

MAKALAH
ALGORITMA SEMESTER SATU

Makalah Ini Disusun untuk Memenuhi Ujian Akhir Semester

Mata Kuliah Algoritma



Disusun Oleh :

Nama : Farkhan

NPM : 20081010060

Kelas : B

PROGRAM STUDI INFORMATIKAFAKULTAS ILMU KOMPUTER

UNIVERSITAS PEMBANGUNAN NASIONAL "VETERAN"

JAWA TIMUR

2021

Daftar Isi

1. ALGORITMA DAN PROGRAM	1
1.1. Algoritma	1
1.2. Program	1
1.3. Flowchart	5
2. STRUKTUR DASAR ALGORITMA	6
2.1. Runutan (<i>Sequential</i>)	6
2.2. Pemilihan (Selection/Branching)	7
2.3. Pengulangan (<i>Looping</i>)	8
3. KOMBINASI STRUKTUR DASAR ALGORITMA.....	11
9.1. Pseudocode	12
4. RAPTOR.....	14
5. PERULANGAN BERSARANG	17
6. KONSEP LARIK	19
7. SUBPROGRAM.....	22
8. REKURS	24
9. PENCARIAN DATA DAN PENGURUTAN DATA	28
9.1. Pencarian Data.....	28
9.2. Pengurutan Data	30

1. ALGORITMA DAN PROGRAM

1.1. Algoritma

Algoritma merupakan sekumpulan langkah-langkah atau instruksi-instruksi yang terbatas untuk menyelesaikan suatu masalah. Algoritma digunakan untuk mengolah data menjadi sebuah informasi (*input* menjadi *output*), pemrogram wajib menyusun langkah detail (runutan) bagaimana komputer akan menyelesaikan masalah-masalah tersebut, langkah detail ini disebut algoritma. Kata algoritma pertama kali diperkenalkan oleh seorang ilmuwan matematika yang berasal dari Persia yang bernama Al Khawarizmi, beliau hidup pada tahun 780-850 masehi.

Berikut ini merupakan suatu contoh algoritma yang digunakan untuk menghitung luas lingkaran.

- Dapatkan jari-jari lingkaran
- Hitung luas lingkaran dengan menggunakan rumus $3.14 \times \text{jari-jari} \times \text{jari-jari}$
- Tampilkan nilai luas lingkaran

Selain menggunakan kalimat, algoritma juga bisa dituliskan menggunakan pseudocode. Contoh algoritma dalam bentuk pseudocode ialah sebagai berikut.

- Jari-jari $\leftarrow 20$
- Luas $\leftarrow 3.14 * (\text{jari-jari}^2)$
- Print (Luas)

1.2. Program

Program adalah kumpulan instruksi yang digunakan untuk mengatur komputer agar melakukan suatu tindakan tertentu. Pada dasarnya, komputer hanyalah mesin kosong yang tidak bisa apa-apa atau tidak berfungsi tanpa adanya program. Komputer memiliki beberapa komponen, di antaranya adalah (1) perangkat keras (*hardware*), (2) perangkat lunak (*software*), dan manusia yang mengendalikannya atau disebut juga perangkat otak (*brainware*). Orang yang membuat program biasa disebut dengan pemrogram atau *programmer*. Aktivitas yang dilakukan ketika membuat program disebut dengan pemrograman atau *programming/coding*.

Program dibuat dari kumpulan baris kode-kode yang ditulis menggunakan kaidah/aturan bahasa pemrograman tertentu. Bahasa pemrograman memiliki fungsi dan peranannya masing-masing yang berbeda dengan bahasa pemrograman lainnya.

Oleh karena itu, bahasa pemrograman yang digunakan untuk membuat suatu program harus sesuai dengan program seperti apa yang akan dibuat. Sama halnya manusia yang saling berkomunikasi, komputer dapat menjalankan pekerjaannya sesuai dengan instruksi yang mengikuti kaidah/aturan tertentu.

Secara garis besar, bahasa pemrograman dapat dibedakan menjadi dua bagian, yaitu sebagai berikut.

- Bahasa Tingkat Rendah (*Low-Level Language*)

Bahasa tingkat rendah ialah bahasa pemrograman yang berorientasi pada mesin. Bahasa tingkat rendah memberikan perintah ke komputer dengan menggunakan kode biner (0 atau 1). Kode biner ialah kode yang sangat sederhana yang berfungsi untuk menggantikan kode-kode tertentu dalam sistem biner. Bahasa yang berorientasi pada mesin membuatnya menjadi sulit dipahami oleh orang awam, bahkan oleh pemrogram sekalipun. Kelebihan dari bahasa tingkat rendah, yaitu sangat cepat dieksekusi karena dekat dengan mesin, sehingga komputer tidak perlu menerjemahkannya terlalu jauh. Berikut adalah contoh bahasa pemrograman tingkat rendah.

- B402 atau 1011 0100 0000 0010 artinya : masukkan angka 2 ke register AH
 - B22A atau 1011 0010 0010 1010 artinya : muatlah angka A2 hex ke register DL
 - CD21 atau 1100 1101 0010 0001 artinya : jalankan interupsi 21 heksadesimal
- Ketiga perintah di atas akan dieksekusi secara berurutan dan menghasilkan karakter * pada layar.

- Bahasa Tingkat Tinggi (*High-Level Language*)

Bahasa tingkat tinggi ialah bahasa pemrograman yang berorientasi pada bahasa manusia. Bahasa tingkat tinggi dirancang untuk dapat mudah dipahami oleh manusia, termasuk oleh pemula. Contoh bahasa pemrograman tingkat tinggi ialah Java, C, C++, Pascal, Basic, PHP, dan masih banyak lagi. Contoh penulisan kode dalam bahasa pemrograman tingkat tinggi adalah sebagai berikut.

- Write ('*') -- PASCAL B22A
- Display "*" -- COBOL
- PRINT "*" -- BASIC
- Printf ("*") -- C

Semua perintah di atas akan menampilkan hasil yang sama, yaitu menampilkan karakter * pada layar.

Bahasa tingkat tinggi membutuhkan penerjemah atau *translator* untuk menerjemahkan bahasa pemrograman ke bahasa mesin agar kemudian bisa dieksekusi oleh komputer. Berdasarkan urutan kerjanya, *translator* dapat dibedakan menjadi dua bagian, yaitu sebagai berikut.

➤ **Compiler**

Compiler bekerja dengan menerjemah seluruh kode bahasa tingkat tinggi secara utuh terlebih dahulu dalam suatu executable code, yang kemudian dieksekusi oleh mesin. File yang dihasilkan dari *compiler* ialah berupa .exe file. Contoh bahasa pemrograman yang menggunakan *compiler* ialah bahasa C, C++, dan Pascal.

Kelebihan Compiler :

- a) Pengerjaan instruksi dilakukan dengan lebih cepat karena telah diterjemahkan ke dalam bahasa mesin secara utuh.
- b) Kode sumber tidak perlu didistribusikan ke pengguna yang menjalankannya, sehingga kerahasiaannya terjamin.

Kekurangan Compiler :

Kode sumber program harus benar sepenuhnya secara sintak agar program dapat dijalankan.

➤ **Interpreter**

Pada interpreter, setiap baris instruksi bahasa tingkat tinggi diterjemahkan menjadi *intermediate code* per baris, untuk kemudian tiap baris *intermediate code* tersebut dieksekusi oleh mesin. Kekurangan dari interpreter ialah proses eksekusinya lebih lambat. Contoh bahasa pemrograman yang menggunakan interpreter ialah Python, Matlab, dan Java.

Kelebihan Interpreter :

Mudah untuk mencari kesalahan andaikan terdapat kesalahan pada program (debugging). Oleh karena itu, program dapat terus berjalan hingga akhirnya komputer menemukan kesalahan.

Kekurangan Interpreter :

- a) Kode sumber harus selalu tersedia – isu pencurian hak cipta.

b) Proses eksekusi program berjalan dengan lebih lambat.

Tujuan dari dibuatnya program ialah untuk menyelesaikan suatu masalah atau tugas yang ada. Dalam membuat program, ada beberapa tahap yang dikerjakan, yaitu (1) melakukan analisis terhadap masalah yang akan diselesaikan, (2) membuat algoritma dari program yang akan dibuat, (3) mengimplementasikan algoritma ke dalam instruksi program, dan (4) mengeksekusi dan menjalankan program.

Analisis masalah merupakan tahapan yang sangat penting, diperlukannya pengalaman, pengetahuan, imajinasi, kreativitas, dan kecerdasan untuk menganalisis informasi apa yang akan menjadi masukan dan keluaran, serta bagaimana mengolahnya.



Di bawah ini adalah contoh kode program untuk menghitung luas lingkaran.







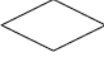


```
#include <stdio.h>

int main ( ) {
    double jari_jari;
    double luas;
    jari_jari = 20;
    luas = 3.14 * jari_jari * jari_jari;
    printf("Luas lingkaran = %lf", luas); return 0;
}
```

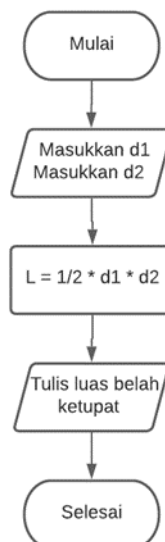
1.3. Flowchart

Flowchart adalah algoritma yang dipresentasikan dalam bentuk diagram alir.

Berikut ini adalah simbol *flowchart* beserta dengan kegunaannya.

SIMBOL	NAMA	FUNGSI
	TERMINATOR	Permulaan/akhir program
	GARIS ALIR (FLOW LINE)	Arah aliran program
	PREPARATION	Proses inisialisasi/ pemberian harga awal
	PROSES	Proses perhitungan/ proses pengolahan data
	INPUT/OUTPUT DATA	Proses input/output data, parameter, informasi
	PREDEFINED PROCESS (SUB PROGRAM)	Permulaan sub program/ proses menjalankan sub program
	DECISION	Perbandingan pernyataan, penyeleksian data yang memberikan pilihan untuk langkah selanjutnya
	ON PAGE CONNECTOR	Penghubung bagian-bagian flowchart yang berada pada satu halaman
	OFF PAGE CONNECTOR	Penghubung bagian-bagian flowchart yang berada pada halaman berbeda

Berikut ini adalah contoh *flowchart* untuk menghitung luas belah ketupat.



2. STRUKTUR DASAR ALGORITMA

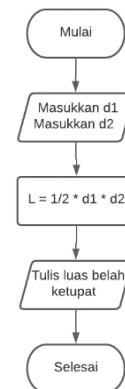
2.1. Runutan (*Sequential*)

Algoritma runutan adalah algoritma yang langkah-langkahnya dikerjakan atau dieksekusi secara berurutan dari awal hingga akhir sesuai dengan urutan program. Setiap instruksi dikerjakan tepat satu kali tanpa ada intruksi yang dikerjakan berulang. Akhir dari instruksi terakhir merupakan akhir dari algoritma.

Contoh *sequential* (1)

Algoritma menghitung luas belah ketupat :

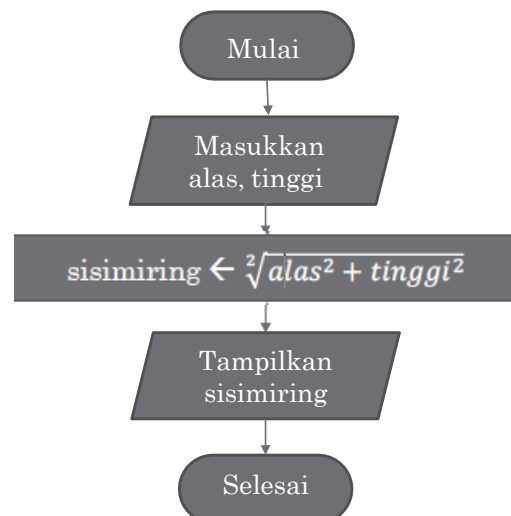
- 1) Masukkan diagonal 1 (d1)
- 2) Masukkan diagonal 2 (d2)
- 3) Hitung luas ($L = 1/2 * d1 * d2$)
- 4) Tulis luas



Contoh *sequential* (2)

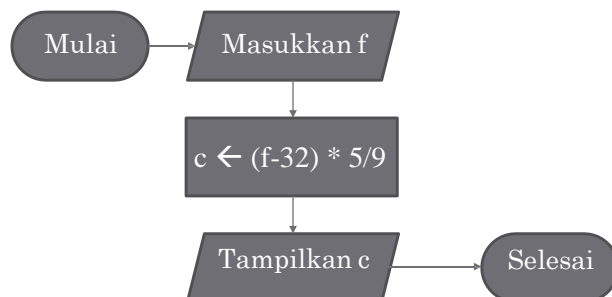
Pseudocode menghitung sisi miring segitiga, dengan asumsi bahwa panjang alas dan tinggi sebuah segitiga siku-siku telah diketahui.

- 1) Masukkan (alas, tinggi)
- 2) $Sisimiring \leftarrow \text{akar pangkat}(\text{alas} * \text{alas} + \text{tinggi} * \text{tinggi})$
- 3) Tampilkan (sisimiring)



Contoh *sequential* (3)

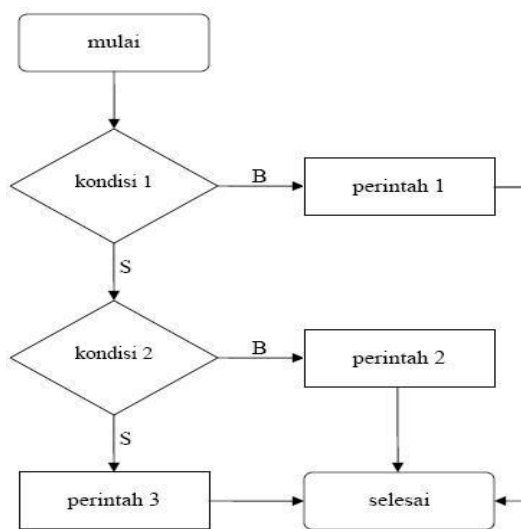
Berikut ini adalah contoh diagram alir untuk menghitung konversi suhu dari Fahrenheit ke Celcius.



2.2. Pemilihan (Selection/Branching)

Struktur pemilihan adalah struktur program yang melakukan proses pengujian untuk mengambil suatu keputusan apakah suatu baris program atau blok kode akan dieksekusi atau tidak. Struktur pemilihan memungkinkan suatu instruksi dijalankan jika suatu kondisi terpenuhi atau tidak terpenuhi. Struktur pemilihan mampu memungkinkan pemroses mengikuti jalur aksi yang berbeda berdasarkan kondisi yang ada. Hanya instruksi program yang terpenuhi saja yang akan dijalankan, sehingga tidak semua baris kode akan dijalankan.

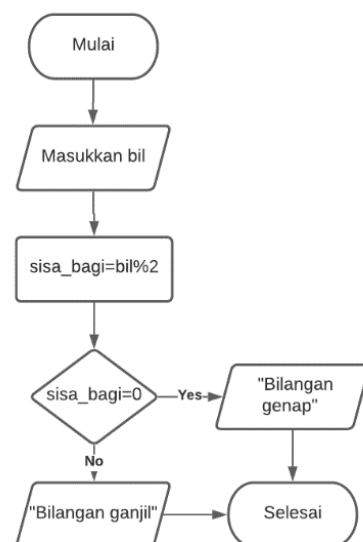
Contoh diagram alir pemilihan biasa secara umum.



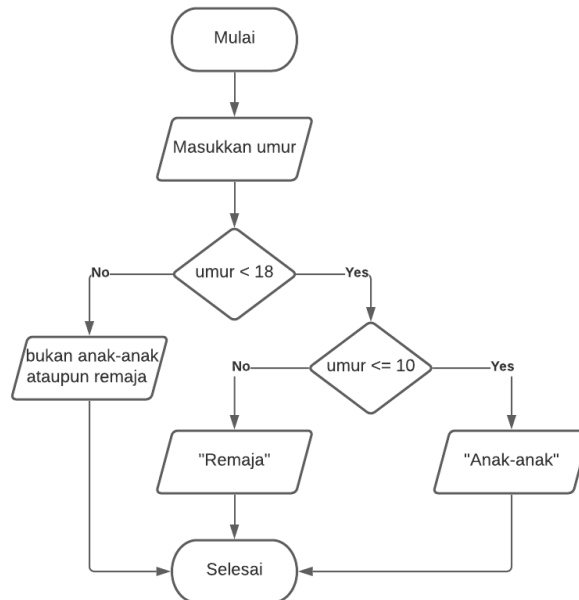
Berikut ini adalah contoh diagram alir pemilihan/percabangan untuk menentukan bilangan ganjil atau genap.

Algoritma :

- Masukkan bilangan.
- Bilangan dimodulus 2 untuk mencari sisa baginya.
- Apabila sisa baginya sama dengan nol, maka bilangan tersebut adalah bilangan genap. Namun, jika sisa baginya tidak sama dengan nol, berarti bilangan tersebut adalah bilangan ganjil.



Pemilihan bersarang adalah seleksi bertingkat. Seleksi ini dilakukan apabila ingin menyeleksi kondisi secara lebih detail, sampai suatu kondisi yang dikehendaki tercapai. Contoh diagram alir pemilihan bersarang adalah sebagai berikut.



2.3. Pengulangan (*Looping*)

Perulangan menyatakan suatu tindakan atau langkah yang dijalankan beberapa kali. Perulangan adalah instruksi yang dapat mengulang sederet instruksi secara berulang-ulang sesuai persyaratan yang ditetapkan. Sebagai contoh, jika ingin menampilkan tulisan “Selamat Belajar” sebanyak 10 kali, maka caranya adalah sebagai berikut.

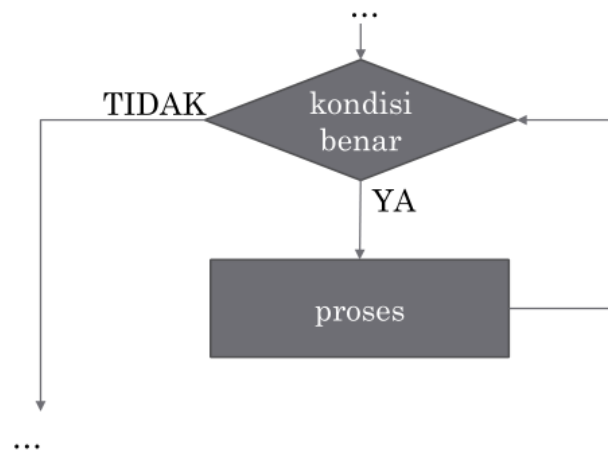
Cara pertama :

```
printf("Selamat Belajar");
printf("Selamat Belajar");
...
printf("Selamat Belajar");
// hingga 10 kali.
```

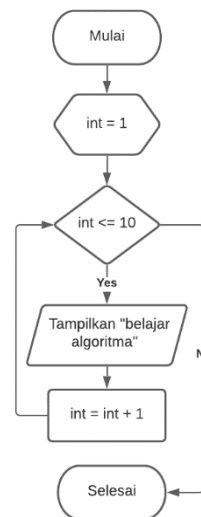
Cara kedua (struktur perulangan) :

```
for (i=1; i<=10; i++)
{
    printf("Selamat Belajar");
}
```

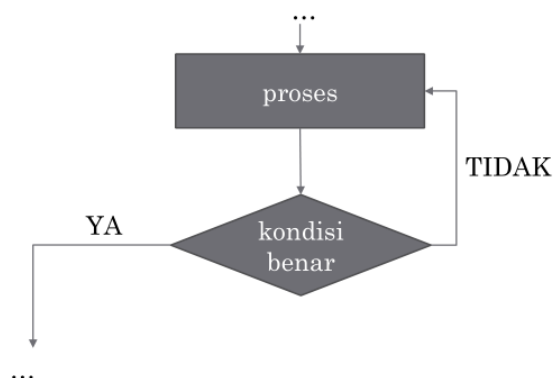
Pada struktur perulangan, proses dapat dilakukan dengan satu atau beberapa langkah. Pada bentuk ini, proses akan dijalankan berulang-ulang selama suatu kondisi benar terjadi.



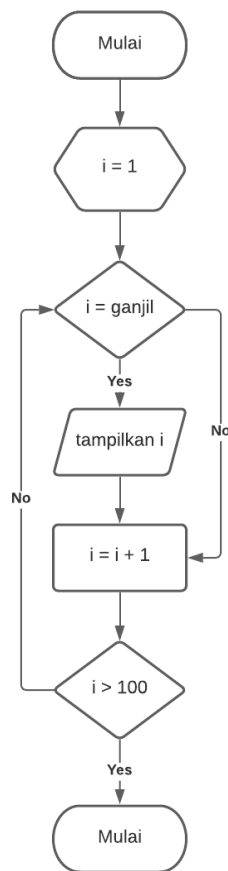
Pada diagram alir di samping, menunjukkan perulangan untuk menampilkan “Belajar Algoritma” sebanyak 10 kali.



Selanjutnya, ada pula perulangan yang akan menjalankan instruksi selama kondisinya bernilai salah (*false*). Perulangan tersebut memiliki bentuk seperti pada gambar di bawah ini.



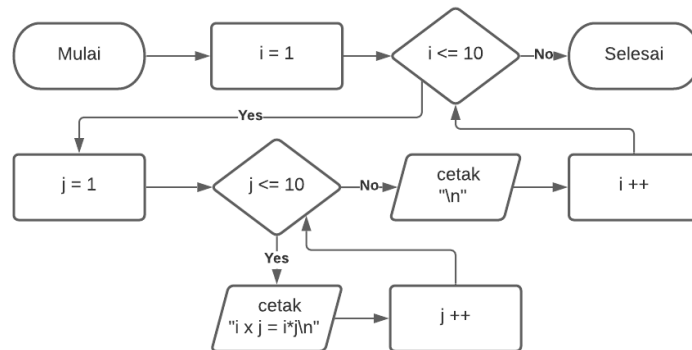
Contohnya, diagram alir di bawah ini menunjukkan perulangan yang menambahkan 1 angka ke sebuah variabel apabila kondisinya bernilai salah (*false*).



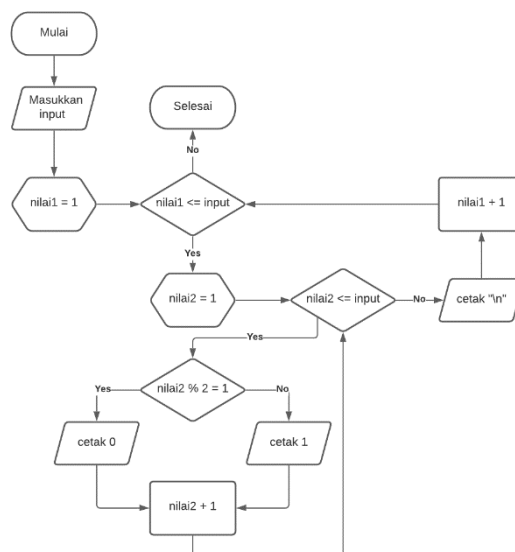
3. KOMBINASI STRUKTUR DASAR ALGORITMA

Algoritma memiliki 3 (tiga) struktur dasar, yang sebelumnya telah dibahas pada subbab sebelumnya. Ketiga struktur dasar tersebut bisa dikombinasikan dengan struktur dasar yang lainnya. Untuk membuat diagram alir dari suatu program yang kompleks, sering kali dua atau bahkan tiga struktur dasar yang berbeda digunakan secara bersamaan dalam satu diagram alir yang sama.

Diagram alir di bawah ini menunjukkan program untuk membuat tabel perkalian, yang menggabungkan pemilihan dengan pengulangan. Hasil yang ditampilkan merupakan perkalian 1 hingga perkalian 10.



Ada pula diagram alir yang menggabungkan ketiga struktur dasar algoritma, yaitu runutan, pemilihan, dan pengulangan. Diagram alir di bawah ini menunjukkan program yang akan menampilkan angka 0 jika nilai yang dimasukkan *user* adalah ganjil dan menampilkan angka 1 jika nilai yang dimasukkan *user* adalah genap.



9.1. Pseudocode

Pseudocode atau disebut juga kode semu merupakan deskripsi tingkat tinggi informal dan ringkas atas algoritma pemrograman komputer yang menggunakan konvensi struktural (suatu bahasa pemrograman) dan ditujukan untuk dibaca oleh manusia dan bukan oleh mesin.

Format penulisan *pseudocode* dibagi menjadi 3 (tiga) bagian, yaitu sebagai berikut.

1) Judul

Bagian judul diawali dengan kata “PROGRAM” dan diikuti dengan nama algoritma. Umumnya, nama algoritma hanya terdiri dari satu kata. Jika terdapat lebih dari satu kata, maka penulisan nama algoritma dapat ditulis dengan menghilangkan spasi atau menggantinya seperti PROGRAM HitungSisiMiring, PROGRAM Pencarian_Angka, dan sebagainya.

2) Deklarasi

Bagian deklarasi digunakan untuk mendeklarasikan nama variabel beserta dengan tipe datanya. Tipe data pada aplikasi komputer, yaitu bilangan bulat, bilangan pecahan, teks, tanggal, dan sebagainya. Format penulisan deklarasi variabel diawali dengan tipe data dan diikuti dengan nama variabelnya.

3) Isi

Bagian isi merupakan bagian utama dari jalannya algoritma. Bagian yang berisi perintah-perintah sesuai dengan logika algoritma. Perintah tersebut dapat berupa runutan, pemilihan, pengulangan, dan/atau kombinasinya.

Contoh penulisan algoritma dalam bentuk *pseudocode* untuk algoritma pencarian sisi miring segitiga adalah sebagai berikut.

PROGRAM CariSisiMiring

DEKLARASI

int a, b

float c

ALGORITMA

read(a, b)

$c = \sqrt{a^2 + b^2}$

write("Sisi miring: ", c)

Penjelasan *pseudocode* :

Judul dari algoritma di atas adalah CariSisiMiring yang dituliskan di awal *pseudocode*. Kemudian pada bagian deklarasi, terdapat 3 buah variabel, yaitu a, b dan c. Variabel a dan b dideklarasikan dengan tipe data int, yaitu bilangan bulat, sedangkan variabel c dideklarasikan dengan tipe data float, yaitu bilangan pecahan. Pada bagian isi, terdapat 3 buah perintah, yaitu sebagai berikut.

- 1) Perintah `read(a, b)`, yang digunakan untuk meminta masukan dari pengguna. Pada aplikasi yang telah jadi, yang memberikan masukan adalah pengguna aplikasi. Masukan berupa 2 buah angka dapat dimasukkan melalui papan kunci.
- 2) Perintah `c = sqrt(a*a + b*b)` adalah perintah untuk memberikan nilai variabel c. Nilai variabel c didapatkan dari perhitungan akar kuadrat dari a kuadrat ditambah b kuadrat.
- 3) Perintah `write("Sisi miring: ", c)` merupakan perintah untuk menampilkan teks "Sisi miring: " diikuti dengan isi dari variabel c. Variabel c dicetak di luar tanda petik sebagai penanda bahwa yang dicetak bukan teks "c" melainkan isi dari variabel c.

4. RAPTOR

Raptor merupakan pemrograman yang berbasis *flowchart*, *raptor* dirancang khusus untuk membantu memvisualisasikan algoritma yang telah kita buat. Program *raptor* diciptakan secara visual dan dieksekusi secara visual dengan menelusuri eksekusi melalui *flowchart*. Biasanya kita lebih suka menggunakan *flowchart* untuk mengekspresikan algoritma, dan lebih berhasil menciptakan algoritma menggunakan *raptor* daripada menggunakan bahasa tradisional atau menulis *flowchart* tanpa *raptor*.

Raptor ditulis dalam kombinasi dari bahasa C# dan A# (*port* dari Ada untuk .NET Framework.) dan hanya didukung pada Windows. *Raptor* telah bereksperimen dengan Mono di Mac OS X dan Ubuntu. Versi Mac tidak berjalan sama sekali, tetapi *raptor* dapat berjalan pada Ubuntu dengan beberapa fitur yang harus dihilangkan.

Raptor memiliki beberapa mode, secara default kita memakai mode *Novice*. Mode *Novice* memiliki global *namespace* tunggal untuk setiap variabel. Mode *Intermediate* digunakan untuk membuat prosedur yang memiliki ruang lingkup mereka sendiri (memperkenalkan gagasan lewat parameter dan mendukung rekurs). Mode baru *raptor* adalah mode yang berorientasi objek, yaitu versi 2009.

Raptor bebas untuk didistribusikan sebagai layanan kepada masyarakat. *raptor* pada awalnya dikembangkan oleh dan untuk US Air Force Academy, Departemen Ilmu Komputer, namun penggunaannya telah menyebar dan *raptor* sekarang digunakan untuk pendidikan di lebih 17 negara pada setidaknya 4 benua.

Raptor juga dilengkapi dengan proses *generate flowchart* ke beberapa sumber kode yang sudah banyak di kenal seperti C++, Java, C# dan lain-lain. Sehingga pengguna tidak perlu lagi membangun dari awal sebuah sumber kode, karena dari *flowchart* yang telah di buat langsung di terjemahkan ke sumber kode oleh *raptor*.

Keunggulan dari perangkat lunak Raptor Interpreter Flowchart adalah dapat mengeksekusi *flowchart* yang telah di bangun menjadi sebuah visualisasi yang nyata, sehingga pengguna dapat mengetahui setiap tahapan *flowchart* yang mereka buat melalui eksekusi secara visual dalam tiap langkahnya. Selain itu kelebihan dari Raptor Interpreter Flowchart ini ada pada saat kita membuat *flowchart* dengan penulisan variabel, *raptor* di dukung dengan adanya fitur *auto complete* seperti layaknya pada Pemrograman Visual Basic.

Operator memerintahkan komputer untuk melakukan beberapa perhitungan

pada data. Operator ditempatkan antara data yang dioperasikan (yaitu $X / 3$, $Y + 7$, dan lain-lain), sedangkan fungsi menggunakan tanda kurung untuk menunjukkan data tersebut beroperasi pada (yaitu $\text{sqrt}(4.7)$, $\text{sin}(2,9)$). Ketika dieksekusi, operator dan fungsi melakukan perhitungan dan mengembalikan hasil. RAPTOR memiliki operator dan fungsi sebagai berikut.

Basic math : +, -, *, /, ^, **, rem, mod, sqrt, log, abs, ceiling, floor.

Trigonometry : sin, cos, tan, cot, arcsin, arcos, arctan, arccot.

Relational : =, !=, /, <, >, >=, <=.

Logical : and, or, not.

Miscellaneous : random, Length_of.

Operator matematika dasar dan fungsi yang termasuk biasa (+, -, *, /) serta beberapa yang tidak biasa “**” dan “^” adalah *exponentiation*, seperti $2 ** 4$ adalah 16 dan $3 ^ 2$ adalah 9. Rem (*remainder*) dan mod (*modulus*) mengembalikan sisa (apa yang tersisa) ketika operan kanan membagi operan kiri, contoh : $10 \text{ rem } 3$ adalah 1, $10 \text{ mod } 3$ adalah 1 juga.

sqrt mengembalikan akar kuadrat contoh : $\text{sqrt}(4)$ adalah 2.

log mengembalikan logaritma natural contoh : $\text{log}(e)$ adalah 1.

abs mengembalikan nilai absolut contoh : $\text{abs}(-9)$ adalah 9.

ceiling pada seluruh nomor contoh : $\text{ceiling}(3,14159)$ adalah 4.

floor pada seluruh nomor contoh : $\text{floor}(10/3)$ adalah 3.

Tanda “+” juga bekerja sebagai operasi *concatenation* untuk menggabungkan dua *string* atau *string* dan angka, contoh : “rata-rata adalah” + (Jumlah / Angka). *length_of* mengembalikan jumlah karakter dalam sebuah variabel *string* (juga jumlah elemen dengan sebuah *array*), contoh : Nama ← “Stuff” diikuti dengan *Length_Of* (Nama) adalah 5.

Kita terbiasa dengan fungsi trigonometri (sin, cos, tan, cot, arcsin, arcos, arctan, arccot). Mereka bekerja pada berbagai unit yang bernilai radian. (kita harus mengonversi dari derajat ke radian sebelum menggunakan fungsi tersebut.). arctan dan arccot adalah versi kedua parameter fungsi ini. (yaitu $\text{arctan}(X / Y)$ ditulis dalam RAPTOR sebagai $\text{arctan}(X, Y)$).

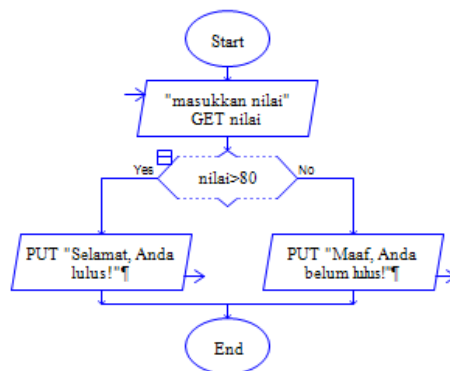
Dalam *raptor*, operator relasional dan operator logika hanya dapat digunakan dalam pengambilan keputusan sebagai bagian dari pemilihan dan pengulangan.

Operator relasional adalah $!=$ (tidak sama dengan), $/=$ (tidak sama dengan), $<$, $>$, $>=$, dan $<=$. Operator relasional mengembalikan nilai “Boolean” dalam “True” atau “False” (ya atau tidak). Sebagai contoh, operasi $X < Y$ akan mengembalikan TRUE jika nilai yang tersimpan dalam variabel X kurang dari nilai yang disimpan dalam variabel Y. Jika tidak, maka nilai FALSE dikembalikan. Hasil dari operasi relasional dapat digunakan oleh operator logika.

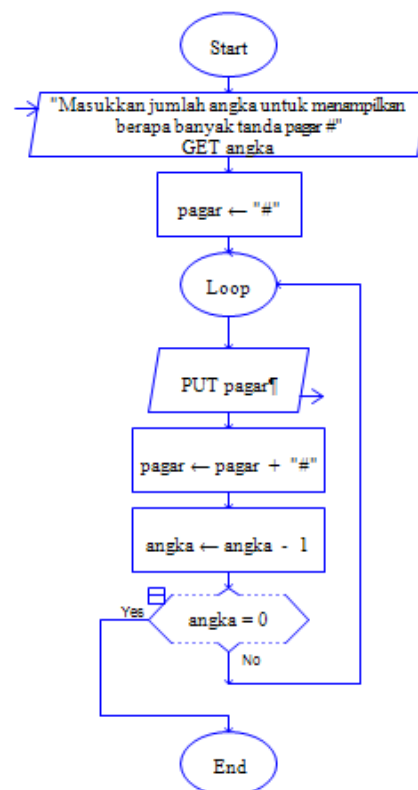
Fungsi secara acak mengembalikan angka antara 0 dan 1, contoh : $X \leftarrow$ secara acak bisa menjadi 0, 0,23, 0,46578, dll. Jika kita memerlukan nomor acak dalam kisaran yang berbeda maka kita bisa menggabungkan fungsi acak dengan operasi lain. Misalnya, $\text{random} * 100$ akan mengevaluasi ke angka antara 0 dan 100. $\text{ceiling}(\text{random} * 100)$ akan mengevaluasi ke seluruh nomor antara 1 dan 100.

Berikut ini adalah contoh *flowchart* yang dibuat menggunakan raptor.

- 1) Program menentukan kelulusan berdasarkan nilai.



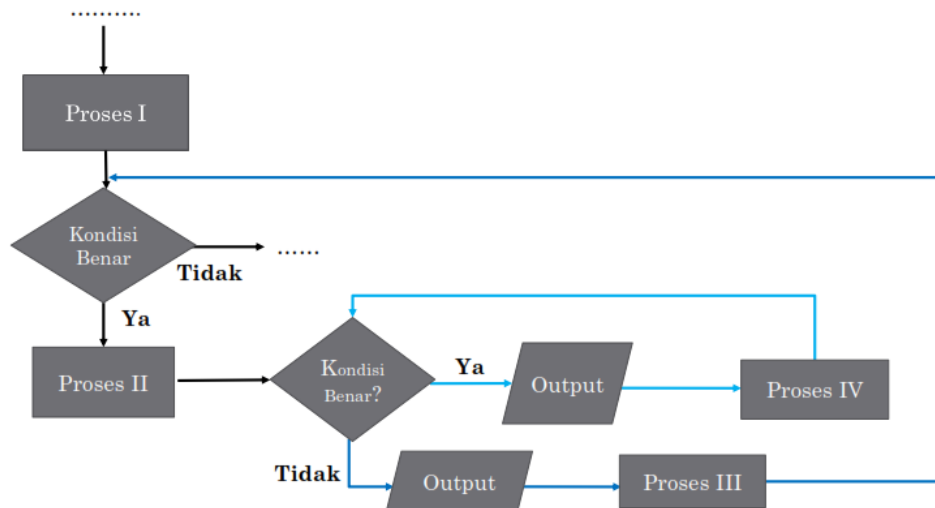
- 2) Program menampilkan tanda # (pagar) sebanyak angka yang dimasukkan dapat dilihat seperti gambar di samping.



5. PERULANGAN BERSARANG

Perulangan bersarang merupakan perulangan yang memiliki perulangan lagi di dalamnya. Perulangan For bersarang adalah perulangan For yang berada pada perulangan For lainnya. Perulangan yang lebih dalam akan diproses terlebih dahulu sampai habis, kemudian perulangan yang lebih luar baru akan bertambah, mengerjakan perulangan yang lebih dalam lagi mulai dari nilai awalnya dan seterusnya. Contoh perulangan bersarang dalam kehidupan sehari-hari bisa berupa pergi ke sekolah, bekerja, memasak, dan lain sebagainya.

Secara umum, struktur perulangan bersarang dapat digambarkan sebagai berikut.



Salah satu contoh program perulangan bersarang, ialah membuat segitiga siku-siku yang disusun dari karakter * (bintang), dengan tinggi segitiga ditentukan melalui papan kunci.

*	*	*
**	**	**
***	***	***
****	****	****
Tinggi 4	*****	*****
	Tinggi 5	*****
		Tinggi 6

Algoritma membuat segitiga siku-siku dengan karakter * (bintang).

1) Tentukan Variabel

Tinggi : Tinggi Segitiga

i : variabel pencacah 1 (jumlah baris)

j : variabel pencacah 2 (jumlah kolom)

2) Input Nilai untuk menentukan jumlah baris bintang * (Tinggi Segitiga)

3) Proses perulangan untuk menghitung jumlah bintang *

for i = 1 to i <= Tinggi

for j = 1 to j <= i

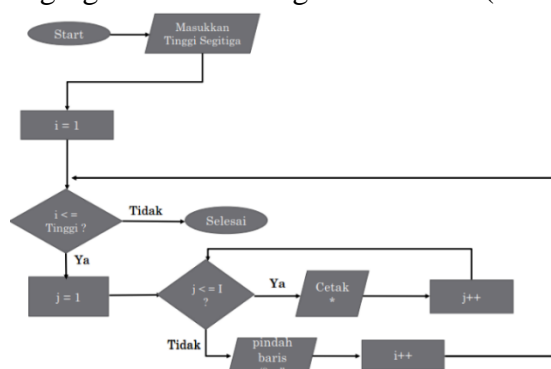
4) Cetak Hasil *

Sumber kode membuat segitiga siku-siku dengan karakter * (bintang) :

```
#include <stdio.h>

main ( )
{
    int i, j, tinggi;
    printf("Masukkan tinggi segitiga : ");
    scanf("%d", &tinggi);
    for(i=1; i <= tinggi; i++)
    {
        for(j=1; j <= i; j++)
            printf("*");
        printf("\n");
    }
}
```

Flowchart membuat segitiga siku-siku dengan karakter * (bintang).



6. KONSEP LARIK

Larik (*array*) adalah kumpulan dari nilai data-data yang bertipe sama dalam urutan tertentu yang menggunakan nama yang sama. Larik menggunakan notasi [] (kurung siku) untuk menyatakan data dalam larik. Sebagai contoh berikut *array* A

A[0]	A[1]	A[2]	A[3]
3	1	2	3

Cara membaca *array* A, yaitu (1) Array A indeks 0 datanya adalah 3. (2) Array A indeks 1 datanya adalah 1. (3) Array A indeks 2 datanya adalah 2. (4) Array A indeks 3 datanya adalah 3. Apabila memberi nilai A[1] = 4; artinya memperbarui indeks ke-1 dengan nilai baru, yaitu 4. Menampilkan nilai printf(“%d”, A[3]); artinya mencetak data indeks ke-3 ke layar, yaitu 3.

Array dapat dideklarasikan dengan mengikuti aturan, yaitu *tipe_data nama_array [jumlah elemen]*. Contoh mendeklarasikan *array* ialah sebagai berikut.

- int cacah[4]; yang berarti *array* dengan nama cacah memiliki 4 buah elemen dan tipe datanya adalah int (bilangan bulat)
- char vokal[5]; yang berarti *array* dengan nama vokal memiliki 5 buah elemen dan tipe datanya adalah char (karakter). Selain itu, *array* ini bisa juga tipe datanya adalah string dengan jumlah 4 buah elemen. Pada string, karakter terakhir digunakan untuk menyimpan karakter penutup ‘\0’.
- char kota[6][20]; yang berarti *array* dengan nama kota memiliki 6 buah elemen dan masing-masing elemen adalah string dengan panjang maksimal 19 karakter.

Di bawah ini adalah contoh algoritma yang menyimpan huruf vokal di dalam larik, kemudian menampilkan isi larik tersebut.

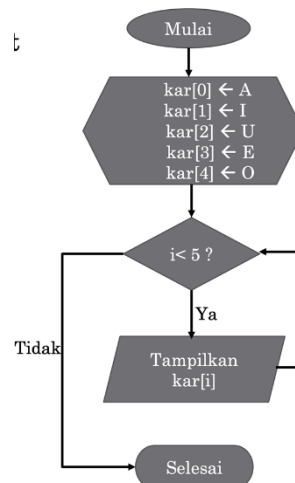
1) kar[] [“A”, ”I”, ”U”, ”E”, ”O”]

- kar[0] □ ‘A’;
- kar[1] □ ‘I’;
- kar[2] □ ‘U’;
- kar[3] □ ‘E’;
- kar[4] □ ‘O’;

2) untuk i □ 0 s/d 4

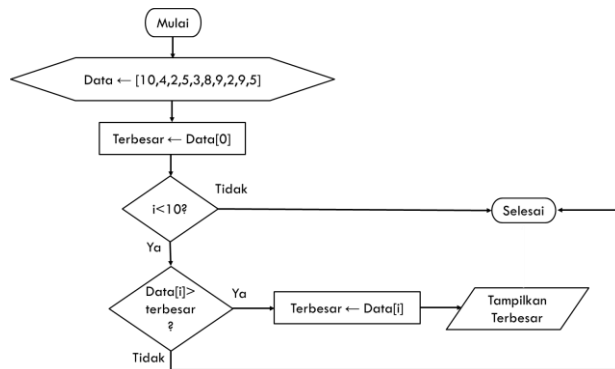
3) tampilkan (kar[i])

4) akhir untuk



Di bawah ini algoritma untuk mencari bilangan terbesar dalam suatu larik.

- 1) Data \leftarrow [10 4 2 5 3 8 9 2 9 5]
- 2) Terbesar \leftarrow Data[0]
- 3) Untuk i \leftarrow 1 s/d 9
- 4) Jika Data[i] > terbesar Maka
- 5) Terbesar \leftarrow Data[i]
- 6) Akhir jika
- 7) Akhir untuk
- 8) Tampilkan (terbesar)



Pada *array* 2 (dua) dimensi, setiap elemen diakses melalui dua buah indeks, yaitu indeks baris dan indeks kolom.

		Indeks Kolom		
		0	1	2
Indeks Baris	0			
	1			
	2			
	3			

Larik pada contoh di samping, dibentuk melalui pernyataan `int [4][3]`. 4 menyatakan jumlah baris dan 3 menyatakan jumlah kolom.

Di bawah ini algoritma untuk menyimpan nama negara dan ibu kota, kemudian menampilkannya dengan pasangan yang benar.

- 1) Negara [0,0] □ “Indonesia”
- 2) Negara [0,1] □ “Jakarta”
- 3) Negara [1,0] □ “Filipina”
- 4) Negara [1,1] □ “Manila”
- 5) Negara [2,0] □ “Austria”
- 6) Negara [2,1] □ “Vienna”
- 7) Negara [3,0] □ “India”
- 8) Negara [3,1] □ “New Delhi”
- 9) Negara [4,0] □ “Iran”
- 10) Negara [4,1] □ “Teheran”
- 11) UNTUK baris □ 0 S/D 4

JIKA Negara [baris,kolom][0]= “I” MAKA

Tampilkan (Negara[baris,0],Negara[baris,1])

AKHIR JIKA

AKHIR UNTUK

Array multidemnsi adalah *array* yang masih memiliki indeks di dalam indeks. Contohnya, kita akan menyimpan nama negara dan ibu kota beberapa negara, dan semuanya memiliki tipe data *string*. Kita memerlukan *array* 3 dimensi untuk melakukan itu, dimensi pertama untuk pemisah antar data, dimensi kedua untuk pemisahan nama negara dan ibukota, dan dimensi ketiga untuk menyimpan datanya (teks).

Contoh program *array multidimensi*.

```
#include <stdio.h>.  
#include<string.h> // Menyajikan negara yang hanya berawalan huruf I saja  
main ( ) {  
    int baris;  
    char negara[5][2][15];  
    strcpy(negara[0][0], "Indonesia");  
    strcpy(negara[0][1], "Jakarta");  
    strcpy(negara[1][0], "Filipina");  
    strcpy(negara[1][1], "Manila");  
    strcpy(negara[2][0], "Austria");  
    strcpy(negara[2][1], "Wina");  
    strcpy(negara[3][0], "India");  
    strcpy(negara[3][1], "New Delhi");  
    for (baris = 0; baris < 5; baris++) {  
        if(negara[baris][0][0] == 'I')  
            printf("%s - %s\n", negara[baris][0], negara[baris][1]);  
    }  
}
```

7. SUBPROGRAM

Dalam menulis program, sering kali kita harus mengetikkan instruksi yang persis sama beberapa kali. Jika itu adalah instruksi yang mudah seperti *print*, maka tidak ada persoalan dalam hal penulisannya. Akan tetapi, penulisannya menjadi suatu masalah apabila yang harus ditulis ulang tersebut cukup banyak, apalagi jika disertai dengan fungsi-fungsi khusus. Tujuan penggunaan *subprogram*, yaitu untuk memudahkan pengelolaan atau pengembangan program dan juga untuk mengurangi jumlah kode pada program yang besar.

Sesuai dengan namanya, *subprogram* adalah satu bagian program yang bisa dikatakan terpisah dari program utamanya. Struktur program yang demikian disebut dengan struktur modular. Tujuan lain dari penulisan *subprogram* adalah dalam hal kemudahan pelacakan dan pembacaan program tersebut. Program tersusun atas modul-modul, di mana setiap modul biasanya tidak terlalu panjang. Jika dalam sebuah modul terjadi kesalahan, kita tidak perlu melacak seluruh program, tetapi cukup kita lihat dari modul di mana kesalahan terjadi.

Subprogram dapat ditulis dengan mengikuti aturan sebagai berikut.

SUBRUTIN NamaSubrutin(daftar-parameter)

 Pernyataan-1

 ...

 Pernyataan-n

AKHIR-SUBRUTIN

Dalam hal ini, bagian SUBRUTIN NamaSubrutin(daftar-parameter) disebut dengan judul *subprogram*. Sebuah *subprogram* dapat memberikan nilai balik ataupun tidak. Nilai balik adalah nilai yang diberikan ke pemanggilannya. Nilai ini ditentukan melalui notasi seperti berikut : NILAI-BALIK nilai

Contoh :

 SUBRUTIN hitung_keliling_kotak(panjang.lebar)

 Keliling $\leftarrow 2 \times (\text{panjang} + \text{lebar})$

 NILAI-BALIK Keliling

 AKHIR-SUBRUTIN

Penjelasan :

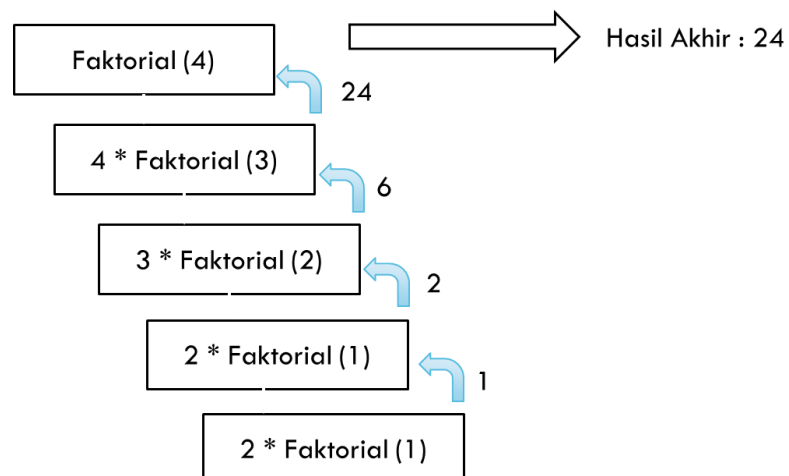
hitung_keliling_kotak adalah nama *subprogram*. Adapun *panjang* serta *lebar* disebut sebagai parameter. Parameter menyatakan bagian untuk berkomunikasi dengan pemanggil *subprogram*. Pada bagian pemanggilan *subprogram*, bagian ini akan di isi dengan argumen. Contoh : hasil \leftarrow hitung_keliling_kotak(10,5)

Pada pemanggilan subrutin di atas, 10 dan 5 berkedudukan sebagai argumen, maka hasil akhirnya adalah 30.

8. REKURS

Rekurs adalah suatu kemampuan dari *subprogram* untuk memanggil dirinya sendiri. Kemampuan untuk memanggil diri sendiri tersebut akan sangat berguna pada beberapa kondisi atau persoalan karena dapat mempermudah solusi. Kelemahan rekurs adalah ketika terjadi *stack overflow* (*stack* tidak mampu lagi menangani permintaan pemanggilan rekurs karena telah kehabisan memori). *Stack* adalah area memori (RAM) yang dipakai untuk variabel lokal dan untuk pengalokasian memori ketika suatu *subprogram* dipanggil. Cara menangani program tersebut, yaitu pemrogram harus memastikan bahwa proses rekurs akan selesai pada suatu titik/kondisi tertentu.

Contoh rekurs dapat kita temui pada perhitungan matematika seperti menghitung faktorial. Contohnya, faktorial 4 dapat diperoleh dengan cara berikut.



Perhitungan tersebut dapat diimplementasikan ke dalam program, yaitu sebagai berikut.

```
#include <stdio.h>
long int faktorial(int n) {
    if(n == 0 || n == 1);
        return 1;
    } else {
        return n* faktorial (n-1);
    }
}
int main () {
    int n;
    long int hasil;
    printf("n= ");
    scanf("%d",&n);
    hasil = faktorial (n);
    printf("%d! = %ld",n, hasil);
    return 0;
}
```

Algoritma *subprogram* faktorial :

Subrutin faktorial (n)

Jika n = 0 atau 1 maka

Nilai-Balik 1

Sebaliknya

Nilai-Balik n x faktorial (n-1)

Akhir-Jika

Akhir Subrutin

Algoritma Program Utama (Fungsi main) :

- 1) Inisialisasi variabel n (bilanganfaktorial) , hasil (hasil dari faktorial bilangan)
- 2) Meminta masukan/input bilangan faktorial dari pengguna
- 3) Memanggil fungsi faktorial, simpan dalam variabel hasil
- 4) Tampilkan hasil

Contoh rekurs *fibonacci* :

Fungsi *fibonacci* dapat dinyatakan dalam bentuk rekurs seperti berikut ini.

$\text{fib}(n) = 0$, untuk $n = 0$

$\text{fib}(n) = 1$, untuk $n = 1$

$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$, untuk $n > 1$

Fibonacci :

$n = 0 \rightarrow \text{fib}(n) = 0$ (bilangan pertama 0)
 $n = 1 \rightarrow \text{fib}(n) = 1$ (bilangan kedua 1)
 $n = 2 \rightarrow \text{fib}(n) = 1$ (bilangan ke tiga $0+1=1$)
 $n = 3 \rightarrow \text{fib}(n) = 2$ (bilangan ke empat $1+1=2$)
 $n = 4 \rightarrow \text{fib}(n) = 3$ (bilangan ke lima $1+2=3$)
 $n = 5 \rightarrow \text{fib}(n) = 5$ (bilangan ke enam $2+3=5$)
 $n = 6 \rightarrow \text{fib}(n) = 8$ (bilangan ke tujuh $3+5=8$)
 $n = 7 \rightarrow \text{fib}(n) = 13$ (bilangan ke delapan $5+8=13$)
 $n = 8 \rightarrow \text{fib}(n) = 21$ (bilangan ke Sembilan $8+13=21$)
dst...

Implementasi rekurs *fibonacci*

```
#include <stdio.h>
long int fib(unsigned int n) {
    if(n == 0) return 0;
    else if (n == 1) return 1;
    else return fib(n-1)+fib(n-2);
}
int main () {
    int n;
    long int hasil;
    printf("n= ");
    scanf("%d",&n);
    hasil = fib(n);
    printf("fib(%d) = %ld",n, hasil);
    return 0;
}
```

Algoritma *subprogram fibonacci* :

Subrutin fib(n)

Jika $n = 0$ maka

Nilai-Balik 0

Sebaliknya

jika $n = 1$ maka

Nilai-Balik 1

Sebaliknya

Nilai-Balik $\text{fib}(n-1)+\text{fib}(n-2)$

Akhir Jika

Akhir-Jika

Akhir Subrutin

Algoritma Fungsi Utama (Fungsi main) :

- 1) Inisialisasi variabel n (untuk nilai bilangan Fibonacci, hasil (untuk menyimpan hasil))
- 2) Meminta masukan/input n dari pengguna
- 3) Memanggil fungsi fib dan simpan dalam variabel hasil
- 4) Tampilkan hasil

9. PENCARIAN DATA DAN PENGURUTAN DATA

9.1. Pencarian Data

Pencarian data (*searching*) sering disebut juga *table look-up* atau *storage and retrieval information* adalah suatu proses untuk mengumpulkan sejumlah informasi di dalam memori komputer dan kemudian mencari kembali informasi yang diperlukan secepat mungkin.

Dalam kehidupan sehari-hari sebenarnya kita sering melakukan pencarian data. Sebagai contoh, saat kita mencari baju yang hendak kita gunakan dari dalam lemari. Contoh lain ketika kita mencari minum dari dalam kulkas dan masih banyak lagi contohnya.

Algoritma pencarian (*searching algorithm*) adalah algoritma yang menerima sebuah argumen kunci dan dengan langkah-langkah tertentu akan mencari rekaman dengan kunci tersebut. Setelah proses pencarian dilakukan, akan diperoleh salah satu dari dua kemungkinan, yaitu data yang dicari ditemukan (*successfull*) atau tidak ditemukan (*unsuccessfull*).

Metode pencarian data dapat dilakukan dengan dua cara, yaitu pencarian internal dan pencarian eksternal. Pada pencarian internal, semua rekaman yang diketahui berada dalam memori komputer sedangkan pada pencarian eksternal, tidak semua rekaman yang diketahui berada dalam pengingat komputer, tetapi ada sejumlah rekaman yang tersimpan dalam penyimpanan luar, misalnya pita atau cakram magnetis.

Selain itu, metode pencarian data juga dapat dikelompokkan menjadi pencarian statis dan pencarian dinamis. Pada pencarian statis, banyaknya rekaman yang diketahui dianggap tetap, pada pencarian dinamis, banyaknya rekaman yang diketahui bisa berubah-ubah yang disebabkan oleh penambahan dan penghapusan suatu rekaman.

Ada macam-macam teknik pencarian, yaitu pencarian sekuensial dan pencarian biner. Perbedaan dari dua teknik ini terletak pada keadaan data. Pencarian sekuensial digunakan apabila data dalam keadaan acak atau tidak terurut. Sebaliknya, pencarian biner digunakan pada data yang sudah dalam keadaan urut dan tambahannya yaitu pencarian beruntun dengan sentinel jika pencarian bertujuan untuk menambahkan elemen baru setelah elemen terakhir larik, maka terdapat sebuah varian dari metode pencarian beruntun yang mangkus.

a) Pencarian Linear atau Sekuensial

Pencarian berurutan sering disebut pencarian linear merupakan metode pencarian yang paling sederhana. Pencarian berurutan menggunakan prinsip sebagai berikut : data yang ada dibandingkan satu per satu secara berurutan dengan yang dicari sampai data tersebut ditemukan atau tidak ditemukan.

Pada dasarnya, pencarian ini hanya melakukan pengulangan dari 1 sampai dengan jumlah data. Pada setiap pengulangan, dibandingkan data ke-i dengan yang dicari. Apabila sama, berarti data telah ditemukan. Sebaliknya apabila sampai akhir pengulangan tidak ada data yang sama, berarti data tidak ada. Pada kasus yang paling buruk, untuk n elemen data harus dilakukan pencarian sebanyak n kali pula.

b) Pencarian Biner

Salah satu syarat agar pencarian biner dapat dilakukan adalah data sudah dalam keadaan urut. Dengan kata lain, apabila data belum dalam keadaan urut, pencarian biner tidak dapat dilakukan. Dalam kehidupan sehari-hari, sebenarnya kita juga sering menggunakan pencarian biner. Misalnya saat ingin mencari suatu kata dalam kamus.

Prinsip dari pencarian biner dapat dijelaskan sebagai berikut : mula-mula diambil posisi awal 0 dan posisi akhir = $n - 1$, kemudian dicari posisi data tengah dengan rumus $(\text{posisi awal} + \text{posisi akhir}) / 2$. Kemudian data yang dicari dibandingkan dengan data tengah. Jika lebih kecil, proses dilakukan kembali tetapi posisi akhir dianggap sama dengan posisi tengah $- 1$. Jika lebih besar, proses dilakukan kembali tetapi posisi awal dianggap sama dengan posisi tengah $+ 1$. Demikian seterusnya sampai data tengah sama dengan yang dicari.

c) Pencarian Beruntun Dengan Sentinel

Jika pencarian bertujuan untuk menambahkan elemen baru setelah elemen terakhir larik, maka terdapat sebuah varian dari metode pencarian beruntun yang mangkus. Nilai x yang akan dicari sengaja ditambahkan terlebih dahulu. Data yang ditambahkan setelah elemen terakhir larik ini disebut sentinel.

9.2. Pengurutan Data

Sorting atau pengurutan merupakan proses yang sering kali harus dilakukan dalam pengolahan data. *Sort* dalam hal ini diartikan mengurutkan data yang berada dalam suatu tempat penyimpanan, dengan urutan tertentu baik urut menaik (*ascending*) dari nilai terkecil sampai dengan nilai terbesar, atau urut menurun (*descending*) dari nilai terbesar sampai dengan nilai terkecil. Pengurutan data memiliki beberapa metode, antara lain sebagai berikut.

a) Bubble Short atau Exchange Short

Ide dari bubble short adalah gelembung air yang akan “mengapung” untuk tabel yang terurut menaik (*ascending*). Elemen bernilai kecil akan “diapungkan” (ke indeks terkecil), artinya diangkat ke “atas” (indeks terkecil) melalui pertukaran. Karena algoritma ini melakukan pengurutan dengan cara membandingkan elemen-elemen data satu sama lain, maka *bubble sort* termasuk ke dalam jenis algoritma *comparison-based sorting*.

Metode gelembung (*bubble sort*) sering juga disebut dengan metode penukaran (*exchange sort*) adalah metode yang mengurutkan data dengan cara membandingkan masing-masing elemen, kemudian melakukan penukaran bila perlu. Metode ini mudah dipahami dan diprogram, tetapi bila dibandingkan dengan metode lain yang kita pelajari, metode ini merupakan metode yang paling tidak efisien.

Proses dalam *Bubble sort* dilakukan sebanyak $N-1$ langkah (*pass*) dengan N adalah ukuran *array*. Pada akhir setiap langkah ke $-I$, *array* $L[0 \dots N]$ akan terdiri atas dua bagian, yaitu bagian yang sudah terurut $L[0 \dots I]$ dan bagian yang belum terurut $L[I+1 \dots N-1]$. Setelah langkah terakhir, diperoleh *array* $L[0 \dots N-1]$ yang terurut menaik.

b) Selection Short

Algoritma *selection sort* memilih elemen maksimum atau minimum *array*, lalu menempatkan elemen maksimum atau minimum itu pada awal atau akhir *array* (tergantung pada urutannya *ascending* atau *descending*). Selanjutnya elemen tersebut tidak disertakan pada proses selanjutnya. Karena setiap kali *selection sort* harus membandingkan elemen-elemen data, algoritma ini termasuk dalam *comparison-based sorting*. Terdapat dua pendekatan dalam metode

pengurutan dengan Selection Sort :

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

c) Insertion Short

Insertion sort adalah sebuah algoritma pengurutan yang membandingkan dua elemen data pertama, mengurutkannya, kemudian mengecek elemen data berikutnya satu persatu dan membandingkannya dengan elemen data yang telah diurutkan. Karena algoritma ini bekerja dengan membandingkan elemen-elemen data yang akan diurutkan, algoritma ini termasuk pula dalam *comparison-based sort*

Ide dasar dari algoritma *Insertion Sort* ini adalah mencari tempat yang “tepat” untuk setiap elemen *array*, dengan cara *sequential search*. Proses ini kemudian menyisipkan sebuah elemen *array* yang diproses ke tempatnya yang seharusnya. Proses dilakukan sebanyak N-1 tahapan (dalam *sorting* disebut sebagai “*pass*”)

Algoritma *insertion sort* pada dasarnya memilah data yang akan diurutkan menjadi dua bagian, yang belum diurutkan dan yang sudah diurutkan. Elemen pertama diambil dari bagian *array* yang belum diurutkan dan kemudian diletakkan sesuai posisinya pada bagian lain dari *array* yang telah diurutkan. Langkah ini dilakukan secara berulang hingga tidak ada lagi elemen yang tersisa pada bagian *array* yang belum diurutkan.

d) Quick Short

Metode *Quick* sering disebut juga metode partisi (*partition exchange sort*). Metode ini diperkenalkan pertama kali oleh C.A.R. Hoare pada tahun 1962. Untuk mempertinggi efektivitas dari metode ini, digunakan teknik menukarkan dua elemen dengan jarak yang cukup besar.

Dasar strateginya adalah “memecah dan menguasai”. *Quick sort* dimulai dengan menscan daftar yang disortir untuk nilai median. Nilai ini, yang disebut tumpuan (*pivot*), kemudian dipindahkan ke satu sisi pada daftar dan butir-butir yang nilainya lebih besar dari tumpuan di pindahkan ke sisi lain.

Pertama-tama $x \leftarrow L[q]$ dipakai untuk membagi larik $L[p \dots r]$ menjadi dua bagian (disebut pemartisian) dengan kondisi semua elemen bagian kiri selalu lebih kecil daripada nilai elemen pivot dan nilai semua elemen bagian kanan selalu lebih kecil daripada nilai elemen pivot.

Pemartisian dilakukan dengan menggunakan variabel i dan j . Dalam hal ini i berupa petunjuk yang bergerak naik, sedangkan j adalah penunjuk bergerak turun. Variabel j bergeser turun secara terus-menerus sehingga $L[j] \leq$ elemen pivot, sedangkan i digeser naik secara terus-menerus sampai memenuhi kondisi $L[i] \geq$ elemen pivot. Proses pengulangan dilakukan sampai nilai $i \geq j$. Pada keadaan seperti ini nilai balik subrutin partisi berupa j .

e) Heap Sort

Metode *heap sort* adalah metode dari pengembangan *tree*. *Heap sort* melakukan suatu pengurutan menggunakan suatu struktur data yang di sebut *heap*. *Heap* memiliki kompleksitas yang besar dalam pembuatan kodenya, tetapi *heap sort* mampu mengurutkan data-data yang sangat banyak dengan waktu yang cepat.

Dalam *sorting* biasanya mempunyai sebuah aturan, berikut adalah aturan dari *heap sort* :

- 1) Untuk mengisikan *heap* dimulai dari level 1 sampai ke level di bawahnya, jika dalam level yang sama semua kunci *heap* belum terisi, maka tidak boleh mengisi di bawahnya.
- 2) *Heap* dalam kondisi teratur apabila *left child* \leq *parent*.
- 3) Penambahan kunci diletakkan pada posisi terakhir dari level dan di sebelah kanan *child* yang terakhir, kemudian diurutkan dengan cara *upheap*.
- 4) Bila menghapus *heap* dengan mengambil kunci pada *parent* di level 1 kemudian digantikan posisi kunci terakhir, selanjutnya di-sort kembali metode *downheap*.