

# BUKU PEDOMAN KULIAH ALGORITMA & PEMROGRAMAN II



Oleh : Drs. Muh. Alhan, ST., M.Eng.

PROGRAM STUDI MANAJEMEN INFORMATIKA  
POLITEKNIK PRATAMA MULIA  
SURAKARTA 2018

# Daftar Isi

<b>KATA PENGANTAR.....</b>	<b>I</b>
<b>DAFTAR ISI.....</b>	<b>II</b>
<b>DAFTAR GAMBAR.....</b>	<b>VI</b>
<b>DAFTAR TABEL .....</b>	<b>VII</b>
<b>BAB 1 PENGANTAR LOGIKA DAN ALGORITMA .....</b>	<b>1</b>
1.1.    PENGERTIAN LOGIKA DAN ALGORITMA.....	1
1.2.    CIRI-CIRI, SIFAT, STRUKTUR DASAR DAN CARA PENULISAN ALGORITMA.....	4
1.1.1 <i>Runtunan (sequence)</i> .....	5
1.1.2 <i>Pemilihan (selection)</i> .....	6
1.1.3 <i>Pengulangan (repetition)</i> .....	6
1.3.    PERBEDAAN ALGORITMA DAN PROGRAM .....	7
1.4.    MEKANISME PELAKSANAAN ALGORITMA OLEH PEMROSES NOTASI PENULISAN ALGORITMA .....	8
1.5.    BELAJAR MEMBUAT PROGRAM DAN BELAJAR BAHASA PEMROGRAMAN .....	10
1.6.    LATIHAN SOAL .....	12
<b>BAB 2 NOTASI PENULISAN ALGORITMA .....</b>	<b>13</b>
2.1.    KALIMAT DESKRIPTIF.....	14
2.1.1 <i>Judul Algoritma</i> .....	14
2.1.2 <i>Bagian Deklarasi</i> .....	15
2.1.3 <i>Bagian deskripsi</i> .....	16
2.2.    PSEUDOCODE .....	17
2.3.    FLOWCHART .....	19
2.4.    LATIHAN SOAL .....	24
<b>BAB 3 STRUKTUR DATA DALAM ALGORITMA .....</b>	<b>28</b>
3.1.    TIPE DATA .....	29
3.1.1 <i>Tipe dasar</i> .....	29
3.1.2 <i>Tipe data bentukan</i> .....	30
3.1.3 <i>Tipe data abstrak (Abstract Data Type)</i> .....	30
3.2.    KONSTANTA DAN VARIABEL .....	31
3.3.    ARRAY .....	32
3.4.    STACK.....	35
3.5.    QUEUE .....	36
3.6.    TREE .....	37
3.7.    GRAPH .....	38
3.8.    LATIHAN SOAL .....	41
<b>BAB 4 STUDI KASUS PERMASALAHAN SEDERHANA .....</b>	<b>42</b>
4.1.    STUDI KASUS 1: MENGHITUNG LUAS DAN KELILING LINGKARAN .....	42
4.1.1 <i>Permasalahan</i> .....	42
4.1.2 <i>Cara Penyelesaian Masalah</i> .....	42
4.1.3 <i>Struktur Data Yang Dibutuhkan</i> .....	43
4.1.4 <i>Input</i> .....	43

4.1.5	Output.....	44
4.1.6	Proses Penyelesaian.....	44
4.1.7	Flowchart Keseluruhan.....	45
4.2	STUDI KASUS 2: KONVERSI SUHU .....	45
4.2.1	Permasalahan .....	45
4.2.2	Cara Penyelesaian Masalah .....	45
4.2.3	Struktur Data Yang Dibutuhkan.....	46
4.2.4	Deklarasi dan Inisialisasi.....	46
4.2.5	Input .....	46
4.2.6	Output.....	46
4.2.7	Proses Penyelesaian.....	47
4.2.8	Flowchart Keseluruhan.....	47
4.3	STUDI KASUS 3: MENAMPILKAN BILANGAN GANJIL.....	47
4.3.1	Permasalahan .....	47
4.3.2	Cara Penyelesaian Masalah .....	48
4.3.3	Struktur Data Yang Dibutuhkan.....	48
4.3.4	Deklarasi dan Inisialisasi.....	48
4.3.5	Input .....	48
4.3.6	Output.....	49
4.3.7	Proses Penyelesaian.....	49
4.3.8	Flowchart Keseluruhan.....	50
4.4	SOAL LATIHAN .....	50
<b>BAB 5 STUDI KASUS PERBANDINGAN .....</b>		<b>52</b>
5.1	CONTOH KASUS 1: TAHUN KABISAT .....	52
5.1.1	Permasalahan .....	52
5.1.2	Proses Penyelesaian Masalah: .....	52
5.1.3	Input: .....	52
5.1.4	Output: .....	52
5.1.5	Struktur Data yang Dibutuhkan: .....	53
5.1.6	Logika Pemrograman: .....	53
5.1.7	Flowchart:.....	53
5.2	CONTOH KASUS 2: DERET BILANGAN GENAP .....	53
5.2.1	Permasalahan .....	53
5.2.2	Cara Penyelesaian Masalah: .....	54
5.2.3	Input: .....	54
5.2.4	Output: .....	54
5.2.5	Struktur Data yang Dibutuhkan: .....	54
5.2.6	Logika Pemrograman: .....	54
5.2.7	Flowchart:.....	55
5.3	SOAL LATIHAN: .....	55
<b>BAB 6 STUDI KASUS KONVERSI .....</b>		<b>56</b>
6.1	CONTOH KASUS 1: KONVERSI JAM KE MENIT .....	56
6.1.1	Permasalahan .....	56
6.1.2	Cara Penyelesaian Masalah: .....	56
6.1.3	Input: .....	56
6.1.4	Output: .....	57
6.1.5	Struktur Data yang Dibutuhkan: .....	57
6.1.6	Logika Pemrograman: .....	57
6.1.7	Flowchart:.....	57
6.2	CONTOH KASUS 2: KONVERSI DETIK KE HARI, JAM, MENIT, DAN DETIK .....	57
6.2.1	Permasalahan .....	57
6.2.2	Cara Penyelesaian Masalah: .....	58

6.2.3	<i>Input:</i> .....	58
6.2.4	<i>Output:</i> .....	58
6.2.5	<i>Struktur Data yang Dibutuhkan:</i> .....	58
6.2.6	<i>Logika Pemrograman:</i> .....	59
6.2.7	<i>Flowchart:</i> .....	59
6.3	SOAL LATIHAN: .....	59
<b>BAB 7 STUDI KASUS PERCABANGAN DAN PERULANGAN.....</b>		<b>60</b>
7.1	CONTOH KASUS 1: KALKULATOR SEDERHANA .....	60
7.1.1	<i>Cara Penyelesaian Masalah:</i> .....	60
7.1.2	<i>Input:</i> .....	61
7.1.3	<i>Output:</i> .....	61
7.1.4	<i>Struktur Data yang Dibutuhkan:</i> .....	61
7.1.5	<i>Logika Pemrograman:</i> .....	61
7.1.6	<i>Flowchart:</i> .....	62
7.2	CONTOH KASUS 2: TUMPUKAN BILANGAN.....	62
7.2.1	<i>Permasalahan</i> .....	62
7.2.2	<i>Cara Penyelesaian Masalah:</i> .....	62
7.2.3	<i>Input:</i> .....	63
7.2.4	<i>Output:</i> .....	63
7.2.5	<i>Struktur Data yang Dibutuhkan:</i> .....	63
7.2.6	<i>Logika Pemrograman:</i> .....	63
7.2.7	<i>Flowchart:</i> .....	63
7.3	SOAL LATIHAN .....	64
<b>BAB 8 STUDI KASUS TUMPUKAN (STACK).....</b>		<b>66</b>
8.1	CONTOH KASUS 1: MEMBALIK KALIMAT .....	66
8.1.1	<i>Permasalahan</i> .....	66
8.1.2	<i>Cara Penyelesaian Masalah:</i> .....	66
8.1.3	<i>Input:</i> .....	67
8.1.4	<i>Output:</i> .....	67
8.1.5	<i>Struktur Data yang Dibutuhkan:</i> .....	67
8.1.6	<i>Logika Pemrograman:</i> .....	67
8.1.7	<i>Flowchart</i> .....	67
8.2	CONTOH KASUS 2: MEMBALIK BILANGAN .....	68
8.2.1	<i>Permasalahan</i> .....	68
8.2.2	<i>Cara Penyelesaian Masalah:</i> .....	68
8.2.3	<i>Input:</i> .....	69
8.2.4	<i>Output:</i> .....	69
8.2.5	<i>Struktur Data yang Dibutuhkan:</i> .....	69
8.2.6	<i>Logika Pemrograman:</i> .....	69
8.2.7	<i>Flowchart</i> .....	69
8.3	SOAL LATIHAN .....	70
<b>BAB 9 STUDI KASUS KONVERSI BILANGAN.....</b>		<b>72</b>
9.1	. STUDI KASUS 1: KONVERSI BILANGAN BINER KE DESIMAL .....	72
9.1.1	<i>Permasalahan</i> .....	72
9.1.2	<i>Cara Penyelesaian Masalah</i> .....	72
9.1.3	<i>Struktur Data Yang Dibutuhkan</i> .....	73
9.1.4	<i>Deklarasi dan Inisialisasi</i> .....	73
9.1.5	<i>Input</i> .....	73
9.1.6	<i>Output</i> .....	74
9.1.7	<i>Proses Penyelesaian</i> .....	74
9.1.8	<i>Flowchart Keseluruhan</i> .....	74

9.2	STUDI KASUS 2: KONVERSI BILANGAN DESIMAL KE BINER .....	75
9.2.1	Permasalahan .....	75
9.2.2	Cara Penyelesaian Masalah .....	75
9.2.3	Struktur Data Yang Dibutuhkan.....	76
9.2.4	Deklarasi dan Inisialisasi.....	76
9.2.5	Input .....	76
9.2.6	Output.....	76
9.2.7	Proses Penyelesaian.....	77
9.2.8	Flowchart Keseluruhan.....	78
9.3	LATIHAN .....	78
9.3.1	Permasalahan .....	78
9.3.2	Cara Penyelesaian Masalah .....	78
<b>BAB 10 STUDI KASUS OPERASI MATRIKS.....</b>		<b>80</b>
10.1	STUDI KASUS: OPERASI PENAMBAHAN MATRIKS .....	80
10.1.1	Permasalahan .....	80
10.1.2	Cara Penyelesaian Masalah.....	80
10.1.3	Struktur Data Yang Dibutuhkan .....	81
10.1.4	Deklarasi dan Inisialisasi .....	81
10.1.5	Input .....	81
10.1.6	Output .....	82
10.1.7	Proses Penyelesaian .....	82
10.1.8	Flowchart Keseluruhan .....	83
10.2	10.2. LATIHAN .....	83
10.2.1	Permasalahan.....	83
10.2.2	Cara Penyelesaian Masalah.....	84
<b>BAB 11 STUDI KASUS SHORTEST PATH PROBLEM.....</b>		<b>86</b>
11.1	STUDI KASUS: SHORTEST PATH PROBLEM DENGAN ALGORITMA DIJKSTRA .....	86
11.1.1	Permasalahan .....	86
11.1.2	Cara Penyelesaian Masalah.....	86
11.1.3	Struktur Data Yang Dibutuhkan .....	90
11.1.4	Deklarasi dan Inisialisasi .....	91
11.1.5	Input.....	91
11.1.6	Output .....	92
11.2	LATIHAN .....	92

# Bab 1

## Pengantar Logika dan Algoritma

---

### Pokok Bahasan

1. Konsep logika dan algoritma
2. Ciri-ciri algoritma
3. Konsep algoritma, program, dan bahasa pemrograman

### Tujuan Pembelajaran

Setelah mempelajari materi dalam bab ini, mahasiswa diharapkan mampu:

1. Mengetahui dan memahami logika dan algoritma.
2. Mampu membuat contoh penyelesaian masalah dengan menggunakan konsep logika

### 1.1. Pengertian Logika dan Algoritma

Pengertian algoritma sangat lekat dengan kata logika, yaitu kemampuan seorang manusia untuk berfikir dengan akal tentang suatu permasalahan menghasilkan sebuah kebenaran, dibuktikan dan dapat diterima akal, logika seringkali dihubungkan dengan kecerdasan, seseorang yang mampu berlogika dengan baik sering orang menyebutnya sebagai pribadi yang cerdas. Dalam menyelesaikan suatu masalahpun logika mutlak diperlukan.

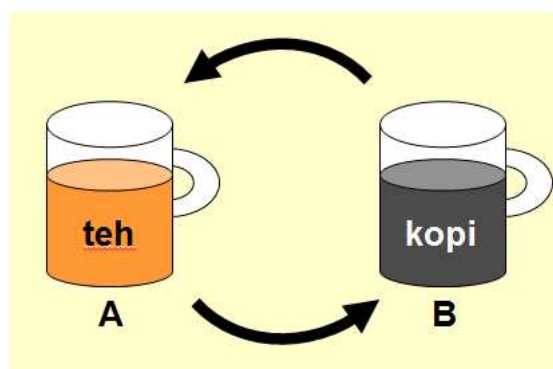
Logika identik dengan masuk akal dan penalaran. Penalaran adalah salah satu bentuk pemikiran. Pemikiran adalah pengetahuan tak langsung yang didasarkan pada pernyataan langsung pemikiran mungkin benar dan mungkin juga tak benar. Definisi logika sangat sederhana yaitu ilmu yang memberikan prinsip-prinsip yang harus diikuti agar dapat berfikir valid menurut aturan yang berlaku. Pelajaran logika menimbulkan kesadaran untuk menggunakan prinsip-prinsip untuk berfikir secara sistematis.

Logika berasal dari bahasa Yunani yaitu LOGOS yang berarti ilmu. Logika dapat diartikan ilmu yang mengajarkan cara berpikir untuk melakukan kegiatan dengan tujuan tertentu. Algoritma berasal dari nama seorang Ilmuwan Arab yang bernama Abu Jafar Muhammad Ibnu Musa Al Khuwarizmi penulis buku berjudul Al Jabar Wal Muqabala.

Kata Al Khuwarizmi dibaca orang barat menjadi Algorism yang kemudian lambat laun menjadi Algorithm diserap dalam bahasa Indonesia menjadi Algoritma. Algoritma dapat diartikan urutan penyelesaian masalah yang disusun secara sistematis menggunakan bahasa yang logis untuk memecahkan suatu permasalahan.

Meski demikian terdapat beberapa definisi algoritma yang lain. Diantaranya menurut Rinaldi Munir, algoritma adalah urutan langkah-langkah logis penyelesaian masalah yang disusun secara sistematis. Sedang menurut Kamus Besar Bahasa Indonesia, definisi algoritma adalah urutan logis pengambilan keputusan untuk pemecahan masalah. Menurut tim Gunadarma:1988, algoritma adalah suatu himpunan berhingga dari instruksi-instruksi yang secara jelas memperinci langkah-langkah proses pelaksanaan, dalam pemecahan suatu masalah tertentu, atau suatu kelas masalah tertentu, dengan dituntut pula bahwa himpunan instruksi tersebut dapat dilaksanakan secara mekanik. Dari pengertian diatas maka dapat disimpulkan bahwa Logika dan Algoritma adalah ilmu yang mempelajari cara penyelesaian suatu masalah berdasarkan urutan langkah-langkah terbatas yang disusun secara sistematis dan menggunakan bahasa yang logis dengan tujuan tertentu.

Untuk lebih mudah memahami arti dari algoritma dicontohkan sebuah permasalahan penukaran isi dari dua gelas. Diberikan dua buah gelas A dan B, gelas A berisi air teh dan gelas B berisi air kopi. Pertukarkan isi gelas tersebut sehingga menghasilkan gelas A yang semula berisi air kopi menjadi berisi air teh dan gelas B yang semula berisi air kopi menjadi berisi air teh. Ilustrasi permasalahan ini dapat dilihat pada Gambar 1.1.

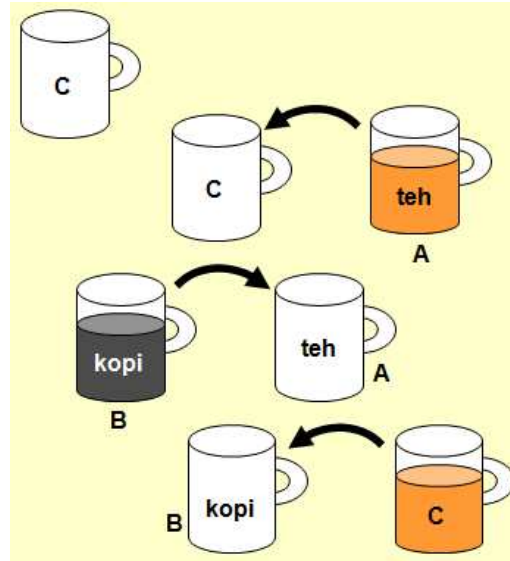


Gambar 1.1. Penukaran gelas isi gelas A dan gelas B.

Cara penyelesaian permasalahan ini adalah sebagai berikut. Untuk mempertukarkan isi gelas dengan benar, maka diperlukan gelas tambahan yang kita namakan gelas C sebagai tempat penampungan sementara. Berikut Algoritmanya:

1. Siapkan gelas cadangan C
2. Tuangkan air teh dari gelas A ke dalam gelas C (gelas A menjadi kosong).
3. Tuangkan air kopi dari gelas B ke dalam gelas A (gelas B menjadi kosong).
4. Tuangkan air teh dari gelas C ke dalam gelas B.

Ilustrasi langkah-langkah algoritma dapat dilihat pada Gambar 1.2.



Gambar 1.2. Langkah-langkah penukaran gelas isi gelas A dan gelas B.

Dari contoh tersebut dapat dilihat bahwa penyelesaian permasalahan penukaran isi dua buah gelas sangat sederhana. Disini digunakan urutan langkah yang masuk akal atau logis sehingga isi dari kedua nya sudah berpindah media, dari A ke B dan B ke A. Inilah yang dinamakan “Algoritma”, urutan penyelesaian sebuah permasalahan dengan urutan dan langkah yang logis dan masuk akal menghasilkan sesuatu langkah yang benar.

Contoh lain penggunaan logika dan algoritma adalah membuat algoritma untuk menghitung luas lingkaran, caranya:

1. Menentukan nilai jari-jari ( $r$ ) lingkaran.
2. Menentukan nilai phi.
3. Menghitung luas lingkaran dengan cara mengkalikan nilai jari-jari ( $r$ ) dengan ( $r$ ) lalu dikalikan dengan nilai phi.
4. Maka luas lingkaran ditemukan.
5. Selesai.

Saat menggunakan logika, sebaiknya jangan berfikir terlalu rumit tentang sebuah masalah, karena belum tentu masalah itu serumit yang kita pikir. Pikirkan hal yang paling sederhana untuk menyelesaikan masalah itu, sehingga tidak terjebak dalam pikiran rumit yang dibuat sendiri. Meski demikian jangan meremehkan masalah sekecil apapun, tapi berfikir sederhana untuk menghasilkan solusi yang efektif



Dalam menentukan algoritma untuk menyelesaikan suatu permasalahan, mungkin kita dihadapkan oleh beberapa pilihan algoritma. Oleh karena itu kita harus memiliki rambu-rambu dalam menentukan pilihan algoritma. Pertimbangan dalam pemilihan algoritma adalah, pertama, algoritma haruslah benar. Artinya algoritma akan memberikan keluaran sesuai seperti yang diharapkan dari sejumlah masukan yang diberikan. Tidak

peduli sebagus apapun algoritma, jika memberikan keluaran yang salah, maka sudah pasti algoritma tersebut bukanlah algoritma yang baik. Pertimbangan kedua yang harus diperhatikan adalah kita harus mengetahui seberapa baik hasil yang dicapai oleh algoritma tersebut. Hal ini penting terutama pada algoritma yang memerlukan aproksimasi hasil yaitu algoritma yang hasilnya hanya berupa pendekatan. Algoritma yang baik harus mampu memberikan hasil yang sedekat mungkin dengan nilai yang sebenarnya. Ketiga adalah efisiensi algoritma. Efisiensi algoritma dapat ditinjau dari dua hal yaitu efisiensi waktu dan memori. Meskipun algoritma memberikan keluaran yang benar atau paling mendekati, tetapi jika kita harus menunggu lama untuk mendapatkan hasil semisal berjam-jam untuk mendapatkan keluarannya maka biasanya algoritma tersebut biasanya tidak akan menjadi pilihan utama, setiap orang menginginkan keluaran yang relatif cepat. Begitu juga dengan memori, semakin besar memori yang terpakai maka semakin jelek algoritma tersebut. Dalam kenyataannya, setiap orang bisa membuat algoritma yang berbeda untuk menyelesaikan suatu permasalahan, walaupun terjadi perbedaan dalam menyusun algoritma, tentunya kita mengharapkan keluaran yang mirip atau sama. Jika dihadapkan pada permasalahan seperti ini maka sebaiknya pilih algoritma yang paling efisien dan cepat.

Tujuan dari belajar logika dan algoritma adalah agar dapat membiasakan diri melakukan suatu perencanaan apabila menyelesaikan suatu masalah. Karena suatu permasalahan yang diselesaikan dengan suatu perencanaan yang matang maka akan mendapatkan solusi yang lebih optimal dibandingkan menyelesaikan masalah tanpa menggunakan suatu perencanaan.

## **1.2. Ciri-ciri, Sifat, Struktur Dasar dan Cara Penulisan Algoritma**

Tidak semua urutan langkah penyelesaian masalah yang logis dapat disebut sebagai algoritma. Menurut Donald E. Knuth, algoritma mempunyai lima ciri penting yang meliputi:

1. Finiteness (keterbatasan), algoritma harus berakhir setelah mengerjakan sejumlah langkah proses.

2. Definiteness (kepastian), setiap langkah harus didefinisikan secara tepat dan tidak berarti ganda.
3. Input (masukan), algoritma memiliki nol atau lebih data masukan (input).
4. Output (keluaran), algoritma mempunyai nol atau lebih hasil keluaran (output).
5. Effectiveness (efektivitas), algoritma harus sangkil (efektif), langkah-langkah algoritma dikerjakan dalam waktu yang wajar.

Sedang sifat algoritma adalah:

1. Tidak menggunakan simbol atau sintaks dari suatu bahasa pemrograman tertentu.
2. Tidak tergantung pada suatu bahasa pemrograman tertentu.
3. Notasi-notasinya dapat digunakan untuk seluruh bahasa manapun.
4. Algoritma dapat digunakan untuk merepresentasikan suatu urutan kejadian secara logis dan dapat diterapkan di semua kejadian sehari-hari

Seperti telah dijelaskan di sub bab sebelumnya bahwa penyusun atau struktur dasar algoritma adalah langkah-langkah. Suatu Algoritma dapat terdiri dari tiga struktur dasar, yaitu runtunan, pemilihan dan pengulangan. Ketiga jenis langkah tersebut membentuk konstruksi suatu algoritma. Berikut adalah penjelasan dari tiga struktur tersebut :

### **1.1.1 Runtunan (sequence)**

Sebuah runtunan terdiri dari satu atau lebih instruksi. Tiap instruksi dikerjakan secara berurutan sesuai dengan urutan penulisannya, yakni sebuah instruksi dilaksanakan setelah instruksi sebelumnya selesai dikerjakan. Urutan dari instruksi menentukan hasil akhir dari suatu algoritma. Bila urutan penulisan berubah maka mungkin juga hasil akhirnya berubah. Sebagai contoh perhatikan operasi aritmatika berikut ini,  $(4+3)*7=49$ , tetapi bila urutan aksinya diubah maka hasil keluaran akan berbeda menjadi  $4+(3*7)=25$ .

Contoh lain dari runtunan aksi adalah algoritma penukaran dua bilangan bulat,yaitu:

1. Deklarasikan A, B, dan C sebagai bilangan bulat
2. Masukkan nilai A dan B
3. Masukkan nilai A ke dalam C
4. Masukkan nilai B ke dalam A
5. Masukkan nilai C ke dalam B

### 1.1.2 Pemilihan (selection)

Kadangkala terdapat suatu kejadian yang baru akan dikerjakan jika suatu kondisi tertentu telah terpenuhi. Pemilihan yaitu instruksi yang dikerjakan dengan kondisi tertentu. Kondisi adalah persyaratan yang dapat bernilai benar atau salah. Satu atau beberapa instruksi hanya dilaksanakan apabila kondisi bernilai benar, sebaliknya apabila salah maka instruksi tidak akan dilaksanakan. Contoh kasus pemilihan adalah dalam penentuan bilangan genap atau ganjil berikut ini:

1. Masukkan **bilangan** sebagai sebuah bilangan bulat
2. Bagi **bilangan** dengan angka 2, simpan nilai sisa pembagian dalam variabel **sisa**
3. Jika nilai **sisa** sama dengan 0 maka kerjakan langkah 4:
4. Tampilkan "GENAP" ke layar
5. Jika nilai **sisa** tidak sama dengan 0 maka kerjakan langkah 6
6. Tampilkan "GANJIL" ke layar
7. Selesai.

### 1.1.3 Pengulangan (repetition)

Salah satu kelebihan komputer adalah kemampuannya untuk mengerjakan pekerjaan yang sama berulang kali tanpa mengenal lelah. Kita tidak perlu menulis instruksi yang sama berulang kali, tetapi cukup melakukan pengulangan dengan instruksi yang tersedia. Pengulangan merupakan kegiatan mengerjakan sebuah atau sejumlah aksi yang sama sebanyak jumlah yang ditentukan atau sesuai dengan kondisi yang diinginkan. Beberapa statemen pengulangan di bahasa pemrograman yaitu for..., while()..., do...while(), repeat....until, for...downto...do, for...to...do dan lain-lain. Sebagai contoh adalah menampilkan huruf tertentu sebanyak n kali ke layar sebagai berikut:

1. Deklarasikan variabel **huruf** untuk menyimpan karakter yang akan ditampilkan.
2. Deklarasikan variabel **n** untuk menyimpan banyaknya perulangan
3. Deklarasikan variabel **counter** yang digunakan sebagai counter perulangan yang sudah dilakukan.
4. Masukkan sebuah karakter dan simpan dalam variabel **huruf**
5. Masukkan banyaknya perulangan yang diinginkan dan simpan dalam variabel **n**
6. Set nilai **counter** dengan 0
7. Tampilkan **huruf** ke layar
8. Lakukan penambahan **counter** dengan 1

9. Jika nilai **counter** < **n**, kerjakan langkah 6
10. Jika **nilai counter** = **n** selesai

### 1.3. Perbedaan Algoritma dan Program

Sebagaimana telah diuraikan di sub bab sebelumnya bahwa algoritma adalah urutan langkah-langkah terbatas yang disusun secara sistematis dan menggunakan bahasa yang logis dengan tujuan menyelesaikan suatu masalah tertentu. Sementara program adalah kumpulan instruksi berupa pernyataan yang ditulis dengan menggunakan bahasa pemrograman yang melibatkan pemilihan struktur data. Beberapa pakar komputer menyatakan program dengan formula  $\text{Program} = \text{Algoritma} + \text{Bahasa Pemrograman}$ .

Bahasa pemrograman dan algoritma berhubungan sangat erat pada sebuah program. Algoritma yang baik tanpa pemilihan struktur data yang tepat akan membuat program menjadi kurang baik, demikian juga sebaliknya. Jika dihubungkan dengan program, maka pembuatan algoritma harus memperhatikan kaidah:

1. Pembuatan atau penulisan algoritma tidak tergantung pada bahasa pemrograman manapun, artinya penulisan algoritma independen dari bahasa pemrograman dan komputer yang memprosesnya.
2. Notasi algoritma dapat diterjemahkan ke dalam berbagai bahasa pemrograman.
3. Apapun bahasa pemrogramannya, output yang akan dikeluarkan sama karena algoritmanya sama.

Algoritma dibuat untuk membantu kita dalam mengkonversikan suatu permasalahan ke dalam bahasa pemrograman. Algoritma merupakan hasil pemikiran konseptual, supaya dapat dilaksanakan oleh komputer, algoritma harus diterjemahkan ke dalam notasi bahasa pemrograman. Ada beberapa hal yang harus diperhatikan pada penerjemahan tersebut, yaitu:

#### 1. Pendeklarasian variabel

Variabel dibutuhkan oleh program dalam rangka menyimpan data masukan, memproses dan mendapatkan hasil komputasi.

#### 2. Pemilihan tipe data

Apabila dalam proses pembuatan program ternyata membutuhkan pendeklarasian variabel maka diwajibkan memilih tipe data, karena setiap variabel pasti membutuhkan tipe data ketika dideklarasikan.

### 3. Pemakaian atau pemilihan instruksi

Terdapat beberapa macam instruksi dalam bahasa pemrograman (sequence, selection dan repetition), urutan langkah dalam algoritma dapat diterjemahkan menjadi salah satu atau beberapa instruksi tersebut.

### 4. Aturan sintaksis

Pada saat menuliskan program kita terikat dengan aturan sintaksis dalam bahasa pemrograman yang akan digunakan. Setiap bahasa pemrograman memiliki aturan penulisan sintaks-nya sendiri.

### 5. Tampilan hasil

Pada saat membuat algoritma kita tidak memikirkan tampilan hasil yang akan disajikan. Hal teknis semacam ini diperhatikan ketika mengkonversikannya menjadi program.

### 6. Cara pengoperasian compiler atau interpreter.

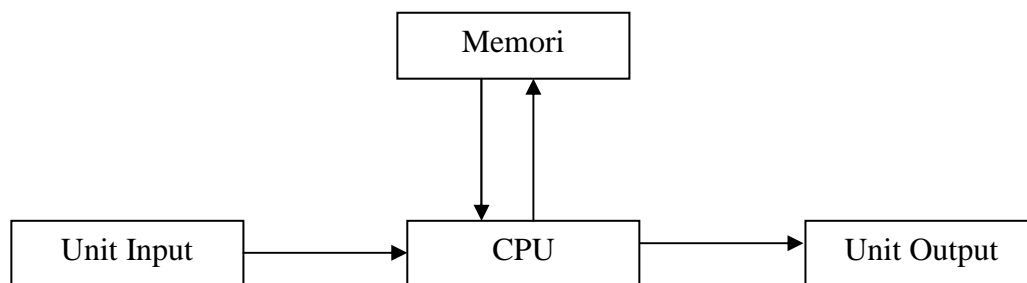
Bahasa pemrograman yang digunakan termasuk dalam kelompok compiler atau interpreter. Masing-masing memiliki cara pengoperasian yang bisa jadi berbeda.

Komputer adalah mesin pemroses. Agar dapat dikerjakan oleh komputer, algoritma harus ditulis dalam notasi bahasa pemrograman berupa instruksi yang dapat dipahami oleh komputer sehingga dinamakan program. Jadi program adalah perwujudan atau implementasi teknis algoritma yang ditulis dalam bahasa pemrograman tertentu sehingga dapat dilaksanakan oleh komputer. Program ditulis dengan menggunakan salah satu bahasa pemrograman. Kegiatan membuat program disebut pemrograman (programming). Orang yang menulis program disebut pemrogram (programmer). Tiap-tiap langkah didalam program disebut pernyataan atau instruksi. Jadi, program tersusun atas sederetan instruksi. Bila suatu instruksi dilaksanakan, maka operasi-operasi yang bersesuaian dengan instruksi tersebut dikerjakan komputer.

## **1.4. Mekanisme Pelaksanaan Algoritma oleh Pemroses Notasi Penulisan Algoritma**

Komputer sebagai mesin pemroses terdiri dari empat komponen utama, yaitu unit input, unit output, unit pemroses sentral (CPU: Central Processing Unit), dan memori seperti digambarkan pada Gambar 1.3. CPU adalah perangkat keras komputer yang memahami dan melaksanakan perintah dan data dari perangkat lunak. Istilah lain, pemroses/prosesor

(processor) sering digunakan untuk menyebut CPU. CPU berperan sebagai otak komputer, yang berfungsi mengerjakan operasi-operasi aritmatika seperti kalkulator, hanya saja CPU jauh lebih kuat dan lebih besar daya pemrosesannya. Fungsi utama dari CPU adalah melakukan operasi aritmatika dan logika terhadap data yang diambil dari memori atau dari informasi yang dimasukkan melalui unit input berupa beberapa perangkat keras, seperti papan tombol, pemindai, tuas kontrol, maupun tetikus. CPU dikontrol menggunakan sekumpulan instruksi perangkat lunak komputer. Perangkat lunak tersebut dapat dijalankan oleh CPU dengan membacanya dari media penyimpanan, seperti cakram keras, disket, cakram padat, maupun pita perekam. Instruksi-instruksi tersebut kemudian disimpan terlebih dahulu pada memori fisik, yang mana setiap instruksi akan diberi alamat unik yang disebut alamat memori. Selanjutnya, CPU dapat mengakses data-data pada memori fisik dengan menentukan alamat data yang dikehendaki. Selama proses ini terjadi, sebuah unit dalam CPU yang disebut dengan penghitung program akan memantau instruksi yang sukses dijalankan supaya instruksi tersebut dapat dieksekusi dengan urutan yang benar dan sesuai sehingga didapatkan hasil yang dapat ditampilkan pada unit output, misalnya dengan menampilkannya di layar monitor.



Gambar 1.3. Komponen-Komponen Utama Komputer

Seperti yang telah diketahui bahwa komputer memerlukan instruksi berupa langkah-langkah perintah sehingga sebuah prosedur dapat dijalankan. Prosedur yang berisi langkah-langkah penyelesaian masalah inilah yang disebut dengan algoritma. Jadi, sebelum memasuki tahap pemrograman komputer dengan menggunakan bahasa pemrograman, sebaiknya mempelajari algoritma yang merupakan pengantar kepada pemrograman komputer tersebut.

Dalam ilmu komputer, algoritma dikenal dengan langkah-langkah komputasi yang terdiri dari masukan dan keluaran. Karena itu Algoritma biasanya dijadikan dasar atau

pengantar bahasa pemrograman dalam studi yang berkecimpung atau berhubungan dengan ilmu komputer, misalnya Teknik Informatika.

### **1.5. Belajar Membuat Program dan Belajar Bahasa Pemrograman**

Belajar membuat program berbeda dengan belajar bahasa pemrograman. Belajar membuat program adalah belajar tentang strategi atau cara menyelesaikan suatu masalah, yang diungkapkan dalam bentuk algoritma yang mudah dibaca dan dipahami yang selanjutnya dituangkan dalam bahasa pemrograman. Belajar memprogram bersifat pemahaman persoalan, analisis dan sintesis. Titik berat belajar membuat program adalah desain program. Sedangkan belajar bahasa pemrograman adalah belajar mengenai tata cara atau tata aturan penulisan pernyataan atau statement untuk membuat program dalam bahasa tertentu saja. Tata aturan ini biasa disebut juga dengan sintaks bahasa. Sintaks bahasa ini meliputi cara mendeklarasikan variabel, cara melakukan perulangan, percabangan dan lain-lain. Tiap bahasa pemrograman memiliki aturan penulisan sendiri-sendiri walaupun terkadang ada kemiripan aturan antar bahasa. Titik berat belajar bahasa pemrograman adalah coding program.

Sampai saat ini terdapat puluhan bahasa pemrogram, antara lain bahasa Assembly, Fortran, Cobol, PL/I, Algol, Pascal, C, C++, Basic, Prolog, LISP, PRG, CSMP, Simscript, GPSS, Dinamo, dan lain-lain. Berdasarkan terapannya, bahasa pemrograman dapat digolongkan atas dua kelompok besar:

1. Bahasa pemrograman bertujuan khusus. Yang termasuk kelompok ini adalah Cobol untuk terapan bisnis dan administrasi. Fortran untuk terapan komputasi ilmiah, bahasa Assembly untuk terapan pemrograman mesin, Prolog untuk terapan kecerdasan buatan, bahasa-bahasa simulasi, dan sebagainya.
2. Bahasa pemrograman bertujuan umum, yang dapat digunakan untuk berbagai aplikasi. Yang termasuk kelompok ini adalah bahasa Pascal, Basic, Java, dan C. Tentu saja pembagian ini tidak kaku. Bahasa bertujuan khusus tidak berarti tidak bisa digunakan untuk aplikasi lain. Cobol misalnya, dapat juga digunakan untuk terapan ilmiah, hanya saja kemampuannya terbatas. Yang jelas, bahasa pemrograman yang berbeda dikembangkan untuk bermacam-macam terapan yang berbeda pula.

Berdasarkan pada tingkat kerumitan sebuah bahasa pemrograman, apakah notasi bahasa pemrograman lebih mendekati bahasa mesin atau ke bahasa manusia, maka bahasa

pemrograman dikelompokkan atas dua macam, yaitu bahasa tingkat tinggi dan bahasa tingkat rendah. Istilah "bahasa pemrograman tingkat tinggi" tidak serta merta menjadikan bahasa tersebut lebih baik dibandingkan dengan bahasa pemrograman tingkat rendah. Akan tetapi, maksud dari "tingkat tinggi" di sini merujuk kepada abstraksi yang lebih tinggi dibandingkan dengan bahasa tingkat rendah terhadap bahasa mesin. Dibandingkan dengan harus berurusan dengan register, alamat memori dan stack-stack panggilan, bahasa pemrograman tingkat tinggi akan berurusan dengan variabel, larik, dan ekspresi aritmetika atau aljabar boolean. Selain itu, tidak seperti bahasa rakitan, bahasa tingkat tinggi tidak memiliki kode operasi yang dapat secara langsung menjadikan bahasa tersebut menjadi kode mesin. Fitur lainnya seperti fungsi-fungsi penanganan string, fitur pemrograman berorientasi objek, input/output terhadap berkas juga terdapat di dalam jenis bahasa ini.

Secara umum, bahasa tingkat tinggi akan membuat pemrograman komputer yang kompleks menjadi lebih sederhana, sementara bahasa tingkat rendah cenderung untuk membuat kode yang lebih efisien. Dalam sebuah bahasa tingkat tinggi, elemen-elemen kompleks dapat dipecah ke dalam beberapa elemen yang lebih sederhana, meski masih dapat dianggap kompleks, di mana bahasa tersebut menyediakan abstraksi. Karena alasan ini, kode-kode yang harus berjalan dengan efisien dapat ditulis dalam bahasa pemrograman tingkat rendah, sementara bahasa tingkat tinggi digunakan untuk mempermudah pemrograman.

Bahasa tingkat rendah dirancang agar setiap instruksinya langsung dikerjakan oleh komputer, tanpa harus melalui penerjemah (translator). Contohnya adalah bahasa mesin. CPU mengambil instruksi dari memori, langsung mengerti dan langsung mengerjakan operasinya. Bahasa tingkat rendah bersifat primitif, sangat sederhana, orientasinya lebih dekat ke mesin, dan sulit dipahami manusia. Bahasa Assembly dimasukkan ke dalam kelompok ini karena alasan notasi yang dipakai dalam bahasanya lebih dekat ke mesin, meskipun untuk melaksanakan instruksinya masih perlu penerjemahan ke dalam bahasa mesin.

Bahasa tingkat tinggi membuat pemrograman lebih mudah dipahami, lebih "manusiawi", dan berorientasi ke bahasa manusia (bahasa Inggris). Hanya saja, program dalam bahasa tingkat tinggi tidak dapat langsung dilaksanakan oleh komputer. Ia perlu diterjemahkan terlebih dahulu oleh sebuah translator bahasa, yang disebut kompilator atau compiler, ke dalam bahasa mesin sebelum akhirnya dieksekusi oleh CPU. Contoh bahasa tingkat tinggi adalah Pascal, PL/I, Ada, Cobol, Basic, Fortran, C, C++, dan sebagainya.



Dengan bertambah rumitnya arsitektur mikroprosesor modern, kompilator-kompilator bahasa pemrograman tingkat tinggi dapat membuat kode yang lebih efisien dibandingkan dengan para programmer bahasa pemrograman tingkat rendah yang melakukannya secara manual. Perlu dicatat bahwa istilah "tingkat tinggi" dan "tingkat rendah" adalah relatif. Pada awalnya, bahasa Assembly dianggap sebagai bahasa tingkat rendah, sementara COBOL, C, dan lain-lainnya dianggap sebagai bahasa tingkat tinggi, mengingat mereka mengizinkan abstraksi terhadap fungsi, variabel, dan evaluasi ekspresi. Akan tetapi, banyak programmer saat ini mungkin menganggap bahasa C sebagai bahasa pemrograman tingkat rendah, mengingat bahasa pemrograman tersebut mengizinkan akses memori secara langsung dengan menggunakan alamatnya, dan juga dapat menggunakan beberapa direktif bahasa Assembly.

## **1.6. Latihan Soal**

1. Apakah yang dimaksud dengan algoritma?
2. Apa perbedaan antara algoritma dan program?
3. Suatu algoritma terdiri dari tiga struktur dasar, yaitu runtunan, pemilihan, dan perulangan. Jelaskan masing-masing!
4. Apa perbedaan antara program dan bahasa pemrograman?
5. Buatlah algoritma menulis dan mengirimkan surat!
6. Buatlah algoritma mengambil uang di ATM!
7. Buatlah algoritma membuat kopi yang rasa manis dan pahitnya pas menurut anda!
8. Buatlah algoritma untuk menghitung luas segitiga!
9. Buatlah algoritma untuk proses aktivitas dari pagi sampai malam!
10. Buatlah algoritma mengurutkan 3 bilangan acak!

# Bab 2

## Notasi Penulisan Algoritma

---

### Pokok Bahasan

1. Jenis penulisan algoritma
2. Tata aturan penulisan algoritma dengan bahasa deskriptif
3. Tata aturan penulisan algoritma dengan pseudocode
4. Notasi flowchart
5. Tata aturan penulisan algoritma dengan flowchart

### Tujuan

1. Menjelaskan tentang pilihan cara penulisan algoritma.
2. Mengetahui tata aturan penulisan algoritma dengan bahasa deskriptif.
3. Memahami tata aturan penulisan algoritma dengan pseudocode.
4. Mengetahu berbagai notasi flowchart.
5. Mengetahui cara penulisan algoritma dengan flowchart.

Algoritma berisi langkah-langkah penyelesaian masalah. Notasi algoritma merupakan hal dasar yang harus diketahui oleh setiap orang yang ingin membuat suatu program, karena dalam notasi algoritma inilah terdapat kerangka-kerangka suatu program. Deskripsi langkah-langkah dalam algoritma tidak mengacu pada sintaks bahasa pemrograman apapun dan tidak tergantung pada spesifikasi komputer yang mengeksekusinya. Tidak ada aturan baku dalam menuliskan algoritma, yang penting mudah dibaca dan menggunakan bahasa yang mudah dipahami. Meskipun demikian untuk menghindari kekeliruan, ketaatan terhadap notasi perlu diperhatikan. Terdapat tiga cara yang umum digunakan dalam menuliskan algoritma yaitu:

1. Kalimat deskriptif
2. Pseudocode
3. Flowchart

Uraian berikutnya akan membahas lebih detail tentang notasi penulisan algoritma disertai dengan contoh.

## **2.1 Kalimat deskriptif**

Notasi penulisan algoritma dengan menggunakan bahasa deskriptif biasa juga disebut dengan notasi alami. Dilakukan dengan cara menuliskan instruksi-instruksi yang harus dilaksanakan dalam bentuk untaian kalimat deskriptif dengan menggunakan bahasa yang jelas. Dasar dari notasi bahasa deskriptif adalah Bahasa Inggris, namun dapat dimodifikasi dengan bahasa sehari-hari termasuk Bahasa Indonesia. Karena tidak ada aturan baku dalam menuliskan algoritma dengan notasi deskriptif maka tiap orang dapat membuat aturan penulisan dan notasi algoritma sendiri. Hal ini dapat dimengerti karena teks algoritma tidak sama dengan teks program. Program adalah implementasi algoritma dalam notasi bahasa pemrograman tertentu. Namun, agar notasi algoritma mudah ditranslasi ke dalam notasi bahasa pemrograman, maka sebaiknya notasi algoritma tersebut berkoresponden dengan notasi bahasa pemrograman pada umumnya. Kata kerja adalah jenis kata yang biasa digunakan dalam penulisan bahasa deskriptif, contohnya tulis, baca, hitung, tampilkan, ulangi, bandingkan, dan lain-lain.

Notasi jenis ini cocok untuk algoritma yang pendek. Tapi untuk masalah algoritma yang panjang, notasi ini kurang efektif. Cara penulisan algoritma dengan notasi bahasa deskriptif paling mudah dibuat, namun demikian cara ini paling sulit untuk diterjemahkan ke dalam bahasa pemrograman. Pada dasarnya teks algoritma dengan bahasa deskriptif disusun oleh tiga bagian utama yaitu:

1. Bagian judul (header)
2. Bagian deklarasi (kamus)
3. Bagian deskripsi

Setiap bagian disertai dengan komentar untuk memperjelas maksud teks yang dituliskan. Komentar adalah kalimat yang diapit oleh pasangan tanda kurung kurawal ('{' dan '}').

### **2.1.1 Judul Algoritma**

Merupakan bagian yang terdiri atas nama algoritma dan penjelasan (spesifikasi) tentang algoritma tersebut. Dibagian ini juga digunakan untuk menentukan apakah teks algoritma yang dibuat tersebut adalah program, prosedur, atau fungsi. Nama algoritma sebaiknya singkat namun cukup menggambarkan apa yang dilakukan oleh algoritma tersebut.

Di bawah nama algoritma disertai dengan penjelasan singkat (intisari) tentang apa yang dilakukan oleh algoritma. Penjelasan dibawah nama algoritma sering dinamakan juga spesifikasi algoritma yang dituliskan dalam kurung kurawal ({}). Algoritma harus ditulis sesuai dengan spesifikasi yang didefinisikan. Gambar 2.1 adalah contoh judul algoritma menghitung luas lingkaran yang disertai dengan penjelasan singkat.

<b>Algoritma Luas_Lingkaran</b> ← Judul Algoritma {Menghitung luas lingkaran untuk ukuran jari-jari tertentu. Algoritma menerima masukan jari-jari lingkaran, menghitung luasnya, lalu cetak luasnya ke piranti keluaran}      ← Spesifikasi
---

Gambar 2.1. Contoh bagian judul algoritma.

### 2.1.2 Bagian Deklarasi

Di dalam algoritma, deklarasi atau kamus adalah bagian untuk mendefinisikan semua nama yang dipakai di dalam algoritma. Nama tersebut dapat berupa nama variabel, nama konstanta, nama tipe, nama prosedur atau nama fungsi. Semua nama tersebut baru dapat digunakan di dalam algoritma jika telah didefinisikan terlebih dahulu didalam bagian deklarasi. Penulisan sekumpulan nama dalam bagian deklarasi sebaiknya dikelompokkan menurut jenisnya. Pendefinisian nama konstanta sekaligus memberikan nilai konstanta. Pendefinisian nama fungsi atau prosedur sekaligus dengan pendefinisian spesifikasi dan parameternya. Gambar 2.2 adalah bentuk umum bagian deklarasi. Sedangkan gambar 2.3 adalah contoh bagian deklarasi algoritma menghitung luas lingkaran.

<b>Deklarasi :</b> {Nama Type variabel yang bukan tipe data dasar} type waktu:<hh:mm:ss: integer> {Type waktu terdiri dari 3 data masukan yaitu "hh" sebagai jam, "mm" sebagai menit, dan "ss" sebagai detik}  {Nama konstanta, harus menyebutkan tipe data dan nilai} constant PHI: real = 3.141592653589793 constant E: real = 2.718281828459045  {Nama variabel yang menggunakan tipe data dasar} nama               : String        {suatu nilai yang merupakan kumpulan karakter} ketemu           : boolean       {suatu nilai logika (true atau false)} beratBadan       : real           {suatu nilai bilangan pecahan} jumlahAnak       : integer       {suatu nilai bilangan bulat}  {Nama fungsi, mendefinisikan domain, nama, jumlah, tipe dan jumlah parameter, serta tipe data keluaran} function tambah(x:int, y:int): int {proses: menambahkan dua nilai data dan hasil penambahan sebagai nilai keluaran fungsi}
---

Gambar 2.2. Bentuk umum bagian deklarasi algoritma.

```
Deklarasi :  
jari_jari = real {tipe data bilangan pecahan}  
luas = real {tipe data bilangan pecahan}  
PHI = 3.14
```

Gambar 2.3. Contoh bagian deklarasi algoritma.

### 2.1.3 Bagian deskripsi.

Deskripsi adalah bagian inti dari struktur algoritma. Bagian ini berisi uraian langkah-langkah penyelesaian masalah. Langkah-langkah ini dituliskan dengan notasi yang lazim dalam penulisan algoritma. Setiap langkah algoritma dibaca dari langkah paling atas hingga langkah paling bawah. Urutan penulisan menentukan urutan pelaksanaan perintah. Seperti telah dijelaskan di bab satu bahwa penyusun atau struktur dasar algoritma adalah langkah-langkah. Suatu Algoritma dapat terdiri dari tiga struktur dasar, yaitu runtunan, pemilihan dan pengulangan. Ketiga jenis langkah tersebut membentuk konstruksi suatu algoritma. Pada bagian deskripsi inilah letak tiga struktur algoritma tersebut.

Gambar 2.3 adalah contoh bagian deskripsi algoritma menghitung luas lingkaran. Gambar 2.4 adalah contoh algoritma menghitung luas lingkaran yang dituliskan menggunakan kalimat deskriptif secara lengkap.

```
Deskripsi :  
1. Baca jari_jari  
2. Hitung luas = jari_jari * jari_jari * PHI  
3. Tampilkan luas ke layar  
4. Selesai
```

Gambar 2.4. Contoh bagian deskripsi algoritma.

```
Algoritma Luas Lingkaran  
{Menghitung luas lingkaran untuk ukuran jari-jari tertentu.  
Algoritma menerima masukan jari-jari lingkaran, menghitung  
luasnya, lalu cetak luasnya ke piranti keluaran}  
  
Deklarasi :  
jari_jari = real {tipe data bilangan pecahan}  
luas = real {tipe data bilangan pecahan}  
PHI = 3.14  
  
Deskripsi:  
1. Baca jari_jari  
2. Hitung luas = PHI*jari_jari * jari_jari  
3. Tampilkan luas ke layar  
4. Selesai
```

Gambar 2.5. Contoh penulisan algoritma menggunakan kalimat deskriptif.

## 2.2 Pseudocode

Pseudocode adalah cara penulisan algoritma yang menyerupai bahasa pemrograman tingkat tinggi. Pseudocode menggunakan bahasa yang hampir menyerupai bahasa pemrograman. Biasanya pseudo-code menggunakan bahasa yang mudah dipahami secara universal dan juga lebih ringkas dari pada algoritma. Pseudocode berisi deskripsi dari algoritma pemrograman komputer yang menggunakan struktur sederhana dari beberapa bahasa pemrograman tetapi bahasa tersebut hanya di tujukan agar dapat di baca manusia. Sehingga pseudocode tidak dapat dipahami oleh komputer. Supaya notasi pseudocode bisa dipahami oleh komputer maka harus diterjemahkan terlebih dahulu menjadi sintaks bahasa pemrograman komputer tertentu.

Dalam pseudocode, tidak ada sintaks standar yang resmi. Karena itu, pseudocode ini dapat diterapkan dalam berbagai bahasa pemrograman. Disarankan untuk menggunakan keyword yang umum digunakan seperti : if, then, else, while, do, repeat, for, dan lainnya. Keuntungan menggunakan notasi pseudocode adalah kemudahan mentranslasi ke notasi bahasa pemrograman, karena terdapat korespondensi antara setiap pseudocode dengan notasi bahasa pemrograman. Tabel 2.1. menunjukkan perbandingan beberapa kata yang biasa digunakan dalam penulisan algoritma dengan menggunakan kalimat deskriptif dan pseudocode.

Tabel 2.1. Perbandingan beberapa kata yang biasa digunakan dalam penulisan algoritma dengan menggunakan kalimat deskriptif dan pseudocode.

Kalimat Deskriptif	Pseudocode
Masukkan panjang	Input panjang
	Read panjang
	Baca panjang
Hitung luas dengan rumus panjang x lebar	luas $\leftarrow$ panjang * lebar
Tampilkan luas	Output luas
	Print luas
	Write luas
Jika sudah selesai, cetak luas	If kondisi_selesai == true then print luas
Nilai B ditambah 5	B $\leftarrow$ B+5
Jika nilai A lebih kecil dari 5 maka nilai B dibagi 3	If A<5 then B $\leftarrow$ B/3

Jika nilai A lebih besar dari nilai B maka tampilkan A, jika A lebih kecil dari B maka tampilkan nilai B	If A>B then print A else print B
--	----------------------------------

Struktur penulisan pseudocode secara umum sama dengan struktur penulisan algoritma dengan menggunakan kalimat deskriptif yaitu dimulai dari judul/header, deklarasi/kamus dan diakhiri dengan deskripsi. Meskipun tidak ada sintaks khusus dalam penulisan pseudocode, tetapi terkadang pseudocode dituliskan dengan menggunakan style atau gaya penulisan dari beberapa bahasa pemrograman yang ada, seperti Fortran, Pascal, C dan lain-lain. Gambar 2.5 adalah contoh penulisan pseudocode dengan menggunakan gaya penulisan beberapa bahasa pemrograman. Gambar 2.6 adalah contoh pseudocode menentukan bilangan terbesar dari 3 masukan bilangan. Sedangkan Gambar 2.7 adalah contoh pseudocode konversi nilai angka menjadi nilai huruf.

Fortran style	Pascal style	C style
<pre> program TikTok do i=0to100     set flag to true if i is divisible by 3     print "Tik"     set flag to false if i is divisible by 5     print "Tok"     set flag to false if flag, print i     print a newline end do </pre>	<pre> procedure TikTok for i:=0to100 do     set flag to true; if i is divisible by 3 then     print "Tik";     set flag to false; if i is divisible by 5 then     print "Tok";     set flag to false; if flag, print i;     print a newline; end </pre>	<pre> void function TikTok for(i=0;i&lt;=100;i++){     set flag to true; if i is divisible by 3     print "Tik";     set flag to false; if i is divisible by 5     print "Tok";     set flag to false; if flag, print i;     print a newline; } </pre>

Gambar 2.5. Contoh pseudocode menggunakan gaya penulisan Fortran, Pascal dan C.

<p><b>Algoritma : Bilangan Maksimum</b>  {Dibaca tiga buah bilangan dari piranti masukan. Carilah bilangan bulat maksimum diantara ketiga bilangan tersebut}</p> <p><b>Deklarasi :</b>  Bil1,Bil2,Bil3 : integer {bilangan yang dicari maksimumnya}  Max : integer {variabel bantu}</p> <p><b>Deskripsi :</b>  1. Read (Bil1,Bil2)  2. If Bil1 &gt;= Bil2 then  3.     Bil1 = Max  4. Else Bil2 = Max  5. Read (Bil3)  6. If Bil3 &gt;= Max then  7.     Bil3 = Max  8. Write (Max)</p>
---

Gambar 2.6 Contoh pseudocode menentukan bilangan terbesar dari 3 masukan bilangan.

**Algoritma Konversi Nilai Angka ke Huruf**

{Dibaca tiga buah bilangan dari piranti masukan. Carilah dan tampilkan bilangan bulat maksimum diantara ketiga bilangan tersebut}

**Deklarasi:**

nama dan nim = String

nilai = integer

**Deskripsi:**

1. Read (nama)
2. Read (nim)
3. Read (nilai)
4. If (nilai < 45) then
5.     Grade = E
6. Else if (nilai >= 45) and (nilai < 60) then
7.     Grade = D
8. Else if (nilai >= 60) and (nilai < 70) then
9.     Grade = C
10. Else if (nilai >= 70) and (nilai < 80) then
11.     Grade = B
12. Else
13.     Grade = A
14. Write (nama)
15. Write (NIM)
16. Write (nilai)
17. Selesai

Gambar 2.7. Contoh pseudocode konversi nilai angka menjadi nilai huruf.

## 2.3 Flowchart

Flowchart adalah cara penulisan algoritma dengan menggunakan notasi grafis. Flowchart merupakan gambar atau bagan yang memperlihatkan urutan atau langkah-langkah dari suatu program dan hubungan antar proses beserta pernyataannya. Gambaran ini dinyatakan dengan simbol. Dengan demikian setiap simbol menggambarkan proses tertentu. Sedangkan antara proses digambarkan dengan garis penghubung. Dengan menggunakan flowchart akan memudahkan kita untuk melakukan pengecekan bagian-bagian yang terlupakan dalam analisis masalah. Disamping itu flowchart juga berguna sebagai fasilitas untuk berkomunikasi antara pemrogram yang bekerja dalam tim suatu proyek. Flowchart



menolong analis dan programmer untuk memecahkan masalah kedalam segmen-segmen yang lebih kecil dan menolong dalam menganalisis alternatif-alternatif lain dalam pengoperasian.

Pada dasarnya terdapat berbagai macam flowchart, diantaranya yaitu Flowchart Sistem (System Flowchart), Flowchart Paperwork / Flowchart Dokumen (Document Flowchart), Flowchart Skematik (Schematic Flowchart), Flowchart Program (Program Flowchart), Flowchart Proses (Process Flowchart). Untuk keperluan pembuatan program maka digunakan Flowchart Program.

Flowchart program menggambarkan urutan instruksi yang digambarkan dengan simbol tertentu untuk memecahkan masalah dalam suatu program. Dalam flowchart program mengandung keterangan yang lebih rinci tentang bagaimana setiap langkah program atau prosedur seharusnya dilaksanakan. Flowchart ini menunjukkan setiap langkah program atau prosedur dalam urutan yang tepat saat terjadi. Programmer menggunakan flowchart program untuk menggambarkan urutan instruksi dari program komputer. Analis Sistem menggunakan flowchart program untuk menggambarkan urutan tugas-tugas pekerjaan dalam suatu prosedur atau operasi.

Dalam pembuatan flowchart program tidak ada rumus atau patokan yang bersifat mutlak. Karena flowchart merupakan gambaran hasil pemikiran dalam menganalisis suatu masalah yang nantinya akan diubah menjadi program komputer. Sehingga flowchart yang dihasilkan dapat bervariasi antara satu pemrogram dengan yang lainnya. Namun demikian terdapat beberapa anjuran yang harus diperhatikan, yaitu :


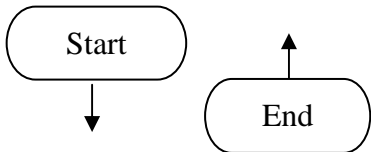

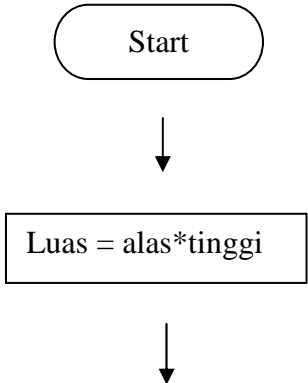
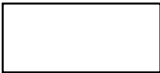
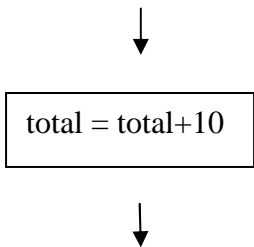
1. Flowchart digambarkan di suatu halaman dimulai dari sisi atas ke bawah dan dari sisi kiri ke kanan.
2. Aktivitas yang digambarkan harus didefinisikan dengan menggunakan bahasa dan simbol yang tepat dan definisi ini harus dapat dimengerti oleh pembacanya.
3. Kapan aktivitas dimulai dan berakhir harus ditentukan secara jelas. Hanya terdapat satu titik awal dan satu titik akhir.
4. Setiap langkah dari aktivitas harus diuraikan dengan menggunakan deskripsi kata kerja, misalkan MENGHITUNG NILAI RATA-TARA.
5. Setiap langkah dari aktivitas harus berada pada urutan yang benar.
6. Lingkup dan range dari aktifitas yang sedang digambarkan harus ditelusuri dengan hati-hati. Percabangan-percabangan yang memotong aktivitas yang sedang digambarkan tidak perlu digambarkan pada flowchart yang sama. Simbol konektor harus digunakan dan

percabangannya diletakan pada halaman yang terpisah atau hilangkan seluruhnya bila percabangannya tidak berkaitan dengan sistem.

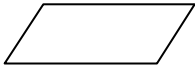
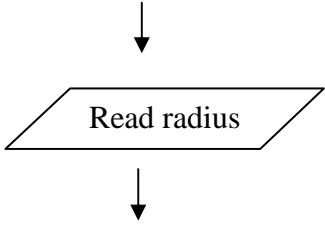
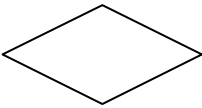
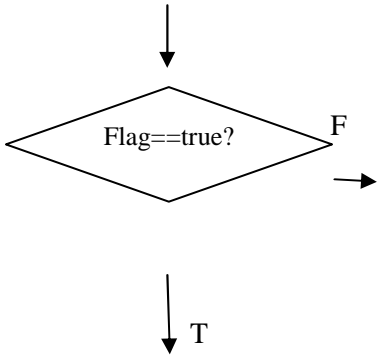
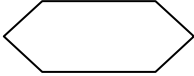
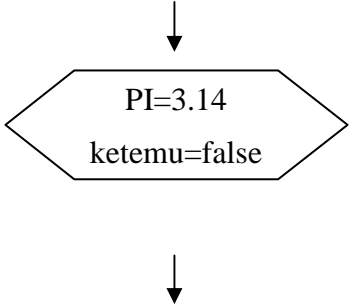
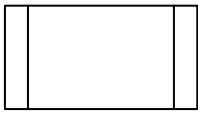
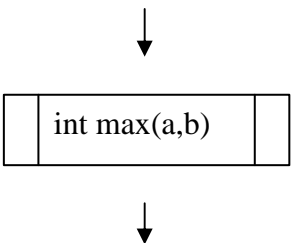


7. Gunakan simbol-simbol flowchart yang standar.

Simbol-simbol flowchart yang biasanya dipakai adalah simbol-simbol flowchart standar yang dikeluarkan oleh ANSI dan ISO. Tabel 2.2 merupakan beberapa simbol flowchart yang digunakan dalam menggambar suatu flowchart:

Tabel 2.2.Simbol-simbol Flowchart

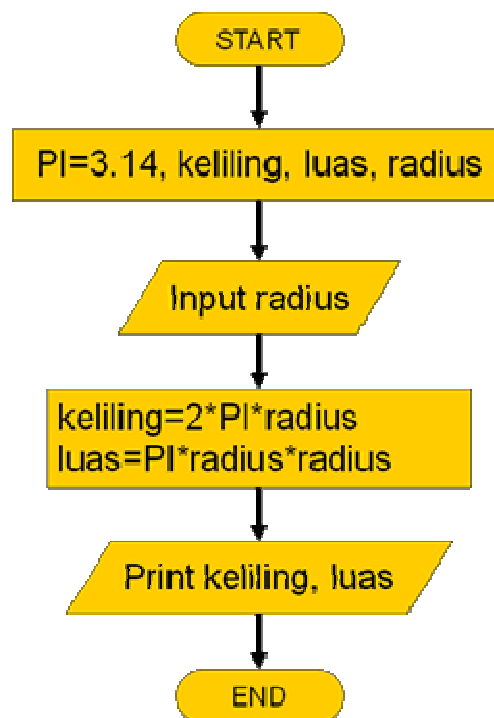
SIMBOL	NAMA	FUNGSI	CONTOH
	Terminator	Simbol Awal (Start) / Simbol Akhir (End)	
	Flow Line	Simbol aliran / penghubung	
	Proses	Perhitungan / pengolahan	

Tabel 2.2.Simbol-simbol Flowchart (lanjutan)

SIMBOL	NAMA	FUNGSI	CONTOH
	Input / Output Data	Mempresentasikan pembacaan data (read) / penulisan (write).	
	Decision	Simbol pernyataan pilihan, berisi suatu kondisi yang selalu menghasilkan 2 nilai keluaran yaitu benar atau salah	
	Preparation	Inisialisasi / pemberian nilai awal	
	Predefined Process (subprogram)	Proses menjalankan sub program / fungsi / prosedur	
	On Page Connector	Penghubung Flow chart pada satu halaman	

			↓ (A)
	Off Page Connector	Penghubung Flow chart pada halaman berbeda	 ↓ 

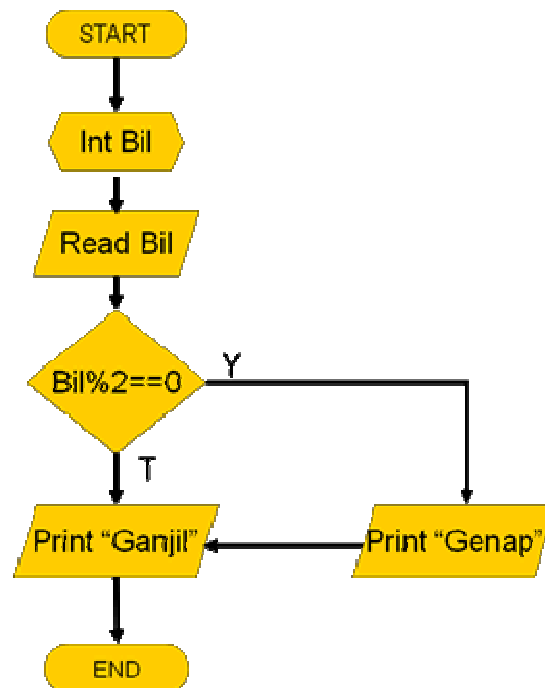
Untuk memahami lebih dalam mengenai flowchart ini, dibuat sebuah kasus sederhana. Misalnya buatlah sebuah rancangan program dengan menggunakan flowchart untuk menentukan keliling dan luas lingkaran. Perumusan untuk menentukan luas lingkaran adalah:  $\text{luas} = \text{PI} \times \text{radius} \times \text{radius}$ , dan keliling lingkaran adalah  $\text{keliling} = 2 \times \text{PI} \times \text{radius}$ , dengan PI adalah sebuah konstanta 3.14. Flowchart permasalahan ini dapat dilihat di Gambar 2.8.



Gambar 2.8. Flowchart luas dan keliling lingkaran

Selanjutnya akan dibuat contoh flowchart yang mengandung percabangan atau decision. Misalnya untuk permasalahan penentuan apakah suatu bilangan termasuk bilangan ganjil atau genap. Cara menyelesaikan permasalahan ini adalah dengan membagi bilangan

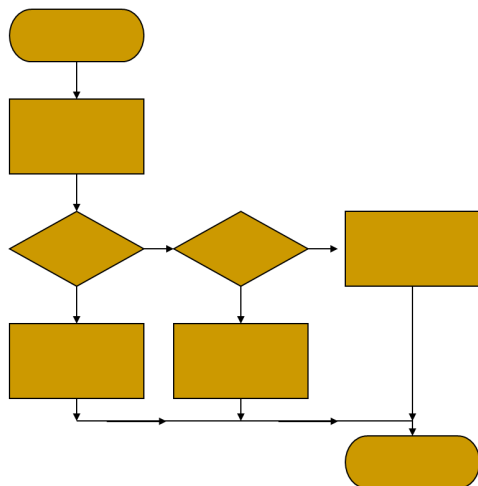
dengan angka 2. Jika nilai sisa pembagian nya adalah 0 maka bilangan tersebut adalah bilangan genap, jika nilai sisa pembagiannya adalah 1 maka bilangan tersebut adalah bilangan ganjil. Operasi aritmatika yang digunakan untuk menentukan nilai sisapembagian adalah perasi modulo (%). Flowchart permasalahan ini dapat dilihat di Gambar 2.9. Dalam hal ini Bil adalah bilangan yang akan di cek status ganjil atau genapnya.



Gambar 2.9. Flowchart Penentuan Bilangan Ganjil-Genap

## 2.4 Latihan Soal

1. Buatlah algoritma untuk menentukan nilai terbesar dari bilangan bulat yang dibaca dari keyboard dan menuliskan hasilnya ke layar! ALgoritma dibuat dalam bentuk kalimat deskriptif, pseudocode dan flowchart.
2. Buat algoritma dalam bahasa deskriptif dan flowchart untuk kegiatan mengambil dan menabung uang di bank melalui teller!
3. Buat algoritma dalam bahasa deskriptif dan pseudocode untuk menentukan apakah suatu bilangan merupakan bilangan genap atau ganjil!
4. Membuat flowchart untuk proses pembuatan kopi yang rasa manisnya tepat
5. 2. membuat flowchart mengurutkan 3 bilangan acak
6. Buat algoritma untuk menghitung nilai N suku dari deret aritmatika berikut:

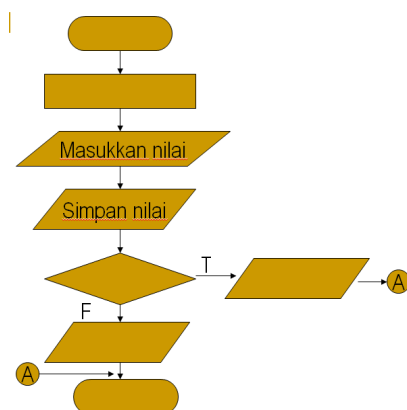


1. Apakah suara bell pintu?
2. Start
3. Angkat dan menjawab telepon
4. Mematikan alarm
5. Apakah suara telepon?
6. Mendengar suara alarm berbunyi
7. Stop
8. Membuka dan memeriksa siapa yang berada didepan pintu.

$$S_n = 3+7+11+\dots+(4n-1)$$

7. Lengkapi penulisan flowchart berikut ini:
8. Analisalah potongan algoritma dibawah ini! Apakah tujuan algoritma ini? Selanjutnya ubahlah potongan algoritma berikut ini menjadi sebuah flowchart!

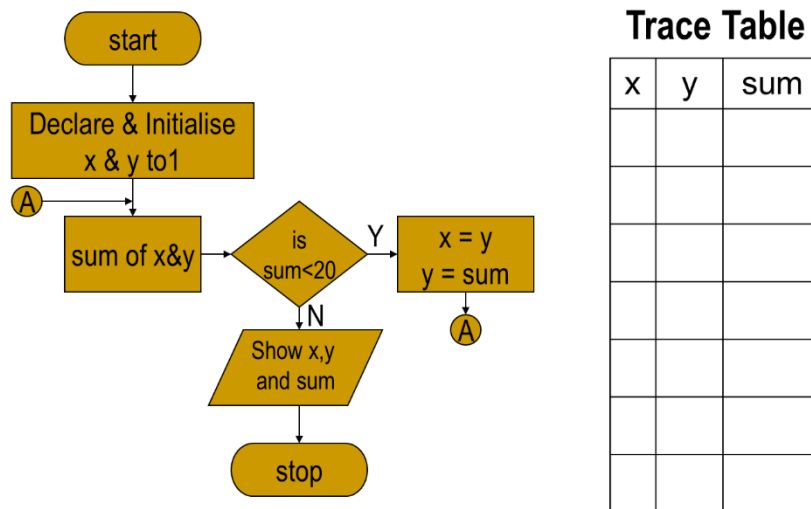
1. **const PI := 3.14**
2. **var radius, volume : real**
3. **put "please enter..."..**
4. **get radius**
5. **volume := (4/3)\*PI\*radius\*\*3**
6. **put ""**
7. **put "The volume is", volume**



1. Memulai dan mendeklarasikan variabel nilai
2. Berapa besarnya nilai?
3. Mematikan alarm
4. Cek apakah nilai  $\geq 50$
5. Jika nilai  $\geq 50$  tampilkan "LULUS"
6. Jila nilai  $< 50$  tampilkan "GAGAL"
7. Stop

9. Lengkapi kerangka flowchart dibawah ini berdasarkan potongan algoritma yang ada!

10. Analisalah flowchart dibawah ini dan selanjutnya lengkapilah trace table!.



# Bab 3

## Struktur Data Dalam Algoritma

---

### Pokok Bahasan

1. Definisi struktur data dan tipe data serta jenis-jenisnya
2. Pengertian konstanta dan variabel
3. Pengertian Array
4. Pengertian Stack
5. Pengertian Queue
6. Pengertian Tree
7. Pengertian Graph

### Tujuan

1. Memahami pengertian struktur data dan tipe data serta mengetahui jenis-jenisnya.
2. Memahami pengertian, cara mendeklarasikan dan memakai konstanta dan variabel.
3. Memahami pengertian, cara mendeklarasikan dan menggunakan array.
4. Memahami pengertian, cara mendeklarasikan, operasi dan menggunakan stack.
5. Memahami pengertian, cara mendeklarasikan, operasi dan menggunakan queue.
6. Memahami pengertian Tree dan contoh penggunaannya.
7. Memahami pengertian Pengertian Graph dan contoh penggunaannya.

Dalam istilah ilmu komputer, struktur data adalah cara penyimpanan, penyusunan dan pengaturan data di dalam media penyimpanan komputer sehingga data tersebut dapat digunakan secara efisien. Dengan kata lain struktur data adalah sebuah skema organisasi, seperti variabel dan array dan lain-lain, yang diterapkan pada data sehingga data dapat diinterpretasikan dan sehingga operasi-operasi spesifik dapat dilaksanakan pada data tersebut. Pemakaian struktur data yang tepat didalam proses pemrograman akan menghasilkan algoritma yang lebih jelas dan tepat, sehingga menjadikan program secara keseluruhan lebih efisien dan sederhana. Data adalah representasi dari fakta dunia nyata.



Fakta atau keterangan tentang kenyataan yang disimpan, direkam atau direpresentasikan dalam bentuk tulisan, suara, gambar, sinyal atau simbol. Data merupakan suatu nilai yang bisa dinyatakan dalam bentuk konstanta atau variabel. Konstanta menyatakan nilai yang tetap, sedangkan variabel menyatakan nilai yang dapat diubah-ubah selama eksekusi berlangsung.

### **3.1 Tipe Data**

Setiap data memiliki tipe data, apakah merupakan angka bulat, angka pecahan, atau berupa karakter, dan sebagainya. Jadi, tipe data adalah pengelompokan data berdasarkan isi dan sifatnya. Dalam bidang informatika tipe data adalah jenis data yang dapat diolah oleh komputer untuk memenuhi kebutuhan dalam pemrograman komputer.

Setiap variabel atau konstanta yang ada dalam kode program, sebaiknya kita tentukan dengan pasti tipe datanya. Ketepatan pemilihan tipe data pada variabel atau konstanta akan sangat menentukan pemakaian sumberdaya komputer (terutama memori komputer). Salah satu tugas penting seorang programmer adalah memilih tipe data yang sesuai untuk menghasilkan program yang efisien dan berkinerja tinggi. Ada banyak tipe data yang tersedia, tergantung jenis bahasa pemrograman yang dipakai. Secara garis besar tipe data dapat dikategorikan menjadi tiga macam yaitu tipe data dasar (primitive data type) tipe data bentukan (composite data type) dan tipe data abstrak (abstract data type).

#### **3.1.1 Tipe dasar**

Tipe data dasar atau tipe data sederhana atau biasa juga disebut dengan tipe data primitif adalah tipe data yang sudah ada dan dijadikan standar dalam bahasa pemrograman tertentu. Isi dari tipe data sederhana ini adalah data-data tunggal. Tipe data dasar sudah disediakan oleh program sehingga programmer bisa langsung memakai.

##### **1. Integer (Bilangan Bulat)**

Yang dimaksud bilangan bulat adalah, -1, -2, -3, 0, 1, 2, 3, 4 dan lain lain yang bukan merupakan bilangan pecahan.

##### **2. Float atau double (Bilangan Real)**

Bilangan real adalah bilangan yang mengandung pecahan desimal. Contoh : 3.45, 6,233.

##### **3. Char (Karakter)**

Karakter adalah semua huruf yang ada di dalam alfabet, tanda baca maupun karakter spesial. Karakter ditulis diantara dua tanda petik tunggal. Contoh : 'A'.

#### 4. Boolean (logika)

Boolean adalah tipe data logika yang terdiri dari dua pernyataan benar atau salah. Pernyataan benar biasanya ditulis True atau angka 1, sedangkan pernyataan salah ditulis dengan False atau angka 0. Sedangkan operasi aritmatika yang umum digunakan adalah or, not, and dan xor.

### 3.1.2 Tipe data bentukan

Tipe data bentukan atau tipe data komposit adalah tipe data yang dibentuk dari tipe data dasar dengan maksud mempermudah pekerjaan programmer. Yang masuk dalam tipe data bentukan adalah array, string, record, union, struct, dan lain-lain. Tujuan dibuatnya tipe data bentukan adalah :

1. Mempermudah proses pemrograman
2. Mempermudah dalam penambahan variabel
3. Mempermudah pengelompokan data sehingga lebih teratur.

### 3.1.3 Tipe data abstrak (Abstract Data Type)

Tipe data abstrak atau yang dikenal sebagai Abstract Data Type adalah model matematika dari obyek data yang menyempurnakan tipe data dengan cara mengaitkannya dengan fungsi-fungsi yang beroperasi pada data yang bersangkutan. Tipe data abstrak adalah tipe data yang didefinisikan sendiri oleh pemrogram untuk suatu keperluan tertentu yang tidak memungkinkan untuk mendeklarasikan dari tipe data yang sudah ada. Contoh tipe data abstrak adalah stack, queue, list, tree, graph, dan lain-lain.

Harus dibedakan antara pengertian struktur data dan tipe data abstrak. Struktur data hanya memperlihatkan bagaimana data-data di organisir, sedangkan tipe data abstrak mengemas struktur data tertentu sekaligus dengan operasi-operasi yang dapat dilakukan pada struktur data tersebut. Dengan demikian, definisi umum tentang tipe data abstrak dapat dipahami bahwa tipe data abstrak adalah struktur data yang mengandung operasi-operasi atau aturan-aturan tertentu. Pada sub bab selanjutnya akan dibahas beberapa jenis tipe data dari tipe data – tipe data yang telah disebutkan sebelumnya.

## 3.2 Konstanta dan Variabel

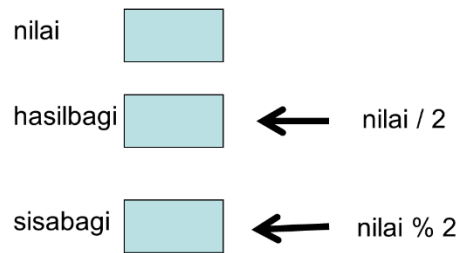
Konstanta dan variabel adalah suatu pengenal (identifier) yang digunakan untuk mewakili suatu nilai tertentu didalam proses program. Berbeda dengan konstanta yang nilainya tidak bisa diubah atau selalu tetap selang eksekusi berlangsung, nilai dari suatu variabel dapat berubah sesuai kebutuhan. Konstanta dan variabel merupakan tempat di memori komputer untuk menyimpan data berupa nilai dengan tipe data tertentu. Konstanta dan variabel harus diberi nama sebagai identifikasi.

Secara logika dapat dibayangkan sebuah konstanta atau variabel sebagai sebuah kotak kosong yang dapat diisi dengan sesuatu tipe data 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. Ilustrasi logika ini dapat dilihat pada Gambar 3.1. Pada Gambar 3.1 dimisalkan membuat dua buah konstanta atau variabel dengan nama identifier **nilai** dan **X** yang masing-masing dapat digunakan untuk menyimpan suatu nilai dalam memori sesuai dengan tipe data yang telah ditentukan.

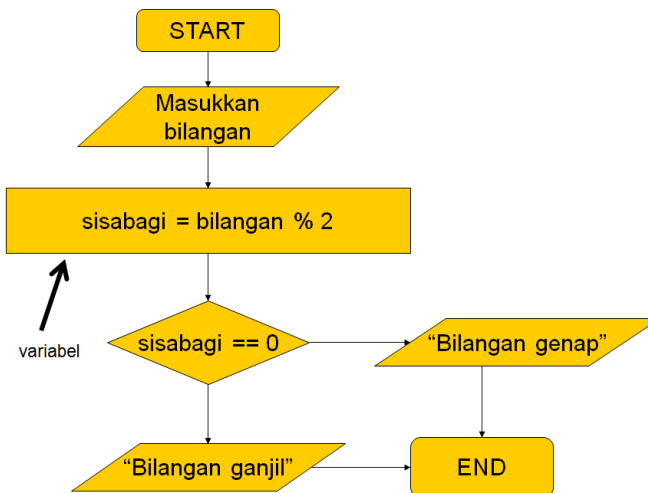


Gambar 3.1. Ilustrasi pembuatan konstanta atau variabel

Semisal **nilai** di Gambar 3.1 adalah sebuah variabel. Maka variabel nilai yang telah dibuat ini selanjutnya dapat digunakan dalam program, semisal dilakukan operasi aritmatika berupa operasi pembagian (/) atau modulo (%). Gambar 3.2 menunjukkan ilustrasi operasi yang terjadi pada variabel nilai. Dalam hal ini variabel nilai dibagi dengan angka 2, dan hasil operasi pembagian disimpan dalam variabel baru yang bernama **hasilbagi**. Operasi aritmatika lain yang terjadi pada Gambar 3.2 adalah variabel nilai dimodulo dengan angka 2 dan hasilnya disimpan dalam variabel baru yang bernama **sisabagi**. Operasi pada suatu konstanta atau variabel tidak hanya terbatas pada operasi aritmatika saja, tetapi juga dapat berupa operasi perbandingan. Misalnya nilai apakah lebih besar dari angka 10 ( $\text{nilai} > 10$ ) dan lain-lain. Contoh penggunaan variabel nilai dalam permasalahan sederhana adalah penentuan suatu bilangan termasuk dalam kategori ganjil atau genap seperti terlihat pada Gambar 3.3.



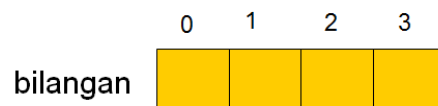
Gambar 3.2. Ilustrasi penggunaan variabel



Gambar 3.3. Contoh penggunaan variabel

### 3.3 Array

Array adalah suatu alokasi beberapa tempat di memori yang tersimpan secara berurutan yang digunakan untuk menyimpan beberapa nilai dengan tipe data yang homogen. Ukuran atau jumlah elemen maksimum array telah diketahui dari awal yaitu ketika array dibuat. Sekali ukuran array ditentukan maka tidak dapat diubah. Ukuran array adalah bilangan bulat positif. Array harus diberi nama sebagai identifikasi. Cara mengaksesnya adalah dengan menyebutkan nama array dan indeksinya. Indeks array dimulai dari 0 sampai dengan  $n-1$  ( $n$  adalah ukuran array). Ilustrasi array dapat dilihat pada Gambar 3.4.

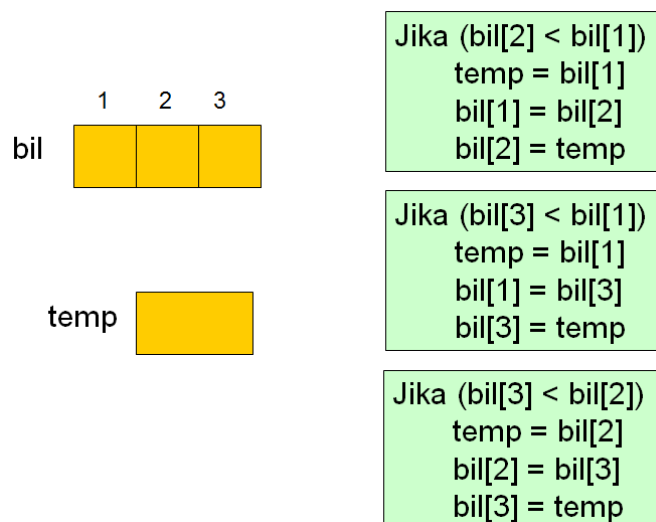


Gambar 3.4. Ilustrasi array

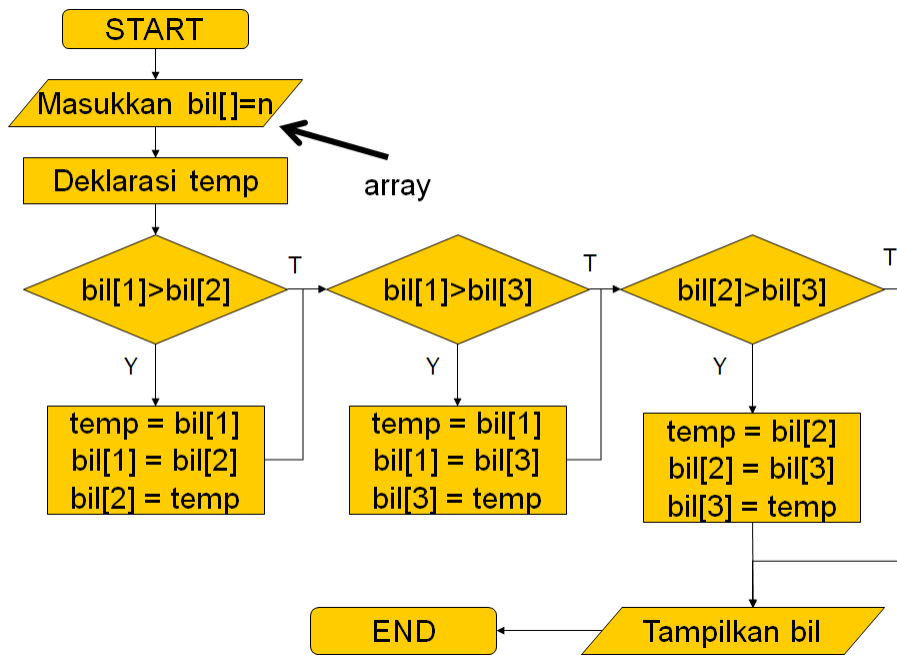
Operasi terhadap elemen array dilakukan dengan pengaksesan langsung. Artinya nilai di masing-masing posisi elemen dapat diambil dan nilai dapat disimpan tanpa melewati posisi-posisi lain. Dua operasi paling dasar terhadap satu elemen array adalah penyimpanan nilai elemen ke posisi tertentu di array dan pengambilan nilai elemen dari posisi tertentu di array. Biasanya bahasa pemrograman menyediakan sintaks tertentu untuk penyimpanan dan pengambilan nilai elemen pada posisi tertentu di array. Contohnya

- NilaiMhs[7] = 80, berarti menyimpan nilai 80 ke posisi ke-7 dari array NilaiMhs.
- Nama = Mahasiswa[20], berarti mengambil nilai elemen posisi ke-20 dari array Mahasiswa dan menyimpan nilai tersebut ke variabel yang bernama "Nama".

Contoh cara mengakses elemen array yang lain dapat dilihat di Gambar 3.5. Contoh penggunaan array dalam permasalahan sederhana adalah pengurutan 3 buah bilangan seperti terlihat pada Gambar 3.6. Untuk mengurutkan tiga buah bilangan dibutuhkan operasi perbandingan yang menghasilkan kondisi benar atau salah (> atau <).



Gambar 3.5. Cara mengakses elemen array



Gambar 3.6. Contoh penggunaan array

Keunggulan array adalah sebagai berikut:

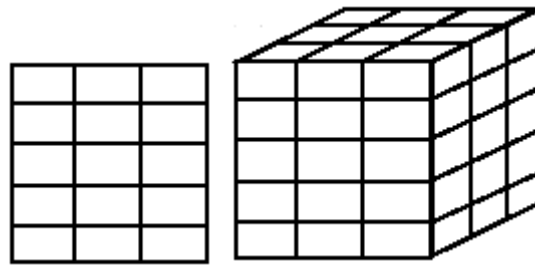
1. Array sangat cocok untuk pengaksesan acak. Sembarang elemen di array dapat diacu secara langsung tanpa melalui elemen-elemen lain.
2. Jika telah berada di suatu lokasi elemen, maka sangat mudah menelusuri ke elemen-elemen tetangga, baik elemen pendahulu atau elemen penerus.

Kelemahan array adalah sebagai berikut:

1. Array mempunyai fleksibilitas rendah, karena array mempunyai batasan harus bertipe homogen. Kita tidak dapat mempunyai array dimana satu elemen adalah karakter, elemen lain bilangan, dan elemen lain adalah tipe-tipe lain
2. Kebanyakan bahasa pemrograman mengimplementasikan array dengan ukuran statik yang sulit diubah ukurannya di waktu eksekusi. Bila penambahan dan pengurangan terjadi terus-menerus, maka representasi statis ini bersifat tidak efisien dalam penggunaan memori.

Dalam bidang pemrograman, array dapat dibuat dengan berbagai macam dimensi. Semisal array dimensi satu akan mirip dengan pola linier seperti pada Gambar 3.4. Sedangkan array 2 dimensi akan tampak seperti tabel atau matrik. Cara mengaksesnya adalah

dengan menyebutkan nama matrik serta baris dan kolomnya. Sedangkan array 3 dimensi akan tampak seperti balok. Ilustrasi array 2 dan 3 dimensi dapat dilihat pada Gambar 3.7.

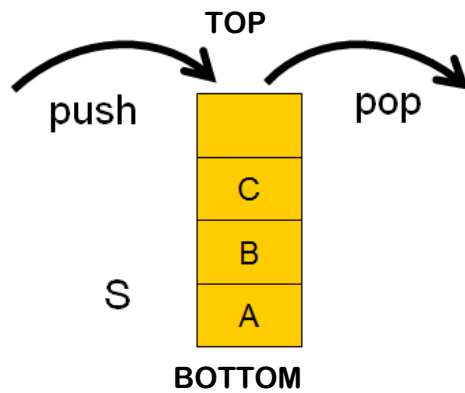


Gambar 3.7. Ilustrasi array 2 dimensi (kiri) dan array 3 dimensi (kanan)

### 3.4 Stack

Secara bahasa, stack berarti tumpukan. Jika dikaitkan dengan struktur data, stack berarti sekumpulan data yang strukturnya menyerupai tumpukan. Stack harus diberi nama sebagai identifikasi. Konsep penyimpanan data pada stack menganut sistem "yang terakhir masuk sebagai yang pertama keluar" (*Last In First Out* / LIFO). Dengan konsep ini, urutan pengambilan data akan berkebalikan dengan urutan penyimpanan data. Elemen yang terakhir disimpan akan menjadi yang pertama kali diambil. Dengan konsep ini maka kita tidak dapat mengambil data yang tersimpan dalam stack secara acak. Data dalam stack harus disimpan dan diambil dari satu sisi atau satu pintu saja. Contoh dalam kehidupan sehari-hari adalah tumpukan piring di sebuah restoran yang tumpukannya dapat ditambah pada bagian paling atas dan jika mengambilnya pun dari bagian paling atas pula.

Cara mengakses stack adalah dengan melakukan operasi dasar pada stack yaitu operasi **push** dan **pop**. Dimisalkan pintu keluar masuknya data pada stack disebut dengan TOP sebagaimana dilihat pada Gambar 3.8. Operasi push adalah proses memasukkan data baru ke stack melalui pintu TOP sehingga data terbaru akan terletak pada posisi teratas. Operasi pop adalah proses mengeluarkan atau mengambil data dari stack dan data yang diambil adalah data yang terletak di posisi teratas. Dengan konsep LIFO maka ketika operasi push dilakukan maka informasi yang diperlukan hanyalah isi atau nilai atau elemen yang akan disimpan atau diambil saja. Operasi push dan pop tidak memerlukan informasi posisi data. Sebagai ilustrasi, stack dapat dilihat seperti pada Gambar 3.8, tampak bahwa stack tersebut bernama S.

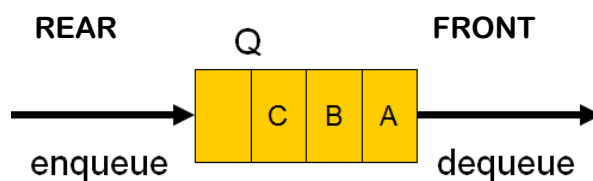


Gambar 3.8. Ilustrasi stack

Bagian Top pada stack merupakan pintu untuk keluar masuknya data – data stack. A, B, dan C merupakan suatu koleksi. Dari ilustrasi dapat diketahui bahwa C merupakan data yang terakhir memasuki stack namun pertama keluar dari stack. Begitu sebaliknya dengan A, A merupakan data pertama yang memasuki tumpukan namun terakhir saat keluar dari tumpukan. Di dalam gambar juga terlihat urutan masuk dan keluar yang berkebalikan. Data yang masuk pertama akan keluar terakhir dan sebaliknya.

### 3.5 Queue

Secara bahasa queue adalah antrian. Queue adalah suatu kumpulan data dengan operasi pemasukan atau penyimpanan data hanya diperbolehkan pada salah satu sisi, yang disebut sisi belakang (rear) dan operasi pengambilan atau penghapusan hanya diperbolehkan pada sisi lainnya yang disebut sisi depan (front). Konsep ini dikenal dengan istilah Last In First Out (LIFO). Ilustrasi queue dapat dilihat pada Gambar 3.9.



Gambar 3.9. Ilustrasi queue

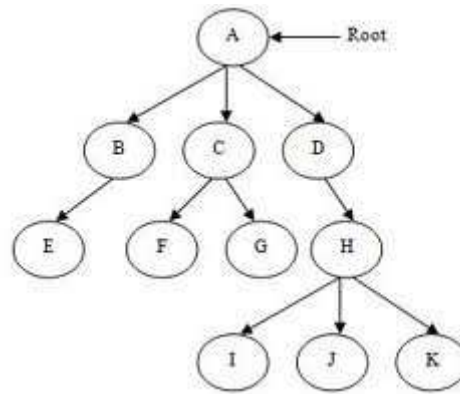


Jenis struktur data queue sering digunakan untuk menstimulasikan keadaan dunia nyata. Antrian banyak dijumpai dalam kehidupan sehari-hari. Contoh yang paling populer untuk membayangkan sebuah queue adalah antrian pada kasir sebuah bank. Ketika seorang pelanggan datang, akan menuju ke belakang dari antrian. Setelah pelanggan dilayani, antrian yang berada di depan akan maju. Pada saat menempatkan data pada ujung (*rear*) dari queue disebut dengan *enqueue*, pada saat memindahkan elemen dari kepala (*front*) sebuah queue disebut dengan *dequeue*. Sama dengan stack, Dengan konsep FIFO maka ketika operasi enqueue dilakukan maka informasi yang diperlukan hanyalah isi atau nilai atau elemen yang akan disimpan atau diambil saja. Operasi enqueue dan dequeue tidak membutuhkan informasi posisi data.

Dari ilustrasi di Gambar 3.9 dapat diketahui bahwa C merupakan data yang terakhir memasuki queue Q dan akan menjadi yang paling akhir keluar dari queue. Begitu sebaliknya dengan A, A merupakan data pertama yang memasuki queue dan akan menjadi yang pertama saat keluar dari queue.

### 3.6 Tree

Tree merupakan salah satu bentuk struktur data tidak linear yang menggambarkan hubungan yang bersifat hirarki (hubungan one to many) antara elemen-elemen. Bentuk tree menyerupai sebuah pohon, yang terdiri dari serangkaian node (simpul) yang saling berhubungan. Node-node tersebut dihubungkan oleh sebuah vektor. Sehingga tree bisa didefinisikan sebagai kumpulan simpul atau node dengan elemen khusus yang disebut root atau akar. Ilustrasi tree dapat dilihat di Gambar 3.10. Contoh data yang dapat direpresentasikan dengan menggunakan tree adalah silsilah keluarga, hasil pertandingan yang berbentuk turnamen, atau struktur organisasi dari sebuah perusahaan.



Gambar 3.10. Ilustrasi tree

Dalam pemrograman, sebuah tree terdiri dari elemen-elemen yang dinamakan *node* (simpul) yang mana hubungan antar simpul bersifat hirarki. Contoh node pada Gambar 3.10 adalah node A, node B, node C dan seterusnya sampai dengan node K. Jadi Gambar 3.10 memiliki node sebanyak 11 node. Node yang paling atas dari hirarki dinamakan *root*, yaitu node A. Simpul yang berada di bawah root secara langsung, dinamakan anak dari root, yang mana biasanya juga mempunyai anak di bawahnya. Sehingga bisa disimpulkan, kecuali root, masing-masing simpul dalam hirarki mempunyai satu induk (parent). Jumlah anak sebuah simpul induk sangat bergantung pada jenis dari pohon.

Setiap node dapat memiliki 0 atau lebih node anak (child). Sebuah node yang memiliki node anak disebut node induk (parent). Sebuah node anak hanya memiliki satu node induk. Sesuai konvensi ilmu komputer, tree bertumbuh ke bawah, tidak seperti pohon di dunia nyata yang tumbuh ke atas. Dengan demikian node anak akan digambarkan berada di bawah node induknya. Node yang berada di pangkal tree disebut node root (akar), sedangkan node yang berada paling ujung tree disebut node leaf (daun).

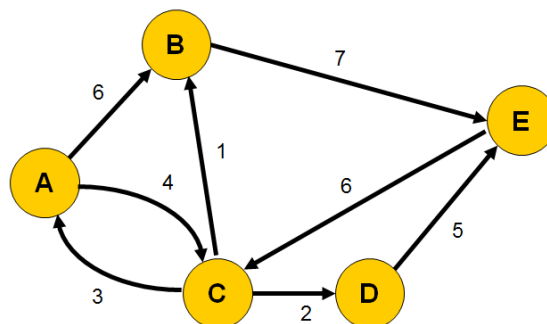
### 3.7 Graph

Dalam bidang matematika dan ilmu komputer, graph adalah struktur yang menggambarkan relasi antar obyek dari sebuah koleksi obyek. Jika struktur linear; misalnya array; memungkinkan pendefinisian keterhubungan sekuensial antara entitas data, struktur data tree memungkinkan pendefinisian keterhubungan hirarkis, maka struktur graph memungkinkan pendefinisian keterhubungan tak terbatas antara entitas data.

Banyak obyek dalam masalah-masalah nyata secara alamiah memiliki keterhubungan langsung secara tak terbatas. Contohnya informasi topologi dan jarak antar kota-kota di suatu pulau. Dalam masalah ini kota x bisa berhubungan langsung dengan hanya satu atau lima kota lainnya. Untuk memeriksa keterhubungan dan jarak tidak langsung antara dua kota dapat diperoleh berdasarkan data keterhubungan langsung dari kota-kota lainnya yang memperantarainya. Contoh lain penggunaan graph adalah untuk menggambarkan jaringan dan jalur kereta api, lintasan pesawat, sistem permipaan, saluran telepon, koneksi elektrik, ketergantungan diantara *task* pada sistem manufaktur dan lain-lain. Terdapat banyak hasil dan struktur penting yang didapatkan dari perhitungan dengan graph.

Representasi data dengan struktur data linear ataupun hirarkis pada masalah ini bisa digunakan namun membutuhkan operasi-operasi yang rumit sehingga kurang efisien. Struktur data graph secara eksplisit menyatakan keterhubungan ini sehingga pencariannya langsung dilakukan pada strukturnya sendiri.

Definisi dari suatu graph adalah himpunan obyek-obyek yang disebut node (atau vertek) yang terhubung oleh edge. Biasanya graph digambarkan secara grafis sebagai kumpulan lingkaran yang melambangkan node yang dihubungkan oleh garis yang melambangkan edge. Edge dalam suatu graph bisa berupa edge berarah atau tidak berarah. Ilustrasi graph dapat dilihat pada Gambar 3.11. Pada gambar tersebut terlihat bahwa graph memiliki 5 buah node. Pada ilustrasi ini dimisalkan node mewakili sebuah kota. Maka dapat dilihat bahwa dari kota A menuju kota E bisa dilalui melalui path A-B-E atau path A-C-D-E.

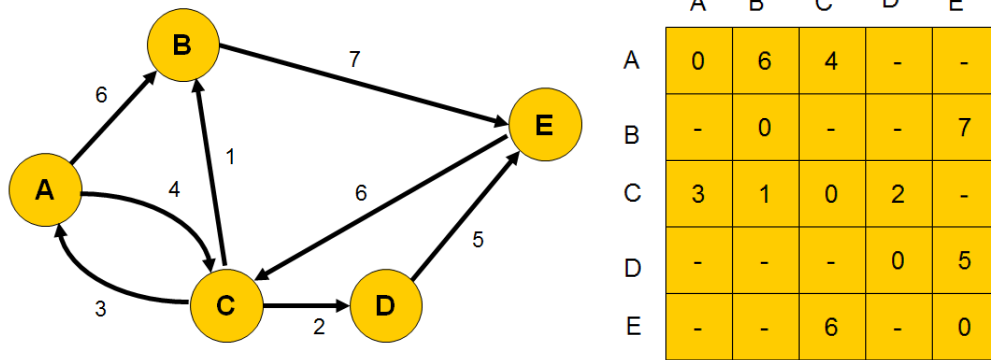


Gambar 3.11. Ilustrasi graph

Contoh lain masalah-masalah yang dapat diselesaikan dengan menggunakan struktur data graph adalah:

- a. Masalah path minimum (Shortest path problem), yaitu mencari route dengan jarak terpendek dalam suatu jaringan transportasi.
- b. Masalah aliran maksimum (maximum flow problem), yaitu menghitung volume aliran BBM dari suatu reservoir ke suatu titik tujuan melalui jaringan pipa.
- c. Masalah pencariiah dalam graph (graph searching problem) yaitu mencari langkah-langkah terbaik dalam program permainan catur komputer.
- d. Masalah pengurutan topologis (topological ordering problem), yaitu menentukan urutan pengambilan mata kuliah yang saling berkaitan dalam hubungan prasyarat.
- e. Masalah jaringan tugas (Task Network Problem), yaitu membuat penjadwalan pengerjaan suatu proyek yang memungkinkan waktu penyelesaian tersingkat.
- f. Masalah pencarian pohon rentang minimum (Minimum Spanning Tree Problem), yaitu mencari rentangan kabel listrik yang totalnya adalah minimal untuk menghubungkan sejumlah kota.
- g. Travelling Salesperson Problem, yaitu tukang pos mencari lintasan terpendek melalui semua alamat penerima pos tanpa harus mendatangi suatu tempat lebih dari satu kali.
- h. Four-color proble, yaitu dalam menggambar peta, memberikan warna yang berbeda pada setiap propinsi yang saling bersebelahan.

Untuk menyelesaikan permasalahan jalur terpendek (sortest path) dari graph pada Gambar 3.11. Maka harus dilakukan penerjemahan dari bentuk graph ke bentuk matrik keterhubungan langsung (list adjancency) yang dibuat dengan menggunakan array dua dimensi. Hasil pembuatan list adjancency graph dapat dilihat pada Gambar 3.12. Nilai-nilai yang tertera dalam list tersebut adalah jarak antara dua kota. Misalnya nilai 6 yang terletak pada posisi baris 1 kolom 2. Artinya adalah ada jalur dari kota A ke kota B dan jaraknya 6 satuan. Kebalikannya dari kota B ke kota A (baris 2 kolom 1) tidak ada jalur sehingga bernilai -. Untuk mendapatkan jalur terpendek dan nilai jaraknya dapat diselesaikan dengan menggunakan beberapa algoritma sortest path, misalnya algoritma Dijkstra, yang akan dibahas di bab 11.



Gambar 3.12. List adjancency

### 3.8 Latihan Soal

1. Apakah yang dimaksud dengan tipe data?
2. Sebutkan dan jelaskan penggolongan tipe data!
3. Apakah yang dimaksud dengan struktur data?
4. Apakah yang dimaksud dengan konstanta dan variabel?
5. Apakah beda antara konstanta dan variabel?
6. Apakah yang dimaksud dengan array? Jelaskan!
7. Apakah yang dimaksud dengan stack? Jelaskan operasi pada stack dan beri contoh penerapannya dalam kehidupan sehari-hari!
8. Apakah yang dimaksud dengan queue? Jelaskan operasi pada stack dan beri contoh penerapannya dalam kehidupan sehari-hari!
9. Apakah yang dimaksud dengan tree? Berikan contoh penerapannya!
10. Apakah yang dimaksud dengan graph? Berikan contoh penerapannya!

# Bab 4

## Studi Kasus Permasalahan Sederhana

---

### Pokok Bahasan

1. Menghitung luas dan keliling lingkaran
2. Konversi suhu
3. Menampilkan deret bilangan ganjil

### Tujuan

1. Memahami algoritma menghitung luas dan keliling lingkaran
2. Memahami algoritma konversi suhu
3. Memahami algoritma menampilkan deret bilangan ganjil
4. Menjelaskan pemakaian logika seleksi menampilkan deret bilangan ganjil dengan syarat tertentu.

### 4.1 Studi Kasus 1: Menghitung luas dan keliling lingkaran

#### 4.1.1 Permasalahan

Buatlah flowchart untuk menghitung luas dan keliling lingkaran! Panjang jari-jari dimasukkan oleh pengguna. Luas dan keliling lingkaran ditampilkan ke layar.

Contoh:

Masukkan panjang jari-jari :7

Luas lingkaran: 153.94

Keliling lingkaran : 43.98

#### 4.1.2 Cara Penyelesaian Masalah

Untuk menghitung luas dan keliling lingkaran maka dibutuhkan data-data berupa radius lingkaran dan nilai konstanta  $\pi$  (kadang kala ditulis PI). Secara matematis rumus menghitung luas dan keliling lingkaran adalah:

Luas lingkaran =  $\pi$  \* radius \* radius

Keliling lingkaran =  $2 * \pi$  \* radius

Dalam kasus ini nilai  $\pi$  adalah sebuah konstanta dalam matematika yang merupakan perbandingan keliling lingkaran dengan diameternya. Nilai  $\pi$  dalam 20 tempat desimal adalah 3.14159265358979323846.

#### 4.1.3 Struktur Data Yang Dibutuhkan

- **PI** sebagai konstanta yang didefinisikan bertipe data pecahan dan bernilai 3.14
- **radius** sebagai variabel yang menampung panjang radius lingkaran yang didefinisikan bertipe data pecahan
- **luas** sebagai variabel penampung hasil perhitungan luas lingkaran
- **keliling** sebagai variabel penampung hasil perhitungan luas lingkaran

#### Deklarasi dan Inisialisasi

PI didefinisikan sebagai konstanta dan diinisialisasi sebagai sebuah angka dengan 2 angka dibelakang koma (jumlah angka dibelakang koma bersifat optional), sehingga PI bernilai 3.14. Variabel radius, luas dan keliling juga dideklarasikan bertipe pecahan. Karena bilangan yang dimasukkan oleh pengguna nilainya terserah pengguna (fleksibel) maka radius dideklarasikan sebagai variabel. Variabel luas dan keliling dideklarasikan sebagai variabel karena nilai yang tersimpan didalamnya adalah hasil kalkulasi rumus luas dan keliling lingkaran yang nilainya dapat berubah tergantung dengan nilai radius yang dimasukkan oleh pengguna. Dari sini, deklarasi dan inisialisasi konstanta dan variabel pada flowchart dapat dibuat sebagai berikut:

PI=3.14, keliling, luas, radius

#### 4.1.4 Input

Masukan yang diperlukan dalam permasalahan ini adalah bilangan yang melambangkan panjang radius lingkaran dan disimpan dalam variabel radius, sehingga input pada flowchart dapat dibuat sebagai berikut:

Inputradius

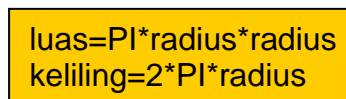
#### 4.1.5 Output

Keluaran permasalahan ini adalah 2 buah nilai yaitu luas dan keliling lingkaran, sehingga output pada flowchart dapat dibuat sebagai berikut:

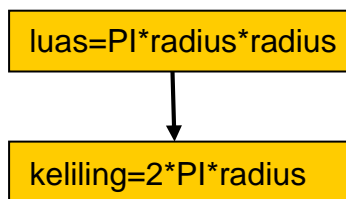


#### 4.1.6 Proses Penyelesaian

Terdapat dua buah kegiatan yang terjadi dalam proses dan dilakukan secara berurutan (sequences), yaitu perhitungan luas dan keliling lingkaran sesuai rumus yang telah dideklarasikan sebelumnya. Dalam penulisan flowchart dua kegiatan ini bisa digabung seperti berikut ini

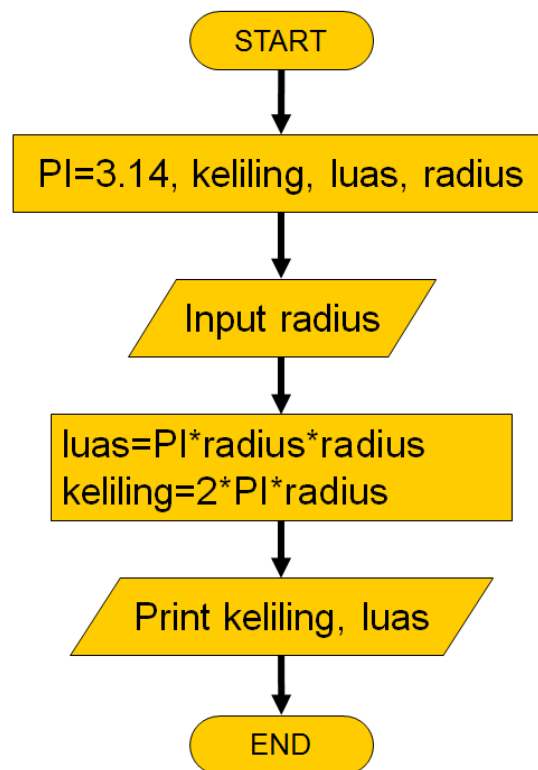


atau dipisah seperti berikut





#### 4.1.7 Flowchart Keseluruhan



## 4.2 Studi Kasus 2: Konversi suhu

### 4.2.1 Permasalahan

Buatlah flowchart untuk mengkonversi suhu dari Celcius ke Fahrenheit! Suhu dalam satuan Celcius dimasukkan oleh pengguna. Suhu dalam satuan Fahrenheit ditampilkan ke layar.

Contoh:

Masukkan suhu dalam satuan Celcius :28

Suhu dalam satuan Fahrenheit: 153.94

### 4.2.2 Cara Penyelesaian Masalah

Untuk menghitung konversi suhu dari satuan Celcius menjadi Fahrenheit dibutuhkan data-data berupa nilai suhu dalam satuan Celcius. Secara matematis rumus menghitung konversi suhu dari satuan Celcius menjadi Fahrenheit adalah:

$$F = C * 1.8 + 32$$

#### 4.2.3 Struktur Data Yang Dibutuhkan

- **F** sebagai variabel yang didefinisikan bertipe data pecahan untuk menyimpan suhu dalam satuan Fahrenheit.
- **C** sebagai variabel yang didefinisikan bertipe data pecahan untuk menyimpan suhu dalam satuan Celcius.

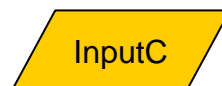
#### 4.2.4 Deklarasi dan Inisialisasi

Variabel F dan C dideklarasikan bertipe pecahan karena sifat nilai besaran suhu yang dapat direpresentasikan dalam bilangan pecahan. Nilai C dimasukkan oleh pengguna dan nilainya terserah pengguna (fleksibel). Variabel F dideklarasikan sebagai variabel karena nilai yang tersimpan didalamnya adalah hasil kalkulasi rumus konversi suhu yang nilainya dapat berubah tergantung dengan nilai C yang dimasukkan oleh pengguna. Dari sini, deklarasi variabel pada flowchart dapat dibuat sebagai berikut:



#### 4.2.5 Input

Masukan yang diperlukan dalam permasalahan ini adalah bilangan yang melambangkan suhu dalam satuan Celcius dan disimpan dalam variabel C, sehingga input pada flowchart dapat dibuat sebagai berikut:



#### 4.2.6 Output

Keluaran permasalahan ini adalah sebuah nilai yaitu suhu dalam satuan Fahrenheit, sehingga output pada flowchart dapat dibuat sebagai berikut:

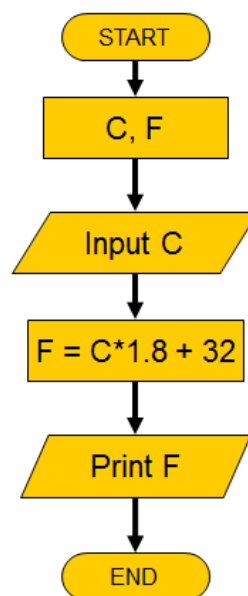


#### 4.2.7 Proses Penyelesaian

Proses yang terjadi adalah perhitungan konversi suhu dari Celcius menjadi Fahrenheit dengan menggunakan rumus yang telah dideklarasikan sebelumnya. Dari sini, flowchart Input dapat dibuat sebagai berikut:

$$F = C * 1.8 + 32$$

#### 4.2.8 Flowchart Keseluruhan



### 4.3 Studi Kasus 3: Menampilkan bilangan ganjil

#### 4.3.1 Permasalahan

Buatlah flowchart untuk menampilkan sederetan bilangan ganjil dari 10 sampai 30 kecuali 21 dan 27!

Contoh:

Output : 11 13 15 17 19 23 25 29

#### 4.3.2 CaraPenyelesaian Masalah

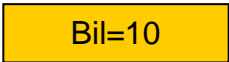
Karena batas atas dan batas bawah bilangan sudah didefinisikan dalam soal maka pengguna tidak perlu memberi masukan ke sistem. Dalam flowchart cukup didefinisikan bahwa bawah atas bilangan adalah 10 dan batas atas bilangan adalah 30. Flowchart akan menampilkan bilangan ganjil saja sehingga harus dilakukan pengecekan kondisi apakah bilangan termasuk bilangan ganjil. Untuk itu dilakukan operasi aritmatika yang melibatkan operator % (modulo), jika bilangan di modulo 2 menghasilkan nilai 1 maka bilangan tersebut adalah bilangan ganjil dan harus ditampilkan ke layar, sedang bila hasilnya adalah 0 maka bilangan tersebut adalah bilangan genap dan tidak perlu ditampilkan ke layar. Tetapi harus ditambahkan pengecekan kondisi sekali lagi yaitu bahwa tidak semua bilangan ganjil akan ditampilkan ke layar, bilangan 21 dan 27 tidak ditampilkan. Oleh karena itu tambahkan pengecekan apakah bilangan=21 atau bilangan=27, jika iya maka tidak perlu ditampilkan ke layar. Karena lebih dari satu bilangan yang di cek apakah ganjil atau genap dan apakah bilangan sama dengan 21 atau sama dengan 27 maka dalam penyelesaian masalah dibutuhkan perulangan proses.

#### 4.3.3 Struktur Data Yang Dibutuhkan

Untuk menyelesaikan permasalahan ini hanya dibutuhkan satu variabel saja yaitu **Bil** sebagai variabel yang menyimpan nilai bilangan yang akan ditampilkan ke layar.

#### 4.3.4 Deklarasi dan Inisialisasi

Variabel terdiri dari **Bil** saja dan nilainya diinisialisasi dengan 10 yang menandakan batas bawah bilangan. Variabel **Bil** dibuat sebagai variabel karena nilai **Bil** akan selalu berubah yaitu selalu ditambah dengan angka 1 karena harus mengecek semua bilangan antara 10 sampai dengan 30. Perubahan nilai **Bil** ini terjadi dalam perulangan proses. Dari sini, deklarasi dan inisialisasi variabel pada flowchart dapat dibuat sebagai berikut:



Bil=10

#### 4.3.5 Input

Dalam flowchart ini tidak diperlukan masukan sama sekali karena batas atas dan batas bawah bilangan sudah didefinisikan dalam soal.

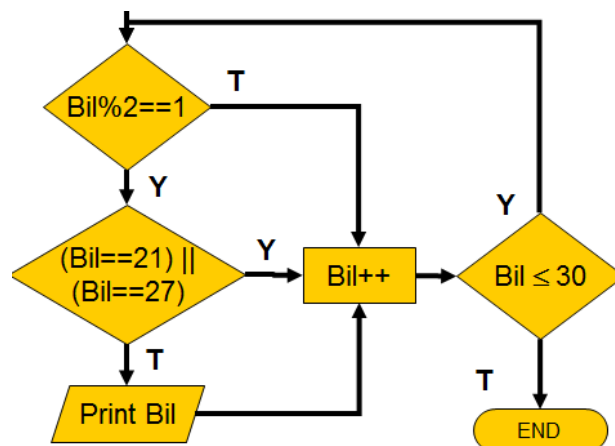
#### 4.3.6 Output

Keluaran permasalahan ini adalah sejumlah bilangan ganjil antara 10 dan 30 kecuali 21 dan 27. Bilangan ini akan di tampilkan beberapa kali sesuai kondisi yang ada sehingga kegiatan menampilkan bilangan ke layar dilakukan dalam perulangan proses, sehingga output pada flowchart dapat dibuat sebagai berikut:

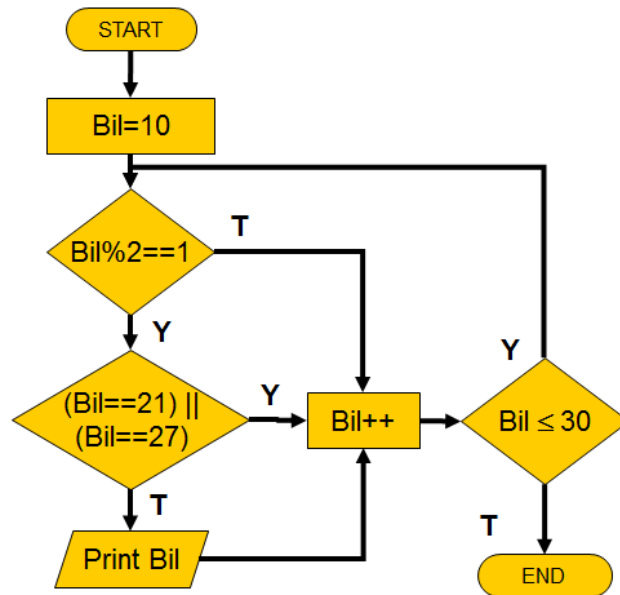


#### 4.3.7 Proses Penyelesaian

Sebagai dijelaskan pada sebelumnya, flowchart akan menampilkan bilangan ganjil saja sehingga harus dilakukan pengecekan kondisi apakah bilangan termasuk bilangan ganjil. Untuk itu dilakukan operasi aritmatika yang melibatkan operator % (modulo), jika bilangan di modulo 2 menghasilkan nilai 1 maka bilangan tersebut adalah bilangan ganjil dan harus ditampilkan ke layar, sedang bila hasilnya adalah 0 maka bilangan tersebut adalah bilangan genap dan tidak perlu ditampilkan ke layar. Tetapi harus ditambahkan pengecekan kondisi sekali lagi yaitu bahwa tidak semua bilangan ganjil akan ditampilkan ke layar, bilangan 21 dan 27 tidak ditampilkan. Oleh karena itu tambahkan pengecekan apakah bilangan=21 atau bilangan=27, jika iya maka tidak perlu ditampilkan ke layar. Karena lebih dari satu bilangan yang di cek apakah ganjil atau genap dan apakah bilangan sama dengan 21 atau sama dengan 27 maka dalam penyelesaian masalah dibutuhkan perulangan proses. Dari sini, flowchart Proses dapat direpresentasikan sebagai berikut:



#### 4.3.8 Flowchart Keseluruhan



#### 4.4 Soal Latihan

Buatlah flowchart untuk menghitung determinan dan mencari akar-akar dari persamaan kuadrat :  $ax^2 + bx + c = 0$ , dengan ketentuan sbb :

$$D = b^2 - 4ac$$

- Jika  $D = 0$  , maka terdapat 2 akar real yang kembar, yaitu:

$$x_1 = x_2 = -b / 2a$$

- Jika  $D > 0$  , maka terdapat 2 akar real yang berlainan, yaitu:

$$x_1 = (-b + \sqrt{D}) / 2a$$

$$x_2 = (-b - \sqrt{D}) / 2a$$

- Jika  $D < 0$  , maka terdapat 2 akar imaginair yang berlainan, yaitu:

$$x_1 = -b / 2a + (\sqrt{-D} / 2a) i$$

$$x_2 = -b / 2a - (\sqrt{-D} / 2a) i$$

# Bab 5

## Studi Kasus Perbandingan

---

### Pokok Bahasan

1. Tahun kabisat
2. Deret bilangan genap

### Tujuan

1. Memahami logika proses perhitungan tahun kabisat
2. Memahami logika proses penentuan deret bilangan genap
3. Menjelaskan pemakaian operasi perbandingan dan operasi perhitungan sisa hasil bagi untuk mengecek tahun kabisat
4. Menjelaskan pemakaian operasi perbandingan, operasi perhitungan sisa hasil bagi, dan operasi increment untuk menentukan deret bilangan genap

### 5.1 Contoh Kasus 1: Tahun kabisat

#### 5.1.1 Permasalahan

Buatlah program untuk menentukan apakah suatu bilangan merupakan tahun kabisat atau bukan.

#### 5.1.2 Proses Penyelesaian Masalah:

Sebuah tahun disebut tahun kabisat jika bilangan tahun tersebut habis dibagi 4.

#### 5.1.3 Input:

Input yang dibutuhkan untuk program ini adalah:

- Tahun dalam bentuk angka (integer)

#### 5.1.4 Output:

Program ini akan menghasilkan output yang berupa:

- Nilai boolean. Menampilkan kalimat “Tahun Kabisat” jika angka yang dimasukkan merupakan tahun kabisat, dan kalimat “Bukan Tahun Kabisat” jika tidak.

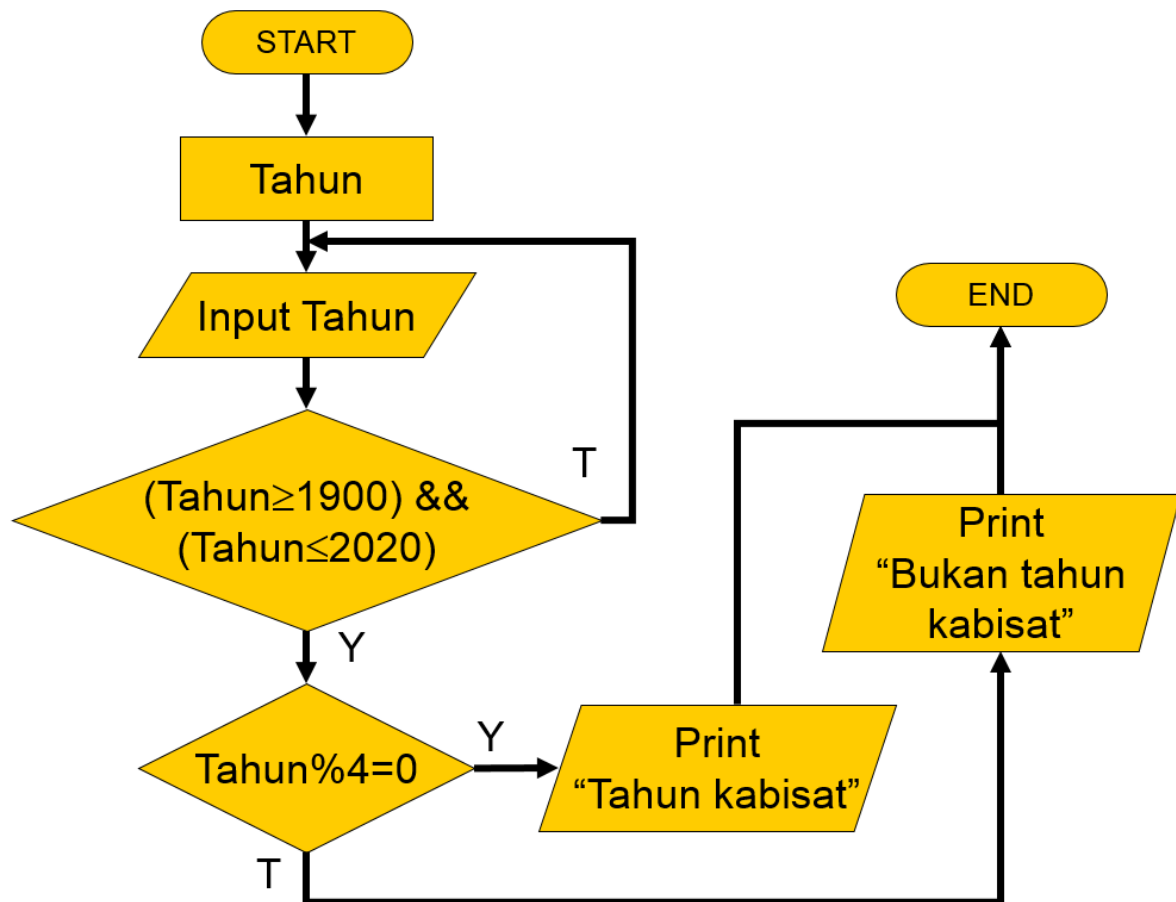
### 5.1.5 Struktur Data yang Dibutuhkan:

- Variabel bertipe integer untuk menyimpan data tahun --> variabel *tahun*

### 5.1.6 Logika Pemrograman:

- Operasi perbandingan menggunakan operator ' $>$ ' dan ' $<$ '.
- Operasi perhitungan sisa hasil bagi (modulus) menggunakan operator ' $\%$ '

### 5.1.7 Flowchart:



## 5.2 Contoh Kasus 2: Deret bilangan genap

### 5.2.1 Permasalahan

Buatlah program yang dapat menampilkan bilangan genap dari 2 sampai n deret kecuali bilangan genap tersebut kelipatan 4.

#### Contoh:

Input:            n: 5  
output:          2, 6, 10, 14, 18



### 5.2.2 Cara Penyelesaian Masalah:

Sebuah bilangan disebut bilangan genap jika bilangan tersebut habis dibagi dengan 2. Dan bilangan tersebut bukan kelipatan 4 jika bilangan tersebut tidak habis dibagi dengan 4.

### 5.2.3 Input:

Input yang dibutuhkan untuk program ini adalah:

- Jumlah bilangan yang ingin ditampilkan dalam bentuk angka

### 5.2.4 Output:

Program ini akan menghasilkan output yang berupa:

- Bilangan genap yang bukan kelipatan 4 sebanyak  $n$  ( $n$  adalah angka yang diterima sebagai masukan).

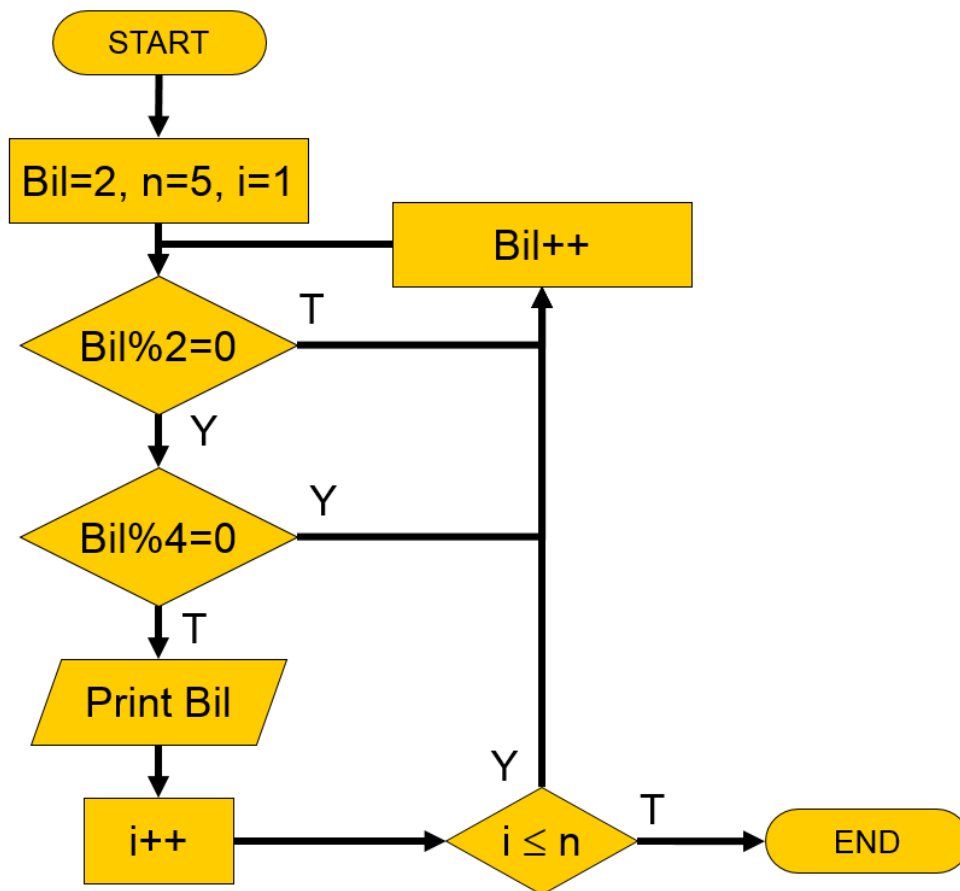
### 5.2.5 Struktur Data yang Dibutuhkan:

- Variabel bertipe integer untuk menyimpan nilai masukan --> variabel  $n$
- Variabel bertipe integer untuk mencatat jumlah bilangan yang sudah ditemukan --> variabel  $i$
- Variabel bertipe integer untuk menyimpan bilangan yang akan dicek --> variabel  $Bil$

### 5.2.6 Logika Pemrograman:

- Operasi perbandingan menggunakan operator '='
- Operasi perhitungan sisa hasil bagi (modulus) menggunakan operator '%'
- Operator increment '++'

### 5.2.7 Flowchart:



### 5.3 Soal Latihan:

Buatlah flowchart untuk menentukan harga yang harus dibayar oleh seorang pembeli bila setiap pembelian barang mendapatkan diskon dengan aturan:

- Jika total harga pembelian > 1.500.000,- maka dapat diskon 10%

# Bab 6

## Studi Kasus Konversi

---

### Pokok Bahasan

3. Konversi jam ke menit
4. Konversi detik ke hari, jam, menit, dan detik

### Tujuan

5. Memahami logika proses konversi jam ke menit
6. Memahami logika proses konversi detik ke hari, jam, menit, dan detik
7. Menjelaskan pemakaian operasi penjumlahan dan perkalian untuk melakukan konversi jam ke menit
8. Menjelaskan pemakaian operasi penjumlahan dan perkalian untuk melakukan konversi detik ke hari, jam, menit, dan detik

### 6.1 Contoh Kasus 1: Konversi jam ke menit

#### 6.1.1 Permasalahan

Buatlah program untuk mengkonversi dari jam ke menit.

#### *Contoh*

Input:            Masukkan jam: 11  
                    Masukkan menit: 7  
Output:           11 jam 7 menit setara dengan 667 menit

#### 6.1.2 Cara Penyelesaian Masalah:

Untuk mengkonversi dari jam ke menit adalah dengan mengalikan nilai jam dengan 60 kemudian ditambahkan dengan nilai menitnya.

#### 6.1.3 Input:

Input yang dibutuhkan untuk program ini adalah:

- Jam dalam bentuk angka (integer)

- Menit dalam bentuk angka (integer)

#### 6.1.4 Output:

Program ini akan menghasilkan output yang berupa:

- Jumlah menit dalam bentuk angka

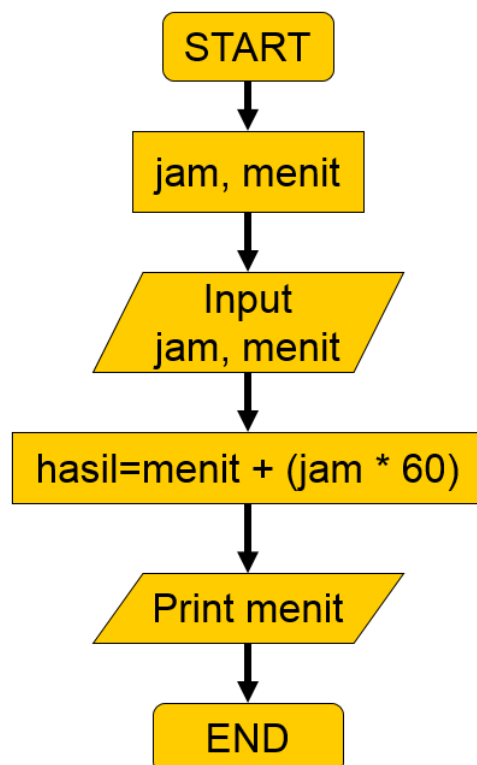
#### 6.1.5 Struktur Data yang Dibutuhkan:

- Variabel bertipe integer untuk menyimpan data jam --> variabel *jam*
- Variabel bertipe integer untuk menyimpan data menit --> variabel *menit*
- Variabel bertipe integer untuk menyimpan data hasil --> variabel *hasil*

#### 6.1.6 Logika Pemrograman:

- Operasi penjumlahan menggunakan operator '+'
- Operasi perkalian menggunakan operator '\*'

#### 6.1.7 Flowchart:



## 6.2 Contoh Kasus 2: Konversi detik ke hari, jam, menit, dan detik

### 6.2.1 Permasalahan

Buatlah program untuk mengkonversi dari detik ke hari, jam, menit, dan detik.

### **Contoh**

Input:           Masukkan detik: 189005  
Output:         189005 detik setara dengan 2 hari, 4 jam, 30 menit, dan 5 detik

#### **6.2.2 Cara Penyelesaian Masalah:**

Untuk mengkonversi dari detik ke hari, jam, menit, dan detik dapat dilakukan dengan 4 tahap. Tahap pertama adalah menghitung nilai detik. Nilai detik merupakan nilai input (dalam detik) yang tidak bisa dikonversikan kedalam menit. Dengan kata lain, nilai detik adalah nilai sisa hasil bagi dari nilai input dibagi 60 (1 menit adalah 60 detik). Misal jika input adalah 130 detik, maka nilai detik adalah 10. Untuk menghitung sisa hasil bagi dapat dilakukan menggunakan operator ‘%’ (modulus).

Tahap kedua menghitung nilai menit. Sebelum kita dapat menghitung nilai menit, kita perlu menghitung jumlah total menit terlebih dahulu. Jumlah total menit adalah nilai total detik (input yang didapat adalah nilai total detik) dibagi dengan 60, menggunakan operator ‘/’. Kemudian nilai menit bisa didapat dengan menghitung sisa hasil bagi dari jumlah total menit. Nilai menit adalah nilai sisa hasil bagi dari jumlah total menit dibagi 60 (1 jam adalah 60 menit).

Tahap ketiga menghitung nilai jam. Terlebih dulu kita harus menghitung jumlah total jam. Jumlah total jam adalah jumlah total menit dibagi dengan 60, menggunakan operator ‘/’. Kemudian nilai jam didapat dengan menghitung sisa hasil bagi dari jumlah total jam dibagi dengan 24 (1 hari adalah 24 jam).

Tahap keempat menghitung nilai hari. Nilai hari didapat dengan membagi jumlah total jam dengan 24 menggunakan operator ‘/’.

#### **6.2.3 Input:**

Input yang dibutuhkan untuk program ini adalah:

- Nilai detik Jam dalam bentuk angka (integer)

#### **6.2.4 Output:**

Program ini akan menghasilkan output yang berupa:

- Jumlah hari, jam, menit, dan detik dalam bentuk angka

#### **6.2.5 Struktur Data yang Dibutuhkan:**

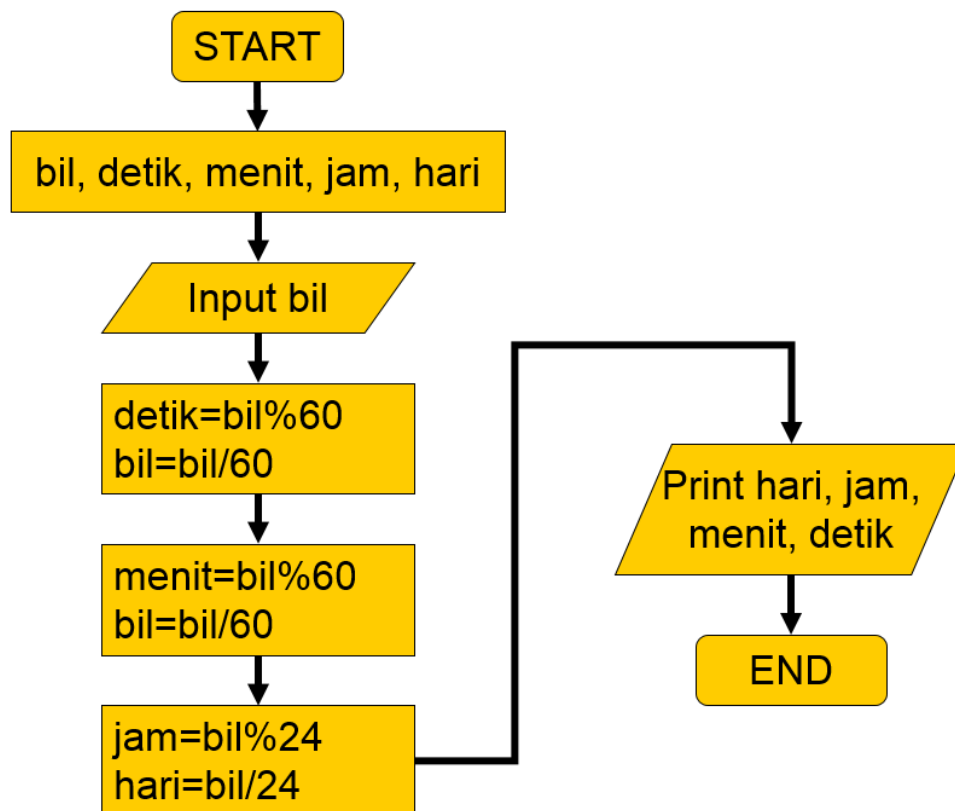
- Variabel bertipe integer untuk menyimpan nilai inputdetik --> variabel *bil*
- Variabel bertipe integer untuk menyimpan data detik --> variabel *detik*

- Variabel bertipe integer untuk menyimpan data menit --> variabel *menit*
- Variabel bertipe integer untuk menyimpan data jam --> variabel *jam*
- Variabel bertipe integer untuk menyimpan data hari --> variabel *hari*

#### 6.2.6 Logika Pemrograman:

- Operasi pembagian menggunakan operator `'/'`.
- Operasi perhitungan sisa hasil bagi (modulus) menggunakan operator `'%'`

#### 6.2.7 Flowchart:



### 6.3 Soal Latihan:

Buatlah flowchart untuk mencari max/min bilangan dari suatu deret bilangan

#### Contoh

Input	Bilangan : 8, 3, 5, 2, 7, 9, 6, 4
Output	Bilangan maksimum = 9
	Bilangan minimum = 2

# Bab 7

## Studi Kasus Percabangan dan Perulangan

---

### Pokok Bahasan

5. Kalkulator sederhana
6. Tumpukan bilangan

### Tujuan

1. Memahami logika proses kalkulator sederhana
2. Memahami logika proses untuk menampilkan tumpukan bilangan
3. Menjelaskan pemakaian operasi percabangan untuk pembuatan kalkulator sederhana
4. Menjelaskan pemakaian operasi perulangan untuk membuat tumpukan bilangan

### 7.1 Contoh Kasus 1: Kalkulator sederhana

Buatlah program kalkulator sederhana yang dapat melakukan operasi  $+$ ,  $-$ ,  $*$  dan  $/$  terhadap 2 bilangan.

#### Contoh

Input:            Masukkan bilangan 1: 16

                  Masukkan bilangan 2: 4

                  Masukkan operator: /

Output:           Hasilnya dari  $16 / 4$  adalah 4

#### 7.1.1 Cara Penyelesaian Masalah:

Program akan menerima input berupa bilangan 1, bilangan 2, dan operator. Kemudian program akan mengecek nilai operator, jika operator yang dimasukkan adalah  $+$  maka operasinya adalah penjumlahan, jika operator yang dimasukkan adalah  $-$  maka operasinya adalah pengurangan, jika operator yang dimasukkan adalah  $*$  maka operasinya adalah perkalian, dan jika operator yang dimasukkan adalah  $/$  maka operasinya adalah pembagian.

### 7.1.2 Input:

Input yang dibutuhkan untuk program ini adalah:

- Bilangan pertama dalam bentuk angka (integer)
- Bilangan kedua dalam bentuk angka (integer)

### 7.1.3 Output:

Program ini akan menghasilkan output yang berupa:

- Nilai hasil perhitungan yang bertipe floating point>

### 7.1.4 Struktur Data yang Dibutuhkan:

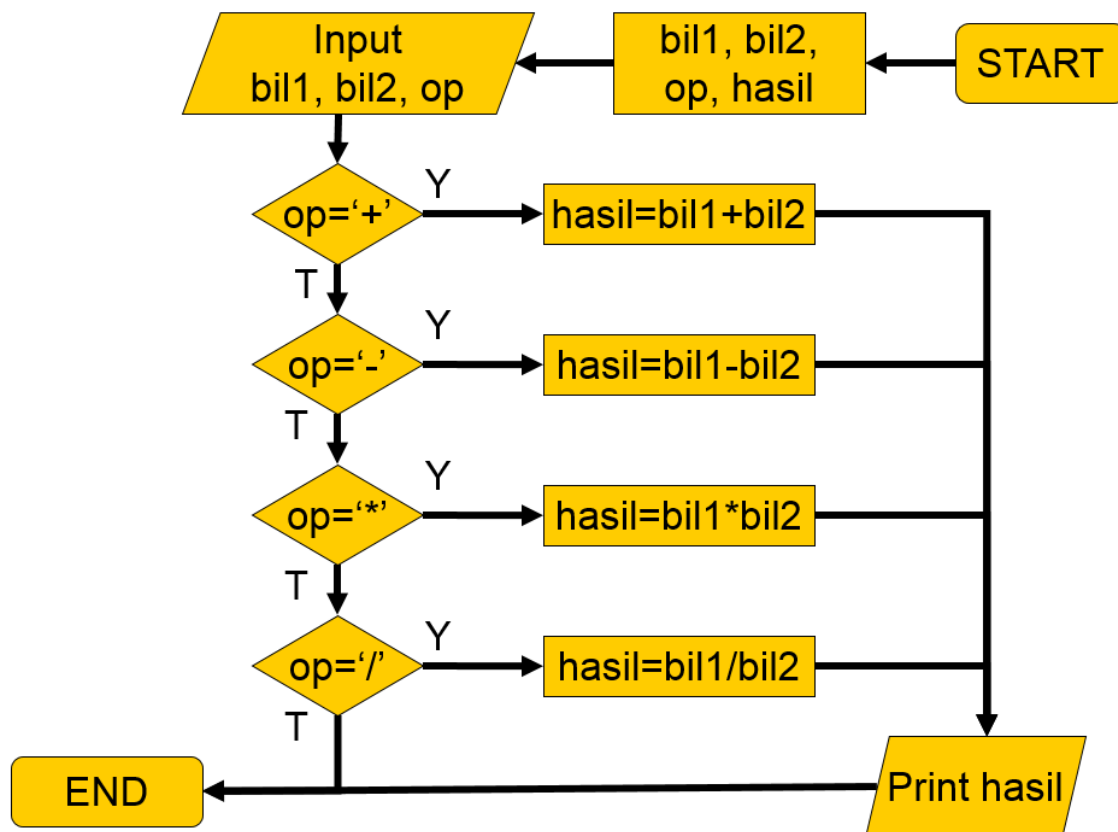
- Variabel bertipe integer untuk menyimpan data bilangan 1 --> variabel *bil1*
- Variabel bertipe integer untuk menyimpan data bilangan 2 --> variabel *bil2*
- Variabel bertipe char untuk menyimpan jenis operator --> variabel *op*
- Variabel bertipe float untuk menyimpan hasil perhitungan --> variabel *hasil*

### 7.1.5 Logika Pemrograman:

- Operasi percabangan dengan menggunakan 'IF ... THEN ... ELSE ...'
- Operasi penjumlahan menggunakan operator '+'
- Operasi pengurangan menggunakan operator '-'
- Operasi perkalian menggunakan operator '\*'
- Operasi pembagian menggunakan operator '/'.



### 7.1.6 Flowchart:



## 7.2 Contoh Kasus 2: Tumpukan bilangan

### 7.2.1 Permasalahan

Buatlah program untuk menampilkan tumpukan bilangan.

#### Contoh

Input: Masukkan jumlah tumpukan: 55

Output: 1  
222  
33333  
4444444  
555555555

### 7.2.2 Cara Penyelesaian Masalah:

Untuk membuat tumpukan bilangan seperti pada contoh kasus diatas, diperlukan dua level perulangan. Perulangan pertama untuk tingkatan, sedangkan perulangan kedua untuk mencetak angka pada tiap tingkatan.

### 7.2.3 Input:

Input yang dibutuhkan untuk program ini adalah:

- Jumlah tumpukan dalam bentuk angka (integer)

### 7.2.4 Output:

Program ini akan menghasilkan output yang berupa:

- Tumpukan angka dengan jumlah tumpukan sesuai nilai angka yang dimasukkan

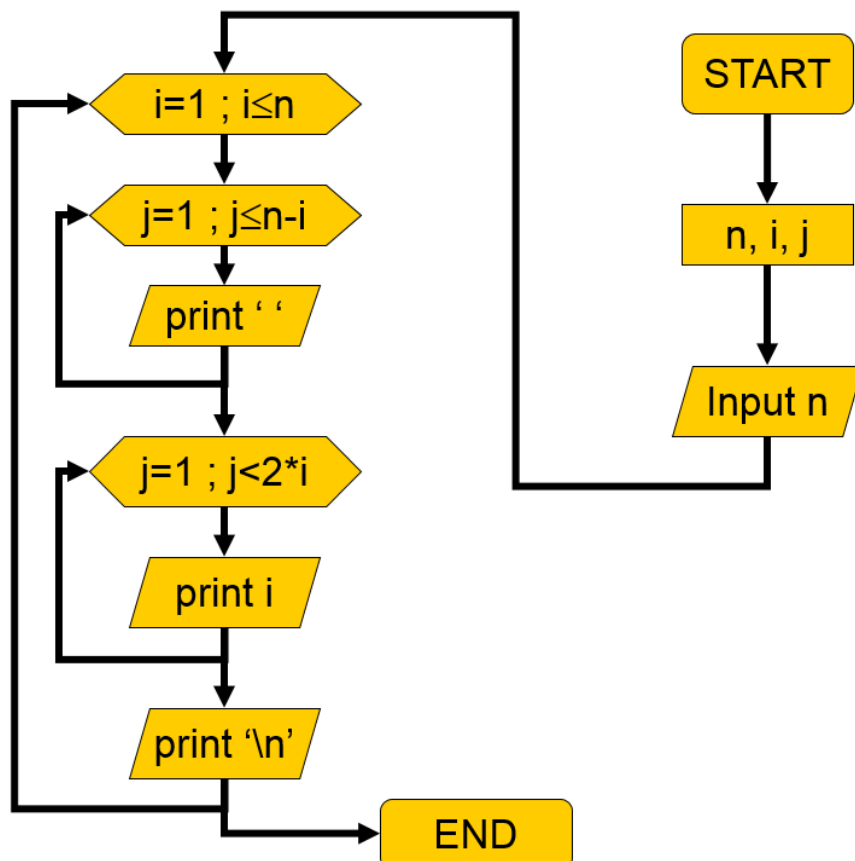
### 7.2.5 Struktur Data yang Dibutuhkan:

- Variabel bertipe integer untuk menyimpan data jumlah tumpukan --> variabel  $n$
- Variabel bertipe integer untuk digunakan pada perulangan pertama --> variabel  $i$
- Variabel bertipe integer untuk digunakan pada perulangan kedua --> variabel  $j$

### 7.2.6 Logika Pemrograman:

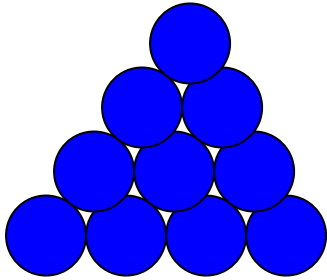
- Operasi perulangan menggunakan statement **For**

### 7.2.7 Flowchart:



### 7.3 Soal Latihan

Buatlah program untuk menghitung jumlah bola dalam suatu triangular. Triangular adalah suatu susunan benda (contoh: bola) yang disusun sedemikian rupa sehingga menyerupai segitiga. Dengan mengetahui jumlah bola yang paling bawah, maka dapat dihitung jumlah seluruh bola yang menyusun triangular tersebut.



**Contoh:**

Input:            Masukkan jumlah triangular: 4

Output:          Jumlah bola: 10

*Halaman ini sengaja dikosongkan*

# Bab 8

## Studi Kasus Tumpukan (Stack)

---

### Pokok Bahasan

7. Membalik kalimat
8. Membalik bilangan

### Tujuan

5. Memahami logika proses pembalikan kalimat
6. Memahami logika proses pembalikan bilangan
7. Menjelaskan pemakaian stack untuk pembalikan kalimat
8. Menjelaskan pemakaian stack untuk pembalikan bilangan

### 8.1 Contoh Kasus 1: Membalik kalimat

#### 8.1.1 Permasalahan

Buatlah flowchart untuk membalik kalimat.

#### Contoh

Input:                Masukkan kalimat = “APA KABAR”

Output:             “RABAK APA”

#### 8.1.2 Cara Penyelesaian Masalah:

Untuk membalik kalimat dapat diselesaikan menggunakan dua cara. Pertama dengan menggunakan array. Sedangkan yang kedua dengan memanfaatkan stack. Stack (tumpukan) merupakan struktur data yang dalam pengelolaan datanya bersifat Last In First Out (LIFO), yaitu data yang terakhir dimasukkan akan dikeluarkan pertama kali. Data yang dimasukkan kedua dari terakhir akan dikeluarkan yang kedua. Demikian seterusnya, sehingga data yang pertama kali dimasukkan akan dikeluarkan terakhir.

### 8.1.3 Input:

Input yang dibutuhkan untuk program ini adalah:

- Kalimat yang akan dibalik dalam bentuk array of character (string)

### 8.1.4 Output:

Program ini akan menghasilkan output yang berupa:

- Kalimat yang sudah dibalik dalam bentuk string

### 8.1.5 Struktur Data yang Dibutuhkan:

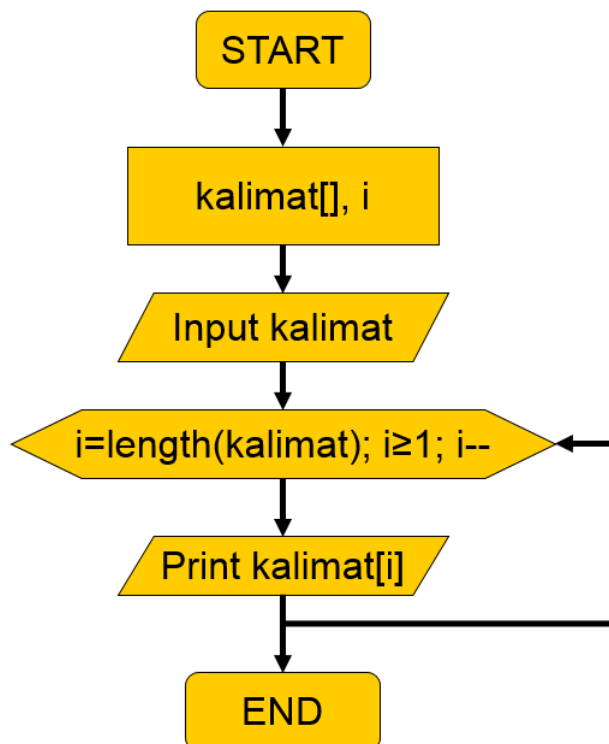
- Variabel bertipe string (array of character) untuk menyimpan kalimat awal --> variabel *kalimat*

### 8.1.6 Logika Pemrograman:

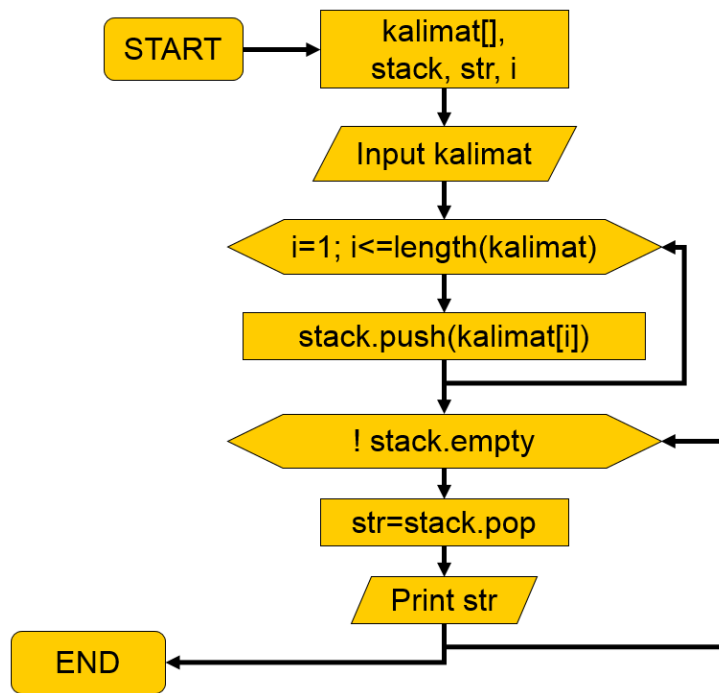
- Operasi perulangan dengan menggunakan 'DO ... WHILE' atau 'WHILE ... DO'

### 8.1.7 Flowchart

Flowchart untuk penyelesaian menggunakan array of character



Flowchart untuk penyelesaian menggunakan stack



## 8.2 Contoh Kasus 2: Membalik bilangan

### 8.2.1 Permasalahan

Buatlah flowchart untuk menampilkan bilangan dalam urutan terbalik.

#### Contoh

Input: Masukkan kalimat : 1234

Output: Hasil : 4321

### 8.2.2 Cara Penyelesaian Masalah:

Proses untuk membalik urutan bilangan berbeda dengan membalik kalimat. Suatu kalimat bisa dianggap sebagai array dari karakter, sedangkan bilangan tidak bisa.

Cara yang bisa dilakukan adalah dengan menggunakan operator modulus untuk mengambil angka terakhir dari bilangan. Kemudian dengan membagi bilangan tersebut dengan 10 menggunakan operator pembagian '/' kita akan mendapatkan sisa bilangan setelah bilangan yang terakhir diambil.

Contoh, misalkan kita mempunyai bilangan 1234. Dengan melakukan operasi  $1234 \% 10$ , kita akan mendapatkan angka terakhir yaitu 4. Kemudian dengan operasi  $1234 / 10$  kita akan mendapatkan bilangan 123. Berikutnya kita dapat mengulangi lagi mulai langkah yang pertama (menggunakan operator modulus).

### 8.2.3 Input:

Input yang dibutuhkan untuk program ini adalah:

- Bilangan yang akan dibalik dalam bentuk integer

### 8.2.4 Output:

Program ini akan menghasilkan output yang berupa:

- Bilangan dengan urutan angka yang berupa kebalikan dari urutan angka pada bilangan hasil input.

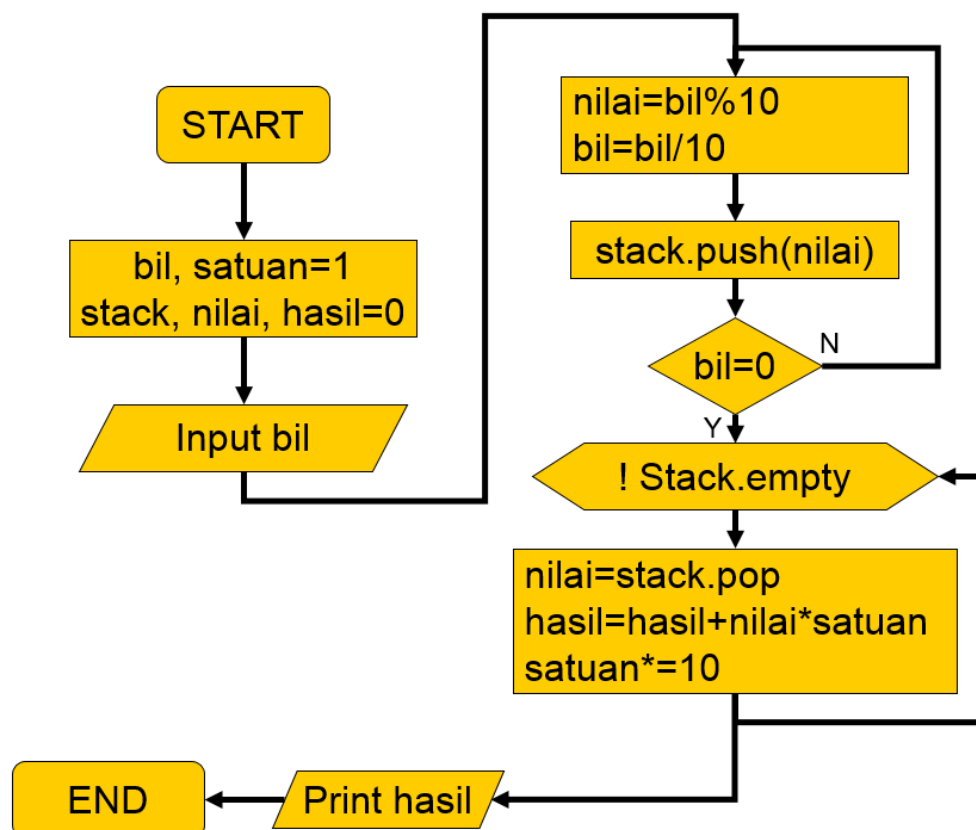
### 8.2.5 Struktur Data yang Dibutuhkan:

- Variabel bertipe integer untuk menyimpan bilangan awal --> variabel *bil*
- Variabel bertipe integer untuk menyimpan bilangan hasil --> variabel *hasil*
- Variabel untuk menyimpan stack --> variabel *stack*
- Variabel bertipe integer untuk menyimpan nilai sementara --> variabel *nilai*
- Variabel bertipe integer untuk menyimpan angka satuan --> variabel *satuan*

### 8.2.6 Logika Pemrograman:

- Operasi perulangan dengan menggunakan 'DO ... WHILE' atau 'WHILE ... DO'

### 8.2.7 Flowchart





### 8.3 Soal Latihan

Buatlah flowchart untuk mengecek suatu kalimat termasuk palindrom apa bukan. Kalimat palindrom adalah kalimat yang susunannya sama dengan keadaan terbaliknyanya.

**Contoh:**

Input:            Masukkan kalimat = “KASUR RUSAK”

Output:            Kalimat termasuk palindrom

Input:            Masukkan kalimat = “MAKAN MALAM”

Output:            Kalimat tidak termasuk palindrom

*Halaman ini sengaja dikosongkan*

# Bab 9

## Studi Kasus Konversi Bilangan

---

### Pokok Bahasan

1. Konversi bilangan biner ke desimal
2. Konversi bilangan desimal ke biner

### Tujuan

1. Memahami logika melakukan konversi bilangan
2. Memahami algoritma konversi bilangan
3. Menjelaskan pemakaian looping untuk konversi bilangan.
4. Menjelaskan pemakaian operasi stack untuk konversi bilangan.

## 9.1 . Studi Kasus 1: Konversi bilangan biner ke desimal

### 9.1.1 Permasalahan

Buatlah flowchart untuk konversi bilangan biner ke desimal (maksimum bilangan=11111111).

Contoh:

Masukkan bilangan biner : 00110101

Bilangan desimal : 53

### 9.1.2 Cara Penyelesaian Masalah

Konversi bilangan biner ke desimal dilakukan dengan cara mengalikan setiap elemen bilangan biner (bi) dengan deretan bilangan 2 yang berpangkat sebagai berikut:

$$b_1 * 2^{n-1} + b_2 * 2^{n-2} + \dots + b_{n-2} * 2^2 + b_{n-1} * 2^1 + b_n * 2^0$$

$$\text{atau} : b_1 * 2^{n-1} + b_2 * 2^{n-2} + \dots + b_{n-2} * 4 + b_{n-1} * 2 + b_n * 1$$

Dalam kasus permasalahan diatas, input bilangan biner yang dimasukkan sepanjang 8 elemen bilangan dapat diselesaikan sebagai berikut:

$$\begin{array}{r} 00110101 \\ \downarrow \\ 0*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0 \\ = 0*128 + 0*64 + 1*32 + 1*16 + 0*8 + 1*4 + 0*2 + 1*1 \\ = 0 + 0 + 32 + 16 + 0 + 4 + 0 + 1 \\ = 53 \end{array}$$

### 9.1.3 Struktur Data Yang Dibutuhkan

- n sebagai variabel banyaknya elemen bilangan (dalam hal ini n=8)
- biner sebagai array bilangan biner dengan panjang n elemen
- hasil sebagai variabel penampung hasil konversi biner ke desimal
- i sebagai variabel penunjuk elemen array

### 9.1.4 Deklarasi dan Inisialisasi

Variabel terdiri dari n, biner, hasil dan i. Untuk n, pada permasalahan n=8. Sedangkan hasil, karena dipakai sebagai penampung hasil konversi, saat awal hasil harus diset menjadi 0. Dari sini, deklarasi dan inisialisasi variabel pada flowchart dapat dibuat sebagai berikut:

n=8, biner=[n],  
hasil=0, i

### 9.1.5 Input

Bilangan biner yang terdiri dari 8 elemen, sehingga input pada flowchart dapat dibuat sebagai berikut:

Input biner

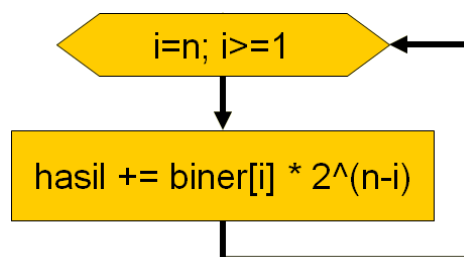
### 9.1.6 Output

Bilangan desimal hasil konversi dari bilangan biner yang disimpan pada variabel *hasil*, sehingga output pada flowchart dapat dibuat sebagai berikut:

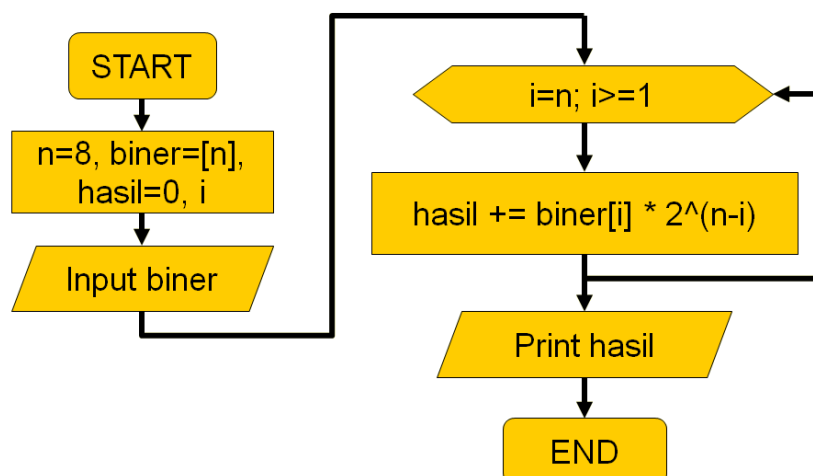


### 9.1.7 Proses Penyelesaian

Karena proses perhitungan konversi dilakukan dari kanan ke kiri, maka looping juga dilakukan menurun dari  $i=n$  sampai  $i=1$ . Saat awal  $i=n$ , dilakukan perhitungan pada  $b_n * 2^0$  dan hasilnya diakumulasikan pada *hasil*. Pada iterasi berikut nilai  $i$  menurun menjadi  $i=n-1$ , dilakukan perhitungan pada  $b_{n-1} * 2^1$  dan hasilnya diakumulasikan pada *hasil*. Dari sini, dapat diambil kesimpulan bahwa pada setiap iterasi  $i$ , *hasil* dapat dihitung dengan rumusan  $hasil = hasil + b_i * 2^{n-i}$ . Dengan demikian, untuk proses dapat direpresentasikan dengan flowchart sebagai berikut:



### 9.1.8 Flowchart Keseluruhan



## 9.2 Studi Kasus 2: Konversi bilangan desimal ke biner

### 9.2.1 Permasalahan

Buatlah flowchart untuk konversi bilangan desimal ke biner (maksimum bilangan=255)

Contoh:

Masukkan bilangan desimal : 53

Bilangan biner : 110101

### 9.2.2 Cara Penyelesaian Masalah

Konversi bilangan desimal ke biner dilakukan dengan cara melakukan pembagian dengan nilai 2 dan mengambil sisa baginya. Sisa bagi tersebut kemudian disimpan pada suatu tempat kumpulan sisa bagi. Bilangan hasil bagi tersebut dibagi lagi dan diambil sisa baginya untuk kemudian disimpan pada tempat kumpulan sisa bagi. Demikian seterusnya sampai didapatkan hasil baginya sama dengan nol. Setelah itu, baru diambil sisa bagi dalam keadaan terbalik (yang masuk terakhir, diambil pertama kali) pada kumpulan sisa bagi. Proses pembacaan terbalik ini membutuhkan operasi stack.

Dalam kasus permasalahan diatas, dapat diselesaikan sebagai berikut:

Bilangan	Hasil bagi dengan 2	Sisa bagi dengan 2	Kumpulan sisa bagi
53	26	1	1
26	13	0	10
13	6	1	101
6	3	0	1010
3	1	1	10101
1	0	1	101011

Bilangan biner yang didapatkan adalah dari pembacaan kumpulan sisa bagi yang dibaca terbalik, sehingga hasilnya adalah 110101.

### 9.2.3 Struktur Data Yang Dibutuhkan

- bil sebagai variabel nilai bilangan
- sisa sebagai variabel nilai sisa bagi dengan 2
- stack sebagai penampung sisa bagi sehingga dapat dibaca terbalik

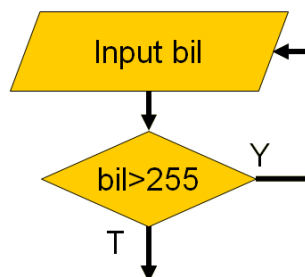
### 9.2.4 Deklarasi dan Inisialisasi

Variabel terdiri dari bil, sisa, dan stack. Dari sini, deklarasi dan inisialisasi variabel pada flowchart dapat dibuat sebagai berikut:

bil, sisa, stack

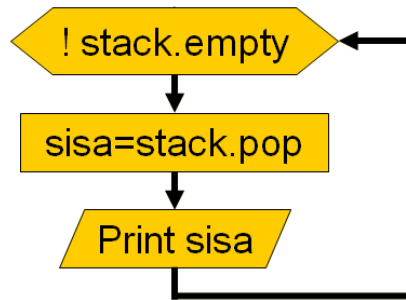
### 9.2.5 Input

Saat pertama kali, user diminta untuk memasukkan nilai bilangan bil yang akan dikonversi ke biner. Karena pada permasalahan terdapat syarat bahwa nilai bilangan tidak boleh lebih dari 255, maka perlu diberikan pengecekan apakah bil sesuai dengan syarat. Dari sini, flowchart Input dapat dibuat sebagai berikut:



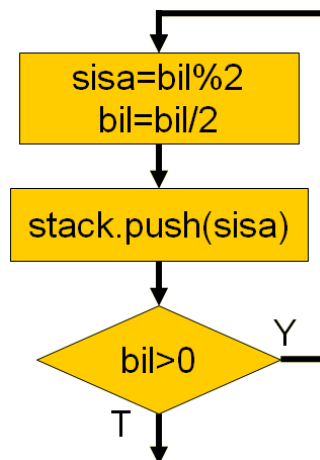
### 9.2.6 Output

Bilangan desimal hasil konversi dari bilangan desimal yang dikeluarkan dari *stack* dengan operasi pop sehingga dapat dibaca dalam keadaan terbalik. Setiap hasil pembacaan kemudian disimpan pada variabel *sisa*, sehingga output pada flowchart dapat dibuat sebagai berikut:



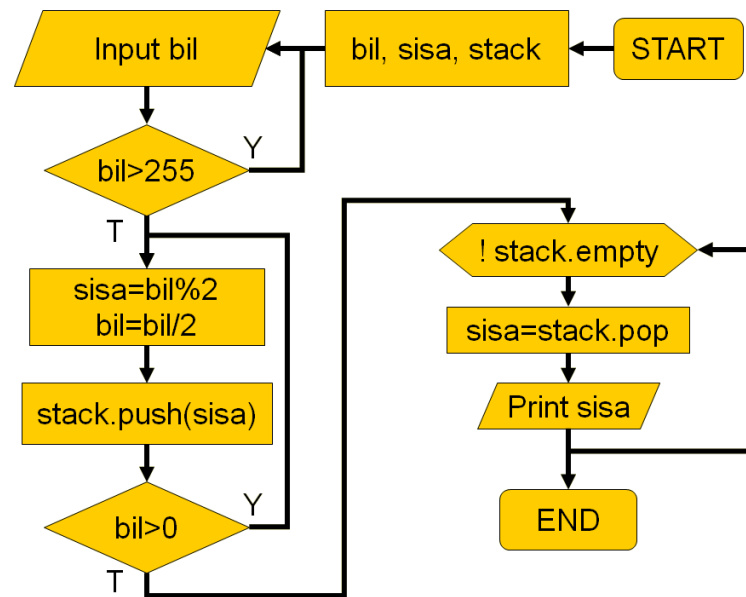
### 9.2.7 ProsesPenyelesaian

Sebagai dijelaskan pada sebelumnya, konversi bilangan desimal ke biner dilakukan dengan cara melakukan pembagian dengan nilai 2 dan mengambil sisa baginya, dimana proses ini dilakukan secara iteratif dan berhenti saat bilangan hasil bagi sama dengan 0. Setiap kali didapatkan sisa bagi, nilai tersebut dimasukkan ke dalam stack melalui operasi push. Adapun tujuan dipergunakannya stack adalah agar data yang masuk ke dalam stack, dapat dibaca dalam keadaan terbalik, sehingga ini berguna untuk membaca terbalik nilai sisa bagi yang dimasukkan ke dalam stack. Dari sini, flowchart Proses dapat direpresentasikan sebagai berikut:





### 9.2.8 Flowchart Keseluruhan



## 9.3 Latihan

### 9.3.1 Permasalahan

Buatlah flowchart untuk konversi bilangan desimal negatif ke biner (maksimum bilangan=-127).

Contoh:

Masukkan bilangan desimal : -1

Bilangan biner : 11111111

Masukkan bilangan desimal : -13

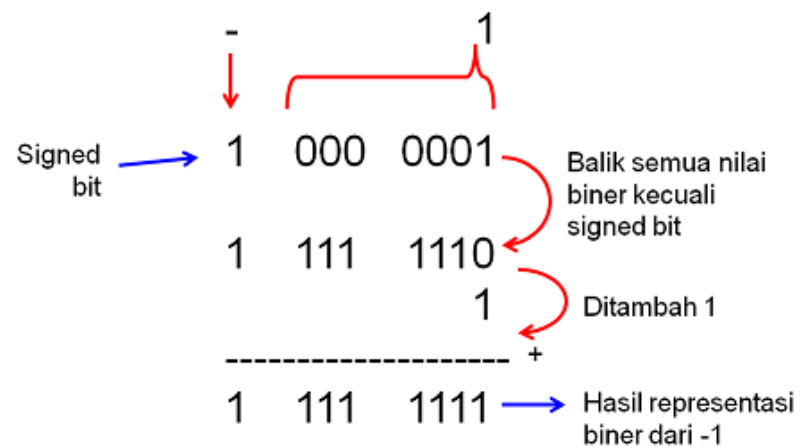
Bilangan biner : 11110011

### 9.3.2 Cara Penyelesaian Masalah

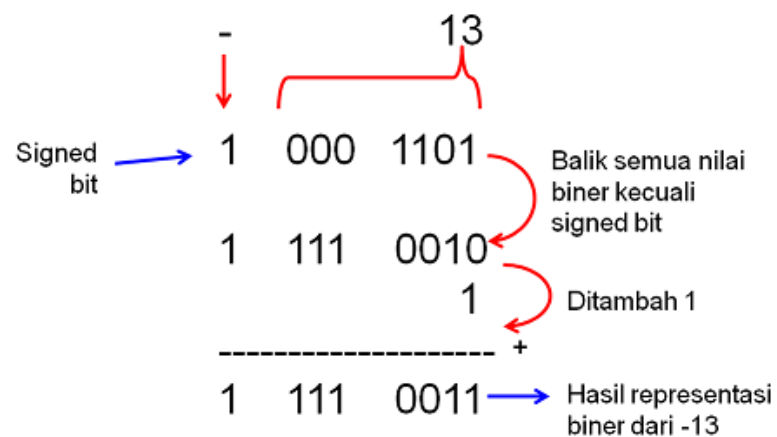
Karena batasan permasalahan bahwa maksimum bilangan adalah -127, maka total bit yang dipakai adalah sebanyak 8 bit. Pertama kali, signed bit (bit pada depan) diset dengan nilai 1 sebagai tanda negatif. Sisa 7 bit yang lainnya adalah nilai bit hasil konversi dari input bilangan desimal yang positif. Selanjutnya, 7 bit sisa tersebut (tidak termasuk signed bit) dibalik nilainya (0 menjadi 1, 1 menjadi 0). Kemudian 7 bit sisa ditambahkan dengan nilai

bit1. Susunan bit keseluruhannya adalah hasil dari konversi desimal negatif. Untuk lebih mudahnya, perhatikan ilustrasi berikut:

Masukkan bilangan desimal : -1



Masukkan bilangan desimal : -13



# Bab 10

## Studi Kasus Operasi Matriks

---

### Pokok Bahasan

1. Operasi Matriks

### Tujuan

1. Memahami logika operasi matriks
2. Memahami penggunaan looping bertingkat untuk menyelesaikan operasi matriks

### 10.1 Studi Kasus: Operasi penambahan matriks

#### 10.1.1 Permasalahan

Buatlah suatu flowchart untuk menyelesaikan operasi penambahan matriks

Contoh:

Masukkan Matriks 1 dan Matriks 2 :

$$\text{Matriks 1 : } \begin{bmatrix} 5 & 2 \\ 1 & 7 \end{bmatrix}$$

$$\text{Matriks 2 : } \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\text{Hasil : } \begin{bmatrix} 6 & 2 \\ 2 & 8 \end{bmatrix}$$

#### 10.1.2 Cara Penyelesaian Masalah

Operasi penambahan matriks dilakukan dengan cara menambahkan setiap elemen Matriks 1 dengan elemen Matriks 2 pada posisi baris dan kolom yang sama untuk kedua matriks.

Pada permasalahan diatas, operasi penambahan matriks dilakukan sebagai berikut:

$$\begin{bmatrix} 5 & 2 \\ 1 & 7 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \\
 = \begin{bmatrix} 5+1 & 2+0 \\ 1+1 & 7+1 \end{bmatrix} \\
 = \begin{bmatrix} 6 & 2 \\ 2 & 8 \end{bmatrix}$$

### 10.1.3 Struktur Data Yang Dibutuhkan

- *brs* sebagai variabel panjang baris untuk matriks
- *klm* sebagai variabel panjang kolom untuk matriks
- *M1* sebagai input matriks pertama bertipe array 2 dimensi
- *M2* sebagai input matriks kedua bertipe array 2 dimensi
- *M3* sebagai matriks penampung hasil operasi penambahan matriks bertipe array 2 dimensi
- *i* sebagai variabel penunjuk baris saat proses penambahan matriks
- *j* sebagai variabel penunjuk kolom saat proses penambahan matriks

### 10.1.4 Deklarasi dan Inisialisasi

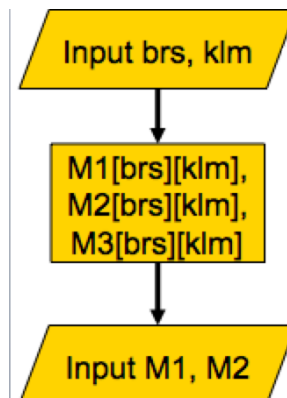
Variabel terdiri dari *brs*, *klm*, *M1*, *M2*, *M3*, *i* dan *j*. Dari sini, deklarasi dan inisialisasi variabel pada flowchart dapat dibuat sebagai berikut:

brs, klm,  
M1[], M2[], M3[],  
i, j

### 10.1.5 Input

Pertama kali, user harus memasukkan panjang baris *brs* dan kolom *klm* untuk matriks. Dari panjang baris dan kolom tersebut, kemudian dibuat suatu 2 buah matriks input *M1* dan

$M2$  dengan panjang baris dan kolom yang diinginkan. Kemudian, user memasukkan nilai dari 2 matriks input. Flowchart untuk input ini dapat dilihat sebagai berikut:



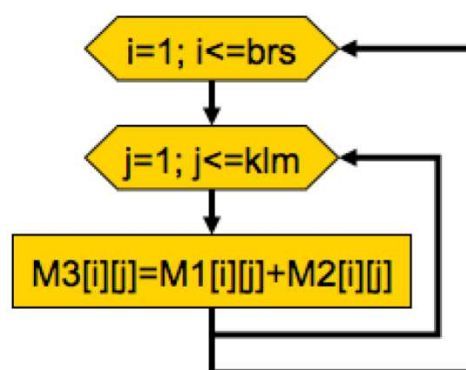
### 10.1.6 Output

Matriks  $M3$  sebagai hasil operasi penambahan matriks  $M1$  dan  $M2$ , sehingga output pada flowchart dapat dibuat sebagai berikut:

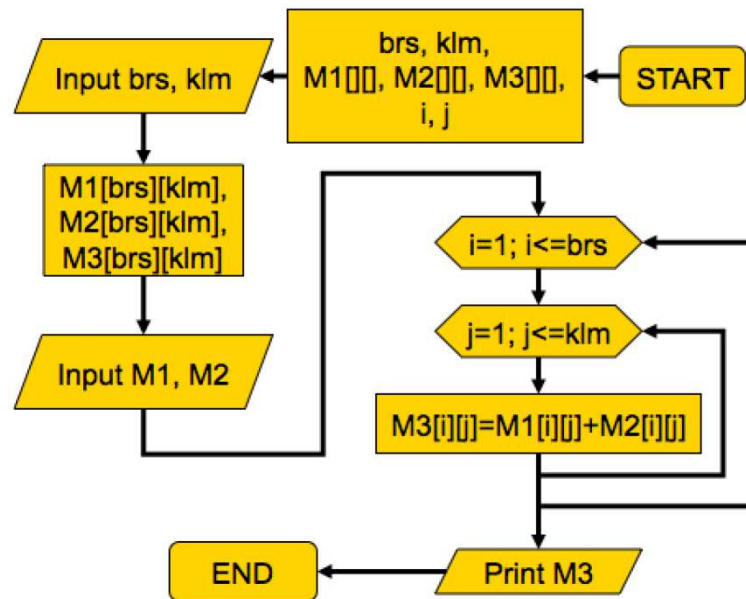


### 10.1.7 Proses Penyelesaian

Proses operasi penambahan matriks dilakukan dengan menambahkan setiap elemen kedua matriks input pada posisi yang sama. Jika proses penambahan dimulai dari baris, untuk baris pertama, dilakukan penambahan nilai pada kolom pertama untuk kedua matriks. Selanjutnya berganti pada kolom kedua. Setelah baris pertama selesai, kemudian berganti pada baris kedua, dilakukan penambahan nilai pada kolom pertama untuk kedua matriks, dan selanjutnya dilakukan hal serupa pada kolom kedua. Pada setiap kali operasi penambahan, hasil penambahan ditampung pada matriks  $M3$ . Dengan demikian, untuk proses dapat direpresentasikan dengan flowchart sebagai berikut:



### 10.1.8 Flowchart Keseluruhan



## 10.2 10.2. Latihan

### 10.2.1 Permasalahan

Buatlah suatu flowchart untuk menyelesaikan operasi penambahan matriks

Contoh:

Masukkan Matriks 1 dan Matriks 2 :

$$\text{Matriks 1 : } \begin{bmatrix} 5 & 2 & 3 \\ 1 & 7 & 1 \end{bmatrix}$$

$$\text{Matriks 2 : } \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 2 & 4 \end{bmatrix}$$

$$\text{Hasil : } \begin{bmatrix} 13 & 14 \\ 10 & 11 \end{bmatrix}$$

### 10.2.2 Cara Penyelesaian Masalah

Operasi penambahan matriks dilakukan dengan cara mengalikan setiap elemen baris pada Matriks 1 dengan elemen kolom pada Matriks 2 secara berturut-turut, dan kemudian masing-masing dari hasil perkalian tersebut diakumulasikan.

Pada permasalahan diatas, operasi perkalian matriks dilakukan sebagai berikut:

$$\begin{bmatrix} 5 & 2 & 3 \\ 1 & 7 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 2 & 4 \end{bmatrix}$$
$$= \begin{bmatrix} 5 \times 1 + 2 \times 1 + 3 \times 2 & 5 \times 0 + 2 \times 1 + 3 \times 4 \\ 1 \times 1 + 7 \times 1 + 1 \times 2 & 1 \times 0 + 7 \times 1 + 1 \times 4 \end{bmatrix}$$
$$= \begin{bmatrix} 13 & 14 \\ 10 & 11 \end{bmatrix}$$

*Halaman ini sengaja dikosongkan*



# Bab 11

## Studi Kasus Shortest Path Problem

---

### Pokok Bahasan

1. Shortest Path Problem (Permasalahan Jalur Tercepat)

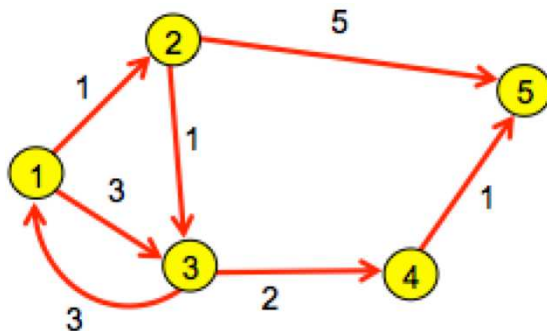
### Tujuan

1. Memahami logika shortest path problem
2. Memahami algoritma Dijkstra untuk menyelesaikan shortest path problem
3. Memahami penggunaan matrik untuk manipulasi variabel pada shortest path problem
4. Memahami penggunaan queue dan stack untuk menyelesaikan shortest path problem

### 11.1 Studi Kasus: Shortest Path Problem dengan Algoritma Dijkstra

#### 11.1.1 Permasalahan

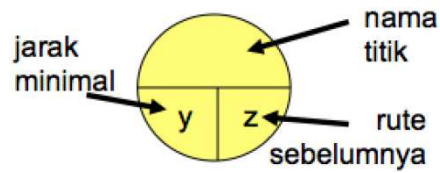
Buatlah flowchart untuk menghitung jarak minimal dan rutenya dari titik 1 ke titik 5 dari jalur berikut ini:



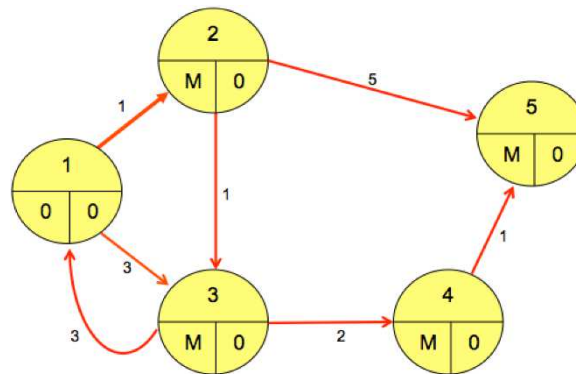
#### 11.1.2 Cara Penyelesaian Masalah

Untuk menyelesaikan shortest path problem satu arah seperti yang ada pada permasalahan, dapat digunakan algoritma Dijkstra. Pertama kali tentukan titik asal dan titik tujuan pada permasalahan. Pada permasalahan, titik asal adalah titik 1 dan titik tujuan adalah titik 5. Karena yang dicari adalah jarak terpendek dan rute pada jarak terpendek tersebut,

maka setiap titik perlu memuat jarak terpendek dan rutenya terhadap titik sebelumnya, sehingga setiap titik terdiri dari informasi berikut:



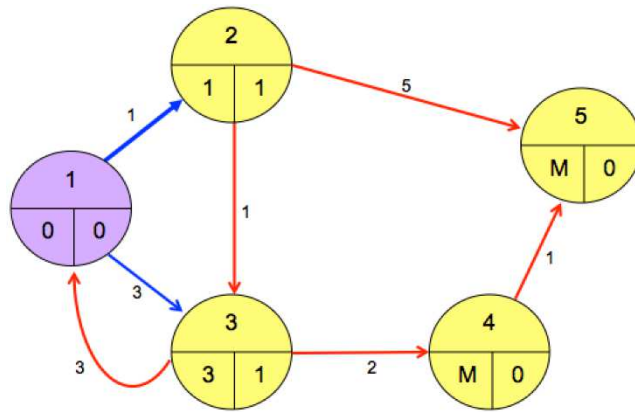
Saat awal, pada setiap titik selain titik asal, nilai y diinisialisasi dengan nilai yang besar (diasumsikan sebagai M) dan nilai z diinisialisasi dengan titik 0, seperti yang tampak pada gambar berikut:



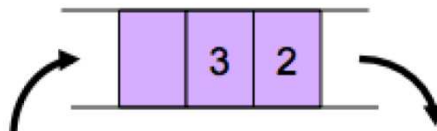
Setelah itu, proses dimulai dari titik asal yaitu titik 1 sebagai titik aktif. Titik 1 mempunyai jalur ke 2 titik, yaitu titik 2 dan titik 3. Saat mendapatkan ada jalur dari titik 1 ke titik 2, maka nilai y pada titik 1 ditambahkan dengan jarak dari titik 1 ke titik 2. Jika nilai penambahan ini lebih kecil dari nilai y pada titik 2, maka akan menyebabkan 3 hal:

1. nilai y pada titik 2 digantikan dengan nilai penambahan tersebut
2. nilai z pada titik 2 digantikan dengan titik 1.
3. Informasi titik 2 dimasukkan ke dalam queue dengan proses enqueue

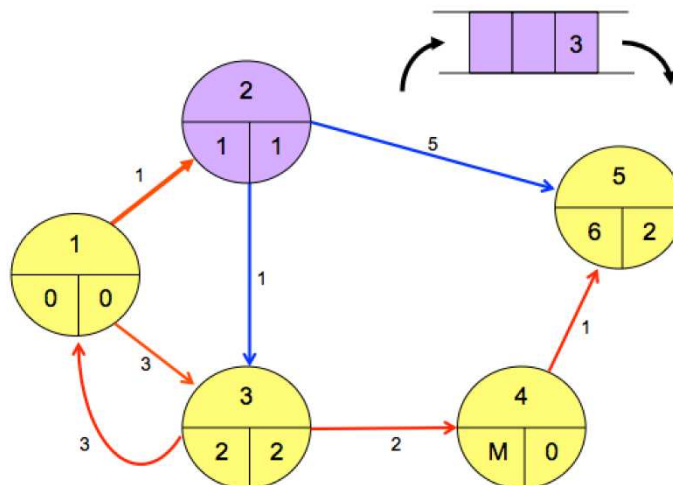
Hal yang sama dilakukan terhadap titik 1 ke titik 3. Ini membuat adanya perubahan nilai pada titik 2 dan titik 3, seperti yang terlihat sebagai berikut.



Adapun kondisi queue, karena titik 2 dan titik 3 dimasukkan ke dalam queue, maka queue terlihat dibawah ini.

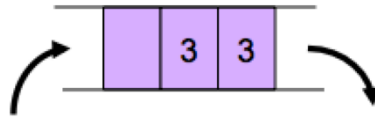


Selanjutnya, isi queue dikeluarkan satu nilai (proses dequeue) sebagai titik aktif. Hasil dari dequeue adalah nilai 2, sehingga sekarang titik aktif adalah titik 2. Titik 2 mempunyai jalur ke titik 3 dan titik 5. Hal yang dilakukan seperti yang sebelumnya, sehingga kondisinya sekarang menjadi seperti berikut:

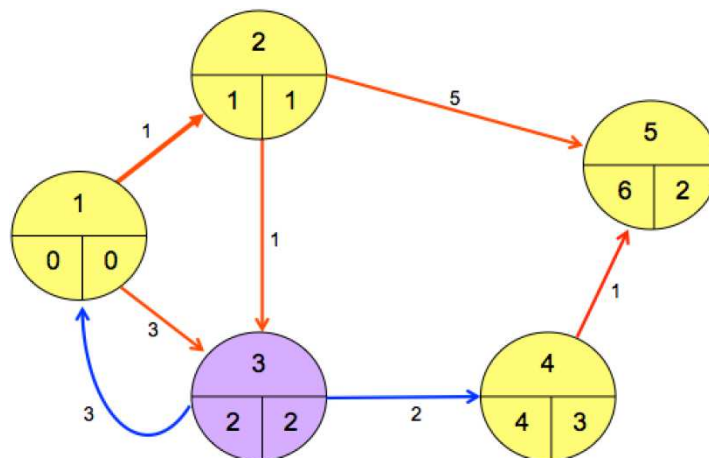


Informasi jalur dari titik 2 ke titik 3 dan titik 5 ini kemudian akan dimasukkan ke dalam queue. Informasi titik 3 kemudian dimasukkan lagi ke dalam queue. Untuk titik 5,

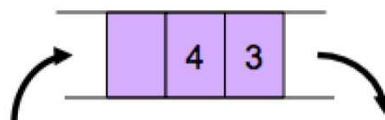
karena titik 5 adalah tujuan, maka tidak perlu dimasukkan ke dalam queue. Sekarang kondisi queue terlihat seperti dibawah ini:



Selanjutnya, isi queue dikeluarkan satu nilai (proses dequeue) sebagai titik aktif. Hasil dari dequeue adalah nilai 3, sehingga sekarang titik aktif adalah titik 3. Titik 3 mempunyai jalur ke titik 1 dan titik 4. Hal yang dilakukan seperti yang sebelumnya, sehingga kondisinya sekarang menjadi seperti berikut:

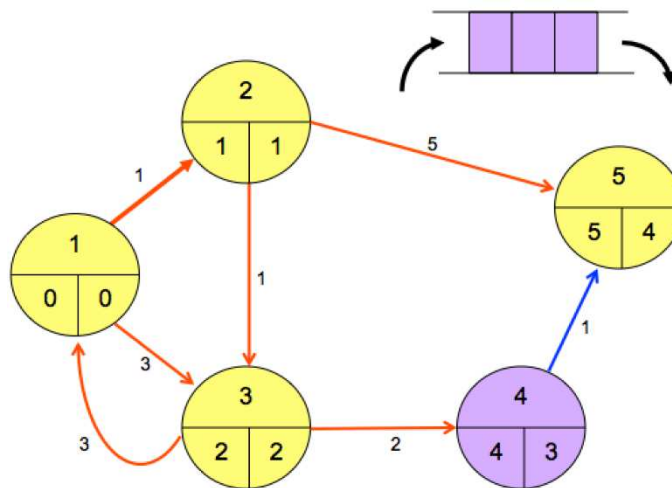


Proses berikutnya adalah enqueue untuk memasukkan titik 1 dan titik 4. Karena titik 1 adalah titik asal, maka tidak dimasukkan ke queue. Kemudian titik 4 dimasukkan ke dalam queue, sehingga kondisi queue sekarang tampak seperti dibawah ini:



Selanjutnya, isi queue dikeluarkan satu nilai (proses dequeue) sebagai titik aktif. Hasil dari dequeue adalah nilai 3, sehingga sekarang titik aktif adalah titik 3, sama seperti keadaan sebelumnya. Proses dilakukan sama, akan tetapi karena nilai penambahan nilai y dan nilai z pada titik 3 ke titik-titik jalurnya tidak ada yang lebih kecil dari nilai y dan nilai z pada titik-titik jalurnya, maka tidak ada perubahan nilai pada nilai y dan nilai z pada titik jalurnya dan tidak ada proses enqueue titik-titik jalur dari titik 3.

Selanjutnya, isi queue dikeluarkan satu nilai (proses dequeue) sebagai titik aktif. Hasil dari dequeue adalah nilai 4, sehingga sekarang titik aktif adalah titik 4. Titik 4 mempunyai jalur ke titik 5. Hal yang dilakukan seperti yang sebelumnya, sehingga kondisinya sekarang menjadi seperti berikut:



Informasi jalur dari titik 4 ke titik 5 ini kemudian akan dimasukkan ke dalam queue. Karena titik 5 adalah tujuan, maka tidak perlu dimasukkan ke dalam queue.

Untuk selanjutnya, kembali pada proses enqueue pada queue. Karena isi queue kosong, maka perubahan nilai  $y$  dan nilai  $z$  pada setiap titik sudah selesai.

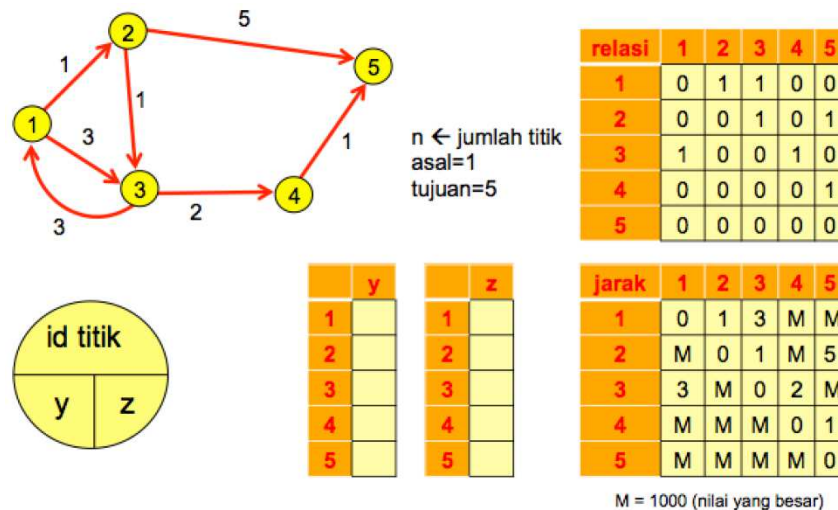
Proses selanjutnya adalah pembacaan nilai jarak terdekat dan jalur terdekat sebagai solusi. Informasi jarak terdekat dapat diambil dari nilai  $y$  pada titik tujuan. Berarti, jarak terdekat pada permasalahan tersebut adalah 5 sebagaimana terlihat pada nilai  $y$  di titik 5. Untuk mendapat informasi jalur terdekat, diperlukan operasi stack. Dimulai dari titik tujuan, titik 5 dimasukkan ke dalam stack. Kemudian, pada titik 5, nilai  $z=4$ , sehingga bergerak ke titik 4. Titik 4 dimasukkan ke dalam stack. Nilai  $z$  pada titik 4 adalah 3, sehingga kemudian bergerak ke titik 3. Titik 3 dimasukkan ke dalam stack. Nilai  $z$  pada titik 3 adalah 2, sehingga kemudian bergerak ke titik 2. Titik 2 dimasukkan ke dalam stack. Kemudian, nilai  $z$  pada titik 2 adalah 1, sama dengan titik asal. Titik asal dimasukkan ke dalam stack. Selanjutnya, informasi jalur tercepat dapat dibaca dari hasil operasi pop terhadap stack, sehingga didapatkan jalur tercepatnya adalah 1-2-3-4-5.

### 11.1.3 Struktur Data Yang Dibutuhkan

- $n$  sebagai variabel banyaknya titik
- *asal* sebagai variabel titik asal
- *tujuan* sebagai variabel titik tujuan
- *relasi* sebagai matriks jalur dari satu titik ke titik lain
- *jarak* sebagai matriks jarak dari satu titik ke titik lain
- $y$  sebagai array untuk menyimpan nilai  $y$  pada setiap titik
- $z$  sebagai array untuk menyimpan nilai  $z$  pada setiap titik

- $Q$  sebagai queue penampung titik akan akan diproses sebagai titik aktif
- $S$  sebagai stack penampung titik jalur terpendek
- $M$  sebagai variabel nilai yang sangat besar
- $i$  dan  $j$  sebagai variabel yang dipakai untuk proses looping

Ilustrasi pemakaian struktur data diatas dapat dilihat pada gambar dibawah ini:



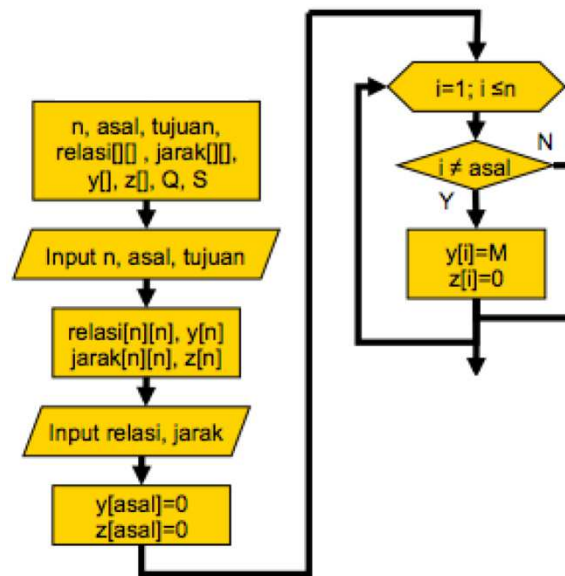
#### 11.1.4 Deklarasi dan Inisialisasi

Variabel terdiri dari  $n$ ,  $asal$ ,  $tujuan$ ,  $relasi$ ,  $jarak$ ,  $y$ ,  $z$ ,  $Q$ ,  $S$ ,  $i$  dan  $j$ . Dari sini, deklarasi dan inisialisasi variabel pada flowchart dapat dibuat sebagai berikut:

```
n, asal, tujuan,
relasi[], jarak[],
y[], z[], Q, S
```

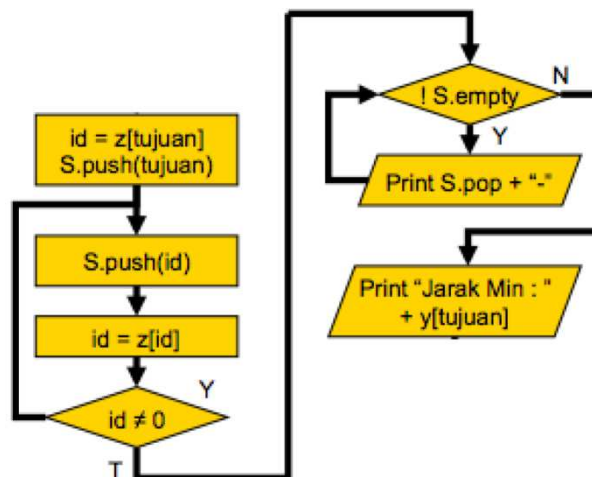
#### 11.1.5 Input

Pertama kali, user harus memasukkan banyaknya titik  $n$ , titik asal dan titik tujuan. Dari banyaknya titik tersebut, dibuatlah matriks input  $relasi$  dan  $jarak$ , array  $y$  dan  $z$ . Kemudian, user memasukkan nilai dari 2 matriks input. Selanjutnya dilakukan proses inisialisasi nilai  $y$  dan  $z$ . Flowchart untuk input ini dapat dilihat sebagai berikut:



### 11.1.6 Output

Solusi yang ingin didapatkan pada permasalahan shortest path problem tersebut adalah jarak terpendek dan jalur untuk jarak terpendek. Informasi jarak terpendek terdapat pada nilai  $z$  di titik tujuan. Sedangkan untuk mendapatkan informasi jalur terpendek, harus dilakukan operasi pop pada stack, sebagaimana yang telah dijelaskan pada bagian Cara Penyelesaian. Flowchart untuk output terlihat seperti dibawah ini:



## 11.2 Latihan

Buatlah flowchart dari permasalahan shortest path problem diatas. Deskripsikan proses penyelesaian dengan menggunakan struktur data yang ada, dan kemudian buatlah flowchart keseluruhan.