihan:

1. Buatlah algoritma dan program dengan bahasa Pascal atau C yang mengisi *array* dan memproses *array* yang berisi elemen bentukan segitiga (tipe terstruktur segitiga terdiri dari x, y, r). *Array* segitiga masukan hanya telah diisi x dan y –nya saja pada proses pengisian *array*. Hitung sisi miring setiap elemen dan isikan pada atribut r dari segitiga.
2. Buatlah algoritma dan program dengan bahasa Pascal atau C yang yang mengisi dan memproses *array* yang berisi elemen string yang menyatakan nilai float dan menampilkan hasil pertambahan dari elemen *string array* yang telah diproses dan diubah ke float (ubah elemen string menjadi float), misal isi array: "2.3", "4.5", "6.9" maka hasil keluaran adalah hasil perubahan menjadi float dari *array* maka hasil keluarannya adalah 2.3 + 4.5 + 6.9 yaitu 12.7
3. Buatlah algoritma dan program dengan bahasa Pascal atau C yang mengisi dan memproses *array* masukan berisi elemen tipe terstruktur lingkaran. Tipe terstruktur lingkaran terdiri jari-jari, luas. Pada proses pengisian, isi hanya jari-jari dari setiap elemen *array*. Hitung masing-masing luas lingkaran dari setiap elemen *array*.
4. Buatlah algoritma dan program dengan bahasa Pascal atau C yang mengisi dan memproses *array* berisi *string* dan menuliskan ke layar isi *string* dengan ketentuan sebagai berikut:

   String yang diawali huruf A harus ditulis dari belakang, misal "Alpha" maka akan menampilkan "ahplA" (tangani huruf 'a' dan 'A')

   String yang diawali huruf i harus ditulis dari dari tengan, misal "Insan" maka akan menampilkan "snIan" (tangani huruf 'i' dan 'I'), dari tengah ke depan ditulis dari tengah dilanjutkan dengan dari tengah ke belakang

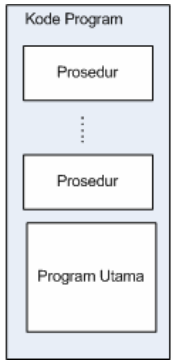
   String yang diawali huruf U harus ditulis dari belakang, misal "Ulfa" maka akan menampilkan "aflU" (tangani huruf 'u' dan 'U')

   String yang diawali huruf E dan O harus ditulis dari depan, misal "Elang" maka akan menampilkan "Elang" (tangani huruf 'o' dan 'O', tangani huruf 'e' dan 'E')
5. Buatlah algoritma dan program dengan bahasa Pascal atau C yang mengisi dan memproses *array* berisi elemen karakter dan menampilkan isi elemen *array* jika isi elemen *array* adalah karakter konsonan.

9 Prosedur

9.1 Pengertian Prosedur

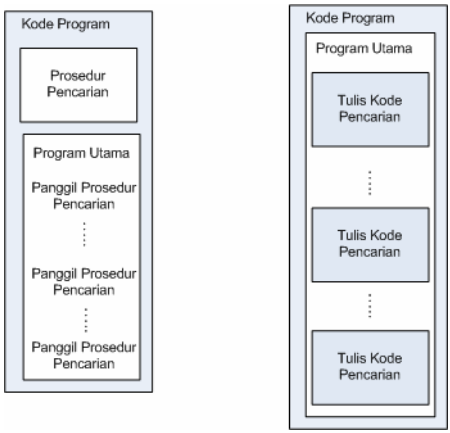
Prosedur adalah sebuah blok program tersendiri yang merupakan bagian dari program lain yang lebih besar. Prosedur dapat dipanggil oleh program utamanya ataupun oleh prosedur lain yang juga merupakan bagian dari program utamanya (masih dalam satu kode program). Sebuah program yang memiliki prosedur biasanya terdiri dari satu atau lebih prosedur dan satu program utama. Sebagai ilustrasi dapat dilihat pada gambar 5.



Gambar 5 Ilustrasi Prosedur pada Kode Program

Orang yang pertama kali belajar memprogram sering kali berpikir kenapa diperlukan prosedur, kenapa tidak hanya satu program utama saja. Prosedur memiliki beberapa keuntungan sebagai berikut :

1. Memecah-mecah program menjadi lebih sederhana, misalnya jika diperlukan proses pencarian berkali-kali jika hanya terdiri dari satu program utama tanpa prosedur, maka kode program pencarian akan beberapa kali ditulis ulang dan hasilnya dapat memperbesar ukuran file. Untuk lebih jelasnya dapat dilihat pada gambar 6.



Gambar 6 Perbedaan Penggunaan Prosedur

2. Blok program yang digunakan jelas jika akan digunakan pada program lain, cukup dengan mengkopi satu prosedur dan meletakkannya pada program lain yang membutuhkannya dan program lain tersebut tinggal memanggil prosedur tersebut.

Perbedaan prosedur dan fungsi yang paling utama adalah prosedur tidak menghasilkan nilai, hanya merupakan proses di dalamnya sedangkan fungsi menghasilkan nilai keluaran (*output*). Jika tidak membutuhkan hasil keluaran dan hanya membutuhkan proses maka prosedur dapat digunakan, tapi jika membutuhkan hasil keluaran (*output*) maka gunakan fungsi.

9.2 Deklarasi Prosedur

Berikut adalah cara mendeklarasikan sebuah prosedur:

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
procedure nama_prosedur(input: variabel_input1 : tipe_data; ......; variabel_inputn : tipe_data) {*proses*} ..... {*end procedure*}	procedure nama_prosedur(variabel_input1 : tipe_data; .....; variabel_inputn : tipe_data); var {variabel} begin { proses } ..... end;	void nama_prosedur(tipe_data variabel_input1, ..., tipe_data variabel_inputn){ // proses ..... }
procedure cariNama(input: nama : string) {*proses*} ..... {*end procedure*}	procedure cariNama(nama : string); var {variabel} begin { proses } ..... end;	void cariNama(char nama[]){ // proses ..... }

sedangkan cara memanggil prosedur adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
nama_prosedur(variabel_masukan)	nama_prosedur(variabel_masukan);	nama_prosedur(variabel_masukan);
cariNama(nama)	cariNama(nama);	cariNama(nama);

Untuk bahasa algoritmik, pendeklarasian prosedur ada beberapa macam sebagai berikut :

Bahasa Algoritmik	Keterangan
procedure nama_procedure(input : nama_variabel : tipe_data)	merupakan prosedur yang hanya memiliki variabel sebagai masukan sehingga setelah prosedur dijalankan tidak ada nilai variabel masukan manapun

Bahasa Algoritmik	Keterangan
	yang berubah, misalnya : `procedure tampil(input : a : nteger)` `  write("nilai a adalah : ", a)` `{end procedure}` selesai prosedur dijalankan, nilai a tidak mengalami perubahan
`procedure nama_procedure(output : nama_variabel : tipe_data)`	merupakan prosedur yang memiliki variabel masukan, tapi nilai dari variabel masukan mengalami perubahan setelah keluar dari prosedur karena proses dalam prosedur setelah prosedur dijalankan tanpa menggunakan nilai dari variabel masukan, misalnya : `procedure hasil(output : a : integer)` `  a <- 2 + 3` `{end procedure}` selesai prosedur dijalankan, nilai a mengalami perubahan menjadi 5 hasil dari 2 + 3
`procedure nama_procedure(input: nama_variabel : tipe_data, output : nama_variabel : tipe_data)`	merupakan prosedur yang memiliki variabel masukan yang tidak berubah nilainya dan yang berubah nilainya setelah prosedur dijalankan, misalnya: `procedure hasil(input : a : integer, output : b : integer)` `  b <- a + 3` `{end procedure}` selesai prosedur dijalankan, nilai a tidak mengalami perubahan sedangkan nilai b mengalami perubahan

Bahasa Algoritmik	Keterangan
procedure nama_procedure(input/output : nama_variabel : tipe_data)	merupakan prosedur yang memiliki variabel masukan, tapi nilai dari variabel masukan mengalami perubahan karena proses dalam prosedur setelah prosedur dijalankan dengan menggunakan nilai dari variabel masukan, misalnya : procedure hasil(input/output : a : integer) a <- a + 3 {*end procedure*} selesai prosedur dijalankan, nilai a mengalami perubahan

dalam aplikasinya pada bahasa pemrograman Pascal dan C adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
procedure nama_procedure(input : nama_variabel : tipe_data) // proses ..... {*end procedure*}	void nama_prosedur(tipe_data nama_variabel){ // proses ..... }	void nama_prosedur(tipe_dat a nama_variabel){ // proses ..... }
procedure nama_procedure(output : nama_variabel : tipe_data) // proses ..... {*end procedure*}	void nama_prosedur(tipe_data *nama_variabel){ // proses ..... }	void nama_prosedur(tipe_dat a *nama_variabel){ // proses ..... }
procedure nama_procedure(input: nama_variabel1 : tipe_data, output : nama_variabel2 : tipe_data) // proses ..... {*end procedure*}	void nama_prosedur(tipe_data nama_variabel1, tipe_data *nama_variabel2){ // proses ..... }	void nama_prosedur(tipe_dat a nama_variabel1, tipe_data *nama_variabel2){ // proses ..... }
procedure nama_procedure(input/outp ut : nama_variabel :	void nama_prosedur(tipe_data *nama_variabel){	void nama_prosedur(tipe_dat a *nama_variabel){

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
tipe_data) // proses ..... {*end procedure*}	// proses ..... }	// proses ..... }

Berikut adalah contoh algoritma program implementasi sebuah prosedur yang menuliskan kata masukan ke layar dengan sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik
Mendeklarasikan prosedur tulis	procedure tulis(input : kata : string)
Menuliskan kata masukan ke layar pada prosedur	output(kata)
Mendeklarasikan program utama, mendeklarasikan variabel kata, dan memanggil prosedur	{*Program Utama*} kata : string kata <- "program utama prosedur tulis" tulis(kata)

Mengubah bahasa algoritmik menjadi bahasa pemrograman Pascal atau bahasa pemrograman C seperti berikut :

Bahasa Algoritmik	Bahasa Pascal
    procedure tulis(input : kata : string) output(kata) {*end procedure*} {*Program Utama*} kata : string kata <- "program utama prosedur tulis" tulis(kata)	var kata : string; procedure tulis(kata : string); begin writeln(kata); end; begin kata := 'program utama prosedur tulis'; tulis(kata); end.

Bahasa Algoritmik	Bahasa C
procedure tulis(input : kata : string) output(kata) {*end procedure*} {*Program Utama*} kata : string kata <- "program utama prosedur tulis" tulis(kata)	#include <stdio.h> void tulis(char kata[]){ printf("%s", kata); } int main(){ char kata[] = "program utama prosedur tulis \n"; tulis(kata); return 1; }

Pada bahasa pemrograman C deklarasi prosedur dapat diletakkan pada file header .h yang biasanya berisi deklarasi struktur, variabel global, fungsi, atau prosedur dengan isi sebagai berikut :

Nama File : prosedur.h

```
#ifndef prosedur_H
#define prosedur_H
#include <stdio.h>

void tulis(char kata[]);

#endif
```

kata kunci define digunakan untuk mendefinsikan sebuah nama lain agar dikenali, misalkan sintaksnya :

```
#define true 1
```

berarti mendefinisikan bahwa angka 1 dapat disebut sebagai true, jadi pemakaiannya dapat sebagai berikut :

```
int ketemu = true;
```

sedangkan jika sintaksnya :

```
#define boolean_H
```

berarti mendefinisikan sebuah file header dengan nama boolean.

Isi dari file .c adalah sebagai berikut :

Nama file : prosedur.c

```
#include "prosedur.h"

void tulis(char kata[]){
  printf("%s", kata);
}

int main(){

  char kata[] = "program utama prosedur tulis \n";
  tulis(kata);

  return 1;

}
```

9.3 Parameter dalam Prosedur

Parameter dalam prosedur adalah variabel masukan (input) dari sebuah prosedur misalnya pada prosedur berikut :

```
     prosedure hitungJumlah(input : hasil : integer;  operan1 : integer;
operan2 : integer)
        hasil <- operan1 + operan2
     {end prosedure}
```

`hasil`, `operan1`, dan `operan2` adalah parameter dalam prosedur hitungJumlah dimana jumlah parameter tidak terbatas, namun prosedur yang baik biasanya memiliki parameter masukan paling banyak tujuh buah parameter. Masukan dari sebuah prosedur tidak harus berupa variabel, dapat langsung berupa nilai karena jika yang dijadikan masukan adalah variabel, tetap saja yang diproses dalam prosedur adalah nilai dari variabel itu, oleh karena itu dalam pemanggilan sebuah prosedur dapat dilakukan dengan cara sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`hitungJumlah( 1, 2, 3)`	`hitungJumlah( 1, 2, 3);`	`hitungJumlah( 1, 2, 3);`

atau jika menggunakan variabel masukan menjadi :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
`hitungJumlah(a, b, c)`	`hitungJumlah(a, b, c);`	`hitungJumlah(a, b, c);`

dimana `a`, `b`, dan `c` adalah variabel bertipe *integer*.

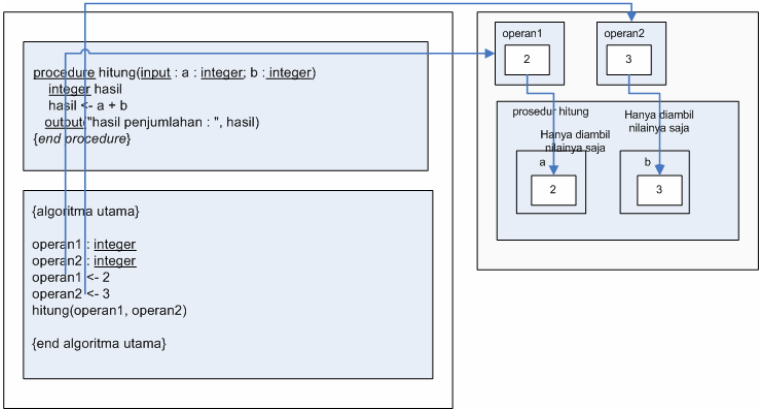
Terdapat dua buah metode pemrosesan parameter dalam penggunaan sebuah prosedur yaitu pemrosesan parameter berdasarkan nilainya (*parameter pass by value*) dan pemrosesan parameter berdasarkan acuan (*parameter pass by reference*).

9.3.1 Pemrosesan Parameter Berdasarkan Nilainya (parameter pass by value)

Pemrosesan parameter berdasarkan nilainya adalah pemrosesan parameter di dalam prosedur, dimana parameter hanya dipakai nilainya saja pada proses yang ada di dalam prosedur sehingga tidak ada nilai dari parameter yang berubah setelah prosedur dijalankan misalnya pada contoh prosedur dengan bahasa algoritmik sebagai berikut :

```
procedure hitung(input : a : integer; b : integer)
    integer hasil
    hasil <- a + b
    output("hasil penjumlahan : ", hasil)
{end procedure}
```

parameter masukan `a` dan `b` hanya dipakai nilainya saja, nilai parameter ini tidak berubah setelah prosedur `hitung` dijalankan. Pada gambar 7 terdapat variabel `operan1` dan `operan2` sebagai paramater masukan prosedur `hitung`, di dalam prosedur `hitung` nilai `a` diisi dengan nilai `operan1` dan nilai `b` diisi dengan nilai `operan2` sehingga nilai `operan1` dan nilai `operan2` tidak berubah setelah prosedur `hitung` dijalankan.



Gambar 7 Parameter *Pass By Value*

Default atau standar pada bahasa pemrograman C adalah *pass by value*.

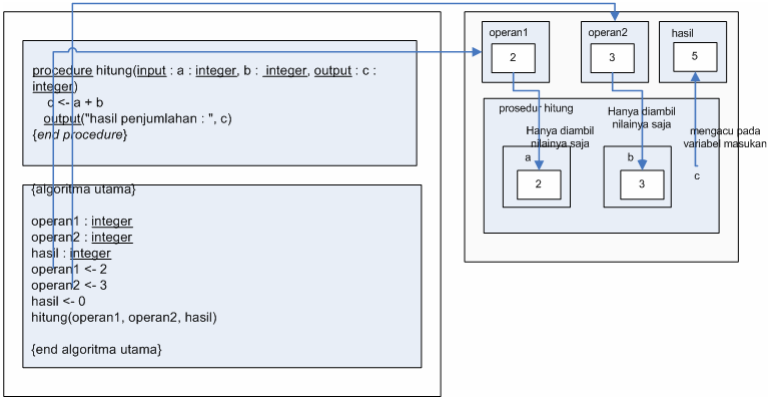
9.3.2 Pemrosesan Parameter Berdasarkan Acuan (parameter pass by reference)

Pemrosesan parameter berdasarkan acuan adalah pemrosesan parameter di dalam sebuah prosedur dimana yang dimasukkan di dalam prosedur adalah tempat dari variabel yang menjadi parameter sehingga dapat terjadi perubahan nilai variabel yang menjadi parameter, misalnya pada contoh prosedur dengan bahasa algoritmik sebagai berikut :

```
      procedure hitung(input : a : integer; b : integer, output : c :
integer)
          c <- a + b
          output("hasil penjumlahan : ", c)
      {end procedure}
```

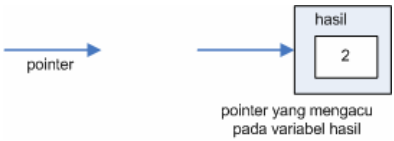
parameter masukan `a` dan `b` hanya dipakai nilainya saja, nilai parameter ini tidak berubah setelah prosedur `hitung` dijalankan sedangkan parameter `c` akan berubah nilainya setelah prosedur `hitung` dijalankan. Pada gambar 8 dapat dilihat bahwa pada saat pemrosesan di dalam prosedur `hitung operan1` dan `operan2` hanya diambil nilainya saja sedangkan `c` merupakan variabel yang mengacu pada variabel `hasil`, di dalam prosedur `hitung`, variabel `c` adalah

variabel `hasil` sehingga saat `c` berubah nilai maka hasil juga berubah nilainya, dalam hal ini variabel `c` adalah sebuah pointer (pengacu) (silakan eksplorasi ke *search engine*).



Gambar 8 Parameter *Pass By Reference*

Secara standar pada bahasa pemrograman C *pass by value* namun C mengijinkan mengakses alamat variabel secara langsung dengan menambahkan tanda * (bintang) jika ditulis sebagai parameter prosedur pada deklarasi prosedur, dan menambah tanda & (dan) pada saat memanggil prosedur pada variabel masukan. Karakter * (bintang) pada bahasa pemrograman C bisa berarti *parameter pass by reference* atau menyatakan pointer. Pointer berarti yang dipesan pada memori bukanlah sebuah tempat, tapi hanya sebuah penunjuk yang mengacu ke tempat variabel yang akan diubah nilainya seperti pada gambar 9.



Gambar 9 Ilustrasi Pointer

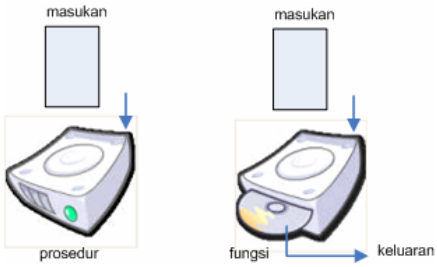
9.4 Prosedur Standar

Prosedur standar adalah prosedur yang dapat langsung dipakai tanpa membuatnya sendiri karena telah ada pada pustaka (*library*), yang perlu dilakukan adalah menyertakan pustaka yang berisi prosedur standar yang dibutuhkan pada kode program dengan kata kunci `include` untuk bahasa pemrograman C, atau uses pada bahasa pemrograman Pascal. Untuk mengetahui prosedur apa saja yang dapat dipakai, perlu dilakukan eksplorasi terhadap pustaka (*library*) yang disediakan. Selain prosedur juga terdapat fungsi standar yang juga dapat langsung dipakai tanpa membuatnya sendiri. Pengertian fungsi akan dibahas pada bab selanjutnya sedangkan beberapa prosedur dan fungsi standar yang ada pada bahasa pemrograman C dapat dilihat pada lampiran buku ini.

10 Fungsi

10.1 Pengertian Fungsi

Fungsi adalah sebuah blok program tersendiri yang merupakan bagian dari program lain yang lebih besar sama halnya dengan prosedur hanya saja fungsi memiliki hasil keluaran sedangkan prosedur hanya berisi proses. Fungsi dapat dipanggil oleh program utamanya ataupun oleh prosedur atau fungsi lain yang juga merupakan bagian dari program utamanya (masih dalam satu kode program). Sebuah program bisa terdiri dari satu atau lebih prosedur, satu atau lebih fungsi, dan satu program utama. Fungsi juga memiliki keuntungan-keuntungan seperti halnya prosedur, hal paling mendasar yang membedakan prosedur dan fungsi adalah fungsi menghasilkan sebuah keluaran sedangkan prosedur tidak. Pada gambar 10, prosedur melakukan proses dan melakukan perubahan nilai di dalam proses yang dilakukan di dalam prosedur, keluaran atau output yang ada pada deklarasi prosedur berarti nilai variabel itu akan berubah bergitu prosedur dijalankan sedangkan fungsi melakukan proses dan perubahan nilai dengan proses di dalam fungsi namun fungsi menghasilkan nilai keluaran yang dapat ditangkap oleh sebuah variabel.



Gambar 10 Ilustrasi Perbedaan Prosedur dan Fungsi

10.2 Deklarasi Fungsi

Berikut adalah cara mendeklarasikan sebuah fungsi pada bahasa algoritmik, bahasa pemrograman Pascal dan C:

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
function nama_fungsi(nama_variabel_masukan : tipe_data) -> tipe_data_keluaran {*proses*} …… -> variabel_keluaran {*end function*}	function nama_fungsi(nama_variabel_masukan : tipe_data) : tipe_data_keluaran; var begin { proses } …… nama_fungsi := variabel_keluaran; end;	tipe_data_keluaran nama_fungsi(tipe_data nama_variabel_masukan){ // *proses* …… return variabel_keluaran; }
function nilaiTerbesar(a : integer; b : integer, c : integer) -> integer {*proses*} integer hasil ……. -> hasil {*end function*}	function nilaiTerbesar(a : integer; b : integer; c : integer) : integer; var hasil : integer; begin { proses } …… nilaiTerbesar := hasil; end;	int nilaiTerbesar(int a, int b, int c){ // *proses* int hasil; …… return hasil; }

sedangkan cara memanggil fungsi adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
nama_variabel <- nama_fungsi (variabel_masukan)	nama_variabel := nama_fungsi(variabel_masukan);	nama_variabel = nama_fungsi(variabel_masukan);
maksimum <- nilaiTerbesar(a, b, c)	maksimum := nilaiTerbesar(a, b, c);	maksimum = nilaiTerbesar(a, b, c);

Berikut adalah sebuah fungsi yang menghasilkan hasil perkalian dari tiga buah parameter masukan:

Bahasa Manusia	Bahasa Algoritmik
Mendeklarasikan fungsi hasilKali yang menghasilkan nilai hasil perkalian dari tiga buah variabel masukan	function hasilKali(a : integer; b : integer; c : integer)-> integer hasil : integer hasil <- a * b * c

Bahasa Manusia	Bahasa Algoritmik
Mendeklarasikan variabel hasil yang nilainya merupakan nilai keluaran fungsi	`-> hasil`
Mendeklarasikan program utama, mendeklarasikan tiga variabel masukan dan variabel hasil, memanggil fungsi hasilKali serta menampilkannya ke layar	`{algoritma utama}` `  a : integer` `  b : integer` `  hasil : integer` `  a <- 1` `  b <- 2` `  c <- 3` `  hasil <- hasilKali(a, b, c)` `  output("hasil perkalian : ", hasil)` `{end algoritma utama}`

Mengubah bahasa algoritmik menjadi bahasa pemrograman Pascal atau bahasa pemrograman C seperti berikut :

Bahasa Algoritmik	Bahasa Pascal
	`var` `a, b, c : integer;`
`function hasilKali(a : integer; b : integer; c : integer)-> integer`	`function hasilKali(a : integer; b : integer; c : integer) : integer;` `var`
`  hasil : integer`	`hasil : integer;`
	`begin`
`  hasil <- a * b * c` `  -> hasil`	`  hasil := a * b * c;` `  hasilKali :=  hasil;`
`{end function}`	`end;`
`{algoritma utama}`	`begin`
`  a : integer` `  b : integer` `  hasil : integer`	
`  a <- 1` `  b <- 2` `  c <- 3`	`  a := 1;` `  b := 2;` `  c := 3;`
`  hasil <- hasilKali(a, b, c)` `  output("hasil perkalian : ", hasil)`	`  hasil := hasilKali(a, b, c);` `  write("hasil perkalian : ", hasil);`
`{end algoritma utama}`	`end.`

Bahasa Algoritmik	Bahasa C
function hasilKali(a : integer; b : integer; c : integer)-> integer hasil : integer hasil <- a * b * c -> hasil {*end function*} *{algoritma utama}* a : integer b : integer hasil : integer a <- 1 b <- 2 c <- 3 hasil <- hasilKali(a, b, c) output("hasil perkalian : ", hasil) {*end algoritma utama*}	#include <stdio.h> int hasilKali(int a, int b, int c){ int hasil; hasil = a * b * c; return hasil; } int main(){ int a; int b; int c; int hasil; a = 1; b = 2; c = 3; hasil = hasilKali(a, b, c); printf("hasil perkalian : %d\n", hasil); return 1; }

10.3 Parameter Fungsi

Seperti halnya prosedur, fungsi juga memiliki parameter. Pengertian parameter fungsi juga sama dengan pengertian parameter pada prosedur seperti yang telah dijelaskan pada bab sebelumnya. Begitu pula dengan pemrosesannya, pada fungsi juga mengenal *pass by value* dan *pass by reference* seperti pada prosedur.

Daftar Pustaka

[1] Griffith, Arthur. 2002. GCC : The Complete Reference. McGraw-Hill Book Co.
[2] Jamsa, Kris. 1996. 1001 Tips C/C++. Penerbit ANDI : Yogyakarta.
[3] Kadir, Abdul. 2001. Pemrograman C++. Penerbit ANDI : Yogyakarta.
[4] Liem, Inggriani. 2003. Diktat Kuliah IF1281 Algoritma dan Pemrograman. ITB : Bandung.
[5] Munir, Rinaldi. 2001. Algoritma dan Pemrograman Buku II. Penerbit Informatika : Bandung.
[6] Shalahuddin, M. dan Rosa A. S. 2007. Belanjar Pemrograman dengan Bahasa Pemrograman C++ dan Java: dari Nol Menjadi Andal. Penerbit Informatika : Bandung.

Lampiran 1 - Proses Kompilasi Sampai Eksekusi Kode Program

Mengenai MinGW (Windows) /GCC (linux)

- MinGW/GCC untuk bahasa C digunakan dengan menggunakan comand prompt/ terminal/console
- Pembuatan kode program dapat menggunakan notepad++ (*freeware*)
- Untuk kompilasi hingga eksekusi, maka langkah-langkah yang harus dilakukan adalah sebagai berikut:
  - Buat kode program dengan menggunakan notepad++ dan simpan dalam file dengan ekstensi .c (c kecil)
  - Buka command prompt/terminal/console
  - Pergi ke tempat file .c disimpan pada command prompt dengan perintah
    - drive: (tekan enter)
      misal
      d: (tekan enter)
      maka akan masuk pada drive d
    - cd huruf_awal_direktori (tekan tab sampai ke direktori yang dituju)
  - Set path dimana MinGW diinstal dengan perintah
    - path=path_mingw/bin;%path% (tekan enter)
      misal
      path=C:/MinGW/bin;%path% (tekan enter)

    set path dimaksudkan agar perintah gcc untuk kompilasi dikenali pada direktori tempat menyimpan file .c (dan .h jika ada)
  - Kompilasi kode program dengan perintah:
    - gcc nama_file.c (tekan enter)
      misal
      gcc titik.c (tekan enter)
      jika masih ada *error* maka perhatikan pesan *error* yang ditampilkan, biasanya ada nama file yang mengalami *error*, ada baris keberapa dari *file* itu yang mengalami *error*, dan ada pesan *error*.
      Perbaiki kode program lalu lakukan kompilasi lagi sampai tidak ada *error*
  - Buat kode .exe dengan perintah
    - gcc nama_file.c -o nama_exe.exe (tekan enter)
      misal
      gcc titik.c -o titik.exe (tekan enter)

- Eksekusi Program dengan perintah:
  - nama_exe.exe (tekan enter)
    misal
    titik.exe
- Untuk keluar dari program sebelum selesai dieksekusi tekan ctrl+c
- Setiap mengubah kode program ulangi proses mulai dari perintah kompilasi (untuk mempermudah dapat menekan tombol up atau down agar perintah yang pernah diketik muncul kembali tanpa harus mengetik ulang)

Mengenai Free Pascal Compiler (FPC)

- FPC untuk bahasa Pascal dapat digunakan dengan menggunakan comand prompt/ terminal/console
- Pembuatan kode program dapat menggunakan notepad++ (*freeware*)
- Untuk kompilasi hingga eksekusi, maka langkah-langkah yang harus dilakukan adalah sebagai berikut:
  - Buat kode program dengan menggunakan notepad++ dan simpan dalam file dengan ekstensi .pas
  - Buka command prompt/terminal/console
  - Pergi ke tempat file .pas disimpan pada command prompt dengan perintah
    - drive: (tekan enter)
      misal
      d: (tekan enter)
      maka akan masuk pada drive d
    - cd huruf_awal_direktori (tekan tab sampai ke direktori yang dituju)
  - Set path dimana FPC diinstal dengan perintah
    - path=path_fpc/bin;%path% (tekan enter)
      misal
      path=C:/FPC/bin;%path% (tekan enter)

    set path dimaksudkan agar perintah fpc untuk kompilasi dikenali pada direktori tempat menyimpan file .pas
  - Kompilasi kode program dengan perintah:
    - fpc nama_file.pas (tekan enter)
      misal
      fpc titik.pas (tekan enter)
      jika masih ada *error* maka perhatikan pesan *error* yang ditampilkan, biasanya ada nama file yang mengalami *error*, ada baris keberapa dari *file* itu yang mengalami *error*, dan ada pesan *error*.
      Perbaiki kode program lalu lakukan kompilasi lagi sampai tidak ada *error*

- Buat kode .exe dengan perintah
  - fpc -Fu:\path_rtl_fpc nama_exe (tekan enter)
    misal
    fpc -Fu:\fpc\2.0.4\units\i386-win32\rtl titik (tekan enter)

- Eksekusi Program dengan perintah:
  - nama_exe.exe (tekan enter)
    misal
    titik.exe
- Untuk keluar dari program sebelum selesai dieksekusi tekan ctrl+c

Setiap mengubah kode program ulangi proses mulai dari perintah kompilasi (untuk mempermudah dapat menekan tombol up atau down agar perintah yang pernah diketik muncul kembali tanpa harus mengetik ulang)

Lampiran 2 - Prosedur dan Fungsi Standar pada Bahasa Pemrograman C

Bahasa Pemrograman C	
Matematika	
pustaka : math.h	
Prosedur/Fungsi	**Keterangan**
abs(bil_masukan) contoh : x = abs(y);	menjadikan bilangan bulat masukan sebagai bilangan absolut (positif)
fabs(bil_masukan) contoh : x = fabs(y);	menjadikan bilangan riil masukan sebagai bilangan absolut (positif)
sqrt(bil_masukan) contoh : x = sqrt(y);	untuk menghitung akar pangkat dua dari suatu bilangan positif
exp(bil_masukan) contoh : x = exp(y);	untuk menghitung $e^{bil\_masukan}$ dimana e adalah bilangan natural
log(bil_masukan) contoh : x = log(y);	untuk menghitung logaritma naturalis (berbasis e)

	dari bilangan masukan
log10(bil_masukan) contoh : x = log10(y);	untuk menghitung logaritma basis 10 dari bilangan masukan
pow(x,y) contoh : z = pow(x,y);	untuk menghitung hasil x^y
acos(bil_masukan) contoh : x = acos(y);	menghitung arcus cosinus bilangan masukan
asin(bil_masukan) contoh : x = asin(y);	menghitung arcus sinus bilangan masukan
atan(bil_masukan) contoh : x = atan(y);	menghitung arcus tangent bilangan masukan
cos(bil_masukan) contoh : x = cos(y);	untuk menghitung cosinus bilangan masukan
cosh(bil_masukan) contoh : x = cosh(y);	untuk menghitung cos hiperbolis dari bilangan masukan
sin(bil_masukan) contoh : x = sin(y);	untuk menghitung sinus dari bilangan masukan
sinh(bil_masukan)	untuk

contoh : x = sinh(y);	menghitung sinus hiperbolis dari bilangan masukan
tan(bil_masukan) contoh : x = tan(y);	untuk menghitung tangent bilangan masukan
tanh(bil_masukan) contoh : x = tanh(y);	untuk menghitung tangent hiperbolis dari bilangan masukan
atof(string_masukan) contoh : x = atof(“65.78”);	untuk mengubah string masukan menjadi bilangan riil, pustaka ditambah dengan stdlib.h
atoi(string_masukan) contoh : x = atoi(“23”);	untuk mengubah string masukan menjadi bilangan integer pustaka ditambah dengan stdlib.h
rand() contoh : x = rand();	menghasilkan bilangan bulat secara acak pustaka

	ditambah dengan stdlib.h
random(bil_masukan) contoh : x = random(100);	menghasilkan bilangan bulat secara acak diantara 0 sampai bilangan masukan dikurangi 1 pustaka ditambah dengan stdlib.h
randomize() contoh : randomize();	menginisialisasi proses pengacakan bilangan pustaka ditambah dengan stdlib.h dan time.h
srand(bil_masukan) contoh : srand(100);	menginisialisasi proses pengacakan mulai dari 0 sampai bilangan masukan dikurangi 1 pustaka ditambah dengan stdlib.h
max(a, b) contoh : float x = (float) max(a,b);	untuk menentukan bilangan terbesar antara a dan b

min(a,b) contoh ; int x = (int) min(a,b);	untuk menentukan bilangan terkecil antara a dan b
Manipulasi String	
pustaka :string.h	
Prosedur/Fungsi	**Keterangan**
strcpy(string_hasil, string_asal) contoh : strcpy(hasil, asal);	menyalin string asal ke string hasil
strcat(string_hasil, string_tambah) contoh : strcat(hasil, tambah);	menggabungkan isi string tambah ke string hasil
strcmp(string1, string2) contoh : if(strcmp(string1, string2) == 0){ ..... }	membandingkan dua buah string masukan dengan memperhatikan huruf besar dan huruf kecil menghasilkan bilangan negatif jika string1 lebih kecil dari string2, menghasilkan 0 jika kedua string sama, menghasilkan bilangan positif lebih besar 0 jika string1 lebih

	besar dari string2
strcmpi(string1, string2) contoh : if(strcmpi(string1, string2) == 0){ ..... }	membandingkan dua buah string masukan tanpa memperhatikan huruf besar dan huruf kecil menghasilkan bilangan negatif jika string1 lebih kecil dari string2, menghasilkan 0 jika kedua string sama, menghasilkan bilangan positif lebih besar 0 jika string1 lebih besar dari string2
strlen(string_masukan) contoh : x = strlen("program");	menghitung panjang karakter pada string masukan
strstr(string1, string2) contoh : strstr(string1, string2);	menentukan ada atau tidaknya suatu string2 di dalam string1 jika tidak ada maka hasilnya adalah null
tolower (string_masukan) contoh :	mengubah semua karakter pada string

tolower(teks);	masukan menjadi berhuruf kecil semua perlu ditambahkan pustaka ctype.h
strupr(string_masukan) contoh : char teks[] = "informatika"; strupr(teks); atau toupper(string_masukan) contoh : toupper(teks);	mengubah semua karakter pada string masukan menjadi berhuruf besar semua jika menggunakan toupper() perlu ditambahkan pustaka ctype.h
Waktu	
pustaka : time.h	
Prosedur/Fungsi	**Keterangan**
_strdate(string_masukan) contoh : char tanggal[9]; _strdate(tanggal);	mengembalikan tanggal saat ini
_strtime(string_masukan) contoh : char waktu[9]; _strtime(waktu);	mengembalikan waktu saat ini