

Tim Penulis:

Meidyan Permata Putri, Guntoro Barovich, Rezania Agramanisti Azdy, Yuniansyah,
Andri Saputra, Yesi Sriyeni, Arsia Rini, Fadhila Tangguh Admojo.

ALGORITMA DAN STRUKTUR DATA



ALGORITMA DAN STRUKTUR DATA

Tim Penulis:

Meidyan Permata Putri, Guntoro Barovih, Rezania Agramanisti Azdy, Yuniansyah,
Andri Saputra, Yesi Sriyeni, Arsia Rini, Fadhila Tangguh Admojo.



ALGORITMA DAN STRUKTUR DATA

Tim Penulis:

Meidyan Permata Putri, Guntoro Barovih, Rezania Agramanisti Azdy, Yuniansyah,
Andri Saputra, Yesi Sriyeni, Arsia Rini, Fadhila Tangguh Admojo.

Desain Cover:

Fawwaz Abyan

Tata Letak:

Handarini Rohana

Editor:

Meidyan Permata Putri, S.Kom., M.Kom

ISBN:

978-623-459-182-8

Cetakan Pertama:

September, 2022

Hak Cipta 2022, Pada Penulis

Hak Cipta Dilindungi Oleh Undang-Undang

Copyright © 2022

by Penerbit Widina Bhakti Persada Bandung

All Right Reserved

Dilarang keras menerjemahkan, memfotokopi, atau memperbanyak sebagian atau seluruh isi buku ini tanpa izin tertulis dari Penerbit.

PENERBIT:

WIDINA BHAKTI PERSADA BANDUNG

(Grup CV. Widina Media Utama)

Komplek Puri Melia Asri Blok C3 No. 17 Desa Bojong Emas
Kec. Solokan Jeruk Kabupaten Bandung, Provinsi Jawa Barat

Anggota IKAPI No. 360/JBA/2020

Website: www.penerbitwidina.com

Instagram: @penerbitwidina

PRAKATA

Rasa syukur yang teramat dalam dan tiada kata lain yang patut kami ucapkan selain rasa syukur, karena berkat rahmat dan karunia-Nya buku yang berjudul Algoritma dan Struktur Data ini telah dapat di terbitkan untuk dapat dikonsumsi oleh khalayak banyak.

Struktur data dan algoritma berhubungan sangat erat pada sebuah program. Algoritma merupakan tahapan terencana yang menyatakan dengan jelas pemecahan suatu permasalahan dalam rentang waktu tertentu. Pemilihan struktur data dan algoritma yang kurang tepat akan membuat program menjadi kurang baik, demikian juga sebaliknya. Struktur data dan algoritma digunakan untuk memecahkan masalah kemungkinan bisa lebih dari satu. Struktur Data adalah cara mengumpulkan dan mengatur data sedemikian rupa sehingga kita dapat melakukan operasi pada sebuah data dengan cara yang efektif.

Struktur Data adalah tentang merender elemen data dalam beberapa hubungan, untuk organisasi dan penyimpanan yang lebih baik. Sebagai contoh, kami memiliki beberapa data yang memiliki, nama pemain “Virat” dan usia 26. Di sini “Virat” adalah tipe data *String* dan 26 adalah tipe data *integer*. Kami dapat mengatur data ini sebagai catatan seperti catatan Pemain, yang akan memiliki nama dan usia pemain di dalamnya. Sekarang kita dapat mengumpulkan dan menyimpan catatan pemain dalam file atau *database* sebagai struktur data. Misalnya: “Dhoni” 30, “Gambhir” 31, “Sehwag” 33. Jika Kamu mengetahui konsep pemrograman Berorientasi Objek, maka *Class* juga melakukan hal yang sama, ia mengumpulkan berbagai jenis data di bawah satu entitas tunggal. Satu-satunya perbedaan adalah, struktur data menyediakan teknik untuk mengakses dan memanipulasi data secara efisien.

Dalam bahasa sederhana, Struktur Data adalah struktur yang diprogram untuk menyimpan data yang dipesan, sehingga berbagai operasi dapat dilakukan dengan mudah. Ini mewakili pengetahuan tentang data yang akan diatur dalam memori. Ini harus dirancang dan

diimplementasikan sedemikian rupa sehingga mengurangi kompleksitas dan meningkatkan efisiensi. Sedangkan Algoritma adalah seperangkat instruksi atau logika yang terbatas, ditulis dalam rangka, untuk menyelesaikan tugas tertentu yang telah ditentukan sebelumnya. Algoritma bukanlah kode atau program yang lengkap, itu hanya logika inti (solusi) dari suatu masalah, yang dapat dinyatakan baik sebagai deskripsi tingkat tinggi informal sebagai kode semu atau menggunakan diagram alur.

Oleh karena itu buku yang berjudul Algoritma Dan Struktur Data ini hadir sebagai bagian dari upaya untuk menambah khazanah, diskusi Algoritma Dan Struktur Data. Akan tetapi pada akhirnya kami mengakui bahwa tulisan ini terdapat beberapa kekurangan dan jauh dari kata sempurna, karena sejatinya kesempurnaan hanyalah milik tuhan semata. Maka dari itu, kami dengan senang hati secara terbuka untuk menerima berbagai kritik dan saran dari para pembaca sekalian, hal tersebut tentu sangat diperlukan sebagai bagian dari upaya kami untuk terus melakukan perbaikan dan penyempurnaan karya selanjutnya di masa yang akan datang.

Terakhir, ucapan terimakasih kami sampaikan kepada seluruh pihak yang telah mendukung dan turut andil dalam seluruh rangkaian proses penyusunan dan penerbitan buku ini, sehingga buku ini bisa hadir di hadapan sidang pembaca. Semoga buku ini bermanfaat bagi semua pihak dan dapat memberikan kontribusi bagi pembangunan ilmu pengetahuan di Indonesia, khususnya terkait Algoritma dan Struktur Data.

September, 2022

Tim Penulis

DAFTAR ISI

PRAKATA	iii
DAFTAR ISI	v
BAB 1 PENGANTAR ALGORITMA	1
A. Pengertian Algoritma.....	1
B. Kegunaan Algoritma	2
C. Rangkuman Materi	6
BAB 2 FLOWCHART	9
A. Pendahuluan.....	9
B. Jenis-Jenis <i>Flowchart</i>	10
C. Simbol <i>Flowchart</i>	13
D. Keuntungan dan Kerugian <i>Flowchart</i>	17
E. Teknik Pembuatan <i>Flowchart</i>	18
F. Rangkuman Materi	19
BAB 3 PEMROGRAMAN JAVA	25
A. Pendahuluan.....	25
B. Struktur Program	43
C. Variabel	47
D. Tipe Data.....	48
E. Operator	50
F. Contoh	52
G. Rangkuman Materi	53
BAB 4 STRUKTUR PERCABANGAN	57
A. Pendahuluan.....	57
B. Konsep Percabangan	58
C. <i>Statement IF</i>	58
D. <i>Statement IF-ELSE</i>	60
E. <i>Statement IF-ELSE-IF</i>	62
F. <i>Statement Switch</i>	68
G. Rangkuman Materi	73
BAB 5 PERULANGAN	75
A. Pendahuluan.....	75
B. Perulangan <i>For</i>	76

C. Perulangan <i>While</i>	79
D. Perulangan <i>Do While</i>	83
E. Rangkuman Materi	87
BAB 6 ARRAY	89
A. Pendahuluan	89
B. Definisi <i>Array</i> dan Deklarasi <i>Array</i>	90
C. <i>Array</i> 1 Dimensi	91
D. <i>Array</i> 2 Dimensi	96
E. Manipulasi <i>String</i>	101
F. Rangkuman Materi	106
BAB 7 PROCEDURE DAN FUNCTION	109
A. <i>Procedure</i>	109
B. <i>Function</i>	117
C. Rangkuman Materi	124
BAB 8 SORTING DAN SEARCHING PADA ARRAY	127
A. Pendahuluan	127
B. <i>Sorting</i>	128
C. <i>Searching</i>	145
D. Rangkuman Materi	150
GLOSARIUM	152
PROFIL PENULIS	158



PENGANTAR ALGORITMA

Meidyan Permata Putri, S.Kom., M.Kom
Institut Teknologi dan Bisnis Palcomtech

PENDAHULUAN

A. PENGERTIAN ALGORITMA

Terdapat 3 aspek yang mesti dipertimbangkan dalam menciptakan suatu program, yakni sintaksis, semantik, serta kebenaran logika (Hanief & Jepriana, 2020). Sintaksis ialah struktur tata bahasa yang digunakan dalam program (Ed.D & Afifah, 2021). Semantik merupakan iktikad yang dikandung pada tiap *statment* di dalam program. Meskipun kebenaran logika berhubungan dengan benar-tidaknya urutan pernyataan dan prosedur yang terdapat dalam program, ataupun yang biasa diucap algoritma.

Algoritma matematika dan ilmu komputer adalah kumpulan perintah terkait untuk memecahkan masalah (Suhendar, Ali, & Suratman, 2021). Perintah-perintah ini dapat dikonversi langkah demi langkah dari awal sampai akhir. Persiapan memerlukan urutan serta logika supaya algoritma yang dihasilkan cocok dengan yang diharapkan. Algoritma merupakan bagian yang sangat berarti serta tidak terpisahkan dari pemrograman. Apalagi bila sintaks serta semantiknya benar, permasalahan yang dituntaskan dengan tata cara pemrograman tidak hendak sukses dengan

algoritma yang salah. Oleh sebab itu, saat sebelum menulis program aplikasi, Kamu wajib terlebih dulu menguasai algoritma ataupun pemecahan. Perihal ini buat membenarkan kalau program yang terbuat cocok dengan yang kamu mau.

Berikut yang wajib dipertimbangkan dalam membuat suatu algoritma (Andreswari., Sari, & Asmika, 2022).

1. Algoritma yang dibikin haruslah benar.
Maksudnya, algoritma wajib membagikan *output* yang sesuai. Dengan algoritma yang sesuai, hasil yang diharapkan bisa dicapai.
2. Selain sesuai, algoritma juga harus efektif.
Maksudnya, seberapa baik algoritma yang digunakan mendekati hasil yang diharapkan. Perihal ini sangatlah berarti paling utama bila kasus yang dialami lumayan rumit serta membutuhkan hasil yang mendekati sama.
3. Algoritma yang digunakan harus efisien.
Efisiensi suatu algoritma bisa ditinjau dari efisiensi waktu serta memori yang digunakan. Walaupun algoritma yang kita pakai mendekati hasil yang diharapkan, tetapi bila memerlukan waktu yang sangat lama, apalagi berjam-jam, umumnya algoritma itu tidak sering dipakai.

RINCIAN PEMBAHASAN MATERI

B. KEGUNAAN ALGORITMA

Algoritma memiliki lima karakteristik kunci yang terkait dengannya (Kusumaningtyas & Wahyuddin, 2022).

1. *Input* merupakan permasalahan di mana Kamu berpengalaman serta mau mencari pemecahan. Algoritma mempunyai nol ataupun lebih nilai *input*.
2. Proses, yakni langkah-langkah yang harus dicoba buat mencapai tujuan akhir.
3. *Output* yakni pemecahan ataupun penanda akhir yang diperoleh dari algoritma. Algoritma mempunyai paling tidak satu *output*.
4. Instruksi-instruksi yang jelas dan tidak ambigu, yakni instruksi yang jelas dalam algoritma sehingga tidak terjalin kesalahan dalam menghasilkan *output*.

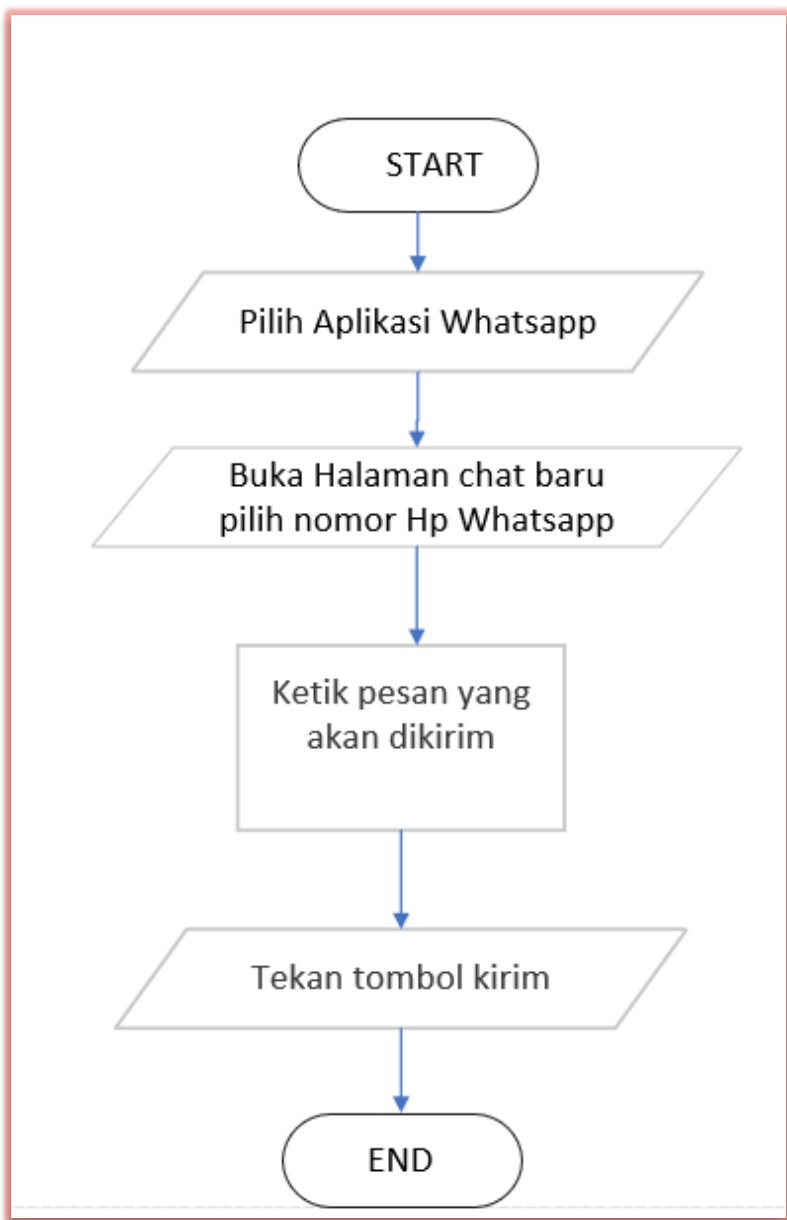
5. Tujuan akhir yang dicapai merupakan akhir program, serta program berakhir kala tujuan akhir tercapai.

Pada dasarnya, pemakaian utama dari algoritma merupakan buat membongkar permasalahan. Adapun Fungsi algoritma sebagai berikut (Sulasmoro, 2022):

1. Dapat Menyederhanakan program yang lingkungan serta besar.
2. Buat memudahkan memprogram buat sesuatu permasalahan tertentu.
3. Algoritma bisa digunakan kesekian kali buat menuntaskan permasalahan.
4. Menolong mengklasifikasikan permasalahan secara logis serta sistematis.
5. Meminimalkan persiapan program.
6. Sanggup mempraktikkan pendekatan *top-down, divide and rule*.
7. Buat lebih gampang membuat program yang lebih baik serta terstruktur, lebih gampang dimengerti serta diperluas.
8. Mempermudah proses modifikasi pada program karena bisa dicoba hanya pada satu modul tanpa harus mengubah modul yang lain.
9. Kala terjalin kesalahan, algoritma dapat membantu menciptakannya karena alur kerja yang jelas.
10. Mempermudah proses dokumentasi.

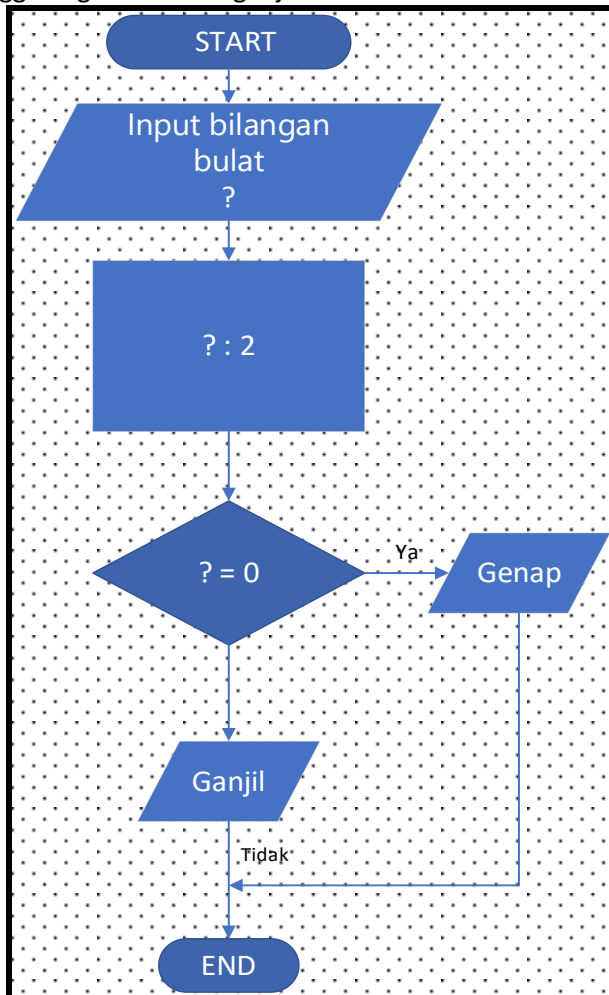
Algoritma tidak hanya dalam ilmu komputer dan matematika saja Algoritma juga dapat kita jumpai dalam kehidupan sehari-hari misalnya:

1. Algoritma mengirim Pesan Whatsapp baru
 - a. Buka aplikasi Whatsapp pada ponsel
 - b. Buka Halaman chat baru pilih nomor Hp Whatsapp
 - c. *Input* pesan yang akan dikirim
 - d. Klik tombol kirim
 - e. Chat Whatsapp akan terkirim



Gambar 1.1 *Flowchart* Mengirim Pesan Whatsapp Baru.

2. Algoritma memastikan Angka ganjil ataupun genap:
- Input* angka bulat.
 - Angka di untuk 2.
 - Bila angka tersebut habis dipecah dengan 2 hingga angka tersebut genap.
 - Bila angka tersebut tidak habis dipecah dengan 2 (ada sisa 1) hingga angka tersebut ganjil.



Gambar 1.2 Flowchart Menentukan Angka Ganjil Genap.

C. RANGKUMAN MATERI

Algoritma adalah kumpulan instruksi terkait untuk memecahkan masalah. Perintah dapat dikonversi langkah demi langkah dari awal hingga akhir. Persiapan membutuhkan urutan dan logika agar algoritma yang dihasilkan sesuai dengan yang diharapkan. Ada tiga aspek yang perlu dipertimbangkan saat menulis program: sintaks, semantik, dan kebenaran logika.

TUGAS DAN EVALUASI

1. Buatlah algoritma untuk menentukan *input* angka genap atau ganjil:
output dari angka: genap/ganjil
2. Buatlah algoritma untuk menentukan suatu bilangan adalah bilangan prima atau bukan.
3. Buatlah algoritma untuk untuk menghitung akar-akar persamaan kuadrat dengan rumus $A = C^2 - 8 * D * E$
Jika $A < 0$ maka didapat akar imajiner
Jika $A = 0$ maka $X1 = X2$ yang didapat dari $A = -C / (2 * A)$
Jika $A > 0$ maka ada dua akar $X1 = -C + \sqrt{A / 2 * D}$ dan $X2 = -C - \sqrt{A / 2 * D}$
4. Jelaskan pengertian dari algoritma ?
5. Jelaskan Fungsi algoritma dan buatlah contoh algoritma yang sering kita gunakan sehari-hari.

DAFTAR PUSTAKA

- Hanief, S., & Jepriana, I. (2020). *Konsep Algoritme dan Aplikasinya Dalam Bahasa Pemrograman C++*. Penerbit Andi. Yogyakarta: Andi.
- Kusumaningtyas, G. Y., & Wahyuddin, M. I. (2022). Implementasi Algoritma C4.5 dan Simple Additive Weight Untuk Menentukan KPIKaryawan. *Building of Informatics, Technology and Science (BITS)*, 519-527.
- Suhendar, A. M., Ali, S., & Suratman, A. (2021, Desember). MEMBANGUN BERPIKIR KREATIF, SISTEMATIS DAN LOGIS MATEMATIS. *Perspektif*, 15, 176-190.
- Andreswari., D., Sari, J. P., & Asmika, V. (2022, Maret). IMPLEMENTASI CASE BASED REASONING UNTUK MENDIAGNOSIS GANGGUAN PADA SISTEM PENCERNAAN MANUSIA MENGGUNAKAN ALGORITMA SIMILARITAS NEYMAN BERBASIS WEB. *Rekursif*, 10, 12-22.
- Ed.D, P. Z., & Afifah, W. (2021). *Analisis Konten Etnografi & Grounded Theory, dan Hermeneutika Dalam Penelitian*. Jakarta Timur: PT Bumi Askara.
- Sulasmoro, A. H. (2022). *Buku ajar algoritma dan pemrograman I*. Lombok Tengah NTB: Pusat Pengembangan Pendidikan dan Penelitian Indonesia.



BAB 2

FLOWCHART

Guntoro Barovich, S.Kom., M.Kom
Institut Teknologi dan Bisnis PalComTech

A. PENDAHULUAN

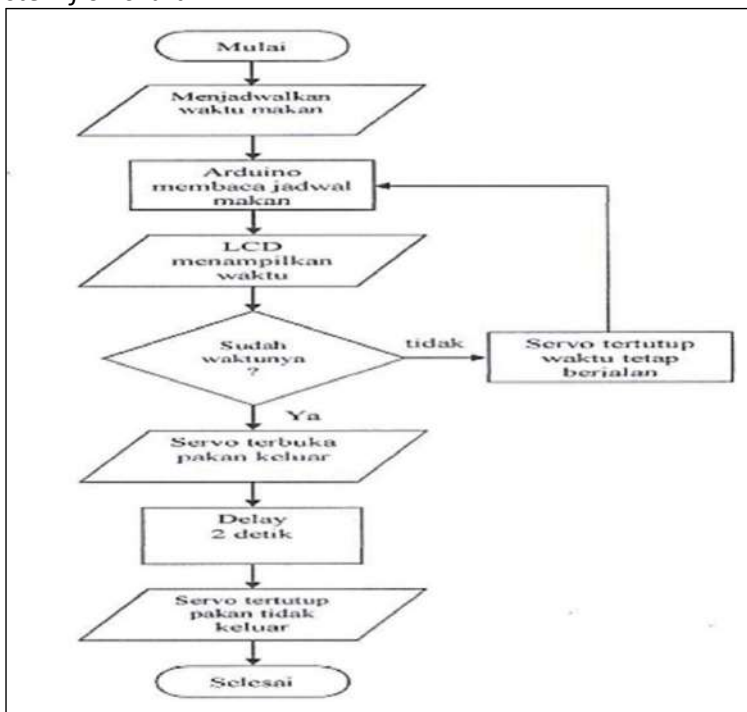
Komputer digunakan untuk membantu manusia dalam memecahkan permasalahan-permasalahan yang kompleks. Agar komputer bisa memecahkan permasalahan yang kompleks tersebut maka diperlukan suatu metode yang mampu menjabarkan kronologis suatu proses kejadian sehingga bisa menciptakan suatu solusi yang dibutuhkan. *Flowchart* merupakan diagram alir yang disajikan secara sistematis dengan tampilan grafis yang menggambarkan suatu proses dan logika dari kegiatan penanganan informasi yang memuat urutan-urutan atau langkah-langkah prosedur pada suatu program yang digunakan dalam penyelesaian masalah untuk dipelajari dan dievaluasi lebih lanjut (Arief *et al.*, 2019). *Flowchart* sangat membantu analis dan *programmer* dalam memecahkan masalah dalam membangun atau mengembangkan aplikasi ke dalam segmen yang lebih kecil. Dalam banyak literatur tentang pengembangan perangkat lunak, *flowchart* berfungsi sebagai dokumen desain sistem di mana analis sistem, *programmer*, dan pengguna berkomunikasi, bernegosiasi, dan mewakili kompleksitas. Namun makna dari *flowchart* tertentu sering kali sangat diperdebatkan, dan kekhususan yang tampak

dari dokumen desain semacam itu jarang mencerminkan kenyataan (Ensmenger, 2016).

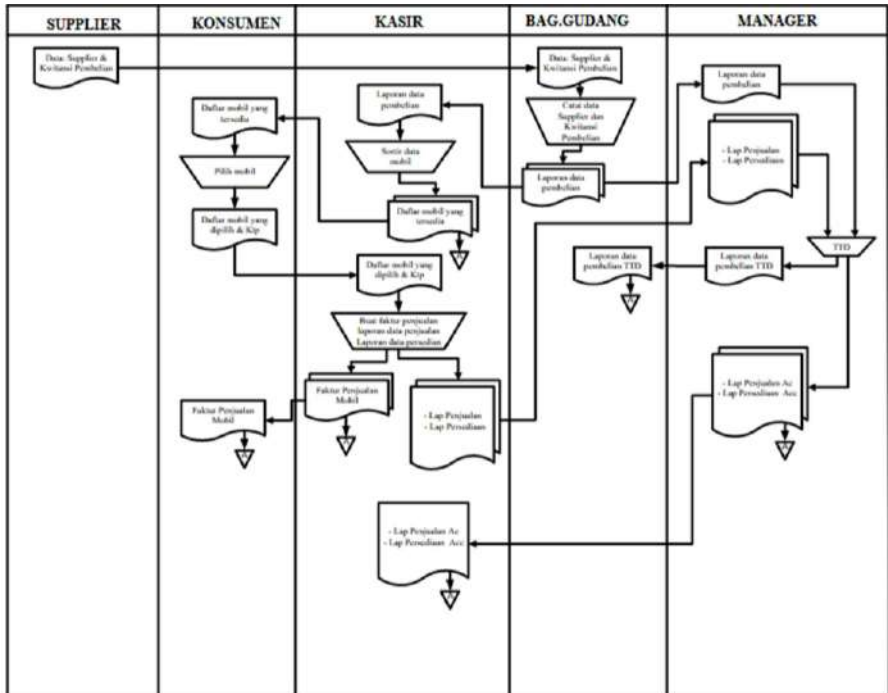
B. JENIS-JENIS *FLOWCHART*

Berdasarkan jenisnya dan kegunaannya *flowchart* di bagi menjadi 5 jenis, yaitu: *system flowchart*, *program flowchart*, *document flowchart*, *schematic flowchart* dan *process flowchart* (Agusvianto, 2017). Adapun pengertian dari jenis *flowchart* tersebut antara lain:

- a. *System Flowchart*: Suatu *flowchart* yang menunjukkan arus pekerjaan dan menjelaskan urutan-urutan dari suatu prosedur yang ada di dalam sistem. *Flowchart* sistem digunakan oleh analis sistem untuk menunjukkan berbagai proses, sub sistem, keluaran dan operasi pada data dalam suatu *system*. Gambar 2.1 memperlihatkan contoh dari *system flowchart*.



Gambar 2.1 Contoh *flowchart* pakan burung otomatis menggunakan Arduino berbasis IoT (Rakhman & Rais, 2020)



Gambar 2.3 Contoh Flowchart AnalisaSistem Inventory data mobil pada PT. Andalas Berlian Motor Bukit Tinggi (Veza, 2017).

- Schematic Flowchart:** Suatu bagan alir yang digunakan untuk menggambarkan suatu prosedur sistem dengan menggunakan simbol-simbol komputer dan perangkat digital yang bertujuan untuk memudahkan komunikasi dengan orang-orang awam yang tidak paham dengan simbol-simbol tersebut. *Schematic flowchart* hampir sama dengan *system flowchart*, perbedaannya terletak dalam penggunaan *symbol* untuk menjelaskan alur proses atau prosedur sistem.
- Process Flowchart:** Suatu diagram yang menunjukkan langkah-langkah berurutan dari suatu proses dan keputusan yang diperlukan untuk membuat proses tersebut bekerja. Dalam representasi visual, setiap langkah ditunjukkan oleh sebuah bentuk, dimana bentuk-bentuk ini dihubungkan oleh garis dan panah untuk menunjukkan gerakan dan

arah proses. Bagan alur proses distandarisasi sedemikian rupa sehingga siapa pun yang memiliki pemahaman tentang bagan alur dapat melihatnya dan mengetahui apa yang terjadi. Mereka mengikuti arus informasi yang logis sehingga pemangku kepentingan bisnis memiliki panduan tentang bagaimana memenuhi proses dengan benar. *Process flowchart* bisa digambarkan dengan tangan atau ditulis dan bisa juga menggunakan bantuan perangkat lunak.

Dalam dunia komputer khususnya yang sering digunakan dalam pembangunan dan pengembangan aplikasi sering menggunakan *system flowchart* atau program *flowchart* untuk mengetahui alur proses sistem yang akan di bangun nanti. *Flowchart* dalam suatu metode pengembangan perangkat lunak atau aplikasi, biasanya diterapkan di fase atau langkah desain sistem. Sehingga *programmer* bisa mengetahui apa saja yang dibutuhkan dalam sistem yang dikembangkan tersebut.







C. SIMBOL *FLOWCHART*

Flowchart merupakan representasi visual dari algoritma, dimana proses komputasi diwakili oleh simbol yang dikaitkan melalui anak panah yang merepresentasikan aliran algoritma. Simbol-simbol pada *flowchart* merujuk pada standarisasi yang sudah ditetapkan oleh *American National Standards Institute* (ANSI) (Penaloza & Gonzalez, 2019). Elemen-elemen simbol *flowchart* digunakan untuk membantu memvisualisasikan proses, memfasilitasi pemahaman aliran data yang diterima oleh program, dapat mengambil jalur dan proses yang berbeda yang tergantung pada desain solusi yang diinginkan. Setelah melakukan desain solusi menggunakan *flowchart*, maka kita bisa memilih untuk menggunakan *pseudocode*, yaitu kombinasi natural *language*, pernyataan-pernyataan logis dan operasi aritmatika untuk merancang solusi yang lebih spesifik dan dapat diimplementasikan dalam bahasa pemrograman. Simbol-simbol pada *flowchart* terbagi menjadi beberapa kategori dan sesuai fungsinya, yaitu *input/output*, pemrosesan, arah aliran/*flowlines* dan anotasi (*American National Standards Institute*, 1970). Kategori *flowchart* jika dilihat dari kebutuhannya antara lain:

a. **Basic symbol (symbol dasar)**

Simbol dasar dibuat untuk setiap fungsi dan selalu dapat digunakan untuk mewakili fungsi tersebut. Simbol dasar ini seperti tampak pada tabel 2.1.




Tabel 2.1 Basic Symbol

Input/Output 	Simbol Masukan/Keluaran merupakan simbol yang merepresentasikan fungsi input/output (I/O), yaitu menyediakan informasi untuk diproses (input), atau merekam informasi yang diproses (output).
Process 	Simbol yang mewakili segala jenis fungsi pemrosesan. misalnya, proses menjalankan operasi atau kelompok operasi tertentu yang menghasilkan perubahan nilai, bentuk, atau lokasi informasi, atau dalam penentuan mana dari beberapa arah aliran yang harus diikuti.
Flowline 	Simbol yang mewakili fungsi menghubungkan antar simbol. menunjukkan urutan informasi dan operasi yang tersedia dan yang dapat dieksekusi
Crossing of Flowlines 	simbol yang menyatakan aliran informasi boleh menyeberang yang berarti aliran informasi memiliki logika hubungan yang tidak timbal balik
Junction of Flowlines 	simbol dengan dua atau lebih banyak aliran masuk dapat bergabung dengan satu alur keluar.
Annotation, Comment 	simbol ini mewakili fungsi anotasi, yaitu penambahan komentar deskriptif atau catatan penjelasan sebagai klarifikasi. Garis putus-putus yang terhubung ke simbol apa pun pada titik di mana anotasinya adalah bermakna dengan memperpanjang garis putus-putus di mode apa pun yang sesuai

b. *Specialized Symbol* (Simbol Khusus)

Specialized symbol atau simbol khusus yang ditetapkan dapat digunakan sebagai pengganti simbol dasar untuk memberikan informasi tambahan. Simbol khusus ini tampak seperti pada tabel 2.2










Tabel 2.2 *Specialized Input/Output Symbol*

	Simbol ini mewakili fungsi I/O di mana mediana adalah kartu berlubang, termasuk: kartu tanda pengenal, kartu parsial, kartu pindai (smart card).		Simbol ini mewakili fungsi I/O di mana: mediana adalah inti magnet.
	Simbol ini merupakan kumpulan punched card.		Simbol ini mewakili fungsi I/O di mana mediana adalah dokumen.
	Simbol ini merupakan kumpulan terkait catatan kartu berlubang.		Simbol yang ditunjukkan di bawah ini mewakili fungsi input di mana informasi dimasukkan secara manual pada saat pemrosesan. misalnya, melalui keyboard, pengaturan, push button.
	Simbol ini mewakili fungsi I/O memanfaatkan segala jenis penyimpanan online, misalnya pita magnetik, drum magnetik, disk magnetik.		Simbol ini mewakili fungsi I/O dimana informasi ditampilkan untuk digunakan manusia pada saat pemrosesan, melalui indikator online, perangkat video, printer konsol, plotters, dan sebagainya.
	Simbol ini mewakili fungsi I/O dimana mediana adalah pita magnetik.		Simbol ini mewakili fungsi di mana informasi ditransmisikan oleh: link telekomunikasi.
	Simbol ini mewakili fungsi I/O dimana mediana adalah pita berlubang.		Simbol ini mewakili fungsi menyimpan informasi secara offline, terlepas dari media dimana informasi tersebut direkam.
	Simbol ini mewakili fungsi I/O di mana mediana adalah cakram magnetik.		Simbol ini mewakili fungsi I/O di mana mediana adalah drum magnet.

c. *Specialized Process Symbol* (Simbol khusus proses)

Simbol proses khusus dapat mewakili fungsi pemrosesan dan, sebagai tambahan, mengidentifikasi jenis operasi spesifik yang akan dilakukan pada informasi. Jika tidak ada simbol khusus, simbol proses dasar digunakan. Simbol proses khusus tampak seperti pada tabel 2.3.




Tabel 2.3 Specialized Process Symbol

Decision 	Simbol ini mewakili keputusan atau operasi tipe switching yang menentukan mana dari sejumlah jalur alternatif yang harus diikuti.	Auxiliary Operation 	Simbol ini menunjukkan operasi offline yang dilakukan pada peralatan yang tidak berada di bawah kendali langsung pusat unit pemrosesan.
Predefined Process 	Simbol ini mewakili proses yang terdiri dari satu atau lebih operasi atau langkah-langkah program yang ditentukan di tempat lain. misalnya, subrutin atau unit logis.	Merge 	Simbol ini mewakili penggabungan dua atau lebih set item menjadi satu set.
Preparation 	simbol ini merupakan modifikasi dari instruksi atau kelompok instruksi yang mengubah program itu sendiri. misalnya, set switch, memodifikasi register indeks, dan menginisialisasi rutin.	Extract 	Simbol ini mewakili penghapusan satu atau lebih set item tertentu dari satu set item.
Manual Operation 	Simbol ini mewakili proses offline yang disesuaikan dengan kecepatan manusia, tanpa menggunakan bantuan mekanik.	Sort 	Simbol ini mewakili pengaturan satu set item ke dalam urutan tertentu.
Collate 	Simbol ini mewakili penggabungan dengan ekstraksi, yaitu pembentukan dua atau lebih set item dari dua atau lebih set lainnya.		

d. Additional Symbol (symbol tambahan)

Simbol tambahan dapat mewakili percabangan dari keluaran atau entri pada suatu diagram alur, mewakili terminal awalan atau akhir dari dua atau lebih operasi simultan. Simbol tambahan ini seperti tampak pada tabel 2.4.

Tabel 2.4 Additional Symbol

Connector 	Simbol ini mewakili jalan keluar ke atau entri dari bagian lain diagram alur. Ini adalah persimpangan dalam garis aliran. Satu set konektor digunakan untuk mewakili arah aliran lanjutan. Satu set konektor digunakan untuk mewakili persimpangan beberapa garis aliran dengan satu garis aliran/flowline alternatif.
Terminal 	Simbol ini mewakili titik terminal dalam diagram alur, misalnya, mulai, berhenti, hentikan, tunda, atau interupsi
Parallel Mode 	Simbol ini mewakili awal atau akhir dari dua atau lebih operasi simultan.

D. KEUNTUNGAN DAN KERUGIAN *FLOWCHART*

Flowchart dalam penerapannya juga memiliki keuntungan dan kerugian. Keuntungan dan kekurangan tersebut antara lain (Rajpoot *et al.*, 2021):

a. Keuntungan penggunaan *flowchart*

1. Komunikasi yang lebih baik

Flowchart merupakan representasi bergambar dari suatu program, dan lebih mudah bagi seorang *programmer* untuk menjelaskan logika program kepada para *programmer* lainnya menggunakan *flowchart* dari pada program itu sendiri. *Flowchart* juga digunakan sebagai cara untuk menjelaskan logika sistem, langkah-langkah yang terlibat dalam mendapatkan solusi khususnya kepada pengguna sistem.

2. Analisis yang efektif

Adanya *flowchart*, keseluruhan program dapat dianalisis secara efektif. Secara jelas menentukan alur langkah-langkah yang membentuk program. *Flowchart* dapat digunakan untuk analisis masalah yang efektif.

3. Integrasi yang Efektif

Flowchart memudahkan para program untuk mengintegrasikan modul-modul yang dikembangkan secara terpisah untuk menjadi 1 program secara utuh.

4. Efisiensi dalam pengkodean

Flowchart menjadi peta jalan bagi para *programmer*, karena menjadi pemandu bagi mereka dalam pengkodean mulai dari awal hingga ketahap akhir dan beserta langkah-langkah yang harus dilakukan

5. Baik dalam proses *debug*

Flowchart membantu dalam proses *debug*.

6. Dokumentasi yang baik

Flowchart program adalah bagian penting dari dokumentasi program yang baik. Dokumen program digunakan untuk berbagai keperluan seperti mengetahui komponen-komponen dalam program, kompleksitas program, dan lain-lain.

7. Kemudahan dalam perawatan program

Setelah sebuah program dikembangkan dan menjadi operasional, maka akan membutuhkan pemeliharaan dari waktu ke waktu. *flowchart* membuat perawatan menjadi lebih mudah karena sudah memuat peta jalan program.

b. Kerugian penggunaan *flowchart*

Terlepas dari banyaknya keuntungan yang didapatkan dari penggunaan *flowchart*, tetapi juga memiliki beberapa kekurangan diantaranya:

1. Membutuhkan waktu yang lama

Flowchart membutuhkan waktu yang lama dalam menggambarkan alur dan langkah-langkah proses algoritma yang sangat kompleks.

2. Perubahan dan modifikasi

Sulit untuk mengubah diagram alur. perubahan langkah logika pada diagram alur akan berpengaruh pada langkah logika lainnya dan bisa berakibat pada perubahan secara keseluruhan dari diagram alur yang sudah di buat.

3. Hanya berupa visual program

Flowchart hanya berupa visualisasi langkah logika program dan bukanlah program yang sebenarnya. Tidak ada standarisasi yang menentukan kompleksitas diagram alur yang digunakan dari langkah logika program yang akan di bangun.

4. Logika yang kompleks

Terkadang logika program cukup rumit. dalam hal ini, diagram alur menjadi kompleks. ini akan menyulitkan pengguna, sehingga membuang-buang waktu untuk memperbaiki suatu masalah

5. Reproduksi *symbol*

Karena simbol *flowchart* tidak dapat diketik, reproduksi *flowchart* menjadi masalah

E. TEKNIK PEMBUATAN *FLOWCHART*

Beberapa petunjuk yang harus diperhatikan dalam pembuatan *flowchart* bagi seorang analis dan *programmer*, antara lain:

- a. Penggambaran *Flowchart* dilakukan dari halaman atas ke bawah dan dari kiri ke kanan.
- b. Aktivitas yang digambarkan dan didefinisikan harus penuh kehati-hatian dan pendefinisian setiap langkah atau alur yang digunakan harus bisa dipahami oleh pembacanya.
- c. Awalan dan akhiran aktivitas harus ditentukan dengan jelas.
- d. Setiap aktivitas dan langkah harus diuraikan dengan menggunakan deskripsi kata kerja yang jelas.
- e. Langkah-langkah aktivitas harus berada pada urutan yang benar.
- f. Lingkup dan batasan dalam aktivitas yang digambarkan harus ditelusuri dengan hati-hati. Setiap percabangan yang memotong suatu aktivitas yang sedang digambarkan tidak perlu digambarkan dalam *flowchart* yang sama. Simbol konektor harus digunakan dan percabangannya diletakkan pada halaman yang terpisah atau bisa dihilangkan seluruhnya bila percabangannya tidak berkaitan dengan sistem.
- g. Gunakanlah simbol-simbol *flowchart* yang standar.

F. RANGKUMAN MATERI

Flowchart merupakan diagram alir dengan tampilan grafis yang digambarkan secara sistematis dan digunakan untuk menggambarkan suatu proses logika program yang disusun secara berurutan sehingga bisa dipelajari untuk mendapatkan pemecahan masalah dalam pengembangan aplikasi atau program. *Flowchart* terdiri dari 5 jenis *flowchart* berdasarkan jenis dan kegunaannya, antara lain:

- a. *Flowchart* sistem yang lebih dominan digunakan oleh analis untuk menjelaskan urutan-urutan prosedur dalam sebuah sistem.
- b. Program *flowchart* terdiri dari 2 macam yaitu program *logic flowchart* dimana digunakan untuk menggambarkan tiap-tiap langkah *logic* dalam program dan *detailed computer program flowchart* digunakan untuk menggambarkan instruksi-instruksi program atau sistem secara terinci. program *flowchart* mencakup beberapa hal-hal, antara lain: Struktur Program, Logika Program, Data *input*, Data *Processing*, *Condition*, Urutan Percabangan & Perulangan, *Result*/Hasil, *Output*. Hal yang paling penting dalam *Flowchart* ini adalah logika.

- c. *Document flowchart* digunakan untuk menelusuri alur *form* dari satu bagian ke bagian lainnya termasuk bagaimana *form* diterima, kemudian di proses, di catat dan di simpan.
- d. *Schematic flowchart* digunakan untuk menggambarkan suatu prosedur sistem dengan menggunakan simbol-simbol komputer dan perangkat digital yang bertujuan untuk memudahkan komunikasi dengan orang-orang awam.
- e. *Process flowchart* digunakan untuk mengurutkan langkah-langkah dari suatu proses hingga didapatkan suatu keputusan yang diperlukan dari proses kerja

Dunia Komputer dan pemrograman biasanya menggunakan 2 jenis *flowchart* saja yaitu *system flowchart* dan *program flowchart* yang digunakan untuk membangun atau mengembangkan suatu aplikasi atau perangkat lunak. *Flowchart* sendiri terdiri dari 4 jenis simbol yaitu:

- a. *Basic simbol* yang merupakan simbol-simbol dasar dari suatu *flowchart* biasanya hanya terdiri dari proses, *input/output*, arah komunikasi antar simbol (*flowlines*).
- b. *Input/output symbol*. Simbol ini berisikan simbol-simbol khusus dalam *flowchart*, dimana simbol ini merepresentasikan *input/output* dari media penyimpanan data mulai dari media penyimpanan data dalam bentuk tape *magenetic*, *punchcard* hingga tipe *magnetic disk* atau biasa dikenal sebagai *storage* media.
- c. *Process symbol*. Simbol ini lebih menitikberatkan pada proses dari suatu alur data. isi dari simbol ini berisikan simbol-simbol yang menjalankan fungsi pemrosesan, penggabungan, penambahan dan identifikasi jenis operasi termasuk perubahan arah proses data.
- d. Simbol tambahan, simbol ini berisikan awal dan akhir dari suatu urutan atau langkah proses, konektor dan *parallel mode*

Keuntungan yang bisa didapatkan dari *flowchart* adalah kemudahan dalam berkomunikasi dalam menyelesaikan permasalahan dalam aplikasi, keefektifan dalam analisis, pengintegrasian antar modul yang efektif jika perangkat lunak dikembangkan sangat kompleks, efisiensi dalam penulisan kode karena pemetaan permasalahan dan instruksi program

sudah sangat jelas, bisa digunakan untuk melakukan proses *debugs*, dokumentasi program yang sangat baik karena memuat peta jalan aplikasi atau program dan kemudahan dalam perawatan program. Sedangkan kerugian dalam penggunaan *flowchart* adalah kebutuhan waktu dalam mendeskripsikan alur atau langkah program sangat lama, sulit dalam melakukan perubahan langkah proses program karena jika terjadi perubahan pada langkah proses program maka akan berdampak pada perubahan pada langkah-langkah yang lain, bentuknya yang hanya diagram alur langkah program sehingga para *programmer* harus menterjemahkan kembali dalam bentuk *code-code* pada bahasa pemrograman yang digunakan, semakin kompleks program yang akan dikembangkan, maka semakin kompleks juga *flowchart* yang digambarkan serta simbol-simbol tersebut harus digunakan satu persatu. Pembuatan *flowchart* dilakukan dari atas kebawah dan dari kiri ke kanan, setiap langkah proses harus didefinisikan dengan baik dan simbol yang digunakan harus sesuai dengan standar.

TUGAS DAN EVALUASI

Berikut beberapa tugas yang bisa anda gunakan untuk meningkatkan pemahaman anda tentang *flowchart* dan bagaimana cara menggunakan simbol-simbol pada *flowchart*.

1. Gambarkan *flowchart* pendaftaran mahasiswa baru, dimana alur prosesnya sebagai berikut:
 - a. Calon mahasiswa datang ke bagian pendaftaran
 - b. Membeli dan mengisi *form* pendaftaran
 - c. Membayar uang pendaftaran dan uang semester awal
 - d. Mengumpulkan dan menyerahkan persyaratan
 - e. Jika berkas persyaratan sudah lengkap? jika kurang lengkap maka akan kembali ke proses pengumpulan dan penyerahan dokumen persyaratan
 - f. Menerima kartu mahasiswa, jadwal orientasi dan kuliah
 - g. Mengikuti orientasi
 - h. Kuliah
 - i. Selesai

2. Gambarkan *flowchart input* nilai semester, dimana alur proses *flowchart* sebagai berikut:
 - a. *Input* nim, nama mahasiswa dan program studi
 - b. *Input* nilai tugas kelas, nilai tugas mandiri, UTS dan UAS
 - c. Rumus rata-rata nilai tugas = $(tk1 + tk2 + tk3 + tk4 + tk5 + tk6 + tk7 + tk8 + tk9 + tk10) / 10$. Nilai total akhir semester = $((0.4 * \text{rata-rata nilai tugas}) + (0.2 * \text{tugas mandiri}) + (0.2 * \text{UTS}) + (0.2 * \text{UAS}))$. Jika nilai akhir semester $0 - 40 = E$ (maaf anda tidak lulus), $41-59 = D$ (maaf anda tidak lulus), $60-69 = C$ (lulus dengan nilai cukup), $70-84 = B$ (selamat anda lulus) , $85-100 = A$ (selamat anda lulus dengan memuaskan)
 - d. Tampilkan nilai akhir dan keterangannya
 - e. Selesai
3. Gambarkan *flowchart login* dan daftar *user* baru pada suatu aplikasi dimana alur proses *flowchart* sebagai berikut:
 - a. Buka aplikasi
 - b. *Sign up* atau *sign in*
 - c. *Sign in*, tampil halaman *sign in*
 - d. Masukkan *user* dan *password*
 - e. Periksa data *user* dan *password*
 - f. Jika salah masuk ke halaman awal *login*.
 - g. Jika benar masuk ke menu utama
 - h. Proses selesai
 - i. *Sign up*, tampil halaman pendaftaran *user* baru
 - j. Masukkan nama lengkap, *username*, email, *password*, *re-password*.
 - k. Proses simpan
 - l. Jika data tidak lengkap kembali ke *form* pendaftaran
 - m. Jika lengkap data simpan dalam *database*
 - n. Tampilkan *pop up* berhasil simpan
 - o. Kembali ke halaman *login*
 - p. Selesai

4. Gambarkan *flowchart* dokumen proses pendaftaran untuk menjadi anggota baru perpustakaan. Dalam alur ini ada 3 aktor yang terlibat yaitu Anggota, Administrasi, Kepala Perpustakaan. Alur proses dalam kegiatan ini sebagai berikut:
- a. Anggota menerima dokumen formulir.
 - b. Anggota mengisi formulir yang telah diterima.
 - c. Formulir yang telah diisi selanjutnya diberikan kepada bagian administrasi dan disimpan ke dalam *database*.
 - d. Bagian administrasi menerbitkan kartu anggota.
 - e. Kartu anggota baru diserahkan kepada kepala perpustakaan
 - f. Kepala perpustakaan melakukan paraf dan mengembalikan kartu anggota kepada bagian administrasi
 - g. Bagian administrasi membubuhkan cap stempel ke kartu anggota baru dan memberikan kartu anggota baru ke yang bersangkutan.

DAFTAR PUSTAKA

- Agusvianto, H. (2017). Sistem Informasi Inventori Gudang Untuk Mengontrol Persediaan Barang Pada Gudang Studi Kasus : PT.Alaisys Sidoarjo. *Journal of Information Engineering and Educational Technology*, 1(1), 40. <https://doi.org/10.26740/jieet.v1n1.p40-46>
- American National Standards Institute. (1970). Flowchart symbols and their usage in information processing. *Ansi, X3.5*(American National Standard Institute), 24. <https://nvlpubs.nist.gov/nistpubs/Legacy/FIPS/fipspub24.pdf>
- Arief, S., Safi'i, I., & Laela, N. (2019). Mekanisme Pembuatan Flowchart Penerimaan Pinjaman (Angsuran) pada (Bumdes) Di Desa Pomahan Kecamatan Pulung Kabupaten Ponorogo. *Jurnal Abdikarya : Jurnal Karya Pengabdian Dosen Dan Mahasiswa*, 03(03), 259–264.
- Ensmenger, N. (2016). The Multiple Meanings of a Flowchart. *Information & Culture*, 51(3), 321–351. <https://doi.org/10.7560/IC51302>

- Febriani, O., & Putra, A. (2013). Sistem Informasi Monitoring Inventori Barang Pada Balai Riset Standardisasi Industri Bandar Lampung. *Jurnal Informatika Darmajaya*, 13(1), 90–98.
- INTERNET OF THINGS. *Syntax Literate : Jurnal Ilmiah Indonesia*, 5(5), 18–25. <https://doi.org/http://dx.doi.org/10.36418/syntax-literate.v5i5.1151>
- Penaloza, F. vazquez, & Gonzalez, C. R. J. (2019). Towards a web application to create flowcharts for supporting the teaching-learning process of structured programming courses. *American Journal of Educational Research*, 7(12), 976–982. <https://doi.org/10.12691/education-7-12-12>
- Rajpoot, V., Agarwal, R., & Chaturvedi, P. (2021). *BASIC COMPUTER ENGINEERING*. Horizon Books (A Division of Ignited Minds Edutech P Ltd). <https://books.google.co.id/books?id=EJMSEAAAQBAJ>
- Rakhman, A., & Rais. (2020). ANALISA PAKAN BURUNG OTOMATIS MENGGUNAKAN ARDUINO BERBASIS
- Supaartagorn, C. (2018). Web application for automatic code generator using a structured flowchart. *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS, 2017-Novem*, 114–117. <https://doi.org/10.1109/ICSESS.2017.8342876>
- Veza, O. (2017). PERANCANGAN SISTEM INFORMASI INVENTORY DATA BARANG PADA PT.ANDALAS BERLIAN MOTORS (Studi Kasus : PT Andalas Berlian Motors Bukit Tinggi). *Jurnal Teknik Ibnu Sina (JT-IBSI)*, 2(2), 121–134. <https://doi.org/10.36352/jt-ibsi.v2i2.63>



PEMROGRAMAN JAVA

Rezania Agramanisti Azdy, S.Kom., M.Cs
Institut Teknologi dan Bisnis PalComTech

A. PENDAHULUAN

Dalam membuat sebuah program, diperlukan sebuah bahasa pemrograman yang dapat menerjemahkan logika si pembuat program menjadi sebuah aplikasi atau sistem yang dapat *terinstall* di perangkat komputer. Java merupakan salah satu bahasa pemrograman yang dapat digunakan untuk membuat sebuah aplikasi atau sistem yang dapat menjembatani antara *user* dengan komputer. Bahasa Java pada awalnya dikembangkan pada tahun 1991 dengan nama "*Oak*" dan dirilis pada publik pertama kali pada tahun 1995 (Kadir, 2020).

Versi awal Java menyertakan paket standar yang terus dikembangkan pada versi selanjutnya. Paket-paket tersebut antara lain (EMS, 2015):

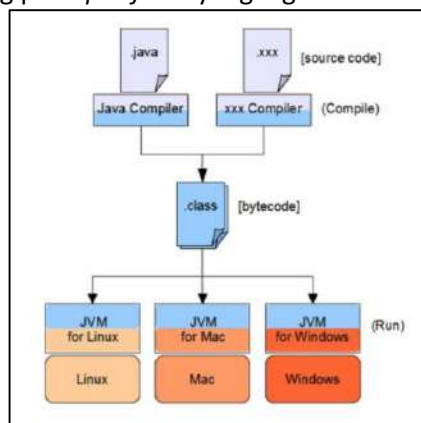
- ✓ `java.lang`: paket yang berisi kelas elemen dasar.
- ✓ `java.io`: paket untuk kelas *input*, *output*, dan *file*.
- ✓ `java.util`: paket untuk kelas struktur data dan penanggalan.
- ✓ `java.net`: paket untuk kelas yang memungkinkan berkomunikasi dengan komputer lain menggunakan TCP/IP.
- ✓ `java.awt`: paket untuk kelas dalam membuat aplikasi berbasis GUI sebagai antarmuka dengan pengguna.

- ✓ `java.applet`: paket untuk kelas berisi aplikasi antarmuka yang dapat diterapkan pada *web browser*.

Java memiliki slogan berupa “*Write Once, Run Everywhere*” yang berarti program yang ditulis menggunakan bahasa Java dapat dijalankan di berbagai perangkat. Syarat utama agar program Java dapat dijalankan di sebuah perangkat adalah dengan ter-*install*-nya sebuah mesin *interpreter* yang disebut dengan *Java Virtual Machine (JVM)*. JVM membaca *bytecode* dari sebuah program dalam bentuk *class file* sebagai representasi dari program tersebut dalam bahasa mesin. Sehingga Java disebut sebagai bahasa pemrograman yang *portable* karena dapat dijalankan pada berbagai sistem operasi selama perangkat yang menjalankannya memiliki JVM (Setiawan, Witama, dan Hikmah, 2020).

a. JRE

JRE (*Java Runtime Environment*) adalah sebuah lingkungan dimana program Java dapat dijalankan. Salah satu komponen utama yang terdapat pada JRE adalah JVM (Hakim dan Sutarto, 2009). Pada JRE terdapat *libraries* serta *file* yang digunakan JVM saat dijalankan (Kurniawan, Muslim, Raihan, Putra, dan Yusuf, 2020). Meskipun JVM merupakan komponen inti yang memungkinkannya program Java untuk dijalankan di berbagai perangkat, namun JVM satu perangkat dengan perangkat lain bisa berbeda tergantung pada *platform* yang digunakan.



Gambar 3.1 Arsitektur JVM (Yang, 2020)

b. JDK

JDK (*Java Development Kit*) merupakan sebuah paket berisi *libraries* maupun *file* yang memungkinkan komputer pengguna dapat digunakan untuk membuat aplikasi Java. Dengan kata lain, agar dapat membuat atau mengembangkan sebuah aplikasi Java maka komputer yang kita gunakan harus memiliki atau telah terinstall JDK. Terdapat dua jenis JDK yang umum digunakan, yang pertama adalah milik *Oracle* (<https://www.oracle.com/>) yang telah mengakuisisi perusahaan pengembang Java pertama kali yaitu *Sun Microsystems*. Jenis berikutnya adalah versi GNU GPL yang merupakan versi *open source* dari JDK yaitu *OpenJDK* (<https://openjdk.java.net/>).

c. PEMROGRAMAN PADA JAVA

Pemrograman menggunakan bahasa Java terdiri dari tiga tahap, yaitu:

1. Menuliskan program Java.

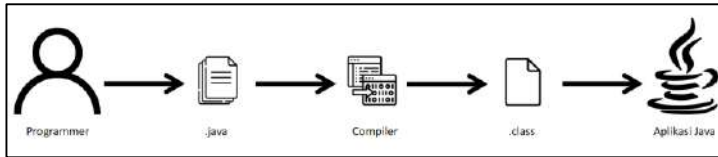
Menuliskan program Java secara sederhana dapat dilakukan dengan menggunakan *text editor* standar seperti *Notepad* dan sejenisnya. Jika menggunakan cara ini maka setelah selesai mengetikkan kode program pada *text editor*, *file* disimpan menggunakan ekstensi *.java* (misal: *HelloWorld.java*).

2. *Compile* (kompilasi) program.

Kompilasi program Java dilakukan dengan menggunakan perintah **javac** nama_file.java (misal: `javac HelloWorld.java`). Hal ini dilakukan untuk mengubah kode program yang telah ditulis menjadi sebuah *binary file* yang dapat dibaca komputer. Kompilasi program dapat gagal dilakukan jika terdapat kesalahan dalam menuliskan kode program maupun sintaks Java yang digunakan sehingga kode program yang ditulis tidak dapat diterjemahkan ke *binary file* dengan baik.

3. *Run* (menjalankan) program.

Menjalankan program Java dapat dilakukan dengan menggunakan perintah **java** nama_file (misal: `java HelloWorld`). Jika kompilasi pada tahapan sebelumnya berhasil dilakukan, maka program akan dijalankan sesuai dengan kode yang telah diterjemahkan saat kompilasi.

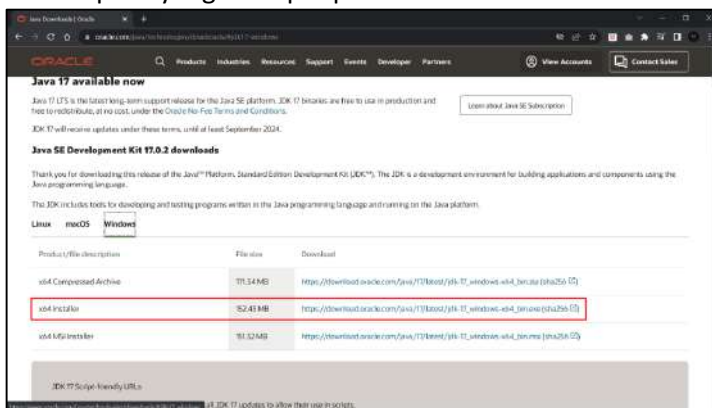


Gambar 3.2 Tahapan Pemrograman pada Java

Tahapan keseluruhan dalam membuat aplikasi Java diperlihatkan pada Gambar 3.2. Saat ini terdapat berbagai *editor* berbasis GUI yang dapat digunakan dalam membuat aplikasi Java seperti NetBeans, Eclipse, IntelliJ IDEA, dan masih banyak lagi.

d. INSTALASI JAVA

Pada buku ini, pemrograman aplikasi Java yang akan dilakukan adalah dengan menggunakan editor Java berupa NetBeans. Sehingga cara instalasi Java yang akan diberikan pertama-tama adalah dengan meng-*install* paket jdk agar komputer dapat digunakan untuk melakukan pemrograman Java. Paket jdk yang akan digunakan adalah paket yang terdapat pada laman *download* situs resmi *oracle* untuk jdk yaitu di <https://www.oracle.com/java/technologies/downloads/>. Versi terkini untuk paket jdk adalah **Java SE Development Kit 17.0.2** yang merupakan rilis LTS (*Long Term Support*) terbaru dari Java SE yang dikeluarkan oleh *Oracle*. Pada versi Java 17 atau JDK 17 ini penulis memilih versi Windows x64 *Installer* seperti yang terdapat pada Gambar 3.3.



Gambar 3.3 Laman *Download* JDK di Situs *Oracle*

Setelah *installer file* terunduh, klik dua *file* tersebut dan akan muncul jendela aplikasi seperti yang diperlihatkan pada Gambar 3.4.



Gambar 3.4 Jendela yang Muncul Saat *Installer* JDK diklik

Ketika diklik **Next >**, maka tampilan selanjutnya adalah jendela dimana kita dapat memilih lokasi tempat JDK akan di-*install* nantinya. Secara *default*, lokasi peng-*install*-an JDK adalah di **C:\Program Files\Java\jdk-17.0.2**. Anda dapat memilih lokasi tujuan lainnya dengan menekan tombol **Change** dan memilih lokasi instalasi yang anda kehendaki, tapi disini penulis memilih lokasi *default* tersebut dan menekan tombol **Next >**.



Gambar 3.5 Laman Pilihan *Destination Folder* Lokasi Instalasi JDK

Setelah menekan tombol *Next >*, proses instalasi akan berjalan seperti diperlihatkan pada Gambar 3.6, dan ketika proses instalasi selesai maka laman aplikasi akan terlihat seperti pada Gambar 3.7.



Gambar 3.6 Proses Instalasi JDK

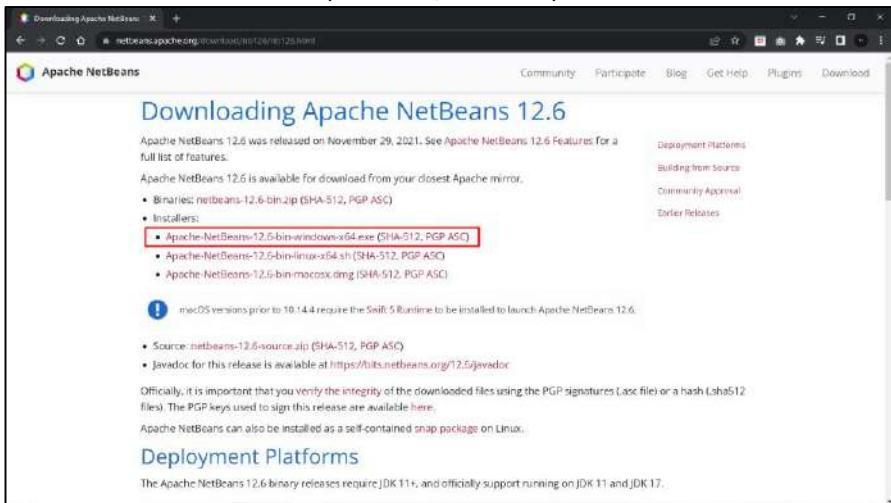


Gambar 3.7 Proses Instalasi JDK Telah Berhasil

Klik tombol **Close** untuk menutup jendela aplikasi instalasi JDK.

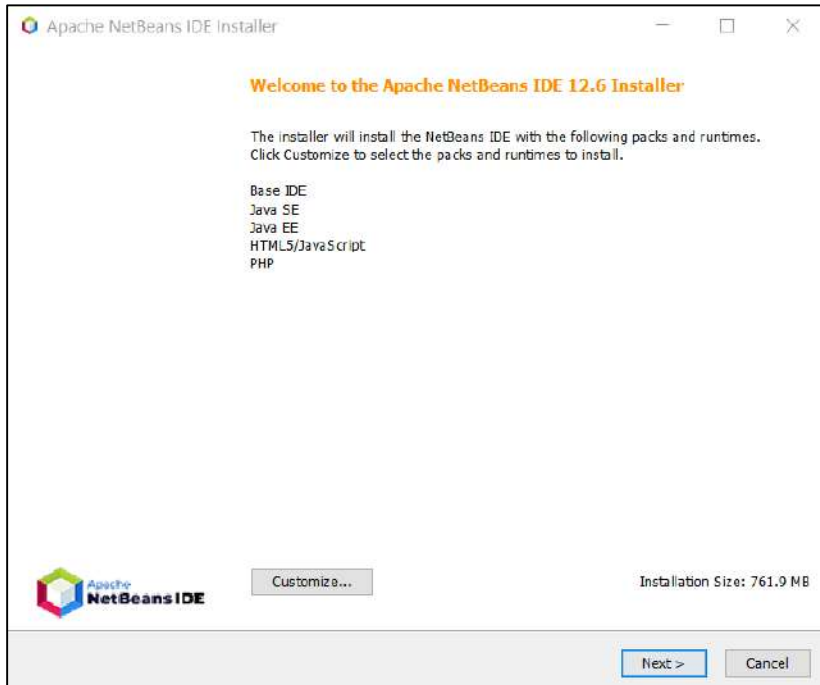
Setelah berhasil meng-*install* JDK, berikutnya kita akan meng-*install* editor sebagai *workspace* yang akan kita gunakan untuk melakukan pemrograman Java. *Editor* yang akan digunakan pada buku ini adalah NetBeans dengan pertimbangan instalasi paket Java *editor* pada NetBeans secara standar sudah menyediakan fungsi pemrograman antarmuka berbasis GUI pada Java tanpa menambahkan paket atau *libraries* lainnya.

NetBeans sendiri dapat diperoleh pada situs resmi *download*-nya yaitu <https://netbeans.apache.org/download/>. Saat buku ini ditulis, versi NetBeans IDE stabil terbaru adalah Apache NetBeans 12.6 yang dirilis pada 29 November 2021. Pada versi ini, terdapat 3 jenis *installer* yang dapat dipilih yaitu versi Windows, Linux, atau MacOSX. Cara instalasi Java pada buku ini menggunakan *installer* versi Windows yaitu Apache-NetBeans-12.6-bin-windows-x64.exe (SHA-512, PGP ASC).



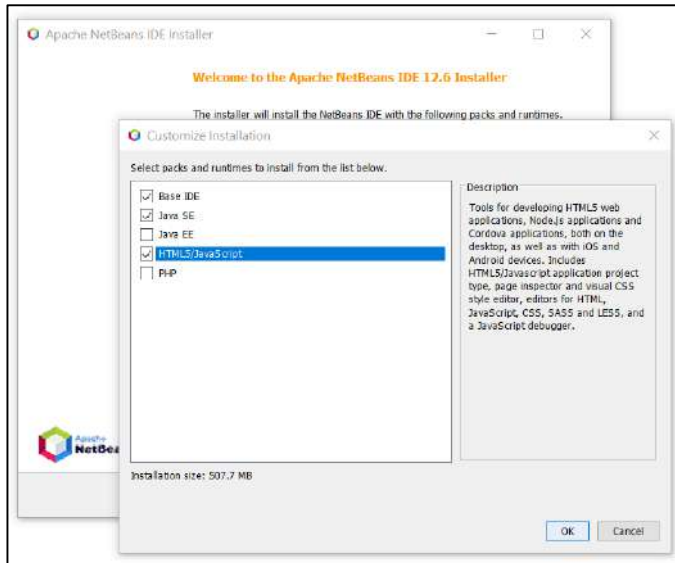
Gambar 3.8 Laman *Download* di Situs Resmi NetBeans

Ketika *installer file* diklik sebanyak 2 kali, maka akan muncul jendela aplikasi seperti yang diperlihatkan pada Gambar 3.9.

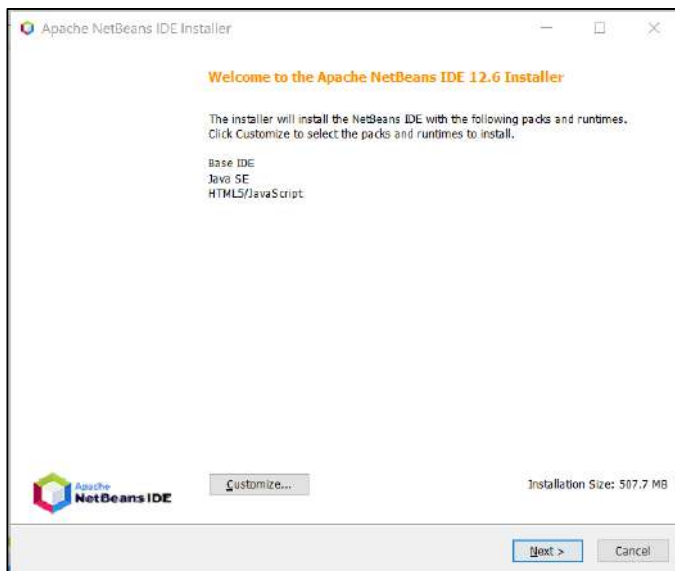


Gambar 3.9 Jendela Aplikasi NetBeans *Installer*

Pada laman tersebut, kita dapat mengubah paket yang akan kita *install* dengan menekan tombol **Customize**. Pada jendela aplikasi **Customize Installation** kita dapat memilih paket untuk *runtime* aplikasi yang kita kehendaki (Gambar 3.10). Untuk pemrograman Java dasar, kita hanya memerlukan paket Base IDE, Java SE, HTML5/JavaScript saja sehingga kita *uncheck* seluruh paket yang ada dan hanya menyisakan 3 paket tersebut untuk di-*install*. Klik **OK**.

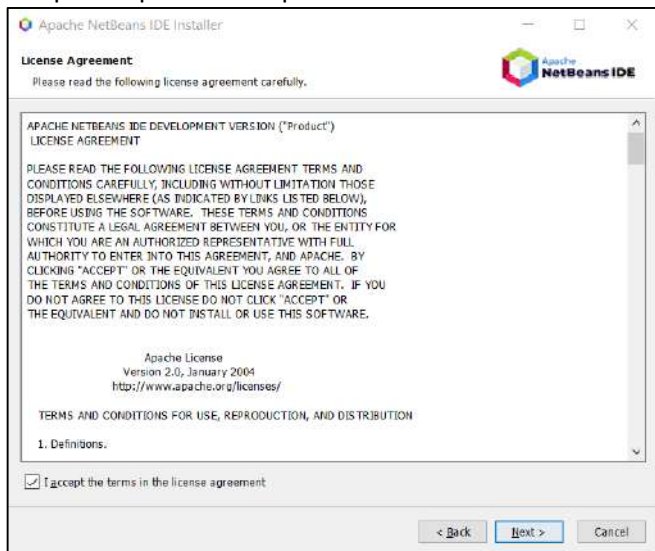


Gambar 3.10 Jendela Aplikasi *Customize Installation*

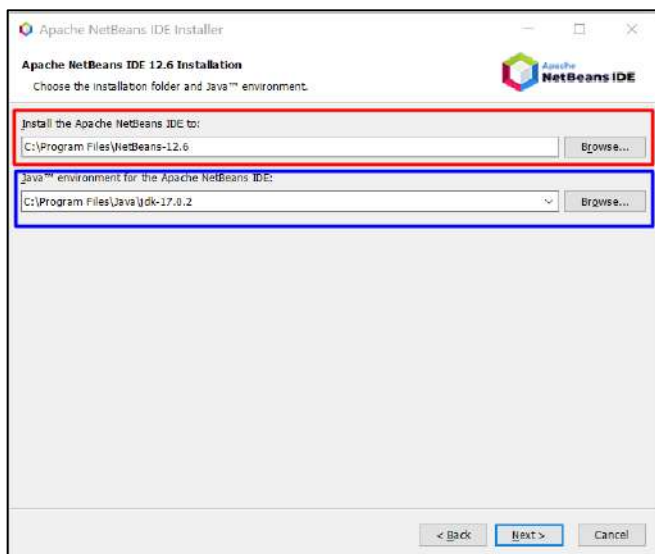


Gambar 3.11 *NetBeans Installer* untuk *install* paket *Base IDE*, *Java SE*, dan *HTML5/JavaScript*

Klik (beri centang) pada *checkbox I accept the terms in the license agreement* seperti diperlihatkan pada Gambar 3.12. Kemudian klik **Next >**.

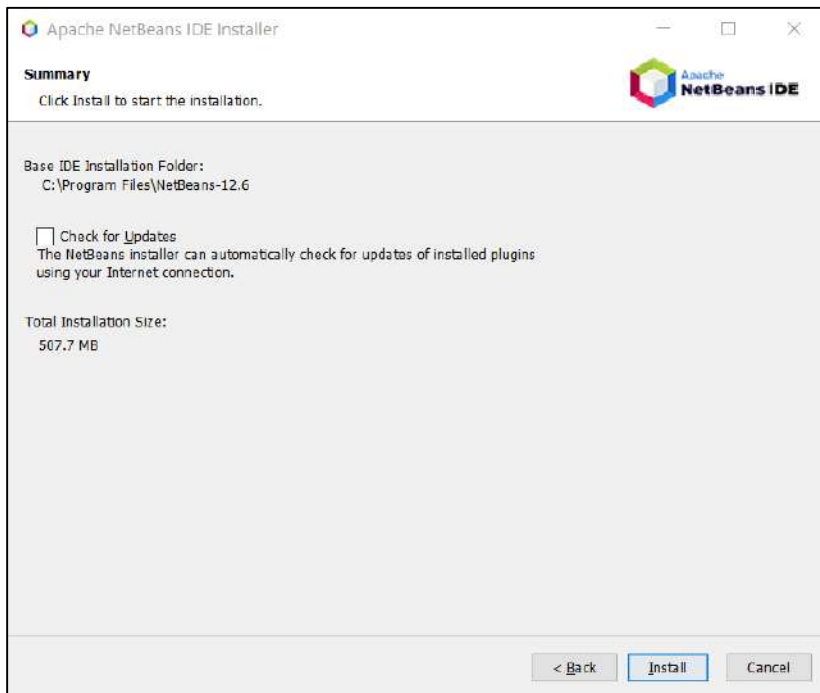


Gambar 3.12 Jendela Aplikasi Persetujuan *License Agreement*



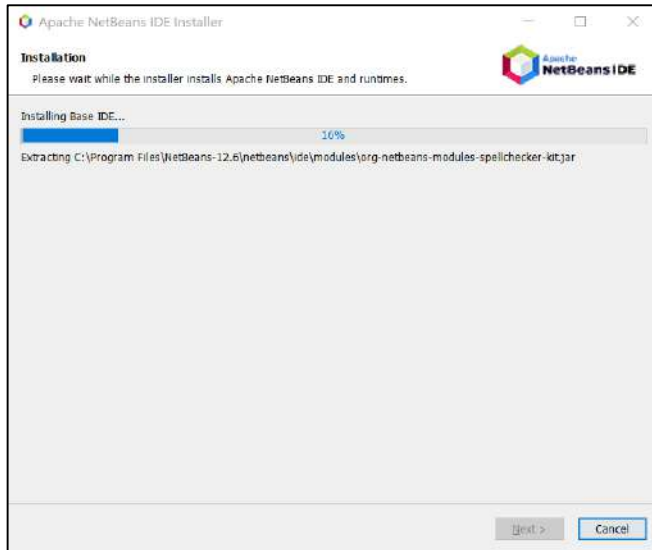
Gambar 3.13 Jendela Aplikasi Pilihan Lokasi Instalasi dan JDK yang digunakan

Pada Gambar 3.13, *field* yang ditandai dengan garis merah merupakan lokasi *default* yang digunakan untuk tempat instalasi NetBeans, dan *field* yang ditandai garis biru merupakan versi JDK yang akan digunakan untuk kompilasi program Java yang dibuat menggunakan NetBeans tersebut. Jika anda menghendaki *folder* instalasi maupun versi JDK yang lain, maka anda dapat menekan tombol **Browse**. Penulis menggunakan *folder* instalasi *default* tersebut dan versi JDK yang telah diinstal sebelumnya (JDK 17). Klik **Next >**.



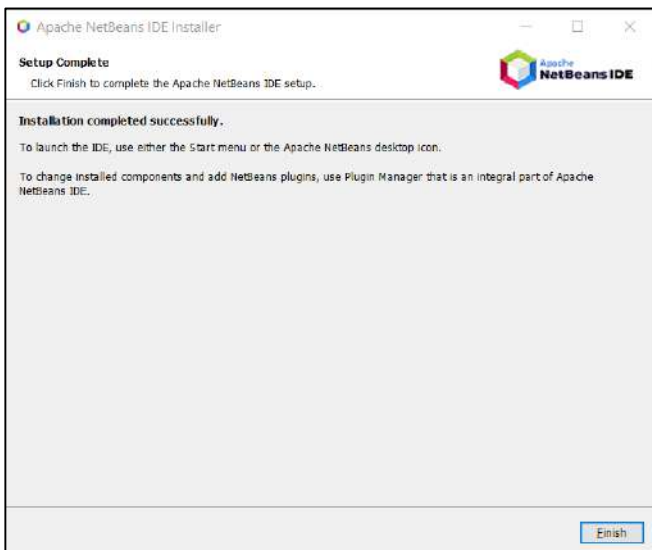
Gambar 3.14 Jendela Aplikasi Pilihan *Updates*

Jendela aplikasi yang muncul berikutnya adalah pilihan untuk memeriksa *update* secara berkala terhadap *plugin* yang telah kita *install* pada NetBeans (Gambar 3.14). Anda dapat melakukan *uncheck* (menghilangkan tanda centang) jika tidak menginginkan pemeriksaan *update* secara otomatis ini. Klik **Install** dan tampil jendela aplikasi berupa progres instalasi seperti yang diperlihatkan pada Gambar 3.15.



Gambar 3.15 Instalasi NetBeans sedang Dilakukan

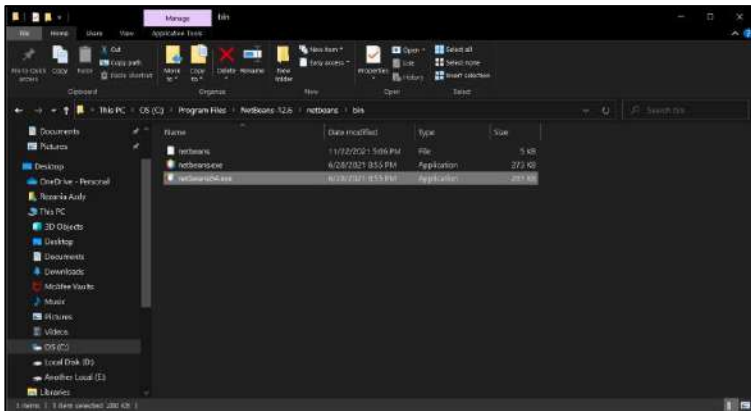
Gambar 3.16 memperlihatkan jendela aplikasi yang menunjukkan bahwa NetBeans telah berhasil di-*install*.




Gambar 3.16 Instalasi NetBeans Berhasil Dilakukan

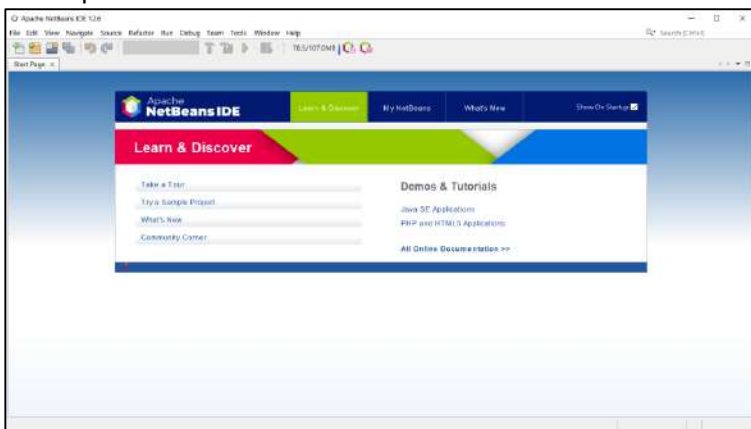
Untuk memastikan bahwa perangkat komputer yang kita miliki dapat digunakan untuk membuat aplikasi Java, maka kita dapat melakukan langkah berikut:

1. Buka aplikasi NetBeans. Hal ini dapat dilakukan dengan mengklik dua kali pada *icon* NetBeans di *Desktop* atau mencari aplikasi NetBeans dari *Start Menu*.



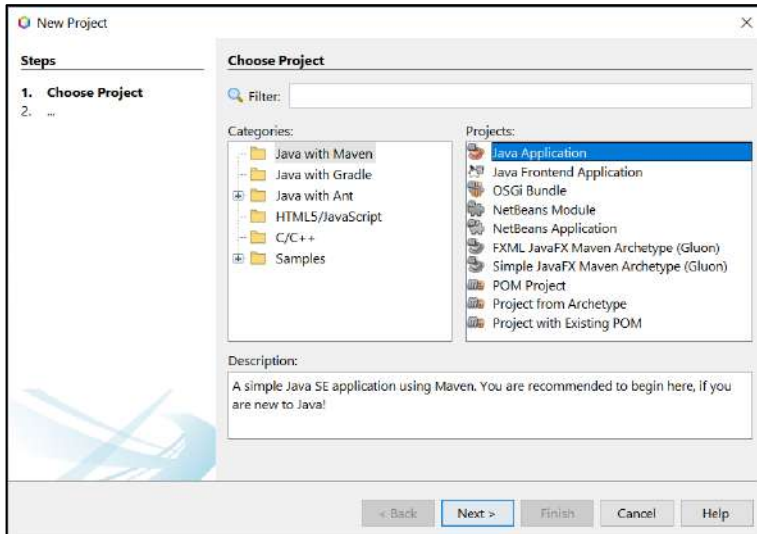
Gambar 3.17 Membuka Aplikasi NetBeans dari *Icon* pada *Installation Folder*

2. Setelah muncul jendela aplikasi NetBeans seperti pada Gambar 3.x, buat *Project* baru dengan mengklik pada menu **File** → **New Project** atau klik pada *icon* .



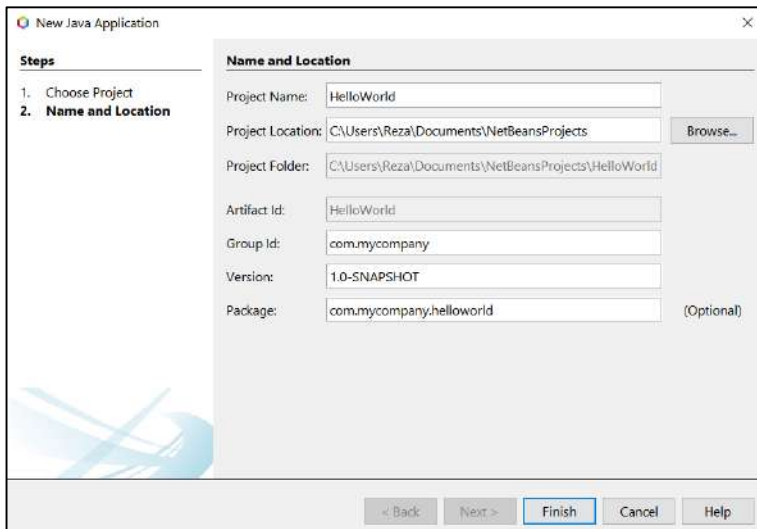
Gambar 3.18 Tampilan Awal NetBeans

3. Pilih **Java with Maven** → **Java Application** → **Next**.



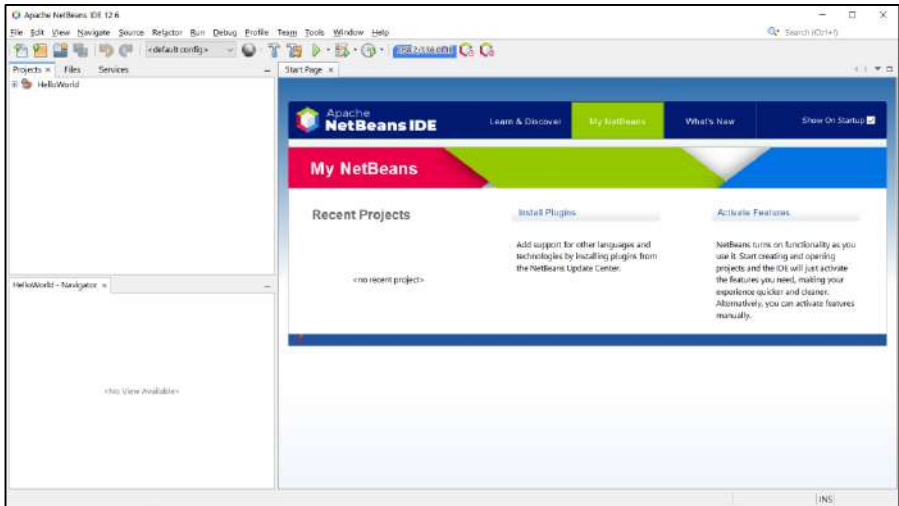
Gambar 3.19 Membuat *Project* Baru pada NetBeans

4. Masukkan nama *project*, dalam hal ini isi nama *project* dengan **HelloWorld**. Klik **Finish**.



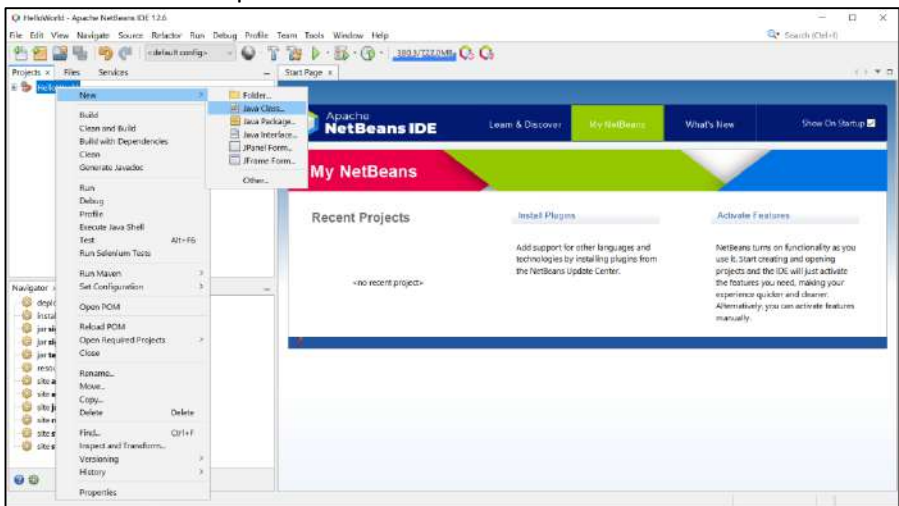
Gambar 3.20 Memberi Nama *Project* Baru

5. Ketika *project* HelloWorld selesai dibuat, maka akan muncul jendela aplikasi seperti berikut.



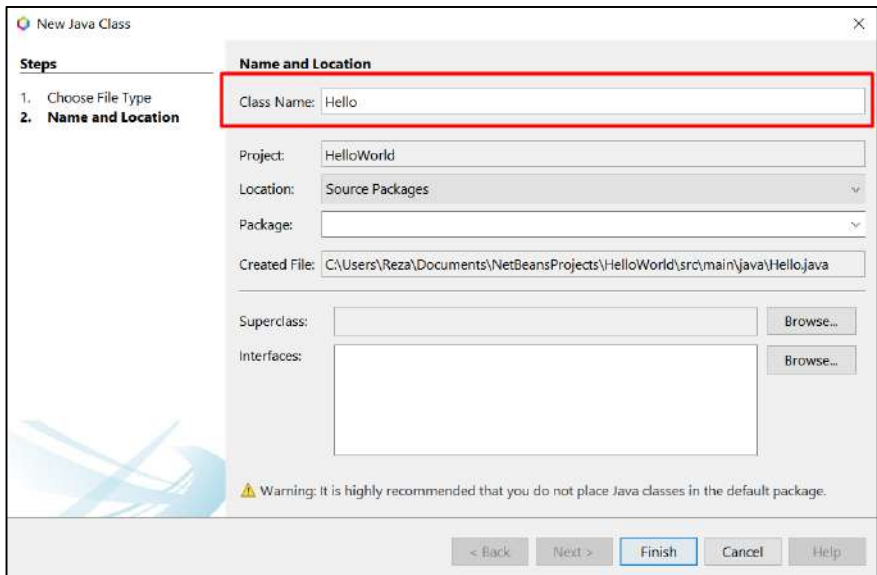
Gambar 3.21 Tampilan NetBeans Setelah Membuat *Project* Baru

6. Berikutnya buat kelas Java baru dengan cara mengklik kanan *project* HelloWorld dan pilih **New** → **Java Class**.



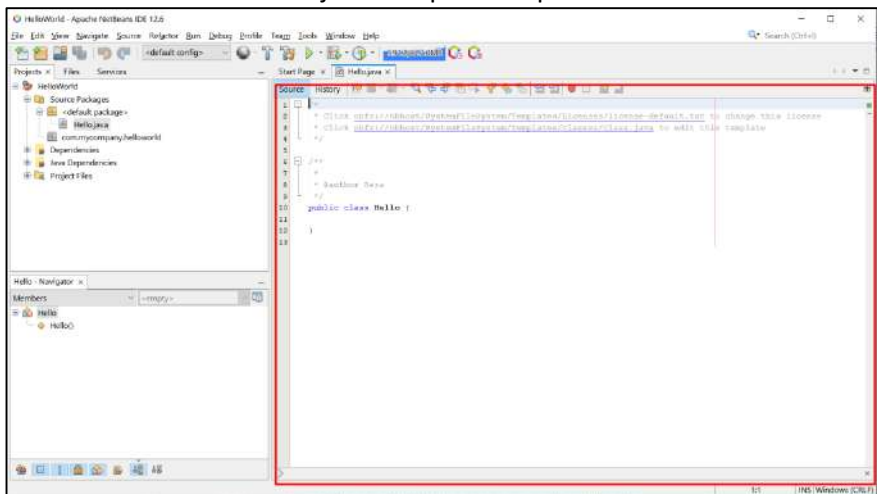
Gambar 3.22 Membuat Kelas (Java File) Baru pada *Project*

7. Ketika muncul jendela aplikasi berikut, isi nama kelas dengan Hello. Klik **Finish**.



Gambar 3.23 Memberi Nama Kelas

8. Kemudian akan muncul jendela aplikasi seperti berikut.



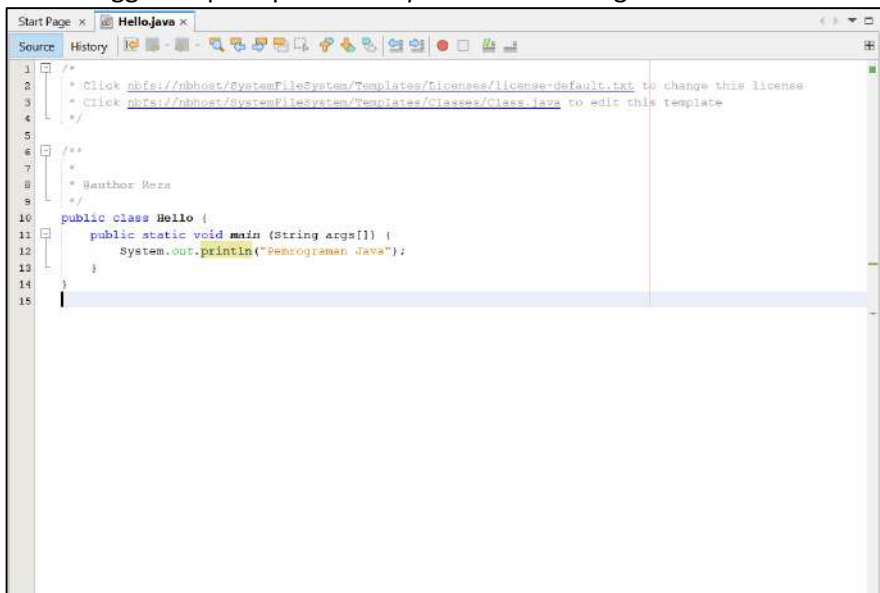
Gambar 3.24 Workspace menampilkan Editor untuk Kelas Hello.java

Field yang ditandai dengan garis merah merupakan sebuah *workspace* dimana kita dapat menuliskan kode program menggunakan bahasa Java.

Ubah kode program pada *workspace* tersebut dengan:

```
public class Hello {  
    public static void main (String args[]) {  
        System.out.println("Pemrograman Java");  
    }  
}
```

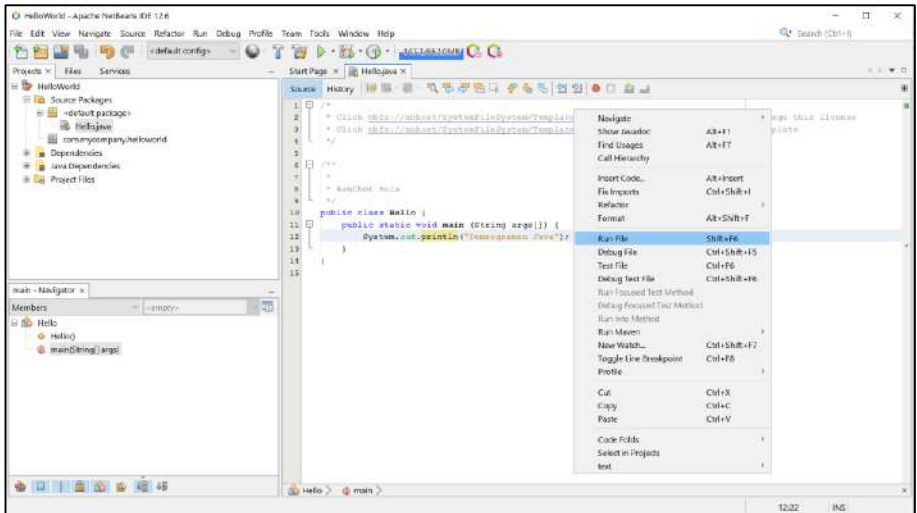
Sehingga tampilan pada *workspace* adalah sebagai berikut:



Gambar 3.25 Menambahkan Kode Program pada Kelas Hello.java

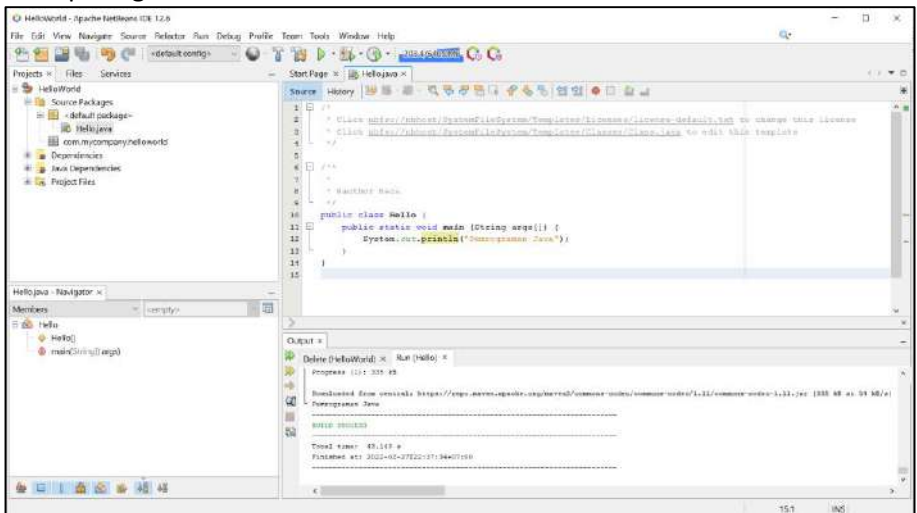
Simpan kode program yang telah dibuat dengan mengklik **File** → **Save** atau tekan **CTRL+Shift**.

9. Untuk menjalankan/melihat *output* dari aplikasi yang telah dibuat tersebut, klik kanan pada sembarang lokasi di *workspace*, dan pilih **Run File**.



Gambar 3.26 Menjalankan Baris Program pada Kelas Hello

10. Jika kode program yang dituliskan benar dan tidak ada kesalahan dalam proses kompilasi, maka NetBeans akan menampilkan jendela **Output** yang secara *default* terletak pada bagian bawah *workspace* seperti gambar berikut.



Gambar 3.27 Tampilan Program dari Kelas hello

Jika telah muncul jendela *Output* seperti pada Langkah 10 diatas, maka instalasi JDK dan NetBeans telah berhasil dilakukan dan dapat digunakan untuk melakukan pemrograman menggunakan Java.

B. STRUKTUR PROGRAM

Pada subbahasan sebelumnya kita telah melihat contoh program sederhana yang ditulis menggunakan bahasa Java. Pada dasarnya, dalam menuliskan program menggunakan Java terdapat sintaks dasar dan struktur program yang harus diikuti. Penulisan struktur program pada Java secara garis besar terdiri dari:

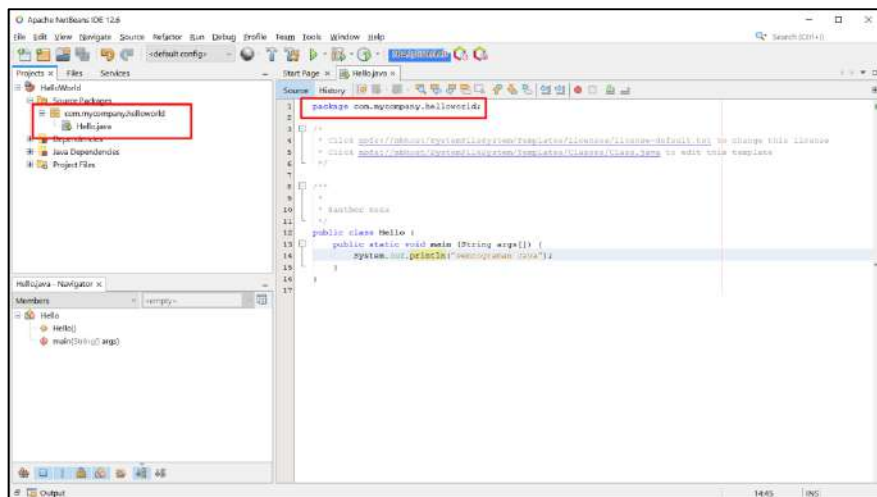
1. *Package*

Package dapat diartikan sebagai sebuah *folder* yang berisi kumpulan *file-file* program Java. Deklarasi *package* dilakukan pada baris paling atas kode program yang ditulis dengan menggunakan kata kunci *package*.

Sebagai contoh, jika kelas Hello yang telah kita buat sebelumnya kita pindahkan ke folder `com.mycompany.helloworld` maka deklarasi *package* yang dituliskan adalah:

```
package com.mycompany.helloworld;
```

Sehingga struktur *project* dan *workspace* terlihat seperti pada Gambar 3.28.



Gambar 3.28 Deklarasi *package* pada Program Java

2. *Import*

Bagian ini digunakan untuk mendeklarasikan *libraries* yang akan digunakan dalam penulisan program Java. Cara mendeklarasikan *libraries* yang akan digunakan dapat dilakukan dengan menggunakan kata kunci *import*. Contoh deklarasi *import* adalah:

```
import java.util.Scanner;
```

Sintaks diatas menyatakan bahwa kita akan meng-*import* kelas *Scanner* yang terdapat pada paket *java.util*.

3. *Class*

Java merupakan bahasa pemrograman yang berorientasi objek atau disebut juga dengan istilah *Object-Oriented Programming* (OOP). OOP merupakan suatu konsep pemrograman dengan cara memodelkan objek-objek dunia nyata (benda, sifat, sistem, dll.) ke dalam sebuah bahasa pemrograman yang diaplikasikan ke dalam kelas, *method*, dan properti/variabel (Komputer, 2010).

Kelas merupakan struktur program utama dan memiliki hierarki tertinggi pada pemrograman Java. Didalam kelas (*class body*) didefinisikan *method*, variabel, maupun *inner class*. Deklarasi kelas secara umum memiliki bentuk :

```
class nama_kelas {  
    ... // class body  
}
```

Contoh deklarasi kelas:

```
class mahasiswa {  
  
}
```

Contoh deklarasi kelas yang memiliki variabel:

```
class mahasiswa {  
    String NPM;  
    String nama;  
    int nilai;  
}
```

4. **Method**

Method merupakan subprogram atau kumpulan kode program yang menjelaskan perilaku dari objek yang akan di-*instance* dari sebuah kelas. *Method* dapat menerima *input* sebagai parameter untuk menjalankan *method* tersebut. *Method* dapat dikategorikan menjadi 4 jenis, yaitu:

a. **Main method**

Main method merupakan *method* utama yang langsung dijalankan ketika kelas dieksekusi. Dengan kata lain, jika program Java yang dibuat tidak memiliki *main method*, maka program tersebut tidak dapat dijalankan.

Contoh kelas Java yang memiliki *main method*:

```
public class Mahasiswa {  
    public static void main (String args[]) { ← Deklarasi main method  
        String nama = "Ani";  
        System.out.println("Nama mahasiswa = "+nama);  
    }  
}
```

b. **Konstruktor**

Konstruktor merupakan *method* yang akan dijalankan ketika sebuah objek di-*instance* dari suatu kelas. Konstruktor umumnya digunakan untuk memberikan nilai awal untuk objek yang dibuat dari sebuah kelas. Syarat utama dari konstruktor adalah *method* memiliki nama yang sama dengan nama kelasnya.

Contoh kelas Java yang memiliki konstruktor:

```
public class Mahasiswa {  
    Mahasiswa () {  
        String nama = "Ani";  
        System.out.println("Nama mahasiswa = "+nama);  
    }  
  
    public static void main (String args[]) {  
        Mahasiswa mh = new Mahasiswa();  
    }  
}
```

Deklarasi konstruktor pada contoh diatas terdapat pada baris 2 dengan nama konstruktor sama dengan nama kelasnya, yaitu Mahasiswa. Pada baris 8, dibuat objek mh dari kelas Mahasiswa, hal ini membuat seluruh kode program yang terdapat di dalam konstruktor (baris 2-4) dijalankan secara otomatis.

c. Prosedur

Prosedur dapat digunakan untuk mengelompokkan kode program sehingga program yang besar dapat dipecah menjadi subprogram atau *modul* yang dapat dijalankan secara berulang sesuai dengan peruntukannya.

Deklarasi *method* sebagai prosedur dilakukan dengan memberikan kata kunci berupa **void** sebelum nama *method*.

Contoh deklarasi *method* sebagai prosedur:

```
public class PersegiPanjang {
    static void hitungLuas() {
        int p = 10;
        int l = 4;
        int L = p*l;
        System.out.println("Luas persegi panjang =
"+L);
    }
    public static void main (String args[]) {
        hitungLuas();
    }
}
```

Pada baris 2, dideklarasikan *method* sebagai sebuah prosedur dengan nama **hitungLuas()**. Ketika *method* ini dipanggil pada *main method*, maka kode program pada baris 3-6 secara otomatis dijalankan.

d. Fungsi

Fungsi memiliki kegunaan yang sama dengan prosedur. Perbedaannya dengan prosedur terletak pada adanya nilai yang harus dikembalikan dari *method* yang berperan sebagai fungsi. Nilai kembalian dari fungsi ini diberikan dengan menggunakan kata kunci **return**.

```

public class PersegiPanjang {
    static int hitungLuas() {
        int p = 10;
        int l = 4;
        int L = p*l;
        return L;
    }

    public static void main (String args[]) {
        System.out.println("Luas persegi panjang =
"+hitungLuas());
    }
}

```

Method sebagai fungsi memiliki cara deklarasi yang berbeda dengan prosedur. Pada baris 2, dapat dilihat terdapat kata kunci `int` yang merupakan tipe data untuk bilangan bulat. Kata kunci ini dimaksudkan untuk memberikan tipe data bagi nilai kembalian yang dihasilkan oleh fungsi, yaitu berupa bilangan bulat. Nilai kembalian dari fungsi `hitungLuas()` adalah nilai `L` berdasarkan pada kode program baris 6. Sehingga ketika *method* `hitungLuas()` dipanggil pada baris 9, maka nilai yang ditampilkan adalah nilai dari `L` yang dihitung pada baris 5.

C. VARIABEL

Variabel merupakan sebuah *wadah* yang dapat digunakan untuk menampung nilai. Prinsip utama dari variabel adalah nilainya berubah sesuai dengan nilai yang diberikan/disimpan terakhir di variabel tersebut.

Pada Java, penamaan variabel memiliki ketentuan:

1. Nama variabel dapat terdiri dari (kombinasi) huruf, angka, maupun *underscore*.
2. Nama variabel tidak dapat diawali dengan angka.
3. Nama variabel sebaiknya ditulis menggunakan gaya *camelCase*, sebagai contoh: `panjangPersegi`, `namaMahasiswa`, `nipPegawai`.
4. Nama variabel tidak boleh sama dengan kata kunci yang dimiliki oleh Java.

***) Tentang Kata Kunci**

Kata kunci (*keywords*) pada Java, sering juga disebut dengan *reserved words*, merupakan kata-kata yang digunakan Java dan telah memiliki arti tertentu sehingga tidak dapat digunakan sebagai penamaan variabel, kelas, *method*, atau objek. Beberapa *keywords* pada Java diperlihatkan pada Tabel 3.1.

Tabel 3.1 Keywords pada Java

<i>abstract</i>	<i>char</i>	<i>enum</i>	<i>implements</i>	<i>new</i>	<i>short</i>	<i>throw</i>
<i>assert</i>	<i>class</i>	<i>extends</i>	<i>import</i>	<i>null</i>	<i>static</i>	<i>throws</i>
<i>boolean</i>	<i>continue</i>	<i>final</i>	<i>instanceof</i>	<i>package</i>	<i>strictfp</i>	<i>transient</i>
<i>break</i>	<i>default</i>	<i>finally</i>	<i>int</i>	<i>private</i>	<i>super</i>	<i>try</i>
<i>byte</i>	<i>do</i>	<i>float</i>	<i>interface</i>	<i>protected</i>	<i>switch</i>	<i>void</i>
<i>case</i>	<i>double</i>	<i>for</i>	<i>long</i>	<i>public</i>	<i>synchronized</i>	<i>volatile</i>
<i>catch</i>	<i>else</i>	<i>if</i>	<i>native</i>	<i>return</i>	<i>this</i>	<i>while</i>

Contoh penamaan variabel yang benar adalah:

1. nama_lengkap
2. _nilai
3. nilaiMahasiswa1

Contoh penamaan variabel yang salah adalah:

1. 9luas, karena diawali dengan angka.
2. *void*, karena merupakan kata kunci pada Java.

D. TIPE DATA

Pada Java, dalam mendeklarasikan variabel juga diharuskan untuk mendeklarasikan tipe datanya. Dengan adanya deklarasi tipe data pada variabel ini membatasi jenis nilai yang dapat disimpan di dalam variabel.

Java memiliki 2 kategori tipe data, yaitu tipe data primitif dan tipe data referensi (Yuniansyah, 2020). Tipe data primitif adalah tipe data yang tidak diturunkan dari tipe data lain. Sedangkan tipe data referensi adalah tipe data yang digunakan dalam OOP dan mereferensikan objek atau kelas tertentu.

Yang termasuk dalam tipe data primitif adalah:

1. Tipe data untuk bilangan bulat (*integer*). Terdapat 4 tipe data pada Java untuk bilangan bulat, perbedaan tipe data bilangan bulat ini terletak pada jangkauan nilai yang dapat disimpan pada setiap tipe data.
2. Tipe data untuk bilangan pecahan/desimal. Terdapat 2 jenis tipe data untuk bilangan pecahan (*floating point*).
3. Tipe data untuk karakter. Tipe data **char** digunakan untuk menyatakan sebuah karakter *Unicode*, yaitu seluruh karakter yang terdapat pada semua bahasa, seperti bahasa Latin, Arab, Yunani, dan lainnya. Selain karakter *Unicode*, Java juga menyediakan *escape sequence* yang merupakan gabungan dari beberapa karakter yang telah dikenali Java. Contoh *escape sequence* dan fungsinya diperlihatkan pada Tabel 3.2.

Tabel 3.2 Escape Sequence pada Java

<i>Escape Sequence</i>	Keterangan
<code>\t</code>	Menambahkan <i>tab</i> pada teks
<code>\b</code>	Menambahkan <i>backspace</i> pada teks
<code>\n</code>	Menambahkan baris baru pada teks
<code>\r</code>	Menambahkan <i>carriage return</i> pada teks
<code>\f</code>	Menambahkan <i>form feed</i> pada teks
<code>\'</code>	Menambahkan karakter <i>single quote</i> (') pada teks
<code>\"</code>	Menambahkan karakter <i>double quote</i> (") pada teks
<code>\\</code>	Menambahkan karakter <i>backslash</i> (\) pada teks

4. Tipe data untuk *Boolean*. Tipe data ini ditandai dengan kata kunci **Boolean**, digunakan untuk menampung nilai logika. Tipe data ini hanya bernilai benar yang direpresentasikan dengan **true** dan bernilai salah yang direpresentasikan dengan **false**.

Adapun yang termasuk dalam tipe data referensi adalah:

1. *String*, merupakan tipe data yang digunakan untuk data yang terdiri dari deretan karakter.
2. *Class*, merupakan sebuah *blueprint* yang mendefinisikan variabel dan *method* dari sebuah objek.
3. *Array*, merupakan tipe data yang memungkinkan sebuah variabel dapat menampung banyak nilai sekaligus.

4. *Interface*, merupakan sekumpulan *method* yang hanya terdiri dari deklarasi dan struktur *method*. Detail dari *method* berada pada kelas yang mengimplementasikan *interface* tersebut.

Tabel 3.3 memperlihatkan tipe data yang umum digunakan pada Java beserta keterangan dan jangkauan masing-masing tipe datanya.

Tabel 3.3 Tipe Data pada Java

Tipe Data	Jenis Data	Nilai Default	Ukuran Default
<i>byte</i>	<i>Integer</i>	0	1 <i>byte</i>
<i>short</i>	<i>Integer</i>	0	2 <i>byte</i>
<i>int</i>	<i>Integer</i>	0	4 <i>byte</i>
<i>long</i>	<i>Integer</i>	0L	8 <i>byte</i>
<i>float</i>	<i>Floating point</i>	0.0f	4 <i>byte</i>
<i>double</i>	<i>Floating point</i>	0.0d	8 <i>byte</i>
<i>char</i>	Karakter	'\u0000'	2 <i>byte</i>
<i>boolean</i>	<i>Boolean</i>	<i>False</i>	1 bit

E. OPERATOR

Pada bahasa Java, terdapat 4 jenis operator yang umum digunakan, yaitu:

1. Operator Aritmatika, merupakan operator yang digunakan untuk operasi aritmatika.
2. Operator Penugasan, merupakan operator yang berfungsi memberikan penugasan pada variabel tertentu. Operator ini menggunakan simbol sama dengan (=) untuk mengisi nilai dalam sebuah variabel.
3. Operator Pembandingan, merupakan operator yang digunakan untuk membandingkan dua buah nilai (elemen). Nilai yang dihasilkan dari penggunaan operator ini adalah nilai bertipe *boolean* yaitu *true* atau *false*.
4. Operator Logika, merupakan operator yang digunakan untuk menguji satu atau lebih *operand* yang bertipe *boolean*. Hasil dari operasi menggunakan operator ini adalah nilai bertipe *boolean*, yaitu *true* atau *false*.

Operator-operator yang terdapat pada Java diperlihatkan pada Tabel 3.4.

Tabel 3.4 Operator pada Java

Nama operator (simbol)	Keterangan	Contoh
Operator Aritmatika		
Penjumlahan (+)	Menjumlahkan 2 buah nilai	$a + b$
Pengurangan (-)	Mengurangkan 2 buah nilai	$a - b$
Perkalian (*)	Mengalikan 2 buah nilai	$a * b$
Pembagian (/)	Membagi 2 buah nilai	a / b
Modulus (%)	Memberikan sisa hasil bagi	$a \% b$
<i>Increment</i> (++)	Menaikkan nilai dari variabel sebanyak 1	$++a$
<i>Decrement</i> (--)	Menurunkan nilai dari variabel sebanyak 1	$--a$
Operator Penugasan		
=	Memberikan nilai	$a = 5$
+=	Memberikan nilai dengan menjumlahkannya terlebih dahulu dengan nilai sebelumnya	$a += 5$ Memberi hasil yang sama dengan $a = a + 5$
-=	Memberikan nilai dengan mengurangkannya terlebih dahulu dengan nilai sebelumnya	$a -= 5$ Memberi hasil yang sama dengan $a = a - 5$
*=	Memberikan nilai dengan mengalikannya terlebih dahulu dengan nilai sebelumnya	$a *= 5$ Memberi hasil yang sama dengan $a = a * 5$
/=	Memberikan nilai dengan membaginya terlebih dahulu dengan nilai sebelumnya	$a /= 5$ Memberi hasil yang sama dengan $a = a / 5$
%=	Memberikan nilai dengan mencari sisa hasil baginya terlebih dahulu dengan nilai sebelumnya	$a \% = 5$ Memberi hasil yang sama dengan $a = a \% 5$

Operator Pembanding		
==	Memberikan hasil <i>true</i> jika nilai a sama dengan b	a == b
!=	Memberikan hasil <i>true</i> jika nilai a tidak sama dengan b	a != b
>	Memberikan hasil <i>true</i> jika nilai a lebih dari nilai b	a > b
<	Memberikan hasil <i>true</i> jika nilai a kurang dari nilai b	a < b
>=	Memberikan hasil <i>true</i> jika nilai a lebih dari atau sama dengan nilai b	a >= b
<=	Memberikan hasil <i>true</i> jika nilai a kurang dari atau sama dengan nilai b	a <= b
Operator Logika		
&&	Logika AND (Memberikan nilai <i>true</i> jika pernyataan a dan pernyataan b benar)	a < 3 && b > 8
	Logika OR (Memberikan nilai <i>true</i> jika pernyataan a atau pernyataan b benar)	a = 0 b > 9
!	Logika NOT (Membalikkan nilai pernyataan, membalikkan nilai <i>true</i> jika pernyataan bernilai <i>false</i>)	!(a >= 5)

F. CONTOH

Gambar 3.29 memperlihatkan contoh program sederhana pada Java meliputi penggunaan variabel, tipe data, dan operator.

```

1 public class ContohProgramJava {
2     public static void main (String args[]) {
3         String garis = "-----";
4         String teks = "          Contoh Variabel, Tipe Data, dan Operator";
5         int a = 10;
6         int b = 3;
7         int mod = a%b;
8         Boolean banding1 = a>b;
9         Boolean banding2 = a>=15 && b<20;
10
11         System.out.println(garis);
12         System.out.println(teks);
13         System.out.println(garis);
14         System.out.println("Silas hasil bagi : "+mod);
15         System.out.println("Hasil perbandingan 1 pernyataan : "+banding1);
16         System.out.println("Hasil perbandingan 2 pernyataan : "+banding2);
17         System.out.println(garis);
18     }
19 }

```

Gambar 3.29 Contoh Program Java Sederhana

G. RANGKUMAN MATERI

Pemrograman Java merupakan salah satu cara yang dapat digunakan untuk membuat aplikasi komputer. Java memiliki slogan “*Write Once, Run Everywhere*” karena aplikasi yang dihasilkan dapat dijalankan di berbagai *platform* asalkan di *device* tersebut telah ter-*install Java Virtual Machine* (JVM). Java merupakan bahasa pemrograman yang memiliki konsep pemrograman dengan paradigma *Object-Oriented Program* (OOP), dimana pembuat aplikasi dalam melakukan pemrograman memodelkan permasalahan yang ada dalam objek-objek yang ada di dunia nyata. Untuk mengembangkan aplikasi Java, diperlukan sebuah perangkat yang telah ter-*install Java Development Kit* (JDK) maupun *tools* lainnya yang dapat memudahkan pembuat aplikasi dengan tampilan berbasis GUI-nya seperti NetBeans, Eclipse, maupun IntelliJIdea. Pada Java terdapat beberapa aturan yang dapat digunakan dalam pembuatan aplikasi seperti struktur program, penamaan variabel, tipe data untuk setiap variabel, konstanta, maupun fungsi, serta operator-operator yang dapat digunakan untuk menghasilkan operasi yang sesuai. Seiring berjalannya waktu, Java mengalami banyak perubahan baik untuk *libraries* maupun API yang dapat digunakan. Sekarang ini Java populer dijadikan sebagai bahasa pemrograman yang digunakan untuk membangun aplikasi *mobile* berbasis Android.

TUGAS DAN EVALUASI

Jawablah pertanyaan ini dengan jelas dan benar!

1. Apa kelebihan dari Pemrograman Java?
2. Sebutkan apa saja yang perlu dilakukan agar perangkat (komputer) yang kita miliki dapat digunakan untuk membuat aplikasi Java!
3. Sebutkan 3 nama variabel yang dapat digunakan pada Java!
4. Jelaskan masing-masing operator yang terdapat pada Java!
5. Sebutkan masing-masing 3 operator pada setiap jenis operator yang ada di Java!

DAFTAR PUSTAKA

- EMS, T. (2015). Pemrograman Java dari Nol. https://www.google.co.id/books/edition/Pemrograman_Java_dari_Nol/i4IKDwAAQBAJ?hl=en&gbpv=1&dq=Pemrograman+Java&printsec=frontcover
- Hakim, R., Sutarto. (2009). Mastering JavaTM. https://www.google.co.id/books/edition/Mastering_Java_+_Cd/7pu6hla9_PcC?hl=en&gbpv=1&dq=jvm+adalah&pg=PA2&printsec=frontcover
- Kadir, A. (2020). Logika Pemrograman Java. https://www.google.co.id/books/edition/Logika_Pemrograman_Java/6mXrDwAAQBAJ?hl=en&gbpv=1&dq=Pemrograman+Java&printsec=frontcover
- Komputer, W. 2010. ShortCourse Pengembangan Aplikasi Database Berbasis JavaDB dengan Netbeans. https://www.google.co.id/books/edition/Shortcourse_Series_Pengembangan_Aplikasi/kpheFL6DfQAC?hl=en&gbpv=1&dq=oop+adalah&pg=PA88&printsec=frontcover
- Kurniawan, D.E., Muslim, I., Raihan, M., Putra, A.P., dan Yusuf, P.A. 2020. MOBILE PROGRAMMING Praktik Membuat Aplikasi Berbasis QR Code dan NFC. https://www.google.co.id/books/edition/MOBILE_PROGRAMMING_Praktik_Membuat_Aplik/VhMBEAAAQBAJ?hl=en&gbpv=1&dq=jvm+adalah&pg=PA5&printsec=frontcover
- Setiawan, F.S., Witama, M.N., Hikmah, R. (2020). Perancangan Sistem Pengolahan Data Produksi Konveksi Berbasis Java Pada CV Nirwana Bunga Abadi. Jurnal Nasional Komputasi dan Teknologi Informasi. 3(3). 202-208. <https://ojs.serambimekkah.ac.id/jnkti/article/view/2435>
- Yang, H. 2020. JVM Tutorials - Herong's Tutorial Examples. https://www.google.co.id/books/edition/JVM_Tutorials_Herong_s_Tutorial_Examples/U9wEEAAAQBAJ?hl=en&gbpv=1&dq=jvm&pg=PA13&printsec=frontcover

Yuniansyah. 2020. Algoritma dan Pemrograman Menggunakan Bahasa Java (Teori dan Aplikasinya).
https://www.google.co.id/books/edition/Algoritma_dan_Pemrograman_Menggunakan_Bahasa_Java/2nMCEAAQBAJ?hl=en&gbpv=1&dq=tipe+data+pada+java&pg=PA46&printsec=frontcover



STRUKTUR PERCABANGAN

Yuniansyah, S.Kom., M.Kom
Institut Teknologi dan Bisnis PalComTech

A. PENDAHULUAN

Struktur percabangan mempunyai suatu kondisi dimana terdapat sebuah kondisi yang akan menentukan sebuah perintah yang akan dijalankan ketika kondisi tersebut bernilai benar, ataupun sebaliknya jika kondisi bernilai salah maka perintah didalamnya tidak dapat berjalan. Ada beberapa macam struktur didalam sebuah percabangan diantaranya yaitu percabangan IF, IF *Else*, dan *Swicth Case*.

Hampir setiap program yang telah dibuat akan menggunakan percabangan tersebut dikarenakan struktur percabangan memiliki peran yang sangat penting dalam mengatur setiap jalannya sebuah program. Percabangan merupakan cara untuk mengatur alur sebuah program dengan memberikan satu kondisi atau lebih. Program tersebut dapat dijalankan sesuai *statement* yang terdapat pada percabangan jika kondisi bernilai benar.

B. KONSEP PERCABANGAN

Percabangan adalah struktur program yang digunakan untuk melakukan proses pengujian terhadap satu, dua atau lebih dari dua kondisi. Pemilihan *statement-statement* atau perintah-perintah yang akan di jalankan didasarkan atas kondisi tertentu. Statemen atau perintah tertentu akan dijalankan apabila memenuhi ketentuan yang telah didefinisikan sebelumnya.

Pada pemrograman java terdapat beberapa struktur percabangan, yaitu:

- *Statement if*
- *Statement if-else*
- *Statement if-else if*
- *Staement Switch*

C. STATEMENT IF

Statement-if menentukan sebuah *statement/kondisi* (atau blok kode) yang akan dieksekusi jika dan hanya jika persyaratan *boolean (boolean statement)* bernilai *true*. Pada *statement if* apabila kondisi tidak terpenuhi atau bernilai *false* maka program akan berhenti dan tidak menampilkan pernyataan

Syntax dari *statement if* adalah sebagai berikut:

```
if ( boolean_expression ) statement;
```

atau

```
if ( boolean_expression )
{
    statement1;
    statement2;
.. }
```

Contoh Program Bilangan Ganjil:

```
/*
```

```
Contoh Program
```

```
Menggunakan Struktur Percabangan if */
```

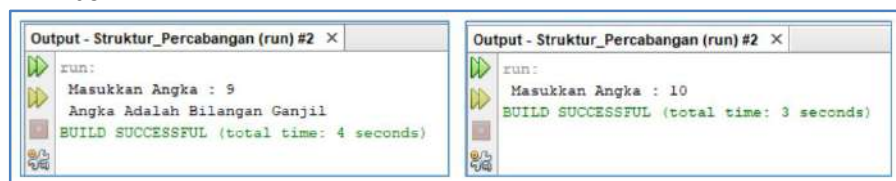
```

package struktur_percabangan;
import java.util.Scanner;
// @author Yuniansyah
public class Program_Ganjil {
    public static void main(String[] args) {
        int Angka;
        Scanner inp = new Scanner(System.in);
        // Input bilangan -----
        System.out.print(" Masukkan Angka : ");
        Angka = inp.nextInt();

        // Cek Kondisi Angka
        if (Angka % 2 > 0)
        {
            System.out.println(" Angka Adalah Bilangan Ganjil
");
        }
    }
}

```

Hasil:



Gambar 4.1 Hasil Eksekusi Program Bilangan Ganjil

Penjelasan:

Pada program diatas terlihat pada tampilan hasil, Program akan meminta pengguna memasukkan *input* angka, berdasarkan angka yang di *input* program melihat kondisi terpenuhi atau tidak, jika kondisi terpenuhi

(*true*) maka Program akan menampilkan “Angka Adalah Bilangan Ganjil”, tetapi apabila kondisi tidak terpenuhi (*false*) sebagai contoh seperti gambar di sisi kanan angka di *input* dengan nilai 10 (bilangan genap), maka program akan selesai tanpa menghasilkan atau tidak menampilkan hasil.

D. STATEMENT IF-ELSE

Statement if-else digunakan apabila kita ingin mengeksekusi sebuah *statement* dengan kondisi *true* dan *statement* yang lain dengan kondisi *false*.

Bentuk *statement if-else*:

```
if (boolean_expression ) {  
    statement;  
}  
else  
{  
    Statement;  
}
```

dapat juga ditulis seperti,

```
if (boolean_expression )  
{  
    statement1;  
    statement2;  
    ...  
}  
else  
{  
    statement1;
```

```

        statement2;

        ...

    }

```

Contoh Program Mencari Suatu Bilangan Ganjil atau Genap

/* Contoh Program

Menggunakan Struktur Percabangan if else */

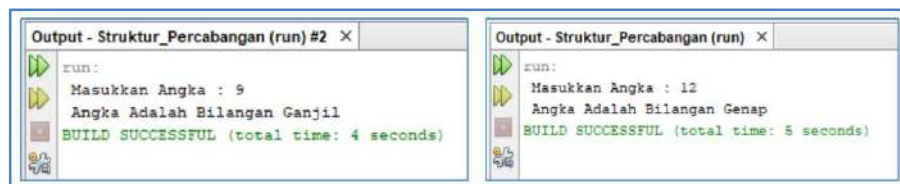
```

package struktur_percabangan;
import java.util.Scanner;
// @author Yuniansyah
public class Program_Genap_Ganjil {
    public static void main(String[] args) {
        int Angka;
        Scanner inp = new Scanner(System.in);
        // Input bilangan -----
        System.out.print(" Masukkan Angka : ");
        Angka = inp.nextInt();

        // Cek Kondisi Angka
        if (Angka % 2 == 0)
        {
            System.out.println(" Angka Adalah Bilangan Genap");
        }
        else
        {
            System.out.println(" Angka Adalah Bilangan Ganjil");
        }
    }
}

```

Hasil:



Gambar 4.2 Hasil Eksekusi Program Ganjil Genap

Penjelasan:

Program di atas adalah pelengkap dari program sebelumnya. Pada struktur *if..else* program akan mengeksekusi program jika kondisi terpenuhi (*true*) atau tidak terpenuhi (*false*). Sama dengan program sebelumnya pertama-tama program akan meminta pengguna memasukkan/*input* angka, berdasarkan angka yang di *input* program melihat kondisi terpenuhi atau tidak, jika kondisi terpenuhi (*true*) maka Program akan menampilkan “Angka Adalah Bilangan Ganjil”, dan apabila kondisi tidak terpenuhi (*false*) sebagai contoh seperti gambar di sisi kanan angka di *input* dengan nilai 12 (bilangan genap), maka program akan menampilkan “Angka Adalah Bilangan Genap”.

E. STATEMENT IF-ELSE-IF

Statement pada bagian *else* dari blok *if-else* dapat menjadi struktur *if-else* yang lain. Struktur seperti ini mengizinkan kita untuk membuat seleksi persyaratan yang lebih kompleks.

Bentuk *statement if-else if*:

```
if (boolean_expression1 )  
  
    {  
        statement;  
        ...  
    }  
else if (boolean_expression2 )  
    {  
        statement;  
        ...  
    }
```

```

    }
    else if (boolean_expression-n )
    {
        statement;
        ...
    }
    else
    {
        statement;
        ...
    }
}

```

Pada *statement if-else if* memiliki banyak kondisi yang akan di uji kebenarannya. *Statement* ini terdiri dari beberapa blok program tergantung kondisi yang ada. Pada *statement* ini *statement else* bersifat *optional*, yang berarti dapat digunakan atau dapat juga dihilangkan berdasarkan kondisi atau kebutuhan program. Pada *statement* ini, jika *boolean_expression1* bernilai *true*, maka program akan mengeksekusi *statement1* dan melewati *statement* yang lain. Jika *boolean_expression2* bernilai *true*, maka program akan mengeksekusi *statement2* dan melewati *statement 3* dan seterusnya. Apabila semua kondisi tidak terpenuhi, maka *statement* pada blok *else* akan dieksekusi.

Contoh Program *if else if (if-else if 1)*:

/*

Contoh Program

Menggunakan Struktur Percabangan *if else if */*

```

package struktur_percabangan;

import java.util.Scanner;

// @author Yuniansyah

public class Program_Positif_Negatif {
    public static void main(String[] args) {
        int Angka;

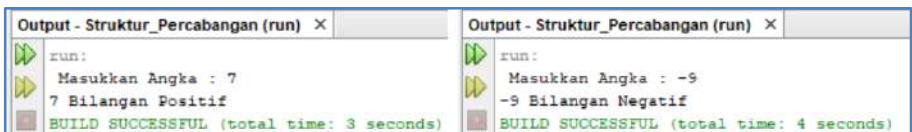
```

```

Scanner inp = new Scanner(System.in);
// Input bilangan -----
System.out.print(" Masukkan Angka : ");
Angka = inp.nextInt();
// Cek Kondisi Angka
if(Angka > 0)
{
    System.out.println(Angka+" Bilangan Positif");
}
else if(Angka < 0)
{
    System.out.println(Angka+" Bilangan Negatif");
}
else
{
    System.out.println(Angka+" Bilangan Nol");
}
}

```

Hasil:



Gambar 4.3 Hasil Eksekusi Program Positif Negatif (if..else if 1)

Penjelasan:

Contoh program *if-else-if* di atas terdiri dari tiga kondisi. Pertama-tama pengguna diminta memasukkan satu Angka, selanjutnya *statement if* akan menguji kondisi pertama yaitu ($\text{Angka} > 0$). Jika kondisi pertama bernilai *true* maka program akan menampilkan keterangan "Angka Adalah Bilangan Positif ", jika tidak program akan menguji kondisi kedua yaitu " $\text{Bilangan} < 0$ ", jika kondisi kedua terpenuhi maka program akan menampilkan keterangan "Angka adalah Bilangan Negatif", jika kondisi kedua bernilai *false* atau tidak terpenuhi, maka program akan menampilkan keterangan "Angka adalah Bilangan 0"

Contoh lain penggunaan struktur percabangan menggunakan *if..then else if* Contoh program *if..else if* adalah program nilai mahasiswa berdasarkan penjumlahan nilai tugas ditambah nilai ujian tengah semester dan ditambah dengan nilai ujian akhir semester, dimana masing-masing nilai mempunyai bobot nilai tertentu. Untuk lebih jelasnya buatlah program berikut ini

Contoh Program Nilai Siswa (*if-else if 2*):

```
/*
 * Contoh Program
 * Menggunakan Struktur Percabangan if else if */
package struktur_percabangan;
import java.util.Scanner;
// @author Yuniansyah
public class Program_Nilai {
    public static void main(String[] args) {
        double ntugas,nuts,nuas,nilaiangka;
        String nhuruf, nama, Predikat;
        Scanner inp = new Scanner(System.in);
        // Input nilai mahasiswa
        System.out.print(" Nama Siswa  : ");
```



```

    nama=inp.next();
System.out.print(" Nilai Tugas : ");
    ntugas=inp.nextDouble();
System.out.print(" Nilai UTS : ");
    nuts=inp.nextDouble();
System.out.print(" Nilai UAS : ");
    nuas=inp.nextDouble();
// nilai Tugas = 30%, UTS = 30%, UAS = 40%
nilaiangka=(30*ntugas)/100 + (nuts*0.30)+(nuas * 0.40);
System.out.println(" Nilai Akhir : " + nilaiangka);
// Cek kondisi untuk mendapatkan nilai huruf -----
if (nilaiangka >=85 && nilaiangka <=100)
{
    nhuruf="A";
    Predikat="Sangat Baik";
}
else if (nilaiangka >=70 && nilaiangka <85)
{
    nhuruf="B";
    Predikat="Baik";
}
else if (nilaiangka >=56 && nilaiangka < 70)
{
    nhuruf="C";
    Predikat="Cukup Baik";
}

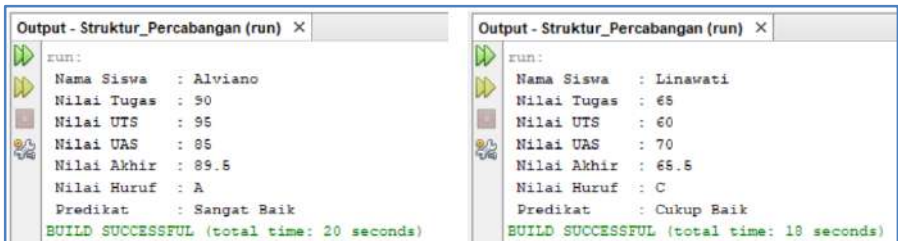
```

```

    }
    else if (nilaiangka >=40 && nilaiangka <56)
    {
        nhuruf="D";
        Predikat="Kurang Baik";
    }
    else
    {
        nhuruf="E";
        Predikat="Gagal Baik";
    }

    System.out.println(" Nilai Huruf : " + nhuruf);
    System.out.println(" Predikat   : " + Predikat);
}
}

```



Gambar 4.4 Hasil Eksekusi Program Nilai (if..else if 2)

Penjelasan:

Contoh program di atas mempunyai banyak kondisi. Program digunakan untuk menampilkan nilai huruf berdasarkan *range* nilai angka. Pertama-tama pengguna diminta memasukkan data nama, nilai tugas, nilai

uts, dan nilai uas. Kemudian program akan menjumlahkan nilai-nilai yang mempunyai bobot yang berbeda-beda untuk mendapatkan nilai angka.

$$\text{nilaiangka} = (30 * \text{ntugas}) / 100 + (\text{nuts} * 0.30) + (\text{nuas} * 0.40);$$

Seperti terlihat diatas nilai tugas =30%, nilai uts=30%, dan nilai uas=40%, penulisan perhitungan persen dapat dilakukan dengan dua cara, yaitu **$(30 * \text{ntugas}) / 100$ atau $(\text{nuts} * 0.30)$** .

Setelah program mendapatkan nilai angka, program akan mencari nilai huruf berdasarkan kondisi *range* nilai, misal nilaiangka ≥ 85 && nilaiangka ≤ 100 = A dan Predikat "Sangat Baik". Terlihat pada potongan program ini nilai A dari 85 sampai 100, simbol && menyatakan dan. Program akan menguji kondisi pertama, apabila terpenuhi atau bernilai *true*, maka akan menjalankan *statement* program yang ada di dalam blok kondisi pertama. Jika kondisi pertama tidak terpenuhi atau bernilai *false*, program akan menguji kondisi kedua. Jika kondisi kedua terpenuhi atau bernilai *true*, maka program akan menjalankan *statement* pada blok kedua, Satu-persatu kondisi akan di uji sampai dengan kondisi terakhir, jika tidak terpenuhi, program akan menjalankan *statement* yang ada pada blok *else*

F. STATEMENT SWITCH

Pada program java terdapat bentuk lain dari *if-else if*, yaitu dengan menggunakan *statement switch – case*. Pada percabangan *switch – case* tidak semua tipe data dapat digunakan. Tipe yang digunakan hanya terbatas pada *integer (byte, int, short)* dan *char*,

Bentuk *statement switch – case*:

```
switch( switch_expression )
{
    case case_selector1:
        statement;
        statement;
        ... //
        break;
    case case_selector2:
        statement;
        statement;
```

```

        ...//
        break;
    case case_selector-n:
        statement;
        statement;
        ...//
        break;
    default:
        statement;
        statement;
        break;
}

```

Pada *switch_expression* akan menjalankan *statement case_selector1*, *case_selector2* dan seterusnya. Ketika ditemukan *case* yang memenuhi kondisi program akan menjalankan *statement* dari awal sampai menemui *statement break*, dan melewati *statement* yang lain sampai akhir struktur *switch*. Jika tidak ditemui *case* yang cocok, maka program akan mengeksekusi blok *default*. Bisa anda catat bahwa blok *default* adalah *optional*. Sebuah *statement switch* bisa tidak memiliki blok *default*.

Catatan:

- Tidak seperti *statement if*, pada struktur *switch statement* dieksekusi tanpa memerlukan tanda kurung kurawal (**{}**).
- Ketika sebuah *case* pada *statement switch* menemui kecocokan, semua *statement* pada *case* tersebut akan dieksekusi. Tidak hanya demikian, *statement* lain yang berada pada *case* yang cocok juga dieksekusi.
- Untuk menghindari program mengeksekusi *statement* pada *case* berikutnya, kita menggunakan *statement break* sebagai *statement* akhir

Contoh Program Bulan (*switch*):

```
/* Program Percabangan
    Menggunakan Statement Switch */
package struktur_percabangan;
import java.util.Scanner;
// @author Yuniansyah
public class Program_Bulan {
    public static void main(String arg[]){
        int bulan, hari,tahun ;
        Scanner inp = new Scanner(System.in);
        System.out.println("Program Menampilkan Bulan ");
        System.out.println("Berdasarkan Input Angka  ");
        System.out.println("===== ");
        System.out.print(" Bulan [1 ..12] : ");
        bulan=inp.nextInt();
        switch (bulan) {
            case 1:
                System.out.println(" Bulan Adalah : Januari : ");
                break;
            case 2:
                System.out.println(" Bulan Adalah : Februari : ");
                break;
            case 3:
                System.out.println("Bulan Adalah : Maret : ");
```

```
break;
```

```
case 4:
```

```
    System.out.println("Bulan Adalah : April : ");
```

```
    break;
```

```
case 5:
```

```
    System.out.println("Bulan Adalah : Mei : ");
```

```
    break;
```

```
case 6:
```

```
    System.out.println ("Bulan Adalah : Juni : ");
```

```
    break;
```

```
case 7:
```

```
    System.out.println("Bulan Adalah : Juli : ");
```

```
    break;
```

```
case 8:
```

```
    System.out.println("Bulan Adalah : Agustus : ");
```

```
    break;
```

```
case 9:
```

```
    System.out.println("Bulan Adalah : September : ");
```

```
    break;
```

```
case 10:
```

```
    System.out.println("Bulan Adalah : Oktober : ");
```

```
    break;
```

```
case 11:
```

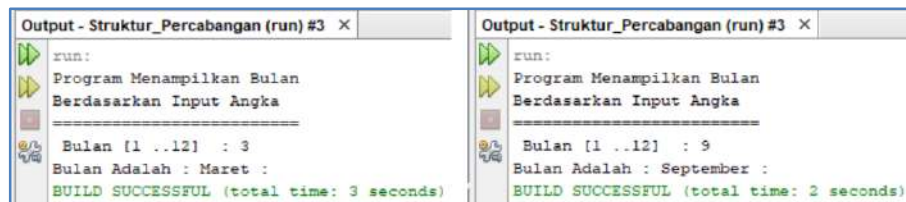
```
    System.out.println("Bulan Adalah : November : ");
```

```

        break;
    case 12:
        System.out.println("Bulan Adalah : Desember : ");
        break;
    default:
        System.out.println("Maaf bulan hanya sampai 12.");
        break;
    }
}
}

```

Hasil:



Gambar 4.5 Tampilan Hasil Eksekusi Program Bulan (Switch 1)

Penjelasan:

Program diatas digunakan untuk menampilkan Bulan berdasarkan angka yang dimasukkan. Angka dimulai dari satu 1 untuk menyatakan Bulan Januari,

case 1:

```

System.out.println(" Bulan Adalah : Januari : ");
break;

```

Angka 2 untuk menyatakan Bulan Februari sampai dengan angka 12 yang menyatakan Bulan Desember. Terlihat pada program Setiap Blok program dimulai dengan **Case:** dan diakhiri dengan **break** untuk menyatakan akhir dari *statement*. Pada bagian akhir terdapat *statement default* yang digunakan untuk menampilkan pesan apabila pengguna salah dalam memasukkan angka.

G. RANGKUMAN MATERI

1. Struktur kondisi terbagi menjadi tiga bagian, yaitu struktur kondisi untuk satu kondisi, struktur kondisi untuk dua kondisi dan struktur kondisi untuk tiga atau lebih
2. Struktur kondisi untuk satu kondisi menggunakan *statement if*
3. Struktur kondisi untuk dua kondisi menggunakan *statement if..else*
4. Struktur kondisi untuk tiga kondisi atau lebih bisa menggunakan *statement if..else if* atau menggunakan *statement switch*
5. Pada struktur kondisi yang menggunakan *statement switch* tipe data yang digunakan terbatas pada *integer (byte, int, short)* dan *char*.

TUGAS DAN EVALUASI

1. Tuliskan perbedaan *statement if* dan *if else* dan buatlah contoh program untuk kedua *statement* ini
2. Tuliskan perbedaan struktur kondisi yang menggunakan *Statement if..else if* dengan struktur kondisi dengan menggunakan *Statement Switch*
3. Buat program untuk menentukan Bilangan Terbesar dari dua bilangan yang di *inputkan*
4. Buatlah program untuk menentukan hari dari angka yang dimasukkan, dengan ketentuan angka 1. adalah Hari Senin dan Angka 7 Adalah Hari Minggu

DAFTAR PUSTAKA

- Hakim, Rachmad & Sutarto. 2009. *Mastering Java*, Jakarta, Elex Media Komputindo.
- Hariyanto, Bambang. 2012. *Esensi-Esensi Bahasa Pemrograman Java*, : Revisi Keempat. Bandung: Informatika.
- Kadir Kadir, 2013, *Teori dan Aplikasi Struktur Data Menggunakan Java*, Yogyakarta, Penerbit Andi
- Kadir, Abdul. 2012. *Algoritma dan Pemrograman Menggunakan Java*. Yogyakarta: Penerbit Andi
- Suarga. 2012. *Algoritma dan Pemrograman*. Edisi Kedua. Yogyakarta: Penerbit Andi.



PERULANGAN

Andri Saputra, S.Kom., M.Kom
Institut Teknologi dan Bisnis PalComTech

A. PENDAHULUAN

Struktur Perulangan adalah salah satu pernyataan terpenting dalam pemrograman. Struktur ini digunakan untuk mengulang satu atau lebih pernyataan selama kondisi terpenuhi. Dengan struktur perulangan, *programmer* tidak perlu menulis kode program sebanyak jumlah pengulangan yang diperlukan.

Pada masing-masing bahasa pemrograman mempunyai sintak untuk struktur perulangan, tetapi secara garis besar struktur perulangan mempunyai aturan yang harus dipenuhi, yaitu:

1. Inisialisasi adalah langkah persiapan untuk mengulang kondisi awal sebuah baris, seperti memasukkan nilai awal dalam variabel. Fase ini berjalan sebelum memasuki bagian perulangan
2. Proses berjalan di dalam bagian perulangan yang berisi semua proses yang perlu dijalankan berulang kali.
3. Iterasi terjadi di dalam perulangan, ini merupakan sebuah kondisi tambahan untuk melanjutkan perulangan agar bisa terus berjalan
4. Terminasi adalah kondisi untuk berhenti dari sebuah perulangan, kondisi berhenti sangat penting untuk pengulangan,

menghindari pengulangan tanpa batas. Kondisi perulangan adalah kondisi yang harus dipenuhi oleh algoritma yang sedang berjalan untuk memasuki blok perulangan.

Pada pemrograman Java, ada tiga struktur perulangan yaitu: ***for***, ***while***, dan ***do-while***. Untuk lebih jelasnya penulis akan uraikan satu persatu struktur penulisan beserta contoh program masing-masing dari struktur perulangan.

B. PERULANGAN ***FOR***

Struktur *For* digunakan untuk mengulang perintah yang sudah ditentukan. Struktur ini memungkinkan anda untuk menentukan seberapa sering perulangan akan berulang dimana jumlah perulangannya biasanya sudah ditentukan dari awal. Dalam struktur ini, variabel *integer* biasanya digunakan sebagai kondisi. Bentuk umum dari struktur ini adalah:

```
for (nilai/kondisi awal; kondisi/nilai akhir ; langkah perulangan)
{
    statement1;
    statement2;
    ...
}
```

Keterangan:

awal: inisialisasi nilai awal perulangan

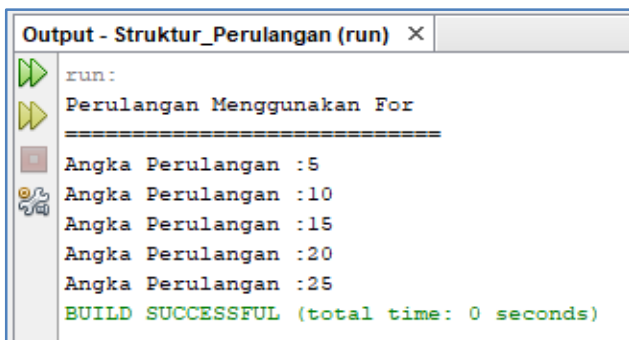
akhir: inisialisasi nilai akhir perulangan

langkah: inisialisasi nilai penambahan/pengurangan perulangan

Contoh Program Perulangan *For* 1:

```
* Program Perulangan Menggunakan For
  Untuk menampilkan Teks dan
  Angka Perulangan dari 5 sampai 25 dengan step 5 */
package struktur_perulangan;
// @Author : Andri Saputra
public class Perulangan_For1 {
    public static void main(String[] args) {
        System.out.println("Perulangan Menggunakan For");
        System.out.println("=====");
        for(int x =5;x<=25;x+=5)
        {
            System.out.println("Angka Perulangan :"+ x);
        }
    }
}
```

Hasil:



```
Output - Struktur_Perulangan (run) ×
run:
Perulangan Menggunakan For
=====
Angka Perulangan :5
Angka Perulangan :10
Angka Perulangan :15
Angka Perulangan :20
Angka Perulangan :25
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 5.1 Hasil Eksekusi Program *For*1 (Perulangan *For* 1)

Penjelasan:

Program diatas digunakan untuk menampilkan angka 5 sampai dengan 25. Perhatikan potongan program berikut: **for(int x =5;x<=25;x+=5)** terlihat variabel yang digunakan adalah x yang bertipe data *integer* (int) yang dimulai dari 5 sampai dengan x<=25 dan angka yang ditampilkan bertambah sebanyak 5 (+=5). Y

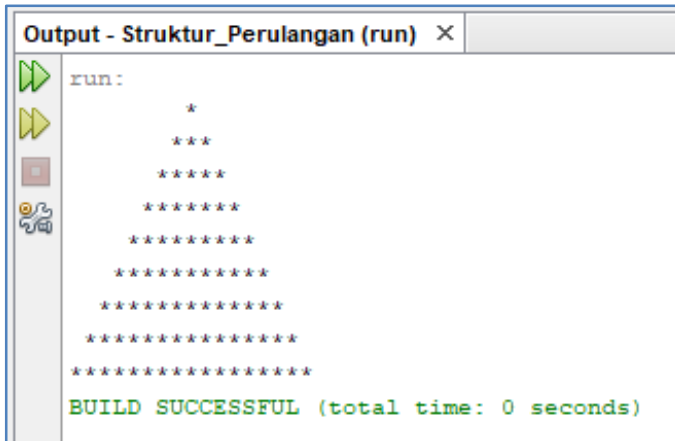
■ ***Nested For***

Nested For atau *for* bersarang adalah bentuk dimana pada perulangan terdapat perulangan. Contoh perulangan ini dapat digunakan untuk program yang di dalamnya banyak terdapat perulangan. Contoh perulangan ini dapat kita gunakan untuk menampilkan pola tertentu, Perhatikan contoh program dibawah ini yang akan menampilkan pola segitiga bintang

Contoh Program Perulangan *For* 2:

```
/* Program Perulangan For
   Untuk menampilkan Pola Bintang */
package struktur_perulangan;
// @author Andri Saputra
public class Perulangan_For2 {
    public static void main(String[] args) {
        int i, j, k;
        for (i=1; i <=9 ; i++)
        {
            for (j=9; j>i; j--)
            {
                System.out.print(" ");
            }
            for (k=1; k<(2*i) ; k++)
            {
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

Hasil:



```
run:
      *
     **
    ***
   ****
  *****
 *****
*****
*****
*****
*****
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 5.2 Hasil Eksekusi Program *For 2 (Nested For)*

Penjelasan:

Program di atas menggunakan *nested for*, yaitu didalam perulangan terdapat perulangan. Perulangan pertama digunakan untuk menambah baris dari 1 sampai 9, kemudian perulangan kedua digunakan untuk mengatur spasi atau baris kosong, dan perulangan ketiga digunakan untuk menampilkan karakter “*”.

C. PERULANGAN *WHILE*

Statement while loop adalah *statement* atau blok *statement* yang diulang-ulang sampai mencapai kondisi terpenuhi.

```
Bentuk statement while,
while( boolean_expression )

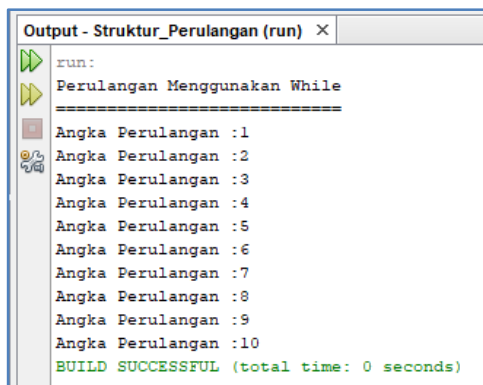
{
    Statement1;
    Statement2;
    ...
}
```

Statement di dalam *while loop* akan dieksekusi berulang-ulang selama *boolean_expression* bernilai *true*.

Contoh Program Perulangan *While* 1

```
/* Program Perulangan
 * Menggunakan Struktur While
  Untuk menampilkan Angka 1 sampai dengan 10 */
package struktur_perulangan;
// @author Andri Saputra
public class Perulangan_While1 {
    public static void main(String[] args) {
        int i=1;
        System.out.println("Perulangan Menggunakan While");
        System.out.println("=====");
        while (i<=10)
        {
            System.out.println("Angka Perulangan :"+i);
            i++;
        }
    }
}
```

Hasil:



```
run:
Perulangan Menggunakan While
=====
Angka Perulangan :1
Angka Perulangan :2
Angka Perulangan :3
Angka Perulangan :4
Angka Perulangan :5
Angka Perulangan :6
Angka Perulangan :7
Angka Perulangan :8
Angka Perulangan :9
Angka Perulangan :10
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 5.3 Hasil Eksekusi Program *While* 1 (Perulangan *While* 1)

Penjelasan:

program di atas hampir sama dengan program perulangan for yang pertama, dimana angka awal dan akhir telah ditentukan pada program, yaitu untuk menampilkan angka 1 sampai dengan 10

```
int i=1;

while (i<=10)
dengan langkah perulangan sebanyak 1
    i++ ;
```

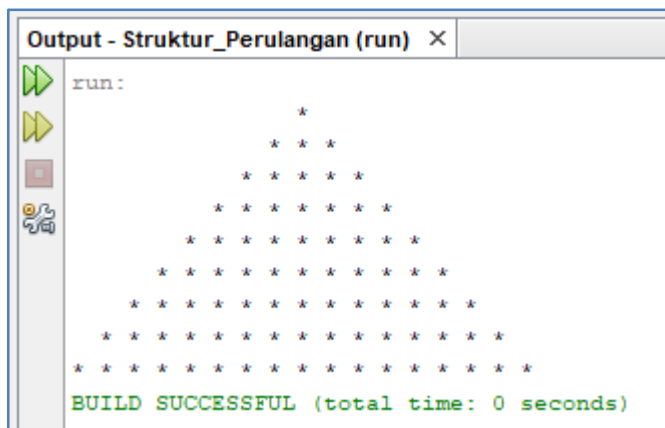
■ **Nested While**

Nested while sama seperti *nested For* dimana pada perulangan terdapat perulangan. Pada contoh disini penulis menggunakan *nested while* juga untuk menampilkan pola bintang.

Contoh Program Perulangan While 2

```
/* Program Perulangan
   Menggunakan While
   Untuk menampilkan Pola Bintang */
package struktur_perulangan;
// @author Andri Saputra
public class Perulangan_While2 {
    public static void main(String[] args) {
        int i=1,j,k,l;
        while (i<=9)
        {
            j=9-i;
            while (j>=1)
            {
                System.out.print(" ");
                j--;
            }
            k=1;
            while (k<=i)
            {
                System.out.print("* ");
                k++;
            }
            l=1;
            while (l<=i-1)
            {
                System.out.print("\n");
                l++;
            }
            i++;
            System.out.println();
        }
        System.out.print("\n");
    }
}
```


Hasil:



```
run:

      *
    * * *
  * * * * *
* * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *
* * * * * * *

BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 5.4 Hasil Eksekusi Program *While* 2 (Perulangan *Nested While*)

Penjelasan:

Program di atas menggunakan *nested while* yang hampir sama dengan *nested for*. Program digunakan untuk menampilkan pola bintang sebanyak 9 baris. Terlihat pada *Statement While* pertama digunakan untuk mengatur jumlah baris dan spasi, kemudian *statement while* kedua untuk mengatur spasi atau baris kosong.

```
int i=1,j,k,l;
while (i<=9)
{
    j=9-1;
    while (j>=i)
    {
        System.out.print(" ");
        j--;
    }
}
```

Statement while ketiga dan keempat untuk mencetak pola bintang dan spasi.

```

while (k<=i)
{
    System.out.print("* ");
    k++;
}

while (l<=i-1)
{
    System.out.print("* ");
    l++;
}

```

D. PERULANGAN *DO WHILE*

Pernyataan perulangan *do-while* dieksekusi beberapa kali selama kondisi bernilai benar. Perbedaan antara perulangan *while* dan perulangan *do-while* adalah bahwa pernyataan pada perulangan *do-while* dieksekusi setidaknya satu kali karena kondisi *do-while* berada di bagian akhir program.

```

Bentuk statement do-while,
do {
    statement1;
    statement2;
    ...
}
while( boolean_expression );

```

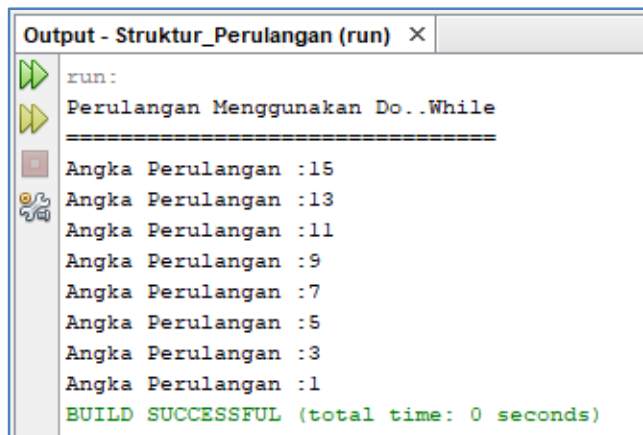
Pernyataan dalam *loop do-while* akan dieksekusi terlebih dahulu dan kemudian dilakukan pemeriksaan kondisi dari *boolean_expression*. Jika nilai tersebut tidak mencapai nilai yang diinginkan, pernyataan akan dieksekusi kembali.

Contoh Program Perulangan *Do...While* 1

```
/* Program Perulangan
   Menggunakan Do While
   Untuk Menampilkan Angka dari 15 sampai 1 dengan step - 2*/
package struktur_perulangan;
// @author Andri Saputra
public class Perulangan_Do_While1 {
    public static void main(String[] args) {
        int i=15;
        System.out.println("Perulangan Menggunakan Do..While ");
        System.out.println("=====");

        do
        {
            System.out.println("Angka Perulangan :"+i);
            i-=2 ;
        }
        while (i>=1);
    }
}
```

Hasil:



```
Output - Struktur_Perulangan (run) ×
run:
Perulangan Menggunakan Do..While
=====
Angka Perulangan :15
Angka Perulangan :13
Angka Perulangan :11
Angka Perulangan :9
Angka Perulangan :7
Angka Perulangan :5
Angka Perulangan :3
Angka Perulangan :1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 5.5 Hasil Eksekusi Program *do while* 1 (*do while* 1)

Penjelasan:

Program di atas sama dengan program pada struktur perulangan *for* dan *while* yang telah dibahas sebelumnya. Program ini digunakan untuk menampilkan angka dari besar ke kecil. Angka yang akan ditampilkan telah ditentukan. Pada awal program telah dideklarasikan variabel *i* bertipe data *integer* dan telah diberi nilai = 15. selanjutnya terdapat perintah *do* yang akan menjalankan *statement* yang ada pada blok program, yaitu:

```
System.out.println("Angka Perulangan :"+i);  
i-=2 ;
```

menampilkan kalimat dan angka *i*, kemudian *i* akan dikurangi dengan 2. Pada bagian akhir program akan menguji kondisi yang ada

while (i>=1);

yaitu selama *i* lebih besar atau sama dengan 1.

■ ***Nested do ..while***

Sama seperti perulangan *for* dan *while*, pada perulangan *do..while* juga bisa digunakan perulangan di dalam perulangan atau *nested do while*. Contoh perulangan *nested do while* disini penulis gunakan untuk menampilkan angka-angka yang berbentuk pola.

Contoh Program Latihan *do while* 2 (*nested do-while* 3)

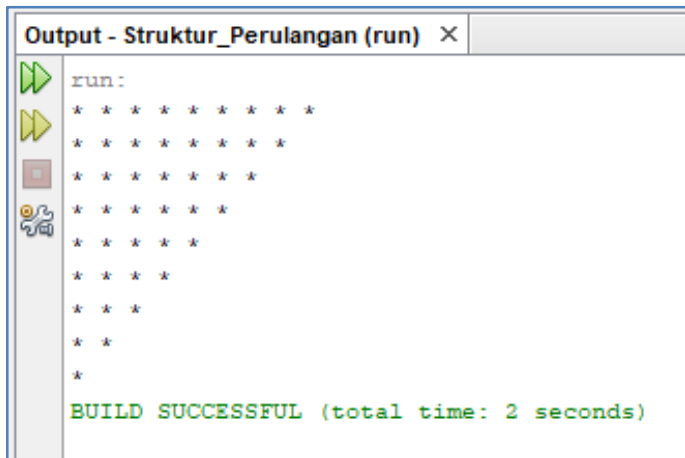
```
/* Program Perulangan  
Menggunakan Do While  
Untuk Menampilkan Pola Bintang */  
package struktur_perulangan;  
// @author Andri Saputra  
public class Perulangan_Do_while2 {  
    public static void main(String[] args){  
        int i=9,j;  
        do {  
            j=i;  
            do  
            {  
                System.out.print("*" + " ");  
                --j;  
            }  
        }  
    }  
}
```

```

        while(j>=1);
        --i;
        System.out.println();
    }
    while (i>=1);
}
}

```

Hasil:



Gambar 5.6 Hasil Eksekusi Program *do while* 2 (*nested do while*)

Penjelasan:

Program di atas digunakan untuk menampilkan pola bintang dengan cara menurun banyak bintangnya sesuai dengan kebalikan barisnya. Pertama pola bintang akan menampilkan sebanyak 9 dan akan menurun/berkurang menjadi sebanyak 8 di baris selanjutnya berkurang satu untuk baris-baris selanjutnya, sampai dengan baris terakhir atau baris ke sembilan hanya ada satu bintang.

E. RANGKUMAN MATERI

Algoritma Perulangan adalah algoritma yang mengulang atau mengulang langkah tertentu. Masalahnya juga memiliki langkah-langkah yang perlu diulang. Semua bahasa pemrograman memiliki sintaks struktur *loop*, tetapi secara umum ada aturan yang harus diikuti. Pemrograman Java memiliki tiga struktur *loop*: **for**, **while**, dan **do-while**.

TUGAS DAN EVALUASI

1. Jelaskan pengertian dari struktur perulangan ?
2. Jelaskan dan berikan contoh dari masing-masing 3 struktur perulangan ?
3. Apa yang dimaksud dengan terminasi ?
4. Perhatikan Sintak di bawah ini, sintak manakah yang *error* Ketika di jalankan ?

```
/* Program Perulangan
   Menggunakan Do While
   Untuk Menampilkan Angka dari 10 sampai 1 dengan step - 2*/
package struktur_perulangan;
// @author Andri Saputra
public class Perulangan_Do_While1 {
    public static void main(String[] args) {
        int i=10;
        System.out.println("Perulangan Menggunakan Do..While ");
        System.out.println("===== ");

        do
        {
            System.out.println("Angka Perulangan :"+i)
            i-=2 ;
        }
        while (i>=1);
    }
}
```

5. Buatlah algoritma untuk mencetak tulisan Selamat Pagi Dunia sebanyak 1000 baris.

DAFTAR PUSTAKA

- Hakim, Rachmad & Sutarto. 2009. *Mastering Java*, Jakarta, Elex Media Komputindo.
- Hariyanto, Bambang. 2012. *Esensi-Esensi Bahasa Pemrograman Java*, : Revisi Keempat. Bandung: Informatika.
- Kadir, Abdul. 2012. *Algoritma dan Pemrograman Menggunakan Java*. Yogyakarta: Penerbit Andi
- Suarga. 2012. *Algoritma dan Pemrograman*. Edisi Kedua. Yogyakarta: Penerbit Andi.



ARRAY

Yesi Sriyeni, S.Kom., M.Kom

Institut Teknologi dan Bisnis PalComTech Palembang

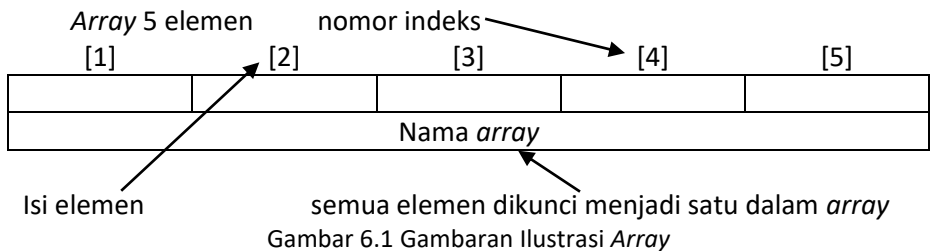
A. PENDAHULUAN

Array adalah salah satu tipe data terstruktur yang digunakan dalam Bahasa pemrograman seperti PHP, Pascal, JAVA, C++, Python. Penggunaan *array* dapat mempermudah penyimpanan data karena *array* berisi komponen-komponen yang memiliki tipe data yang sama. Setiap anggota *array* dapat diakses melalui suatu indeks. *Array* mempunyai beberapa jenis seperti *array* 1 dimensi, *array* 2 dimensi dan *array* multidimensi, dimana masing-masing *array* tersebut memiliki fungsi yang berbeda dalam penggunaannya. *Array* 1 dimensi disebut dengan *vektor*, *array* 2 dimensi sering disebut sebagai matriks sedangkan *array* yang memiliki dimensi lebih dari 2 (dua) yang disebut tensor.

Bab ini akan membahas definisi *array* dan deklarasinya, *array* 1 dimensi, *array* 2 dimensi dan manipulasi *string* serta latihan soal agar pembaca lebih memahami tentang *array*.

B. DEFINISI *ARRAY* DAN DEKLARASI *ARRAY*

Array, sebuah tipe data terstruktur yang dapat diterapkan pada suatu variabel yang dapat menyimpan banyak data dengan tipe sejenis atau homogen. *Array* berfungsi untuk mempermudah dalam penyimpanan data, sebagai contoh suatu data memiliki dua puluh nilai, jika tidak dideskripsikan menggunakan *array*, data tersebut haruslah dibuat menjadi dua puluh variabel? Belum lagi saat nanti data akan diakses berkemungkinan data lebih rumit dibaca karena variabel yang begitu banyak dengan nama yang berbeda. Gambaran dari definisi *array* bisa dilihat dari ilustrasi berikut ini:



Array dapat dideklarasikan dengan beberapa cara, *array* bisa digunakan secara terus menerus dalam suatu algoritma secara konsisten. Untuk mendeklarasikan *array* hal yang harus dilakukan terlebih dahulu adalah mendeklarasikan variabel dengan tipe data yang mengarah kepada *array* dan jumlah elemen *array*, dimana *array* tersebut adalah sebuah objek. Perhatikan contoh sintaks berikut ini:

1. Nama_array : array [1..n] of tipe_data;

Contoh : simpul : array [1..4] of integer

2. Tipe_data nama_array [n];

Contoh : integer simpul[4];

3. Type array : array [1..4] of tipe_data;

Nama_array: array;

4. Contoh : type array : array [1..4] of integer;

i. simpul : array

Sintaks diatas adalah contoh pendeklarasian *array* ke dalam bahasa algoritmik, dimana setiap objek *array* memiliki nama, tipe data dan jumlah elemen *array*. Dalam beberapa Bahasa pemrograman seperti Pascal, Python suatu indeks *array* bisa dimulai dengan angka 0 dan 1, sedangkan untuk Bahasa pemrograman seperti Bahasa C, C++ dan Java indeks *array* dimulai dengan angka 0.

Array hanya dapat berisi sesuai dengan jumlah kapasitas yang sudah dideklarasikan, jika indeks *array* yang diakses lebih dari kapasitasnya maka akan terjadi *error* pada program yang dijalankan karena elemen yang di *input* belum ada didalam deklarasi elemen *array*.

C. ARRAY 1 DIMENSI

1. Definisi dan Deklarasi

Array 1 dimensi atau yang sering disebut dengan vektor adalah suatu *array* yang *value* atau nilainya hanya ditunjukkan oleh satu indeks. Sebagai contoh misalnya Ketika dilakukan pengukuran berat badan untuk 30 orang karyawan, maka dibuatlah variabel berat badan yaitu:

berat_badan₁, berat_badan₂, berat_badan₃, berat_badan₄, ..., berat_badan₃₀,

Contoh ini menjelaskan bahwa setiap nilai dari variabel *berat_badan* hanya ditunjukkan oleh satu nilai indeks saja.

Array 1 dimensi juga dideklarasikan dengan variabel yang sudah dibentuk sebelumnya dan tidak dapat diubah selama program dijalankan. Secara umum, *array* 1 dimensi dideklarasikan dengan bentuk sebagai berikut:

Nama array : array [1..n] of tipe_data

Keterangan:

Nama *array* : nama *array* yang akan disimpan berisi elemen atau nilai
1 : indeks awal elemen atau nilai, awal ini berupa konstanta tidak boleh nama variabel. Indeks awal bisa dimulai dengan angka 1 atau 0 tergantung Bahasa pemrograman.

- N : indeks akhir yang menyatakan posisi/jumlah maksimum *array* yang dapat ditampung.
- Tipe_data : tipe data *array* yang digunakan, bisa berupa tipe data dasar atau tipe data bentukan.

Dari bentuk umum deklarasi *array* 1 dimensi, untuk contoh sebelumnya bisa didapatkan deklarasi *array* sebagai berikut:

berat_badan : array [1..30] of real

Berdasarkan contoh diatas variabel *array* berat_badan akan menyimpan 30 indeks nilai yang dimulai dengan angka 1 sampai dengan 30 yang elemen-elemen *array* memiliki tipe data real. Untuk mengakses *array* yang sudah dideklarasikan dapat dilakukan dengan dua acara yaitu mengisi nilai *array* dengan nilai indeks tertentu dalam rentan indeks nilai yang sudah ditentukan sebelumnya agar program yang dijalankan tidak mengalami *error* atau dengan cara mengakses nilai *array* dengan indeks nilai tertentu. Untuk lebih jelasnya perhatikan tabel 6.1 berikut ini:

Tabel 6.1 Deklarasi dan Akses Array 1 Dimensi

Bentuk Umum	Bahasa Pascal	Bahasa Java	Bahasa C dan C++
<u>Deklarasi</u> berat_badan : array [1..30] of real	type berat_badan = array [1..30] of real; var bb : berat_badan;	real[] berat_badan= new real[30];	real berat_badan [30];
<u>Akses</u> 1. Mengakses nilai <i>array</i> dengan indeks tertentu	nama_array[in deks]; berat_badan[1];	nama_array[ind eks]; berat_badan[0] ;	nama_array[indeks]; berat_badan [0];

2. Mengisi nilai <i>array</i> dengan indeks tertentu	nama_array[indeks]:=nilai berat_badan[1] := 5;	nama_array[indeks]:=nilai berat_badan[0] := 5;	nama_array[indeks]:=nilai berat_badan[0] := 5;
--	---	---	---

Tabel 6.1 menjelaskan cara deklarasi dan akses *array* dari beberapa contoh bahasa pemrograman berdasarkan contoh awal. Jika diperhatikan lebih detail terdapat perbedaan deklarasi indeks pada Bahasa pemrograman Pascal yang dimulai dengan angka 1 sedangkan pada Bahasa JAVA dan C dan C++ indeks dimulai dengan angka 0 (nol).

2. Contoh Array 1 Dimensi

Untuk lebih memahami tentang *array* 1 dimensi perhatikan contoh-contoh berikut ini:

Contoh soal 1:

Buatlah sebuah algoritma untuk membaca 20 bilangan bulat lalu hitung jumlah dan rata-rata bilangan tersebut!

Untuk menjawab contoh soal diatas dimulai dari membaca bilangan bulat yang *diinput* kemudian jumlahkan seluruh bilangan dari awal sampai akhir lalu hitung rata-rata bilangan yang *diinput*. Selengkapnya akan dibahas berikut ini:

Algoritma contoh soal 1

Deklarasi *array*

```
bilangan : array [1..20] of integer
i, jumlah : integer
rata : real
```

Deskripsi

```
{untuk membaca bilangan}
for i ← 1 to 20 do
    input (bilangan[i])
end for
```

```

{untuk menghitung jumlah bilangan}
jumlah ← 0
for i ← 1 to 20 do
    jumlah ← jumlah+bilangan[i]
end for

{menghitung rata-rata}
jumlah/20

{mencetak elemen array}
for i ← 1 to 20 do
    output (bilangan[i])
end for
output ('Jumlah bilangan = ',jumlah)
output ('Rata-rata bilangan= ',rata-rata)

```

Bahasa Pascal contoh soal 1

```

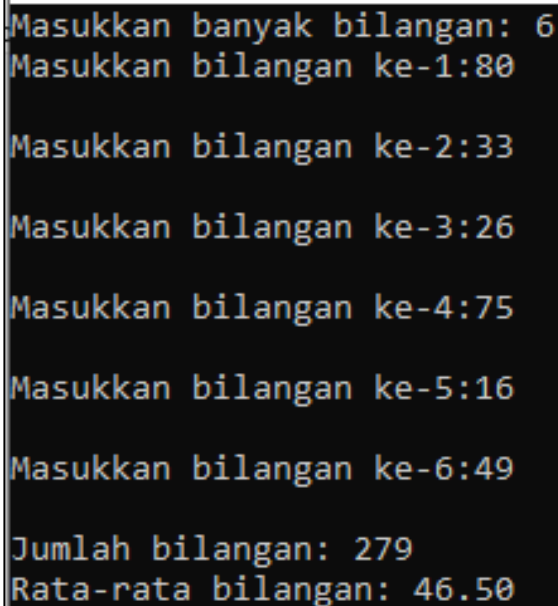
program contohsoalYESI;
uses crt;
var
    bilangan: array [1..20] of integer;
    n,i,jumlah:integer;
    rata:real;
begin
    clrscr;
    write('Masukkan banyak bilangan: ');
    readln(n);
    jumlah:=0;
    for i:= 1 to n do
    begin
        write('Masukkan bilangan ke-',i,':');
        readln(bilangan[i]);
        jumlah:=jumlah+bilangan[i];
        writeln;
    end;
end;

```

```
rata:=jumlah/n;  
writeln('Jumlah bilangan: ',jumlah);  
writeln('Rata-rata bilangan: ',rata:0:2);  
readln;  
  
end.
```

Implementasi *coding* bisa berbeda sesuai dengan Bahasa pemrograman yang digunakan dalam mendeskripsikan *array*.

Hasil *output* contoh soal 1



```
Masukkan banyak bilangan: 6  
Masukkan bilangan ke-1:80  
  
Masukkan bilangan ke-2:33  
  
Masukkan bilangan ke-3:26  
  
Masukkan bilangan ke-4:75  
  
Masukkan bilangan ke-5:16  
  
Masukkan bilangan ke-6:49  
  
Jumlah bilangan: 279  
Rata-rata bilangan: 46.50
```

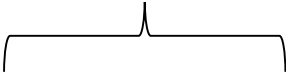
Gambar 6.2 Hasil *output* Bahasa Pascal Contoh Soal 1

D. ARRAY 2 DIMENSI

1. Definisi dan Deklarasi

Array 2 dimensi atau yang lebih sering digambarkan dengan sebuah matriks digunakan untuk menggambarkan *array* yang lebih rumit. Contohnya di perhitungan matematika yang memiliki matriks/grafik/diagram yang memiliki sumbu x dan y. *Array* 2 dimensi juga bisa disebut sebagai perluasan dari *array* 1 dimensi, dimana jika pada *array* 1 dimensi hanya terdapat satu kolom dan satu baris maka pada *array* 2 dimensi tersusun atas beberapa elemen kolom dan beberapa elemen baris dengan tipe data yang sama atau sejenis.

Secara garis besar, jika sebelumnya sudah memahami tentang *array* 1 dimensi maka harusnya tidak sulit untuk memahami alur dari *array* 2 dimensi. Untuk lebih memahami definisi *array* 2 dimensi, perhatikan susunan angka berikut ini:

	Indeks Kolom			
				
Indeks Baris	11	4	90	6
	9	8	23	22
	31	15	29	7
	2	10	42	30
	12	32	11	78
	67	5	9	7
	43	10	11	9

Susunan angka diatas menunjukkan matriks berdimensi 7 x 4, artinya terdiri dari 7 baris dan 4 kolom. Dalam pengisian *array* 2 dimensi dapat dilakukan dengan dua cara yaitu baris per baris (*row major order*) dan kolom per kolom (*column major order*).

Sebagai variabel dalam algoritma, *array* 2 dimensi dideklarasikan dengan bentuk umum seperti berikut ini:

NamaArray : array [1..MaxBaris, 1..MaxKolom] of TypeData

Keterangan :

NamaArray : nama *array* yang dibentuk

MaxBaris : batas maksimal baris *array* atau ukuran baris dari *array*

MaxKolom: batas maksimal kolom *array* atau ukuran kolom dari *array*

TypeData : tipe data dari *array*

Dari bentuk umum deklarasi *array* 2 dimensi, berdasarkan contoh diatas maka hasil deskripsi matriks berdimensi 7 x 4 bisa dijabarkan seperti berikut ini:

matriks : array [1..7, 1..4] of integer

Setelah mendapatkan bentuk umum *array* 2 dimensi langkah selanjutnya adalah menyusun algoritma rinci mengenai cara mengisi *array* matriks 7 x 4 dan menampilkannya.

Algoritma Mengisi_matriks7x4

Deklarasi

```
const baris=7, kolom=4;
```

```
integer row,col
```

```
integer matriks[baris][kolom];
```

Deskripsi

```
for (row=1 to baris step 1)
```

```
    for (col=1 to kolom step 1)
```

```
        write ("Indeks baris ke-", row, "indeks kolom ke-", col);
```

```
        read(matriks[row][col]);
```

```
    endfor.
```

```
Endfor.
```


Selanjutnya susun algoritma untuk menampilkan *array* matriks 7x 4, sebagai berikut:

Algoritma Menampilkan_isi_matriks7x4

Deklarasi

```
const baris=7, kolom=4;

integer row,col

integer matriks[baris][kolom];
```

Deskripsi

```
for (row=1 to baris step 1)
    for (col=1 to kolom step 1)
        write(matriks[row][col]);
    endfor.
write();

endfor.
```

Agar lebih memahami *array* 2 dimensi, berikut ini contoh program dalam Bahasa Pascal untuk *input* dan mencetak *array* 2 dimensi dalam bentuk matriks berdimensi 3 x 2.

```
Program matrikarray2dimensiys;
Uses Crt;
Var
Matrik1: Array[1..3,1..2] of Integer;
b,c : Integer;
Begin
clrscr;
{ input matrik }
Writeln(' Elemen matrik');
```

```

For b := 1 to 3 Do
  Begin
    For c := 1 to 2 Do
      Begin
        Write('Elemen baris ke-',b,' Kolom ke-',c,'= ');
        Readln(matrik1[b,c]);
      End;
    End;

    { mencetak matrik }
    For b:= 1 to 3 Do
      Begin
        For c:= 1 to 2 Do
          Begin
            Write(Matrik1[b,c]);
          End;
          Writeln;
        End;
      End;
    Readln;
  End.

```

Array 2 dimensi berbentuk matrik dalam ilmu matematika memiliki beberapa fungsi seperti penambahan, pengurangan, perkalian dan penggabungan matrik. Setiap fungsi operasi tersebut bisa dimasukkan kedalam program Bahasa Pemrograman dan digunakan untuk membuat sebuah program komputer. Hasil *output* contoh matrik berdimensi 3 x 2 berikut ini berupa tampilan matrik yang belum memiliki operasi bilangan.

```

Elemen matrik
Elemen baris ke-1 Kolom ke-1= 22
Elemen baris ke-1 Kolom ke-2= 14
Elemen baris ke-2 Kolom ke-1= 52
Elemen baris ke-2 Kolom ke-2= 31
Elemen baris ke-3 Kolom ke-1= 20
Elemen baris ke-3 Kolom ke-2= 15
2214
5231
2015

```

Gambar 6.3 Hasil *output* program matrik berdimensi 3 x 2

Dari hasil ini jika ingin menambahkan operasi bilangan matrik seperti penambahan matrik bisa menggunakan *coding* berikut ini dengan mengambil contoh soal sebelumnya:

```

{proses penjumlahan tiap elemen}
For b := 1 to 7 Do
Begin
For c:= 1 to 4 Do
Begin
Hasil[b,c]:=Matrik1[b,c]+Matrik2[b,c]
;
End;
End;
{proses cetak hasil}
For b:= 1 to 7 Do
Begin
For c:= 1 to 4 Do

```

```

Begin
writeln('Hasil Penjumlahan Matriks');
writeln('=====');
Write(Hasil[b,c]:6);
End;
Writeln;
End;
Readln;
End.

```

E. MANIPULASI *STRING*

String adalah tipe data yang dapat menampung kumpulan dari karakter yang membentuk suatu kata atau bisa dikatakan *array* dari karakter. *String* berbeda dengan tipe data lain karena *string* bukan merupakan tipe data dasar melainkan tipe data bentukan yang terdiri dari gabungan huruf, angka, *whitespace* dan berbagai karakter. Deklarasi *string* dalam Bahasa pemrograman seperti Pascal dan C++ langsung bisa ditambahkan dengan menambahkan pada bagian variabel (*var*). Perhatikan contoh deklarasi *string* berikut ini:

```

program contoh_string;
uses crt;
var
  x:string;
begin
  clrscr;
  x:='Halo, Saya Seorang mahasiswa';
  writeln('x: ',x); {menampilkan string}
  x:='Ini pengetahuan tentang string';
  writeln('x: ',x);
  readln;
end.

```

```
x: Halo, Saya Seorang mahasiswa
x: Ini pengetahuan tentang string
```

Gambar 6.4 Hasil *output* sederhana deklarasi *string*

Program Bahasa Pascal diatas mendefinisikan “x” sebagai *string*, kemudian mengubah nilai x menjadi karakter tertentu yang ditampilkan disepanjang program. Pada dasarnya tipe data *string* mempunyai ruang untuk 255 karakter, jadi kita bisa menentukan sendiri seberapa banyak ruang tersebut akan digunakan dalam deklarasi. Cara menentukan kapasitas jumlah karakter dalam *string* adalah dengan menambahkan angka dalam kurung siku sebagai tanda jumlah karakter

Dengan *coding* yang sama hasil deklarasi *string* bisa berubah dari yang sebelumnya karena jumlah karakter *string* sudah dibatasi untuk variabel x hanya sampai 15 karakter. Tipe data *string* dalam operasinya hanya memiliki sebuah operator yaitu “+” yang dapat digunakan untuk menggabungkan dua buah *string* menjadi satu.

```
program contoh_string;
uses crt;
var
  x:string[15];
begin
  clrscr;
  x:='Saya mahasiswa';
  writeln('x: ',x);
  x:='Ini tentang string';
  writeln('x: ',x);
  readln;
end.
```

```
x: Saya mahasiswa
x: Ini tentang string
```

Gambar 6.5 Hasil *Output string* dengan Batasan karakter

String dalam perancangan sebuah program komputer hampir selalu digunakan karena dengan tipe data *string* inilah berbagai jenis karakter bisa dimasukkan kedalam program dalam bentuk variabel. Dalam Bahasa Pascal, *string* memiliki prosedur-prosedur standar yaitu:

1. *Delete*, digunakan untuk menghilangkan sejumlah karakter tertentu yang sudah dideklarasikan mulai dari posisi indeks nilai *string*. Bentuk umum prosedur *delete*, yaitu:

```
Delete (var s:string, index:integer, count:integer);
```

Apabila panjang *string* lebih kecil dibandingkan posisi index, maka tidak ada karakter yang akan dihapus atau dibuang.

Contoh:

```
Program cth_prsdr_delete;
uses crt;
var
  nama:string[12];
  urutan, posisi:integer;
begin
  clrscr;
  nama:= 'mahasiswa';
  for urutan := 1 to 12 do
  begin
    posisi:= 14-urutan;
    delete (nama, posisi, 1);
    writeln(nama);
    readln;
  end;
end.
```

2. *Insert*, digunakan untuk menyisipkan suatu *string* kedalam *string* lainnya yang dimulai dari nilai *integer* index. Bentuk umum prosedur *insert* sebagai berikut:

```
insert (source:string, var s:string, index:integer
```

Contoh:

```
Program cth_prsdr_insert;
uses crt;
var
  nama:string[20];
begin
  clrscr;
  nama:= 'maha';
  insert ('siswa', nama,5);
  writeln (nama);
  write();
  readln;
end.
```

3. *Str*, digunakan untuk merubah nilai *numeric* yang dapat berupa *numeric integer* ataupun *numeric real* menjadi nilai *string*. Bentuk umum dari prosedur *str*, sebagai berikut:

```
str( x:[width[decimals]], var s:string );
```

Contoh:

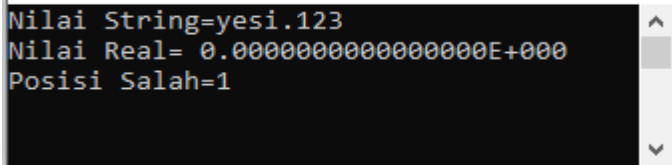
```
Program cth_prsdr_str;
uses crt;
var
  nilai1, nilai2:integer;
  n1,n2:string[6];
begin
  clrscr;
  nilai1:= 7893;
  nilai2:= 123;
  writeln(nilai1+nilai2);
  str (nilai1:3,n1);
  str(nilai2:3,n2);
  writeln (n1+n2);
  write();
  readln;
end.
```

4. *Val* adalah prosedur standar dalam *string* yang digunakan untuk mengkonversi nilai suatu *string* menjadi nilai *numeric*. Nilai *string* ditunjukkan oleh nilai *s* yang harus berupa angka atau tanda plus minus. Nilai variabel *numeric* ditunjukkan dengan *v*. Bentuk umum prosedur *Val*, sebagai berikut:

```
val(s:string, v, var code);
```


Contoh:

```
Program cth_prsdr_val;
uses crt;
var
  nilai_string: string[8];
  nilai_real: real;
  posisi_salah: integer;
begin
  clrscr;
  nilai_string:= 'yesi.123';
  val(nilai_string,nilai_real,
posisi_salah);
  writeln('Nilai String=',
nilai_string);
  writeln('Nilai Real=', nilai_real);
  writeln('Posisi Salah=',
posisi_salah);
  write();
  readln;
end.
```



Gambar 6.6 Hasil *output*

F. RANGKUMAN MATERI

- *Array* adalah salah satu tipe data terstruktur yang digunakan dalam Bahasa pemrograman seperti PHP,Pascal, JAVA, C++, Python. Penggunaan *array* dapat mempermudah menyimpan data karena *array* berisi komponen-komponen yang memiliki tipe data yang sama

- *Array* mempunyai beberapa jenis seperti *array* 1 dimensi, *array* 2 dimensi dan *array* multidimensi, dimana masing-masing *array* tersebut memiliki fungsi yang berbeda dalam penggunaannya. *Array* 1 dimensi disebut dengan *vektor*, *array* 2 dimensi sering disebut sebagai matriks sedangkan *array* yang memiliki dimensi lebih dari 2 (dua) yang disebut tensor.
- *Array* 1 dimensi atau yang sering disebut dengan vektor adalah suatu *array* yang *value* atau nilainya hanya ditunjukkan oleh satu indeks.
- *Array* 2 dimensi juga bisa disebut sebagai perluasan dari *array* 1 dimensi, dimana jika pada *array* 1 dimensi hanya terdapat satu kolom dan satu baris maka pada *array* 2 dimensi tersusun atas beberapa elemen kolom dan beberapa elemen baris dengan tipe data yang sama atau sejenis.
- *String* adalah tipe data yang dapat menampung kumpulan dari karakter yang membentuk suatu kata atau bisa dikatakan *array* dari karakter. *String* berbeda dengan tipe data lain karena *string* bukan merupakan tipe data dasar melainkan tipe data bentukan yang terdiri dari gabungan huruf, angka, *whitespace* dan berbagai karakter

TUGAS DAN EVALUASI

Agar lebih memahami tentang *array*, kerjakanlah tugas-tugas berikut ini:

1. Buatlah algoritma dan *coding* perhitungan bilangan (penjumlahan, pengurangan, perkalian, pembagian dan rata-rata) dengan jumlah data 8 bilangan!
2. Buatlah sebuah program *array* untuk menghitung nilai siswa!
3. Gambarlah 2 buah matriks berdimensi 8x8 ke dalam sebuah Bahasa pemrograman kemudian jumlahkan kedua matriks tersebut!

DAFTAR PUSTAKA

- A.S., R. (2018). *Logika Algoritma dan Pemrograman Dasar*. Bandung: Modula.
- Sitorus, M.Kom., D. (2015). *Algoritma dan Pemrograman*. Yogyakarta: Andi Offset.
- Suarga, S. (2012). *Algoritma dan Pemrograman*. Makassar: Andi Yogyakarta.
- Wadja, P. S., & Kristando, A. (2018). Pengembangan Media Modul Pada Materi Penggunaan Array Mata Pelajaran Algoritma Pemrograman Dasar Kelas X SMK Negeri 1 Jatirejo. *Jurnal Mahasiswa Teknologi Pendidikan*, 9, 216.



PROCEDURE DAN FUNCTION

Arsia Rini, S.Kom., M.Kom
Politeknik Negeri Sriwijaya Palembang

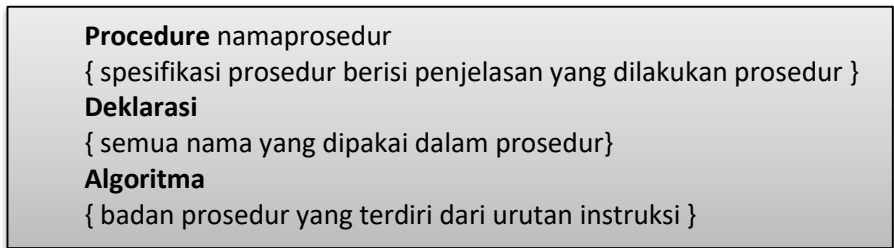
A. *PROCEDURE*

1. Pendahuluan

Algoritma merupakan suatu *sequence* (runtunan) beberapa aksi yang akan dilaksanakan aksi demi aksi mulai dari aksi baris pertama hingga aksi baris terakhir sesuai dengan urutan aksi tanpa ada pengulangan pelaksanaan aksi ataupun lompatan akibat suatu kondisi (Sitorus, 2015). *Procedure* dan *function* merupakan fungsi pada algoritma yang sering dijumpai dalam berbagai aktivitas, tidak hanya dalam kehidupan sehari-hari tetapi juga dalam pemrograman. Secara umum prosedur digunakan untuk fungsi yang tidak mengembalikan nilai sedangkan *function* biasanya ditandai untuk mengembalikan nilai. Beberapa materi prosedur yang akan dibahas adalah mengenai definisi prosedur, pemanggilan prosedur, deklarasi nama global dan lokal, serta penerapan parameter dengan bahasa pemrograman java.

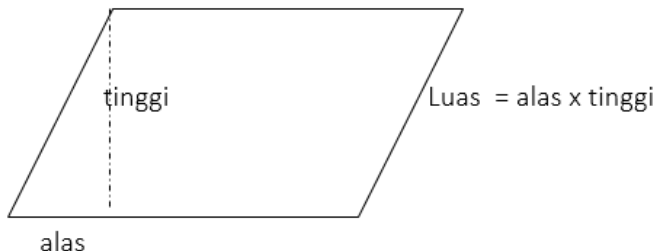
2. Definisi Prosedur

Prosedur merupakan istilah lain dari fungsi yang tidak mengembalikan nilai dan biasanya ditandai dengan *keyword void*. Prosedur memiliki struktur yang terdiri dari tiga bagian yaitu bagian judul, bagian deklarasi dan bagian algoritma (badan prosedur). Prosedur adalah modul program yang melakukan suatu tugas spesifik yang hasilnya dapat diamati setelah semua instruksi di dalamnya selesai dikerjakan (Rinaldi Munir, 2016). Secara umum algoritma penerapan struktur prosedur dapat dilihat pada gambar berikut:



Gambar 7.1 Struktur Prosedur

Selanjutnya agar dapat mendefinisikan prosedur kita bisa menguraikan bagian judul (*header*) kemudian mendeklarasikan keterangan dan selanjutnya mendeskripsikan uraian. Untuk lebih jelasnya penulisan algoritma terdapat pada contoh berikut:



Gambar 7.2 Algoritma Jajargenjang

Penyelesaian:

Misalnya nama prosedur yang digunakan adalah Hitung Luas Jajargenjang. Algoritma dalam prosedur HitungLuasJajargenjang dengan membaca alas dan tinggi, menghitung luas(L) dan mencetak luas (L)

Procedure HitungLuasJajargenjang

{ menghitung luas dengan rumus $L = \text{alas} \times \text{tinggi}$ }

Deklarasi

{bagian untuk mendeklarasikan variabel yang dibutuhkan seperti alas, tinggi dan luas (L)}

Algoritma

{digunakan untuk membaca alas, tinggi, membaca rumus Luas = alas x tinggi dan mencetak luas jajargenjang}

Dari penjelasan tersebut prosedur memiliki tiga bagian yaitu nama prosedur, deklarasi variabel yang dibutuhkan serta algoritma untuk membaca *input* dan mencetak luas.

3. Pemanggilan Prosedur

Pemanggilan prosedur dapat digunakan untuk pertukaran data atau informasi. Program yang dipanggil dapat memberikan masukan (*input*) atau luaran (*output*). Pemanggilan prosedur dengan menggunakan main program (program utama) atau dengan memanggil program yang lain misalnya prosedur x memanggil prosedur y. Berikut merupakan contoh penggunaan prosedur:

```
static void sapa(){  
    System.out.println("Selamat di Jurusan Teknik Komputer");  
}
```

Gambar 7.3 Program penggunaan prosedur

Gambar 7.3 Menjelaskan prosedur dengan nama sapa. *Void* artinya memiliki pengertian tipe data yang kosong, sehingga tidak mengembalikan nilai. Berikut merupakan pemanggilan prosedur sapa:

```
public static void main(String[] args){  
    sapa();  
}
```

Gambar 7.4 Pemanggilan prosedur

Pemanggilan prosedur memerlukan sintak utama seperti gambar diatas, dan sapa(); merupakan nama prosedur yang dipanggil di sintak utama. Berikut merupakan kode lengkap dari penggunaan prosedur sapa:

```
static void sapa(){  
    System.out.println("Selamat di Jurusan Teknik Komputer");  
}  
  
public static void main(String[] args){  
    sapa();  
}
```

Gambar 7.5 Penggunaan dan pemanggilan prosedur

4. Deklarasi Nama Global dan Lokal Pada Prosedur

Deklarasi dalam prosedur memerlukan variabel atau peubah. Variabel merupakan penyimpanan data yang bersifat sementara di memori komputer (RAM) (Dianta, 2021). Seperti contoh dalam menghitung luas jajargenjang memerlukan variabel alas, tinggi dan L(luas). Jika deklarasi variabel dinyatakan dalam ruang lingkup global artinya variabel tersebut dapat digunakan dalam bagian manapun, bisa digunakan di dalam main program atau di dalam prosedur. Jika variabel tersebut digunakan dalam prosedur saja maka variabel tersebut dideklarasikan secara lokal karena hanya digunakan diprosedur tertentu saja.

5. Penerapan Parameter Prosedur dengan Bahasa Pemrograman Java

Parameter dapat diartikan sebagai nama variabel yang dideklarasikan pada bagian *header* prosedur. Parameter yang disertakan ketika pemanggilan prosedur disebut dengan parameter *actual* (*argument*). Parameter yang dideklarasikan di dalam bagian *header* prosedur disebut parameter formal. Penggunaan prosedur parameter formal dilakukan atau diakses dengan memanggil namanya dari program pemanggil disertakan parameter *actual*. Cara memanggilnya seperti contoh berikut:

NamaProsedur(Daftar Parameter Aktual)

Java merupakan bahasa pemrograman yang paling konsisten dalam mengimplementasikan paradigma pemrograman berorientasi objek (Rossa A.S, 2018). Dalam penerapan bahasa pemrograman java biasanya kode program dibungkus dengan *class*. *Class* adalah kumpulan obyek yang memiliki atribut dan *operation* yang sama (Munawar, 2018). Secara umum aplikasi dengan menerapkan bahasa pemrograman java selalu menampilkan kode dasar sebagai berikut:

```
public class namaclass {  
    public static void main(String args[]){  
    }  
}
```

Kode diatas merupakan kode dasar (*default*) dalam penulisan bahasa java dan ketika di *run* (eksekusi) maka tidak menghasilkan informasi (kosong).

Perbedaan umum antara prosedur dan fungsi adalah pada *return* (nilai balik), prosedur tidak memerlukan nilai balik, sedangkan fungsi memerlukan atau memberikan nilai balik, Berikut merupakan perbedaan kode program penggunaan fungsi (a) dan prosedur (b):

a. Kode Fungsi

```
Static int tambah(nama_fungsi)(int i, int j){  
x = i+j;  
return x;  
}
```

b. Kode prosedur

```
static void inData(){  
Scanner in = new Scanner(System.in);  
System.out.println("Masukkan Nilai 1:");  
int a = in.nextInt();  
System.out.println("Masukkan Nilai 2:");  
int b = in.nextInt();  
tambah();  
System.out.println("Hasil penjumlahan =" +x);  
}
```

Gambar 7.6 Kode fungsi dan prosedur

Pada penulisan fungsi tidak menggunakan *void* dan sintak utama diperlukan untuk pemanggilan fungsi dan prosedur. Seperti kode berikut:

```
public static void main(String[] args){  
inData();  
}
```

Contoh 1 penerapan prosedur dengan bahasa java tanpa parameter dapat dilihat pada gambar berikut:

```
public class belajarprosedur {  
    static void halo(){  
        System.out.println("Selamat datang di Jurusan Teknik  
Komputer");  
    }  
  
    public static void main(String[] args){  
        halo();  
    }  
}
```

Gambar 7.7 Prosedur dengan bahasa java tanpa parameter

Dari gambar diatas belajar prosedur adalah nama *class*, *halo()* merupakan nama prosedur dan ketika dijalankan akan mencetak tulisan **Selamat datang di Jurusan Teknik Komputer**.

Contoh 2 adalah penerapan prosedur dengan parameter, nilai merupakan nama parameter dengan tipe data *int* atau *integer*.

```
public void datanilai(int nilai){  
    System.out.println("Nilai = " + nilai);  
}
```

Gambar 7.8 Prosedur dengan bahasa java menggunakan parameter

Pada contoh diatas terdapat *int nilai*. *Int* merupakan tipe data, *nilai* merupakan parameter dan *datanilai* merupakan nama prosedur. Berikut merupakan **contoh 3** penerapan prosedur untuk menghitung luas jajargenjang:

```

public class jajargenjang {
    static void hitungLuas() {
        int a = 8;
        int t = 5;
        int L = a*t;
        System.out.println("Luas jajargenjang = "+L);
    }
    public static void main (String args[]) {
        hitungLuas();
    }
}

```

Gambar 7.9 Prosedur menghitung luas jajargenjang

Dari gambar 7.9 jajargenjang merupakan nama *class*, hitungLuas merupakan prosedur. Hasil kode tersebut adalah: **Luas jajargenjang =40.**

Berikut merupakan **contoh 4** penerapan prosedur dengan metode *input scanner*:

```

1  import java.util.Scanner;
2  public class prosedur1{
3      //deklarasi variabel global
4      public static double nilai_akhir=0;
5      public static char nilaiHuruf=' ';
6      //prosedur Hitung Nilai
7      public static void HitungNilai(){
8          Scanner in=new Scanner(System.in);
9          System.out.print("Nilai UTS : ");
10         double uts = in.nextDouble();
11         System.out.print("Nilai UAS : ");
12         double uas = in.nextDouble();
13         System.out.print("Nilai Tugas : ");
14         double tugas = in.nextDouble();
15         nilai_akhir=0.3*uts+0.5*uas+0.2*tugas;
16         System.out.printf("Nilai Akhir %.2f : \n", nilai_akhir);
17     }
18     //Prosedur Nilai Huruf
19     public static void NilaiHuruf(){
20         if (nilai_akhir>85) nilaiHuruf='A';
21         else if (nilai_akhir>70) nilaiHuruf='B';
22         else if (nilai_akhir>50) nilaiHuruf='C';
23         else if (nilai_akhir>30) nilaiHuruf='D';
24         else nilaiHuruf='E';
25         System.out.println("Nilai Huruf : "+nilaiHuruf);
26     }
27     //Program Utama
28     public static void main(String[] args){
29         HitungNilai();
30         NilaiHuruf();
31     }
32 }

```

Gambar 7.10 Prosedur dengan menerapkan metode *input*

Pada gambar 7.10 prosedur1 merupakan nama *class* dan dideklarasikan dengan variabel global. *HitungNilai* (baris ke 7) merupakan prosedur, *NilaiHuruf* (baris ke 19) juga merupakan prosedur, sedangkan pemanggilan prosedur terdapat pada baris 29 dan 30. Hasil dari kode diatas adalah sebagai berikut:

```
--- exec-maven-plugin:3.0.0:exec (default-cli) @ mavenproject1 ---
Nilai UTS : 90
Nilai UAS : 90
Nilai Tugas : 90
Nilai Akhir 90.00 :
Nilai Huruf : A
-----
BUILD SUCCESS
-----
Total time: 14.007 s
Finished at: 2022-04-11T11:34:05+07:00
-----
```

Gambar 7.11 Luaran prosedur dengan menerapkan metode *input*

Output kode program menghasilkan Nilai Akhir dan Nilai Huruf berdasarkan *input* Nilai UTS, Nilai UAS dan Nilai Tugas.

B. FUNCTION

1. Pendahuluan

Pembahasan tentang *function* atau fungsi hampir sama dengan prosedur. *Function* juga merupakan salah satu bentuk program yang memiliki satu atau lebih parameter sebagai masukan. Beberapa materi yang akan dibahas adalah definisi *function*, pemanggilan fungsi dan penerapan fungsi dengan bahasa pemrograman java.

2. Definisi Function

Function atau fungsi adalah upa-program yang menerima data masukan dan memberikan/mengembalikan (*return*) sebuah nilai dari tipe tertentu (tipe dasar atau tipe bentukan) (Munir & Lidya, 2016). Penerapan fungsi dibidang matematika sama dengan definisi di dalam program, berikut merupakan penerapan fungsi di dalam matematika:

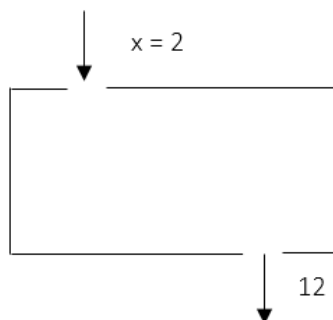
- a. $F(x) = 3x^2 + 4x - 8$
- b. $F(x) = 2x^2 + 6x - 4$
- c. $F(x,y) = 2x - y + xy$

Penyelesaian:

- a. a. $F(x) = 3x^2 + 4x - 8$, misalnya $x = 2$
 $x = 2$, maka
 $F(2) = 3.2^2 + 4.2 - 8 = 12 + 8 - 8 = 12$
- b. $F(x) = 2x^2 + 6x - 4$, misalnya $x = 1$
 $x = 1$, maka
 $F(1) = 2.1^2 + 6.1 - 4 = 2 + 6 - 4 = 4$
- c. $F(x,y) = 2x - y + xy$, misalnya $x = 1$, $y = 2$
 maka
 $F(1,2) = 2.1 - 2 + 1.2 = 2$

Pada contoh a,b dan c, F merupakan fungsi sedangkan x dan y adalah *input* atau masukan. Nilai x dan y tergantung data yang dimasukkan.

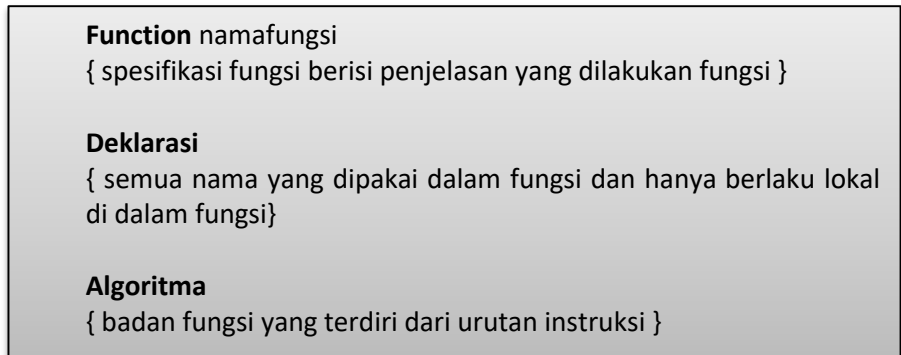
Berdasarkan contoh fungsi bagian a, maka dapat digambarkan contoh fungsi sebagai berikut:



Gambar 7.12 Diagram Fungsi $x = 2$

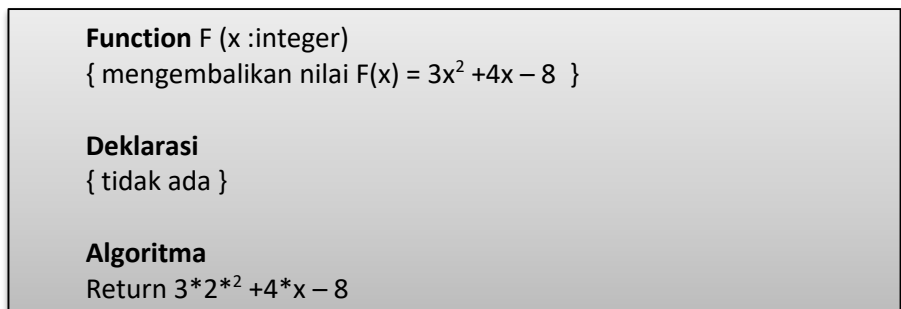
Dari contoh $F(x) = 3x^2 + 4x - 8$, fungsi menerima masukan $x = 2$ dan menghasilkan luaran $F(2) = 12$.

Fungsi memiliki struktur yang terdiri dari *header*, deklarasi dan badan fungsi (algoritma). Bagian *header* merupakan nama dari sebuah *function*, sedangkan deklarasi merupakan nama *function* yang hanya dipakai didalam *function* itu sendiri, dan badan fungsi merupakan sekumpulan instruksi-instruksi yang digunakan dalam *function*. Berikut merupakan struktur dari fungsi:



Gambar 7.13 Struktur Fungsi

Struktur fungsi terdiri dari nama fungsi, deklarasi dan algoritma (badan fungsi), semua variabel hanya berlaku didalam fungsi dan terletak dibagian deklarasi. Berikut merupakan contoh dari penerapan fungsi $F(x) = 3x^2 + 4x - 8$:



Gambar 7.14 Struktur Fungsi $F(x) = 3x^2 + 4x - 8$

Return berfungsi untuk mengembalikan nilai yang dihasilkan, nilai tersebut dapat berupa ekspresi atau konstanta.

3. Pemanggilan Fungsi

Pemanggilan fungsi dilakukan untuk menghasilkan nilai dengan melakukan pemanggilan nama fungsi dari program tersebut. Penerapan fungsi dapat dilihat dari contoh berikut:

```
Void nama_fungsi(parameter1, parameter2,...){  
    Statement_yang_akan_dilakukan;  
}
```

Gambar 7.15 Struktur Fungsi secara umum

Diatas merupakan bentuk fungsi secara umum dan pemanggilan fungsi dapat dilihat dari contoh berikut:

```
Nama_fungsi(parameter1, parameter2,...);
```

Gambar 7.16 Pemanggilan Fungsi

Pemanggilan fungsi dilakukan dengan menuliskan nama fungsi beserta parameternya.

4. Penerapan Fungsi Dengan Bahasa Pemrograman Java

Java merupakan bahasa pemrograman yang menerapkan pemrograman berorientasi objek. Pemrograman dengan bahasa java sudah menyediakan pemeriksaan *debugging*. *Debugging* adalah suatu aktivitas yang ditujukan untuk menemukan penyebab kesalahan suatu program (Kadir, 2019). Pada pemrograman java *function* merupakan bagian dari program utama. Penerapan koding fungsi dapat dilihat sebagai berikut:

```
class NamaClass {  
    static tipeData namaFunction() {  
        // Isi function disini...  
        return nilai;  
    }  
}
```

```

public static void main(String args[]){
    // Jalankan function
    namaFunction()
}
}

```

Berikut merupakan **contoh 1** penerapan fungsi dengan bahasa pemrograman java:

```

public class jajargenjang {
    static int hitungLuas() {
        int a = 8;
        int t = 6;
        int L = a*t;
        return L;
    }

    public static void main (String args[]) {
        System.out.println("Luas jajargenjang = "+hitungLuas());
    }
}

```

Gambar 7.17 Fungsi hitungLuas

Pada gambar 7.17 jajargenjang merupakan nama *class*, hitungLuas adalah fungsi (*function*), int merupakan tipe data yang menyatakan angka pada variabel a, t dan L. Tipe data adalah sebuah pengklasifikasian data berdasarkan jenis datanya (Sulasmoro, 2022). Hasil dari kode tersebut adalah: **Luas jajargenjang =48.**

Pada **contoh 2** kita akan mencari luas kubus dengan cara memasukkan rumus berikut:

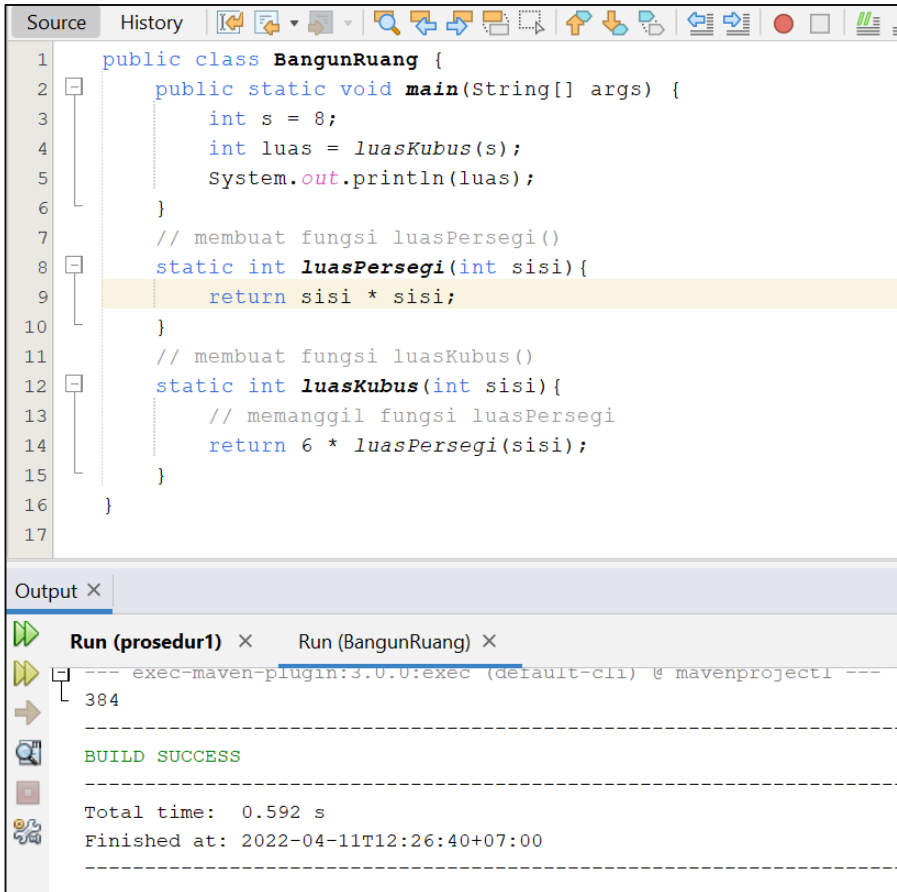
```

Luas Persegi = sisi * sisi
Luas Kubus = 6 * luasPersegi

```

Gambar 7.18 Rumus Luas Kubus

Dari rumus diatas kita akan membuat fungsi luas kubus seperti contoh berikut:



```
1 public class BangunRuang {
2     public static void main(String[] args) {
3         int s = 8;
4         int luas = luasKubus(s);
5         System.out.println(luas);
6     }
7     // membuat fungsi luasPersegi()
8     static int luasPersegi(int sisi){
9         return sisi * sisi;
10    }
11    // membuat fungsi luasKubus()
12    static int luasKubus(int sisi){
13        // memanggil fungsi luasPersegi
14        return 6 * luasPersegi(sisi);
15    }
16 }
17
```

Output X

Run (prosedur1) X Run (BangunRuang) X

exec-maven-plugin:3.0.0:exec (default-cli) @ mavenproject1

384

BUILD SUCCESS

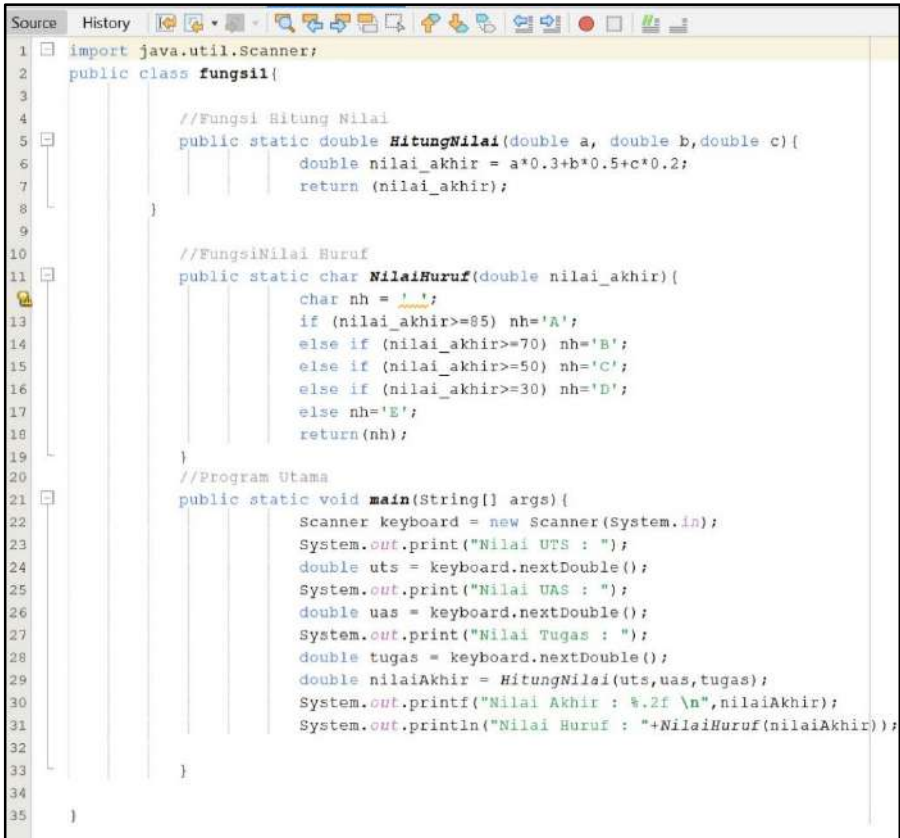
Total time: 0.592 s

Finished at: 2022-04-11T12:26:40+07:00

Gambar 7.19 Penerapan Fungsi luas kubus

BangunRuang adalah nama *class* dari program diatas, untuk mendapatkan luas kubus kita harus membuat fungsi LuasPersegi lalu membuat fungsi luasKubus, dan fungsi tersebut dipanggil dengan melakukan *return* 6 x luasPersegi(sisi). Dari koding tersebut akan menghasilkan *output*: **384**.

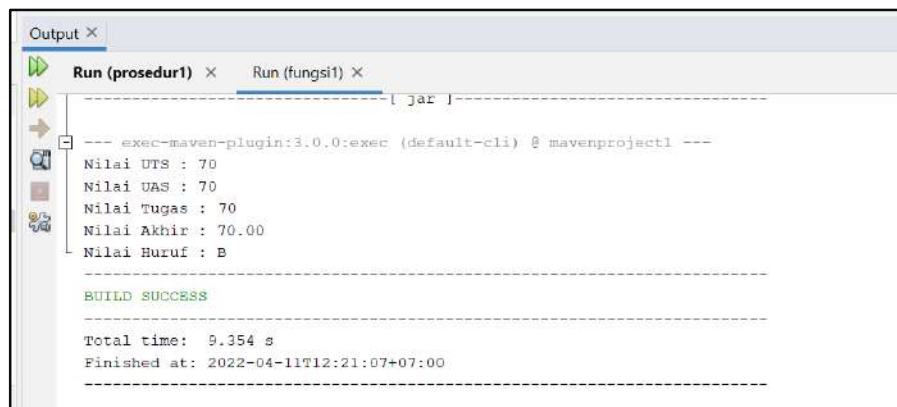
Berikut merupakan **contoh 3** penerapan fungsi untuk mencari nilai akhir dan *grade* (nilai huruf):

The image shows a screenshot of a Java IDE with a source code editor. The code is written in Java and defines a class named 'fungsi1'. It contains two static methods: 'HitungNilai' which calculates a weighted average of three inputs (a, b, c) using the formula $a \times 0.3 + b \times 0.5 + c \times 0.2$, and 'NilaiHuruf' which maps a numerical grade to a letter grade (A, B, C, D, E) based on specific thresholds. A 'main' method is also present, which uses a 'Scanner' to take user input for three scores, calls the 'HitungNilai' function to calculate the final score, and then calls the 'NilaiHuruf' function to determine the corresponding letter grade. The code is well-commented and uses color-coded syntax highlighting.

```
1 import java.util.Scanner;
2 public class fungsi1{
3
4     //Fungsi Hitung Nilai
5     public static double HitungNilai(double a, double b, double c){
6         double nilai_akhir = a*0.3+b*0.5+c*0.2;
7         return (nilai_akhir);
8     }
9
10    //FungsiNilai Huruf
11    public static char NilaiHuruf(double nilai_akhir){
12        char nh = ' ';
13        if (nilai_akhir>=85) nh='A';
14        else if (nilai_akhir>=70) nh='B';
15        else if (nilai_akhir>=50) nh='C';
16        else if (nilai_akhir>=30) nh='D';
17        else nh='E';
18        return (nh);
19    }
20    //Program Utama
21    public static void main(String[] args){
22        Scanner keyboard = new Scanner(System.in);
23        System.out.print("Nilai UTS : ");
24        double uts = keyboard.nextDouble();
25        System.out.print("Nilai UAS : ");
26        double uas = keyboard.nextDouble();
27        System.out.print("Nilai Tugas : ");
28        double tugas = keyboard.nextDouble();
29        double nilaiAkhir = HitungNilai(uts,uas,tugas);
30        System.out.printf("Nilai Akhir : %.2f \n",nilaiAkhir);
31        System.out.println("Nilai Huruf : "+NilaiHuruf(nilaiAkhir));
32    }
33 }
34
35 }
```

Gambar 7.20 Penerapan Fungsi mencari nilai akhir

Sebelumnya penerapan program untuk mencari nilai akhir sudah dibahas di contoh prosedur. Gambar 7.20 merupakan studi kasus yang sama dengan prosedur sebelumnya, hanya saja dicontoh ini kode program menggunakan prosedur diganti dengan *function*. Hasil dari kode tersebut adalah sebagai berikut:



```
Output X
Run (prosedur1) X Run (fungsi1) X
[ jar ]
--- exec-maven-plugin:3.0.0:exec (default-cli) @ mavenproject1 ---
Nilai UTS : 70
Nilai UAS : 70
Nilai Tugas : 70
Nilai Akhir : 70.00
Nilai Huruf : B
BUILD SUCCESS
Total time: 9.354 s
Finished at: 2022-04-11T12:21:07+07:00
```

Gambar 7.21 Hasil Fungsi mencari nilai akhir

Dari kode tersebut dihasilkan *output* berdasarkan gambar 7.21 dengan *input* yang dimasukkan berdasarkan Nilai UTS, Nilai UAS dan Nilai Tugas.

C. RANGKUMAN MATERI

Prosedur digunakan untuk fungsi yang tidak mengembalikan nilai sedangkan fungsi biasanya ditandai untuk mengembalikan nilai. Prosedur dan fungsi memiliki struktur yaitu bagian judul (*header*), bagian deklarasi dan bagian algoritma (badan prosedur). Penggunaan prosedur dan fungsi dilakukan dengan menuliskan nama prosedur atau nama fungsi dan diikuti dengan parameter. Penggunaan prosedur tanpa parameter dilakukan dengan menuliskan nama prosedur dan diikuti tanda kurung() tanpa parameter didalamnya.

TUGAS DAN EVALUASI

Jawablah pertanyaan berikut dengan benar dan tepat

1. Jelaskan struktur dari prosedur dan fungsi ?
2. Apa yang Anda ketahui tentang parameter pada prosedur dan fungsi ?
3. Jelaskan cara memanggil prosedur dan fungsi ?
4. Buatlah contoh program menggunakan *function* dengan menerapkan metode *input scanner* ?
5. Buatlah contoh program menggunakan *function* dengan menerapkan metode *input* selain *scanner*?

DAFTAR PUSTAKA

- Dianta, I. A. (2021). *Logika Dan Algoritma Untuk Merancang Aplikasi Komputer*. Semarang: YPAT.
- Kadir, A. (2019). *Logika Pemrograman Menggunakan C++*. Jakarta: Elex Media Komputindo.
- Munawar. (2018). *Analisis Perancangan Sistem Berorientasi Objek Dengan UML*. Bandung: Informatika.
- Rinaldi Munir, L. L. (2016). *Algoritma dan Pemrograman dalam Bahasa Pascal, C dan C++*. Bandung: Informatika.
- Rossa A.S, M. S. (2018). *Rekayasa Perangkat Lunak Terstruktur Dan Berorientasi Objek*. Bandung: Informatika.
- Sitorus, L. (2015). *Algoritma dan Pemrograman* . Yogyakarta: Andi.
- Sulasmoro, A. H. (2022). *Buku Ajar Algoritma dan Pemrograman I*. NTB: Pusat Pengembangan Pendidikan dan Penelitian Indonesia.



SORTING DAN SEARCHING PADA ARRAY

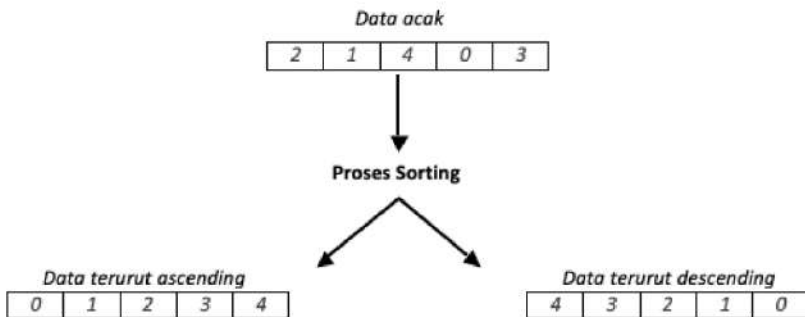
Fadhila Tangguh Admojo, M.Cs
Institut Teknologi dan Bisnis Palcomtech

A. PENDAHULUAN

Seiring berjalannya waktu, jumlah data dan informasi yang dapat disimpan dan diakses melalui komputer telah berubah menjadi struktur data yang sangat besar. Begitu banyak teknik dan algoritma telah dikembangkan untuk memelihara dan memproses informasi secara efisien. Penyortiran dan pencarian merupakan tugas dasar yang perlu dilakukan di banyak program komputer, oleh karena itu *sorting* dan *searching* menjadi salah satu topik terpenting dalam algoritma dan struktur data dan menjadi bidang studi paling mendasar dalam ilmu komputer. Bab ini membahas secara spesifik cara kerja algoritma *sorting* dan *searching* pada *array* beserta contoh implementasi dan penelusuran *output* yang dihasilkan.

B. SORTING

Sorting adalah proses pengurutan sejumlah data yang sebelumnya acak atau tidak teratur menjadi teratur berdasarkan suatu aturan tertentu. Bentuk pengurutan data dapat dibedakan menjadi dua yaitu pengurutan secara *ascending* (pengurutan dari data yang terkecil hingga data yang terbesar) dan pengurutan secara *descending* (pengurutan dari data yang terbesar hingga data yang terkecil). data *ascending* dan *descending* divisualisasikan pada gambar 8.1



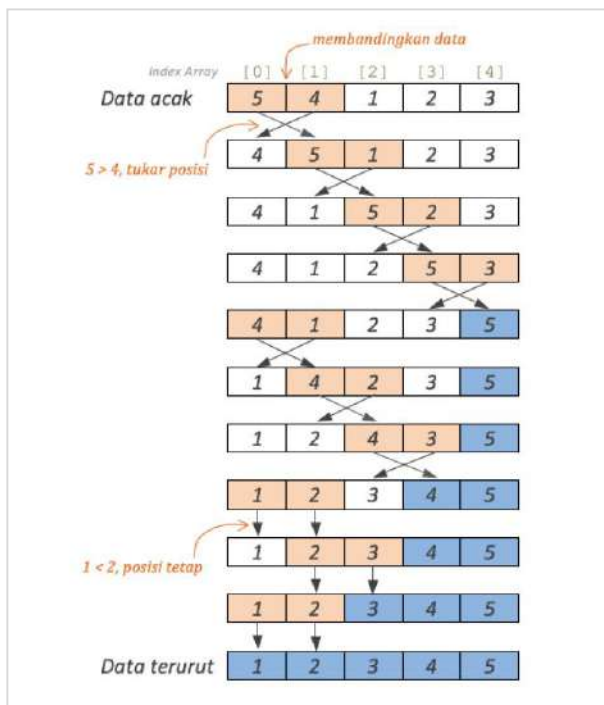
Gambar 8.1 Pengurutan *ascending* dan *desending*

Teknik *sorting* yang umum digunakan adalah dengan membandingkan (*comparing*) dan memindahkan (*swapping*) posisi suatu elemen data mengikuti aturan-aturan tertentu, teknik ini disebut sebagai *comparison-based sorting*. *Comparison-based sorting* dapat dikelompokkan ke dalam dua pendekatan yaitu *brute force* dan *divide and conquer*. *Brute force* mencoba segala kemungkinan untuk mendapatkan solusi menggunakan suatu aturan sistematis tertentu tanpa analisis mendalam dan biasanya tanpa mempertimbangkan efisiensi baik secara waktu maupun *resource* (sumber daya). Sedangkan *Divide and Conquer* mencari solusi dengan membagi/memecah (*devide*) masalah menjadi beberapa bagian yang lebih kecil (*sub-problem*), kemudian menyelesaikan (*conquer*) masalah pada *sub-problem* secara independen hingga solusi akhir didapatkan dari penggabungan (*merging*) solusi yang didapat dari masing-masing *sub problem*. Secara umum *divide and conquer* menghasilkan solusi dengan menggunakan tiga langkah pendekatan yaitu memecah masalah menjadi

sekecil mungkin, menyelesaikan masalah dari bagian yang terkecil, kemudian menggabungkan kembali setiap bagian untuk mendapat solusi akhir yang paling optimal. Algoritma *sorting* yang menggunakan pendekatan *brute force* yaitu *Bubble sort*, *Selection sort* dan *Insertion sort*. Sedangkan algoritma *sorting* yang termasuk dalam kategori *divide and conquer* adalah *Merge sort* dan *Quicksort*.

1. *Bubble Sort*

Bubble Sort merupakan algoritma *sorting* sederhana yang bekerja dengan cara bergerak berulang dari awal hingga akhir data untuk membandingkan semua elemen yang saling bersebelahan. Selama melakukan perbandingan jika ditemukan urutan elemen data yang tidak sesuai berdasarkan kondisi tertentu maka tukar ke dua posisi elemen data tersebut. Ilustrasi cara kerja algoritma *Bubble Sort* dijabarkan pada Gambar 8.2.



Gambar 8.2 Ilustrasi cara kerja algoritma *Bubble Sort*

Sebagai contoh pengurutan data *array* {5, 4, 1, 2, 3} berdasarkan nilai terkecil. Cara kerja algoritma *bubble sort* dimulai dari elemen pada *index*[0] yaitu 5 dibandingkan dengan elemen pada *index*[1] yaitu 4, karena 5 lebih besar dari 4 maka tukar posisi 5 dengan 4 sehingga data *array* berubah menjadi {4, 5, 1, 2, 3}. Perbandingan selanjutnya terhadap *index*[1] yaitu 5 dengan *index*[2] yaitu 1, karena 5 lebih besar tukar posisi 5 dengan 1 sehingga *array* berubah menjadi {4, 1, 5, 2, 3}. Proses yang sama dilakukan secara berulang hingga semua elemen dibandingkan dan data *array* terurut menjadi {1, 2, 3, 4, 5}.

Gambar 8.3 merupakan implementasi algoritma *Bubble* yang dideskripsikan pada *class BubbleSort* (baris 3). Data *array* acak bertipe *integer* yang akan urutkan diinisialisasi pada *method main* (baris 33), sedangkan proses *sorting* dilakukan di dalam *method sort* (baris 4) menggunakan perulangan bersarang (baris 9-22). Baris 9 merupakan perulangan luar untuk melakukan penelusuran sebanyak elemen *array* dimulai dari elemen data paling kiri yaitu pada *index*[0], sedangkan Baris 11 adalah perulangan dalam untuk proses perbandingan elemen. Kondisi untuk perbandingan dua elemen *array* terdapat pada baris 15, yang mana pertukaran posisi elemen (baris 17-19) hanya dilakukan jika kondisi terpenuhi yaitu elemen kiri lebih besar dari kanan. Apabila kondisi tidak terpenuhi maka pertukaran posisi elemen tidak dilakukan, selanjutnya proses diteruskan ke elemen berikutnya hingga semua elemen dibandingkan. Program berhenti ketika perulangan (baris 9) telah selesai. *Output* tahapan *sorting* dengan algoritma *bubble sort* dapat dilihat pada Gambar 8.5a.

```

3 public class BubbleSort { //implementasi algoritma bubble sort
4     public static void sort(int[] data) {
5         int banyak = data.length;
6
7         System.out.print("\nData sebelum sorting : ");
8         cetak(data); //cetak data
9         for (int ulang=0; ulang<banyak-1; ulang++) { //perulangan
10             System.out.print("\nIterasi ke-"+(ulang+1));
11             for (int index=1; index<banyak-ulang; index++) { //perulangan
12                 System.out.print("\n"+(ulang+1)+"."+index+" [ ");
13                 cetak(data);
14                 System.out.print("] cek "+data[index-1]+" > "+data[index]+" ");
15                 if (data[index-1] > data[index]) { //elemen kiri < elemen kanan
16                     System.out.print("tukar "+data[index-1]+" & "+ data[index]);
17                     int temp = data[index-1]; // pertukaran data */
18                     data[index-1] = data[index]; // pertukaran data */
19                     data[index] = temp; // pertukaran data */
20                 }
21             }
22         }
23         System.out.print("\n\nData setelah sorting : ");
24         cetak(data);
25     }
26
27     public static void cetak(int[] data) {
28         for (int x=0; x<data.length; x++) {
29             System.out.print(data[x]+" ");
30         }
31     }
32
33     public static void main(String[] args) {
34         int[] data = {5,4,1,2,3}; //data array acak
35         sort(data);
36     }
37 }

```

Gambar 8.3 Implementasi algoritma *Bubble Sort*

Berdasarkan cara kerjanya *Bubble Sort* merupakan algoritma *sorting* yang sangat sederhana untuk dipahami dan diimplementasikan, akan tetapi jika diperhatikan secara seksama pada Gambar 8.2 selama melakukan perbandingan data terdapat kemungkinan elemen yang sudah dibandingkan dan sudah benar posisinya akan terus ikut dibandingkan, hal ini menyebabkan inefisiensi ketika *bubble sort* digunakan untuk mengurutkan *array* dengan elemen data yang banyak. Modifikasi algoritma *bubble sort* untuk mengurangi inefisiensi dapat dilakukan pada *method sort* (Gambar 8.3 baris 4) dengan memberikan penanda status pada saat perbandingan dan pertukaran elemen data (Gambar 8.3 baris 15), sehingga dengan adanya penanda tersebut perulangan (Gambar 8.3 baris 9) dapat membedakan elemen yang sudah benar posisinya dengan elemen yang masih perlu untuk dilakukan perbandingan.

Modifikasi *method sort* dijabarkan pada Gambar 8.4. Penanda status dilakukan dengan menggunakan variabel `boolean status=true;` (baris 6). Selama `status==true` perulangan (baris 10) akan terus dilakukan. Perubahan `status==true` menjadi `status==false` dilakukan sebelum program menjalankan perbandingan elemen (baris 12-13), perubahan kembali dari `status==false` menjadi `status==true` terjadi jika kondisi (baris 17) terpenuhi namun apabila kondisi (baris 17) tidak terpenuhi status akan

tetap `false` yang artinya elemen sudah berada diposisi yang tepat dan perulangan (baris 10) tidak dilakukan. *Output* modifikasi *Bubble Sort* dapat dilihat pada Gambar 8.5b.

```

3  public class BubbleSortMod { //implementasi algoritma bubble sort
4      public static void sort(int[] data) {
5          int banyak = data.length;
6          boolean status = true;
7
8          System.out.print("\nData sebelum sorting : ");
9          cetak(data); //cetak data
10         for (int ulang=0;ulang<(banyak-1)&&status;ulang++) { //perulangan luar
11             System.out.print("\n\nIterasi ke-"+(ulang+1));
12             status=false;
13             for (int index=1;index<banyak-ulang;index++) { //perulangan dalam
14                 System.out.print("\n"+(ulang+1)+". "+index+"  ");
15                 cetak(data);
16                 System.out.print("  cek "+data[index-1]+" > "+data[index]+" ");
17                 if(data[index-1] > data[index]) { //kondisi elemen kiri < elemen kanan
18                     System.out.print("tukar "+data[index-1]+" & "+ data[index]);
19                     int temp = data[index-1]; /* pertukaran data */
20                     data[index-1] = data[index]; /* pertukaran data */
21                     data[index]= temp; /* pertukaran data */
22                     status=true;
23                 }
24             }
25         }
26         System.out.print("\n\nData setelah sorting : ");
27         cetak(data);
28     }
29
30     public static void cetak(int[] data) {
31         for (int x=0;x<data.length;x++) {
32             System.out.print(data[x]+" ");
33         }
34     }
35
36     public static void main(String[] args) {
37         int[] data = {5,4,1,2,3}; //data array acak
38         sort(data);
39     }
40 }

```

Gambar 8.4 Modifikasi *method Short*

Data sebelum sorting : 5 4 1 2 3

Iterasi ke-1

1.1 [5 4 1 2 3] cek 5 > 4, tukar 5 & 4
1.2 [4 5 1 2 3] cek 5 > 1, tukar 5 & 1
1.3 [4 1 5 2 3] cek 5 > 2, tukar 5 & 2
1.4 [4 1 2 5 3] cek 5 > 3, tukar 5 & 3

Iterasi ke-2

2.1 [4 1 2 3 5] cek 4 > 1, tukar 4 & 1
2.2 [1 4 2 3 5] cek 4 > 2, tukar 4 & 2
2.3 [1 2 4 3 5] cek 4 > 3, tukar 4 & 3

Iterasi ke-3

3.1 [1 2 3 4 5] cek 1 > 2,
3.2 [1 2 3 4 5] cek 2 > 3,

Iterasi ke-4

4.1 [1 2 3 4 5] cek 1 > 2,

Data setelah sorting : 1 2 3 4 5

Data sebelum sorting : 5 4 1 2 3

Iterasi ke-1

1.1 [5 4 1 2 3] cek 5 > 4, tukar 5 & 4
1.2 [4 5 1 2 3] cek 5 > 1, tukar 5 & 1
1.3 [4 1 5 2 3] cek 5 > 2, tukar 5 & 2
1.4 [4 1 2 5 3] cek 5 > 3, tukar 5 & 3

Iterasi ke-2

2.1 [4 1 2 3 5] cek 4 > 1, tukar 4 & 1
2.2 [1 4 2 3 5] cek 4 > 2, tukar 4 & 2
2.3 [1 2 4 3 5] cek 4 > 3, tukar 4 & 3

Iterasi ke-3

3.1 [1 2 3 4 5] cek 1 > 2,
3.2 [1 2 3 4 5] cek 2 > 3,

Data setelah sorting : 1 2 3 4 5

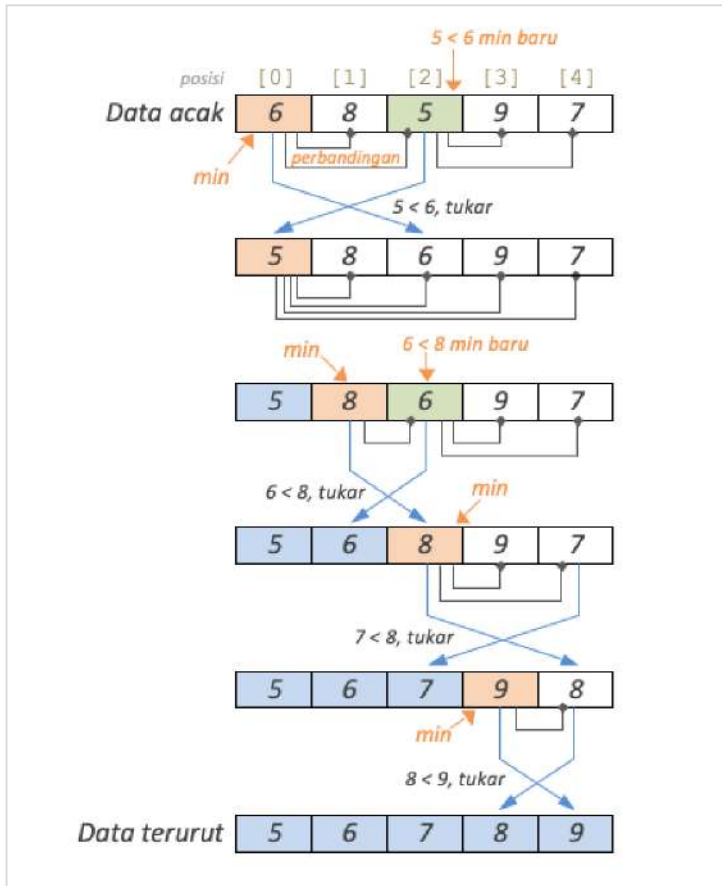
(a)

(b)

Gambar 8.5 (a) Output BubbleSort, (b) Output BubbleSortMod

2. Selection Sort

Selection Sort termasuk algoritma *sorting* sederhana yang bekerja dengan cara membandingkan semua elemen data untuk mencari nilai terkecil/terbesar disetiap perulangan, kemudian menempatkan nilai terkecil/terbesar diposisi yang tepat hingga semua elemen terurut. Sebagai contoh pada pengurutan *array* {6, 8, 5, 9, 7} *Selection Sort* dimulai dengan menentukan elemen pertama yaitu 6 sebagai nilai min (minimum) kemudian membandingkan 6 dengan 8, karena $8 > 6$ maka nilai min tetap 6, selanjutnya bandingkan 6 dengan 5 karena $5 < 6$ maka set 5 sebagai nilai min yang baru, kemudian bandingkan 5 dengan 9, karena $9 > 5$ maka nilai min tetap 5, selanjutnya bandingkan 5 dengan 7, karena $7 > 5$ maka nilai min tetap 5. Setelah perulangan pertama selesai nilai min adalah 5 maka tukarkan 5 dengan 6 sebagai elemen pertama *array* sehingga urutan data menjadi {5, 8, 6, 9, 7}. Pada perulangan kedua 5 tidak lagi dibandingkan maka set 8 sebagai nilai min dan lanjutkan perbandingan data hingga semua elemen data terurut. Cara kerja algoritma *Selection Sort* diilustrasikan pada Gambar 8.6.



Gambar 8.6 Ilustrasi algoritma Selection Sort

Gambar 8.7 menjabarkan deklarasi dari *class SelectionSort* (baris 3). Proses *sorting* terhadap data *array* (baris 40) dilakukan di dalam *method sort* (baris 4-31). Variabel *min* (baris 8) digunakan untuk menampung nilai minimum dan variabel *pos* (baris 7) untuk menandai posisi nilai minimum. Selama proses *sorting method sort* menggunakan perulangan bersarang (baris 6-30) untuk menelusuri (baris 6) dan membandingkan semua elemen data (baris 11-20), sedangkan untuk pertukaran data dilakukan pada baris 22-29. Perubahan nilai *min* dan *pos* untuk setiap iterasi,

kemudian proses pertukaran dapat pada setiap iterasi dapat dilihat pada Gambar 8.8.

```
3 public class SelectionSort {
4     public static void sort(int[] data) {
5         //perulangan sebanyak data
6         for(int ulang=0;ulang<data.length-1;ulang++) {
7             int pos=ulang; //posisi tukar
8             int min=data[ulang]; //nilai minimum
9             System.out.print("\n\nMin: "+min+", Pos : "+pos);
10            //perulangan membandingkan data
11            for(int index=ulang+1;index<data.length;index++) {
12                System.out.print("\nIterasi-"+(ulang+1)+"-"+index+" [ ");
13                cetak(data);
14                System.out.print("] cek "+data[index]+" < "+min);
15                if(data[index] < min) { //kondisi untuk membandingkan data
16                    min=data[index];
17                    pos=index;
18                    System.out.print(", Min: "+min+", Pos: "+pos);
19                }
20            }
21        }
22        if(pos!=ulang) { //kondisi pertukaran data
23            int tmp=data[pos];
24            data[pos]=data[ulang];
25            data[ulang]=tmp;
26            System.out.print("\ntukar "+data[pos]+" & "+data[ulang]+" : [ ");
27            cetak(data);
28            System.out.print("]");
29        }
30    }
31 }
32
33 public static void cetak(int[] data) {
34     for(int i=0;i<data.length;i++) {
35         System.out.print(data[i]+" ");
36     }
37 }
38
39 public static void main(String[] args) {
40     int[] data = {6,8,5,9,7};
41     System.out.print("\nData acak : "); cetak(data);
42     sort(data);
43     System.out.print("\n\nData urut : "); cetak(data);
44 }
45 }
```

Gambar 8.7 Deklarasi class SelectionSort

```

Data acak : 6 8 5 9 7

Min: 6, Pos : 0
Iterasi-1.1 [ 6 8 5 9 7 ] cek 8 < 6
Iterasi-1.2 [ 6 8 5 9 7 ] cek 5 < 6, Min: 5, Pos: 2
Iterasi-1.3 [ 6 8 5 9 7 ] cek 9 < 5
Iterasi-1.4 [ 6 8 5 9 7 ] cek 7 < 5
tukar 6 & 5: [ 5 8 6 9 7 ]

Min: 8, Pos : 1
Iterasi-2.2 [ 5 8 6 9 7 ] cek 6 < 8, Min: 6, Pos: 2
Iterasi-2.3 [ 5 8 6 9 7 ] cek 9 < 6
Iterasi-2.4 [ 5 8 6 9 7 ] cek 7 < 6
tukar 8 & 6: [ 5 6 8 9 7 ]

Min: 8, Pos : 2
Iterasi-3.3 [ 5 6 8 9 7 ] cek 9 < 8
Iterasi-3.4 [ 5 6 8 9 7 ] cek 7 < 8, Min: 7, Pos: 4
tukar 8 & 7: [ 5 6 7 9 8 ]

Min: 9, Pos : 3
Iterasi-4.4 [ 5 6 7 9 8 ] cek 8 < 9, Min: 8, Pos: 4
tukar 9 & 8: [ 5 6 7 8 9 ]

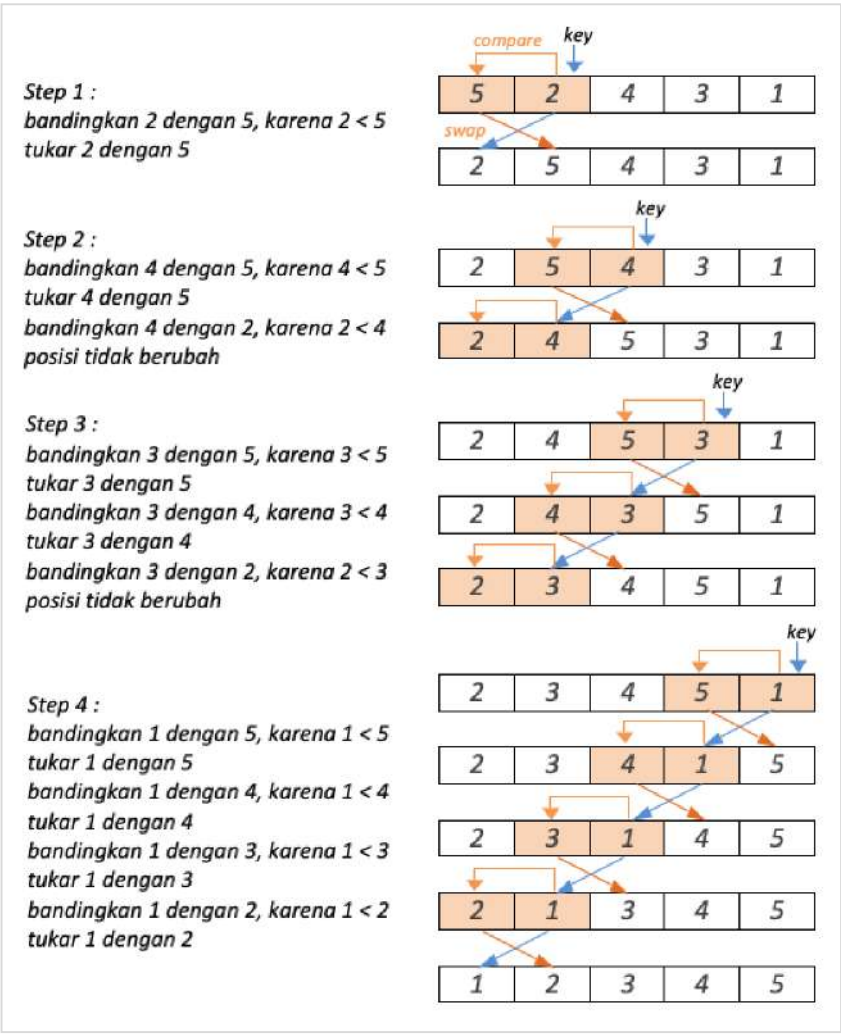
Data urut : 5 6 7 8 9

```

Gambar 8.8 Output proses *sorting Selection Sort*

3. *Insertion Sort*

Insertion sort termasuk ke dalam algoritma *sorting* sederhana yang dimulai dari elemen kedua pada *array* dan membandingkannya dengan elemen pertama, apabila elemen kedua lebih kecil dari elemen pertama maka tukarkan posisi ke dua *element* tersebut, tahapan yang sama dilakukan berulang pada elemen-elemen selanjutnya hingga semua elemen terurut. Cara kerja *Insertion Sort* diilustrasikan pada Gambar 8.9.



Gambar 8.9 Ilustrasi algoritma *Insertion Sort*

Implementasi dari algoritma *Insertion Sort* dijabarkan pada Gambar 8.10. Proses *sorting* dilakukan di dalam *method sort* (baris 4). Step dimulai (baris 9) dengan mengeset elemen kedua *array* pada variabel *key* (baris 10) dan mengeset variabel *index* sebagai penampung pergerakan posisi elemen-elemen yang akan dibandingkan (baris 11). Iterasi perbandingan

elemen dilakukan dengan membandingkan setiap elemen pada posisi *index* dengan *key* (baris 15), apabila elemen pada posisi *index* lebih besar dari *key* maka tukar posisi ke duanya (baris 16-21). Step kedua sama dengan step pertama dimulai dengan mengeset nilai *key* dan nilai *index* sehingga nilai *key* dan *index* akan terus bergerak hingga semua elemen dibandingkan. *Output* dari proses *sorting* dengan algoritma *Insertion Sort* dapat dilihat pada Gambar 8.11.

```

3  public class InsertionSort {
4      public static void sort(int[] data) {
5          int key;
6          int step;
7          int index;
8
9          for(step=1; step<data.length; step++) {
10             key = data[step];
11             index = step;
12             System.out.println("\n\nStep ke "+step);
13             System.out.println("key: "+key);
14             cetak(data);
15             while ((index>0) && (data[index-1] > key)) {
16                 data[index] = data[index-1];
17                 System.out.print(" bandingkan "+key+" dengan "+data[index-1]+
18                     " "+key+"<"+data[index-1]+")");
19                 index = index - 1;
20                 System.out.println(" -> tukar "+key+" dengan "+data[index]);
21                 data[index] = key;
22                 cetak(data);
23             }
24             if (index>0) {
25                 System.out.print(" bandingkan "+key+" dengan "+data[index-1]+
26                     " "+key+">"+data[index-1]+") -> tetap");
27             }
28         }
29     }
30
31     public static void cetak(int data[]) {
32         for(int i=0;i<data.length;i++) {
33             System.out.print(data[i]+" ");
34         }
35     }
36
37     public static void main(String[] args) {
38         int[] data = {5,2,4,3,1};
39         System.out.print("Data awal : "); cetak(data);
40         sort(data);
41         System.out.print("\n\nData urut : "); cetak(data);
42     }
43 }

```

Gambar 8.10 Implementasi *Insertion Sort*

```

Data awal : 5 2 4 3 1

Step ke 1
key: 2
5 2 4 3 1   bandingkan 2 dengan 5 ( $2 < 5$ ) -> tukar 2 dengan 5
2 5 4 3 1

Step ke 2
key: 4
2 5 4 3 1   bandingkan 4 dengan 5 ( $4 < 5$ ) -> tukar 4 dengan 5
2 4 5 3 1   bandingkan 4 dengan 2 ( $4 > 2$ ) -> tetap

Step ke 3
key: 3
2 4 5 3 1   bandingkan 3 dengan 5 ( $3 < 5$ ) -> tukar 3 dengan 5
2 4 3 5 1   bandingkan 3 dengan 4 ( $3 < 4$ ) -> tukar 3 dengan 4
2 3 4 5 1   bandingkan 3 dengan 2 ( $3 > 2$ ) -> tetap

Step ke 4
key: 1
2 3 4 5 1   bandingkan 1 dengan 5 ( $1 < 5$ ) -> tukar 1 dengan 5
2 3 4 1 5   bandingkan 1 dengan 4 ( $1 < 4$ ) -> tukar 1 dengan 4
2 3 1 4 5   bandingkan 1 dengan 3 ( $1 < 3$ ) -> tukar 1 dengan 3
2 1 3 4 5   bandingkan 1 dengan 2 ( $1 < 2$ ) -> tukar 1 dengan 2
1 2 3 4 5

Data urut : 1 2 3 4 5

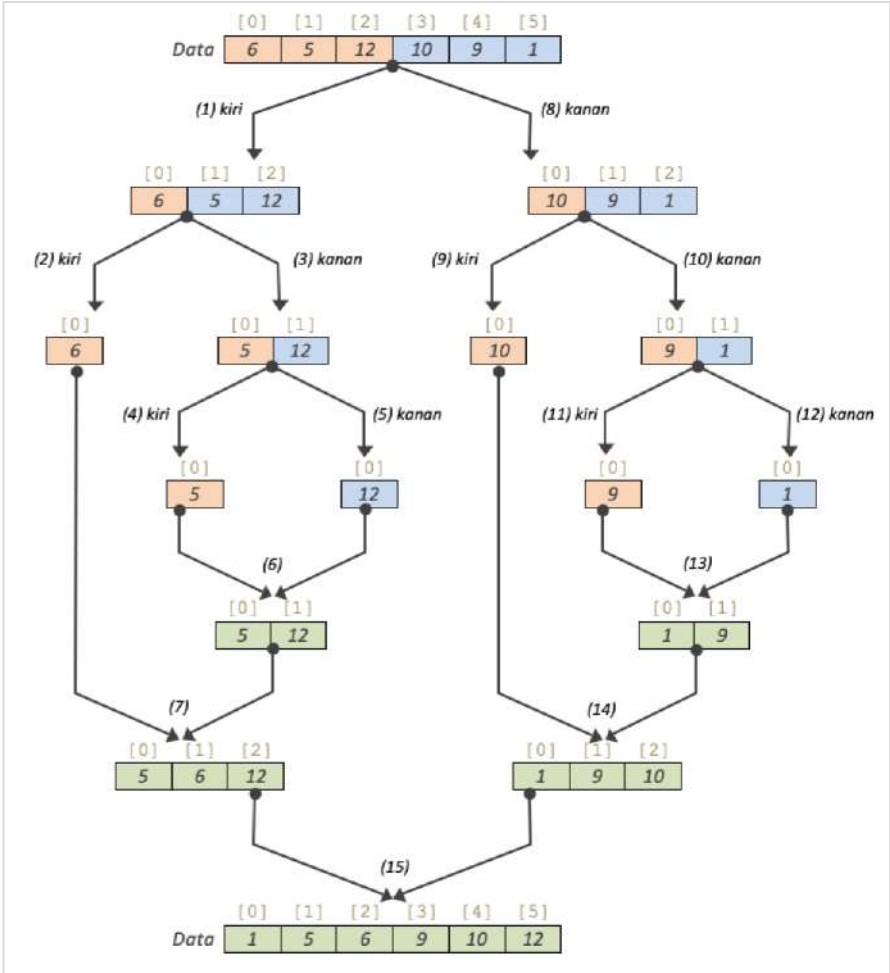
```

Gambar 8.11 Output proses *Insertion Sort*

4. Merge Sort

Merge Sort merupakan algoritma *sorting* yang menerapkan teknik rekursif dalam pendekatan *divide and conquer* berbeda dengan *Bubble Sort*, *Selection Sort* dan *Insertion Sort*. Cara kerja algoritma *Merge Sort* terdiri dari dua tahap, tahap yang pertama yaitu memecah *array* menjadi dua sub-bagian, kemudian dari masing-masing sub-bagian tersebut dipecah lagi menjadi subsub-bagian dan seterusnya hingga masing-masing bagian hanya terdiri dari satu elemen data saja (*atomic*). Tahap yang kedua yaitu penggabungan, selama penggabungan setiap sub-bagian yang sudah *atomic* dibandingkan terlebih dahulu kemudian disusun kembali secara berurutan sesuai dengan posisinya. Penggabungan dilakukan hingga setiap bagian *array* kembali menjadi satu dan sudah terurut. Agar lebih mudah dipahami visualisasi algoritma *Merge sort* dalam bentuk *tree* dapat

dilihat pada Gambar 8.12, nomor urut pada setiap *node* pada *tree* mewakili urutan panggilan rekursif *Merge Sort*.



Gambar 8.12 Visualisasi algoritma *Merge Sort*

Gambar 8.13 merupakan deklarasi dari *class MergeSort* (baris 3). *Method split* (baris 5) berfungsi memecah/membagi data *array* menjadi dua bagian kiri (baris 7-8) dan kanan (baris 12-14), kemudian bagian kiri dan kanan secara *rekursif* dipecah kembali (baris 11 dan 17) hingga tiap

bagian hanya berisi satu elemen saja (baris 6). Proses penggabungan dilakukan oleh *method merge* (baris 23) yang dipanggil oleh *method split* (baris 18). Proses penggabungan dilakukan pada penampung sementara yang panjangnya sama dengan dua *array* yang akan digabungkan (baris 24), sebelum digabungkan tiap elemen data dibandingkan (29-40) dan disusun berdasarkan urutannya.

```

3 public class MergeSort {
4     static int counter=0;
5     public static void split(int[] data) {
6         if(data.length>1) {
7             int[] kiri = new int[data.length/2];
8             System.arraycopy(data, 0, kiri, 0, data.length/2);
9             counter++;
10            System.out.print("\n Split: (" +counter+")kiri: "); cetak(kiri);
11            split(kiri);
12            int pKanan = data.length - data.length /2;
13            int[] kanan= new int[pKanan];
14            System.arraycopy(data, data.length/2, kanan, 0, pKanan);
15            counter++;
16            System.out.print("\t (" +counter+")kanan: "); cetak(kanan);
17            split(kanan);
18            int[] tmp = merge(kiri,kanan);
19            System.arraycopy(tmp, 0, data, 0, tmp.length);
20        }
21    }
22
23    public static int[] merge(int[] kiri, int[] kanan) {
24        int[] tmp = new int [kiri.length+kanan.length];
25        int indexKiri=0;
26        int indexKanan=0;
27        int indexTmp=0;
28        counter++;
29        while (indexKiri < kiri.length && indexKanan < kanan.length) {
30            if (kiri[indexKiri] < kanan[indexKanan]) {
31                tmp[indexTmp++] = kiri[indexKiri++];
32            } else
33                tmp[indexTmp++] = kanan[indexKanan++];
34        }
35        while (indexKiri < kiri.length) {
36            tmp[indexTmp++] = kiri[indexKiri++];
37        }
38        while (indexKanan < kanan.length) {
39            tmp[indexTmp++] = kanan[indexKanan++];
40        }
41        System.out.print("\n Merge: (" +counter+")");
42        cetak(tmp);
43        return tmp;
44    }
45
46    public static void cetak (int[] data) {
47        for(int x=0;x<data.length;x++) {
48            System.out.print(data[x]+" ");
49        }
50    }
51
52    public static void main(String[] a) {
53        int[] data = {6,5,12,10,9,1};
54        System.out.print("Data acak : "); cetak(data);
55        System.out.print("\n");
56        split(data);
57        System.out.print("\n\nData urut : "); cetak(data);
58    }
59 }

```

Gambar 8.13 Implementasi *Merge Sort*

Output dari program *Merge Sort* dapat dilihat pada Gambar 8.14, pada *output* urutan eksekusi *merge sort* ditunjukkan menggunakan (nomor).

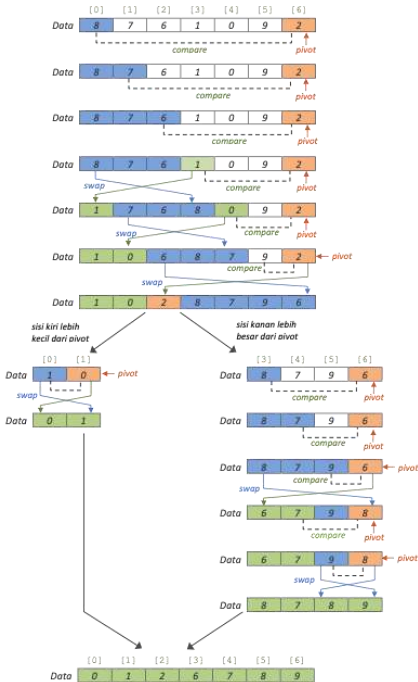
Data acak : 6 5 12 10 9 1

Split: (1)kiri: 6 5 12
Split: (2)kiri: 6 (3)kanan: 5 12
Split: (4)kiri: 5 (5)kanan: 12
Merge: (6)5 12
Merge: (7)5 6 12 (8)kanan: 10 9 1
Split: (9)kiri: 10 (10)kanan: 9 1
Split: (11)kiri: 9 (12)kanan: 1
Merge: (13)1 9
Merge: (14)1 9 10
Merge: (15)1 5 6 9 10 12

Data urut : 1 5 6 9 10 12

Gambar 8.14 *Output* eksekusi *Merge Sort*

5. *Quick Sort*



Gambar 8.15 Ilustrasi Cara kerja *Quick Sort*

Sama seperti *Merge Sort*, algoritma *Quick Sort* menggunakan pendekatan *divide and conquer* dengan teknik rekursif. Namun cara kerja keduanya berbeda. *Merge Sort* membagi *array* menjadi dua bagian secara *recursif* hingga tiap bagian *atomic*, sedangkan *Quick Sort* menggunakan elemen pivot sebagai basis untuk memecah *array* dan mengurutkan elemen data. Visualisasi Cara kerja algoritma *Quick Sort* dijabarkan pada gambar 8.16.

```

3 import java.util.Arrays;
4
5 public class QuickSort {
6     static int counter=0;
7     static int partisi (int data[], int start, int end) { //method partisi
8         int pivot = data[end]; // pivot data paling kanan
9         int i = (start - 1);
10        System.out.print("\npivot = "+pivot+" \n"+Arrays.toString(data));
11
12        for (int index = start; index<=end-1; index++) { //loop untuk membandingkan elemen data
dengan pivot
13            if (data[index] < pivot) { //kondisi membandingkan elemen yang lebih kecil dari pivot
14                i++; //penanda posisi index
15                if((data[index]==data[i]) &&(index == 1) ) {
16                    System.out.println(" :"+data[index]+" < "+pivot+", "+data[index]+" tetap di
["+index+"]");
17                } else {
18                    System.out.println(" :"+data[index]+" < "+pivot+", tukar "+data[index]+" dengan
"+data[i]+"["+i+"]");
19                    int temp = data[i]; //tukar elemen yang < pivot
20                    data[i] = data[index]; //dengan data yang lebih
21                    data[index] = temp; //besar pada index i
22                }
23                System.out.print(Arrays.toString(data));
24            } else if (data[index] > pivot){
25                System.out.print(" :"+data[index]+" > "+pivot+", "+data[index]+"["+index+"]
\n"+Arrays.toString(data));
26            }
27
28            if (data[end] == data[i+1]) {
29                System.out.println(" :end stop, "+data[end]+" tetap (di kiri tidak ada yang >
"+data[end]+" )\n"+Arrays.toString(data) );
30            } else {
31
32                System.out.println(" :end stop, tukar "+data[end]+" dengan "+data[i+1]+"["+i+1+"]");
33                int temp = data[i+1]; //tukar pivot dengan
34                data[i+1] = data[end]; //data yang >
35                data[end] = temp; //pada index i
36                System.out.print(Arrays.toString(data));
37            }
38            System.out.print("\n");
39            return (i + 1); //mengembalikan posisi
40        }
41
42        static void quick(int data[], int start, int end) { //proses sorting
43            if (start < end) {
44                counter++;
45                System.out.print("\nStep "+counter);
46                int pIndex = partisi(data, start, end); //menentukan posisi partisi
47                quick(data, start, pIndex - 1); //rekursif sisi kiri pivot
48                quick(data, pIndex + 1, end); //rekursif sisi kanan pivot
49            }
50        }
51
52        public static void main(String[] args) {
53            int data[] = {8,7,6,1,0,9,2};
54            int n = data.length;
55            System.out.println("\nData Acak : "+Arrays.toString(data));
56            quick(data, 0, n - 1);
57            System.out.print("\nData Druet : "+Arrays.toString(data));
58        }
59 }

```

Gambar 8.16 Implementasi algoritma *Quick Sort*

Tahap pertama pada algoritma *Quick Sort* yaitu memilih satu elemen sebagai pivot, umumnya elemen paling kanan (akhir data) namun dalam beberapa kasus dapat digunakan elemen paling kiri (awal data) atau posisi elemen pivot dapat juga ditentukan secara acak (random). Tahap kedua yaitu mempartisi *array* dengan membandingkan semua elemen dengan pivot, elemen yang lebih kecil atau sama dengan pivot diletakkan pada sisi kiri pivot, sedangkan elemen yang lebih besar ditempatkan pada sisi kanan. Tahap ketiga mengurutkan dan menyusun kembali elemen data, pada tahap ini *quick sort* akan menentukan pivot baru pada sisi kiri dan sisi kanan kemudian membandingkan tiap elemen pada sisi kiri dan sisi kanan dengan pivotnya menggunakan panggilan rekursif dan menempatkan elemen pada urutan yang tepat.

Implementasi algoritma *Quick Sort* dapat dilihat pada Gambar 8.16. Proses *sorting* dimulai dengan mengirimkan data *array* acak (baris 53 dan 56) kepada *method quick* (baris 42), *method* ini berfungsi untuk menentukan posisi pembagian *array* secara *recursif* dengan memanggil *method* partisi (baris 46). *Method* partisi (baris 7) berfungsi menentukan elemen pivot (baris 8), kemudian membandingkan elemen pivot dengan semua elemen data (baris 13) dan melakukan partisi dengan pertukaran elemen data ke posisi kiri atau kanan dari elemen pivot (baris 18-22, 33-35). Setelah melakukan pertukaran data *method* partisi mengirimkan nilai balik berupa posisi terakhir elemen pivot kepada *method quick* yang digunakan kembali sebagai penentu partisi pada pengulangan rekursif berikutnya. Panggilan rekursif berakhir posisi data terakhir (baris 43). Gambar 8.17 menampilkan *output* proses *sorting Quick Sort* beserta perubahan nilai pivot dan proses pertukaran data pada setiap step.

```

Data Acak : [8, 7, 6, 1, 0, 9, 2]

Step 1
pivot = 2
[8, 7, 6, 1, 0, 9, 2] :8 > 2, 8[0]
[8, 7, 6, 1, 0, 9, 2] :7 > 2, 7[1]
[8, 7, 6, 1, 0, 9, 2] :6 > 2, 6[2]
[8, 7, 6, 1, 0, 9, 2] :1 < 2, tukar 1 dengan 8[0]
[1, 7, 6, 8, 0, 9, 2] :0 < 2, tukar 0 dengan 7[1]
[1, 0, 6, 8, 7, 9, 2] :9 > 2, 9[5]
[1, 0, 6, 8, 7, 9, 2] :end step, tukar 2 dengan 6[2]
[1, 0, 2, 8, 7, 9, 6]

Step 2
pivot = 0
[1, 0, 2, 8, 7, 9, 6] :1 > 0, 1[0]
[1, 0, 2, 8, 7, 9, 6] :end step, tukar 0 dengan 1[0]
[0, 1, 2, 8, 7, 9, 6]

Step 3
pivot = 6
[0, 1, 2, 8, 7, 9, 6] :8 > 6, 8[3]
[0, 1, 2, 8, 7, 9, 6] :7 > 6, 7[4]
[0, 1, 2, 8, 7, 9, 6] :9 > 6, 9[5]
[0, 1, 2, 8, 7, 9, 6] :end step, tukar 6 dengan 8[3]
[0, 1, 2, 6, 7, 9, 8]

Step 4
pivot = 8
[0, 1, 2, 6, 7, 9, 8] :7 < 8, 7 tetap di [4]
[0, 1, 2, 6, 7, 9, 8] :9 > 8, 9[5]
[0, 1, 2, 6, 7, 9, 8] :end step, tukar 8 dengan 9[5]
[0, 1, 2, 6, 7, 8, 9]

Data Urut : [0, 1, 2, 6, 7, 8, 9]

```

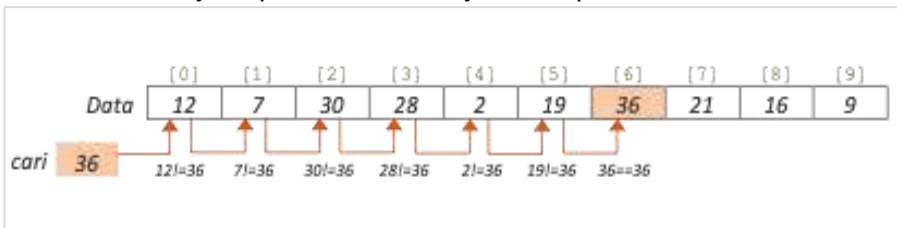
Gambar 8.17 Output proses Quick Sort

C. SEARCHING

Searching (pencarian) pada struktur data mengacu pada proses algoritmik untuk memeriksa, menemukan, mengambil posisi elemen tertentu (target) dari kumpulan elemen di dalam struktur data. Kumpulan elemen dapat berupa *array*, *list*, *linked list*, *tree*, dan lain sebagainya. Berdasarkan jenis operasinya, dua algoritma pencarian yang dibahas pada subbab ini yaitu *Sequential Search* dan *Binary Search* pada data *array*.

1. Sequential Search

Sequential Search (pencarian berurutan) dikenal juga sebagai pencarian linier. *Sequential Search* merupakan algoritma pencarian paling dasar dan paling sederhana yang bekerja menggunakan pendekatan *brute force*, mencoba segala kemungkinan yang ada hingga target yang diinginkan ditemukan. Sebagai contoh, mencari nilai 36 dari data *array* {12, 7, 30, 28, 2, 19, 36, 21, 16, 9}. *Sequential Search* dimulai dengan membandingkan/mencocokkan 36 dengan elemen pertama/*index*[0] yaitu 12, karena 12 tidak sesuai dengan kriteria pencarian ($12 \neq 36$) maka dilanjutkan dengan membandingkan 36 dengan elemen berikutnya hingga ditemukan elemen yang cocok, yaitu pada *index*[6] dimana $36 == 36$. Ilustrasi cara kerja *Sequential Search* dijabarkan pada Gambar 8.18.



Gambar 8.18 Ilustrasi *Sequential Search*

Implementasi algoritma *Sequential Search* dapat dilihat pada Gambar 8.19. *Sequential Search* dideskripsikan pada *class Sequential Search* (baris 6) sedangkan proses pencarian data dilakukan didalam *method search* (baris 7). *Method search* menerima kunci pencarian dari pengguna (baris 12) kemudian menggunakan perulangan (baris 13) untuk membandingkan kunci pencarian dengan semua elemen pada *array* (baris 15). Program berhenti ketika elemen telah ditemukan (baris 16) atau ketika semua elemen telah selesai dibandingkan (baris 20). *Output* program dapat dilihat pada Gambar 8.20, pada *output* dijabarkan tahapan proses pencarian dan hasil pencarian apabila elemen ditemukan dan tidak ditemukan.

```

1 package HapusAja;
2
3 import java.util.Scanner;
4 import java.util.Arrays;
5
6 public class SequentialSearch {
7     public static void search(int[] data) {
8         boolean status=true;
9         Scanner input = new Scanner(System.in);
10        System.out.println("data array : "+Arrays.toString(data));
11        System.out.print("\nMasukkan angka yang anda cari : ");
12        int key=input.nextInt();
13        for(int index=0; index<data.length; index++) {
14            System.out.println(data[index]+" != "+key);
15            if (key == data[index]) {
16                System.out.println(key+" == "+data[index]+" ketemu di index ke-"+index);
17                if ( status == true)
18                    break;
19            }
20            if (index==(data.length-1))
21                System.out.println("data tidak ditemukan");
22        }
23    }
24
25    public static void main(String[] args) {
26        int[] data= {12,7,30,28,2,19,36,21,16,9};
27        search(data);
28    }
29 }

```

Gambar 8.19 Implementasi *Sequential Search*

data array : [12, 7, 30, 28, 2, 19, 36, 21, 16, 9]

Masukkan angka yang anda cari : 36

12 != 36

7 != 36

30 != 36

28 != 36

2 != 36

19 != 36

36 != 36

36 == 36 ketemu di index ke-6

data array : [12, 7, 30, 28, 2, 19, 36, 21, 16, 9]

Masukkan angka yang anda cari : 63

12 != 63

7 != 63

30 != 63

28 != 63

2 != 63

19 != 63

36 != 63

21 != 63

16 != 63

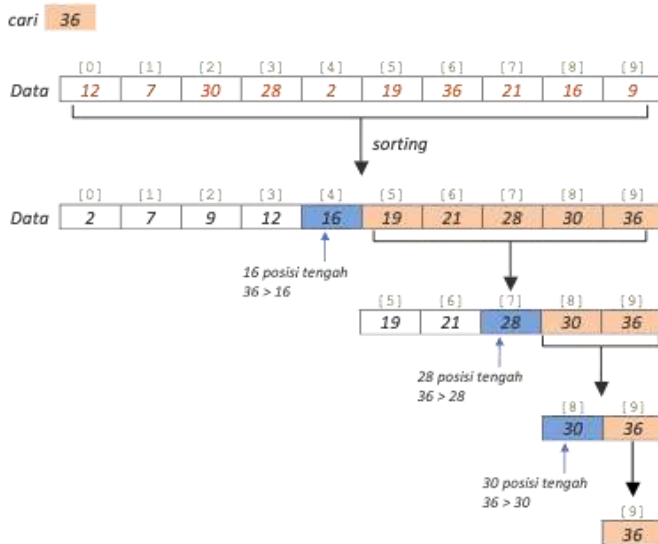
9 != 63

data tidak ditemukan

Gambar 8.20 Output hasil pencarian *Sequential Search*

2. Binary Search

Algoritma *Binary Search* bekerja dengan menggunakan pendekatan *divide and conquer* pada data *array* yang sudah terurut (*sorted*), artinya algoritma *Binary Search* diawali dengan mengurutkan (*sorting*) *array* terlebih dahulu sebelum melakukan proses pencarian (Goodrich *et al*, 2014). Setelah data *array* terurut proses pencarian dimulai dengan perulangan pertama untuk mencari elemen yang posisinya berada di tengah *array*, kemudian membandingkan elemen tengah tersebut dengan kunci pencarian, jika kunci pencarian lebih besar dari elemen tengah, maka abaikan semua elemen *array* yang berada di posisi kiri lanjutkan perulangan kedua dengan mencari elemen tengah pada *array* sisi kanan saja dan dibandingkan lagi dengan kunci pencarian. Namun apabila kunci pencarian lebih kecil dari elemen tengah maka abaikan semua elemen *array* di posisi kanan dan lanjutkan perulangan kedua dengan mencari elemen tengah pada *array* sisi kiri lalu bandingkan dengan kunci pencarian. langkah-langkah ini dilakukan berulang sampai dengan ditemukan elemen tengah yang sama dengan kunci pencarian atau elemen telah habis dibandingkan. Lebih jelas tentang cara kerja algoritma *Binary Search* dapat dilihat pada Gambar 8.21.



Gambar 8.21 Ilustrasi algoritma *Binary Search*

```

3 import java.util.Scanner;
4
5 public class BinarySearch {
6
7     public static void main(String[] args) {
8         int[] data= {12,7,30,28,2,19,36,21,16,9};
9         Scanner input = new Scanner(System.in);
10        MergeSort Sort = new MergeSort();
11        Sort.split(data); //sorting data memanggil method split dari class MergeSort
12        System.out.print("\n\nData setelah sorting : "); Sort.cetak(data);
13        System.out.print("\n\nMasukkan angka yang anda cari : ");
14        int key=input.nextInt();
15        cari(data, 0, data.length-1, key); //method pencarian
16    }
17
18    public static void cari(int[] data, int awal, int akhir, int input) {
19        int tengah = (awal+akhir)/2; //inisialisasi tengah
20
21        while(awal<=akhir) {
22            System.out.print("\ntengah = "+data[tengah]);
23            if (data[tengah]<input) { //kondisi tengah < dari input
24                System.out.print("\n"+data[tengah]+" < "+input);
25                awal = tengah+1;
26            } else if (data[tengah]==input) { //kondisi ditemukan
27                System.out.print("\n"+data[tengah]+" == "+input+", "+input+" ditemukan
pada index ke-"+tengah);
28                break;
29            } else if (data[tengah] > input) { //kondisi tengah > dari input
30                System.out.print("\n"+data[tengah]+" > "+input);
31                akhir=tengah-1;
32            }
33            tengah=(awal+akhir)/2; //perubahan tengah
34            if(awal>akhir) { //kondisi tidak ditemukan
35                System.out.print("\n"+input+" tidak ditemukan");
36            }
37        }
38    }
39 }

```

Gambar 8.22 Implementasi *Binary Search*

Gambar 8.22 menjabarkan implementasi algoritma *Binary Search*. Pada *method* main (Baris 7-16) data acak (baris 8) diurutkan terlebih dahulu dengan memanggil *method split* dari *class MergeSort* (baris 10 dan 11). Setelah data terurut proses pencarian kunci/key (baris 14) dilakukan oleh *method* cari (baris 18-38), diawali dengan menentukan nilai tengah (baris 19) kemudian selama data *array* belum habis (baris 21) secara berulang *method* cari akan membandingkan dan memisahkan posisi pencarian data (baris 23-36). Baris 23-26 merupakan kondisi pencarian jika nilai *key* lebih besar dari nilai tengah maka pencarian difokuskan pada posisi elemen di sebelah kanan, baris 29-32 merupakan kondisi jika nilai *key* lebih kecil dari nilai tengah maka pencarian difokuskan ke semua elemen sebelah kiri. Baris 26-28 merupakan kondisi jika *key* sama dengan elemen tengah maka data ditemukan dan pencarian berakhir (baris 26). Setiap melakukan penentuan menentukan nilai tengah untuk posisi yang

dilakukan pada baris 33. *Output* proses pencarian dapat dilihat pada Gambar 8.23.

```
Data setelah sorting : 2 7 9 12 16 19 21 28 30 36
```

```
Masukkan angka yang anda cari : 36
```

```
tengah = 16
16 < 36
tengah = 28
28 < 36
tengah = 30
30 < 36
tengah = 36
36 == 36, 36 ditemukan pada index ke-9
```

```
Data setelah sorting : 2 7 9 12 16 19 21 28 30 36
```

```
Masukkan angka yang anda cari : 3
```

```
tengah = 16
16 > 3
tengah = 7
7 > 3
tengah = 2
2 < 3
3 tidak ditemukan
```

Gambar 8.23 *Output* hasil pencarian *Binary Search*

D. RANGKUMAN MATERI

Sorting pada dasarnya menempatkan elemen pada struktur data secara berurutan sedangkan *searching* mengacu pada mencari dan mengambil elemen data dari suatu kumpulan data. Bab ini menjabarkan tahapan dan cara kerja algoritma *sorting* dan *searching* pada struktur data *array* menggunakan dua pendekatan yaitu *brute force* dan *divide and conquer*. Algoritma *Bubble Sort*, *Selection Sort* dan *Insertion Sort* menggunakan pendekatan *brute force*, sedangkan *Merge Sort* dan *Quick Sort* menggunakan pendekatan *divide and conquer*. Algoritma *searching* dengan pendekatan *brute force* yaitu *sequential search* dan algoritma *searching* dengan pendekatan *divide and conquer* yaitu *binary search*.

TUGAS DAN EVALUASI

1. Implementasikan proses *sorting* untuk mengurutkan *array* {'d', 'c', 'g', 'h', 'b', 'a', 'f', 'i', 'e', 'c'} menggunakan algoritma *bubble sort*, *selection sort*, *insertion sort*, *merge sort* dan *quick sort*
2. Implementasikan proses pencarian elemen 'a' pada *array* {'d', 'c', 'g', 'h', 'b', 'a', 'f', 'i', 'e', 'c'} menggunakan algoritma *sequential search* dan *binary search*. *Output* program berupa *index* posisi *element* atau "tidak ditemukan"
3. Implementasikan proses pencarian untuk mencari "joko" pada *array* {"budi","LULUS"}, {"andi","TIDAK LULUS"}, {"tini","LULUS"}, {"tono","TIDAK LULUS"}, {"joko","LULUS"}, {"cindi","LULUS"}} menggunakan algoritma *Sequential Search* dan *Binary Search*. *Output* program LULUS atau TIDAK LULUS.
4. Bandingkan kelemahan dan kelebihan algoritma *sorting* yang menggunakan pendekatan *brute force* dengan pendekatan *divide and conquer*.
5. Bandingkan kelemahan dan kelebihan algoritma *searching* yang menggunakan pendekatan *brute force* dengan pendekatan *divide and conquer*.

DAFTAR PUSTAKA

- Deitel, P & Deitel, H. (2011). Java How To Program, 9th Edition, Boston: Pearson.
- Goodrich, Michael T., Tamassia, R., & Goldwasser, Micheal H. (2014). Data structures and algorithms in JAVA, 6th Edition. Hoboken, New Jersey: John Wiley and Sons.
- Levitin, A. (2007). Introduction to the design & analysis of algorithms. Boston: Pearson Addison-Wesley.

GLOSARIUM

A

Aplikasi: Perangkat lunak

Anotasi: Catatan yang dibuat untuk menerangkan suatu kegiatan

Algoritma: Prosedur sistematis untuk memecahkan masalah matematis dalam langkah-langkah terbatas

Array: Kumpulan komponen dengan tipe data yang sama

Algoritma: Seperangkat instruksi atau langkah-langkah yang ditulis secara sistematis dan digunakan untuk memecahkan suatu masalah.

B

Boolean: Suatu tipe data yang hanya mempunyai dua nilai. Yaitu *true* atau *false* (benar atau salah).

C

Char: Tipe data karakter. Pada setiap semua data yang hanya terdiri dari 1 karakter tergolong dalam tipe ini. Misalnya data jenis kelamin yang hanya diisikan huruf L atau P. Penulisan data tipe *char* harus diapit oleh tanda petik tunggal. Karakter-karakter yang diperbolehkan terdefinisi dalam tabel ASCII

D

E

F

Function: Fungsi pada program untuk menjalankan perintah tertentu

G

H

Homogen: Sejenis

Header: Bagian atas (judul) pada program

I

Input: Masukan

Iterasi: Proses yang serangkaian operasinya dilakukan, baik yang jumlahnya telah ditetapkan maupun sampai kondisi tertentu terpenuhi

Index: Akses *array*

Int: *Integer* (tipe data untuk menyatakan angka)

Integer: Merujuk kepada data apapun yang merepresentasikan bilangan bulat.

J

JDK: *Java Developmen Kit*. Sebuah paket aplikasi yang berisi *Java Virtual Machine* (JVM) + *Java Runtime Environment* (JRE) + paket aplikasi atau *libraries* untuk proses pembuatan kode program Java.

JRE: *Java Runtime Environment*. Sebuah paket aplikasi yang berisi *Java Virtual Machine* (JVM) serta *libraries* yang diperlukan untuk menjalankan aplikasi Java.

JVM: *Java Virtual Machine*. Aplikasi dasar yang harus di-*install* pada sebuah perangkat agar dapat menjalankan aplikasi Java.

K

Kelas: Sebuah *blueprint* yang berisi kode program untuk membuat sebuah objek.

Kebenaran Logika: Validitas akal (pengetahuan) tentang objek

L

Loop: Perulangan

Link List: Suatu struktur data linier.

M

N

NetBeans: Sebuah aplikasi *open source* berbasis IDE yang dapat digunakan untuk melakukan pemrograman, salah satunya adalah pemrograman menggunakan bahasa Java.

O

Output: Keluaran

Operator: Simbol yang digunakan untuk mengoperasikan dua buah *operand*.

P

Programmer: Jenis profesi atau pekerjaan yang bertujuan untuk membuat sebuah sistem menggunakan bahasa pemrograman

Package: Sebuah teknik/cara yang dilakukan untuk mengelompokkan kelas.

Pemrograman: Sebuah cara untuk mengubah algoritma menjadi sebuah program menggunakan bahasa pemrograman tertentu.

Project: Sebuah pekerjaan (yang berkaitan dengan pemrograman).

Program: Kata, ekspresi, pernyataan atau kombinasi yang disusun dan dirangkai menjadi satu kesatuan prosedur yang menjadi urutan langkah untuk menyesuaikan masalah yang diimplementasikan dengan bahasa pemrograman.

Percabangan: Struktur program yang digunakan untuk melakukan proses pengujian terhadap satu, dua atau lebih dari dua kondisi

Q

R

Return: Mengembalikan nilai

S

Statement: Pernyataan, laporan dan Pengumuman

Switch Case: Percabangan kode program dimana kita membandingkan isi sebuah variabel dengan beberapa nilai.

String: Tipe data untuk menyatakan teks atau huruf

Sintaksis: Ilmu tentang prinsip dan aturan komposisi kalimat dalam bahasa alami. Terlepas dari aturan-aturan ini, kata sintaksis juga digunakan untuk secara langsung merujuk pada aturan dan prinsip yang mencakup struktur kalimat dalam bahasa apa pun.

Sorting: Proses penyusunan elemen-elemen dengan tata urutan tertentu.

T

Tipe Data: Pengelompokan data berdasarkan jenis data tertentu.

Teknik Rekursif: Metode pengulangan yang bersifat *non*-iterasi.

U

V

Variabel: Sebuah tempat (dalam memori) yang digunakan untuk menampung data.

Value: Nilai sebuah variabel

W

X

Y

Z

PROFIL PENULIS

Meidyan Permata Putri, S.Kom., M.Kom



Penulis bernama Meidyan Permata Putri, S.Kom., M.Kom., lahir di kota Palembang pada tahun 1986. Pada tahun 2010 menyelesaikan pendidikan strata 1 program Studi Sistem Informasi di STMIK MDP Palembang. Penulis melanjutkan Pendidikan strata 2 di Universitas Bina Darma Palembang pada tahun 2013 program studi Magister Teknik Informatika. Saat ini Penulis merupakan dosen tetap di Institut Teknologi dan Bisnis PalComTech pada program studi Sistem Informasi sejak tahun 2015. Fokus bidang keahlian penulis utamanya dibidang sistem informasi, Analisis dan Perancangan sistem, Interaksi manusia dan komputer penulis juga tertarik serta Penjaminan Mutu perangkat lunak.

Guntoro Barovih, S.Kom., M.Kom



Penulis bernama Guntoro Barovih, S.Kom., M.Kom., kelahiran kota Tanjung Enim pada tahun 1986. Menyelesaikan pendidikan strata 1 bidang Teknik Informatika di STMIK PalComTech Palembang pada tahun 2011. Penulis kemudian melanjutkan studi Magister Teknik Informatika dan menyelesaikan pendidikan strata 2 di Universitas Bina Darma Palembang pada tahun 2013. Saat ini Penulis merupakan dosen tetap di Institut Teknologi dan Bisnis PalComTech pada program Studi S1 Informatika sejak tahun 2012. Fokus bidang keahlian penulis utamanya dibidang Informatika, Jaringan Komputer, Komputasi Awan, Pemrograman Web dan penulis juga tertarik pada Sistem Digital.

Rezania Agramanisti Azdy, S.Kom., M.Cs



yang sama.

Penulis kelahiran Yogyakarta 15 November 1986 ini merupakan seorang dosen yang ditugaskan pada Institut Teknologi dan Bisnis PalComTech (Program Studi S1 Informatika). Pendidikan yang telah ditempuh adalah pendidikan jenjang S1 pada jurusan Ilmu Komputer Universitas Gadjah Mada, kemudian dilanjutkan ke jenjang S2 pada Universitas dan Jurusan

Yuniansyah, S.Kom., M.Kom



Mengenal Ilmu Komputer Pada Tahun 1995 dimana penulis mendapatkan kesempatan melanjutkan pendidikan Strata satu di Universitas Bina Darma Palembang dan selesai pada tahun 1999. Pada saat kuliah juga selama 2 tahun (1997-1999) sempat menjadi Asisten Dosen di Laboratorium komputer Universitas Bina Darma untuk beberapa matakuliah pemrograman. Pada awal tahun 2002 melanjutkan pendidikan di Program Studi Magister Ilmu Komputer Fakultas Ilmu Matematika dan Ilmu Pengetahuan Alam (F-MIPA) Universitas Gadjah Mada – Yogyakarta dan selesai pada akhir tahun 2013. Selama di Yogyakarta juga penulis berkesempatan menjadi Dosen di salah satu Perguruan Tinggi yang ada di Yogyakarta. Selepas meraih gelar Magister Komputer penulis menjadi dosen di beberapa Perguruan Tinggi swasta dan Universitas Negeri di Kota Palembang. Penulis juga pernah menjadi Dosen di Kota Lubuk Linggau, Batam, dan Duri. Saat ini penulis menjadi dosen praktisi di salah satu perguruan tinggi ternama di Kota Palembang Penulis ini dapat dihubungi melalui email: yuniansyah.mr@gmail.com serta dapat juga melalui WA/Telegram: 0812 3516 8181.

Andri Saputra, S.Kom., M.Kom



Penulis adalah Dosen Tetap Institut Teknologi dan Bisnis PalComTech di Kota Palembang yang bertugas di Program Sarjana (S1) Informatika, pernah menjabat sebagai Ketua Program Studi S1 Sistem Informasi dari Tahun 2015 s.d 2021. Pendidikan tingginya dimulai dari S1 di Universitas Bina Darma dengan program studi Teknik Informatika, dilanjutkan dengan program S2 di Perguruan tinggi yang sama di program studi Magister Teknik Informatika. Sebelum terjun di dunia pendidikan sebagai Dosen atau dimasa menjalani proses kuliah penulis sempat berkecimpung di dunia bisnis besi tua. Saat ini selain aktif mengajar, melakukan penelitian dan pengabdian (Menjalankan Tri Dharma Perguruan Tinggi) penulis juga saat ini aktif sebagai praktisi bidang dunia bisnis digital.

Yesi Sriyeni, S.Kom., M.Kom



Penulis adalah seorang dosen tetap pada Institut Teknologi dan Bisnis Palcomtech Palembang dengan nama lengkap Yesi Sriyeni, S.Kom., M.Kom., yang lahir di kota Palembang 18 Maret 1989. Pendidikan strata 1 diselesaikan pada tahun 2012 di STMIK Palcomtech Palembang, melanjutkan studi strata 2 pada tahun 2015 dan lulus di tahun 2017 dari Universitas Bina Darma Palembang. Penulis fokus pada bidang keahlian yang meliputi analisis dan perancangan sistem, rekayasa perangkat lunak, analisa proses bisnis, interaksi manusia dan komputer. Saat ini penulis juga tertarik untuk mendalami bidang ilmu penjaminan mutu dan pengujian perangkat lunak.

Arsia Rini, S.Kom., M.Kom



Penulis yang sering disapa Arsi memiliki nama lengkap Arsia Rini, S.Kom., M.Kom., merupakan dosen tetap di Program Studi Teknik Komputer Jurusan Teknik Komputer Politeknik Negeri Sriwijaya Palembang. Fokus keahlian penulis adalah dibidang pemrograman berbasis web, pemrograman java, pemrograman desktop, analisis perancangan sistem informasi, algoritma dan struktur data, pengolahan *database*, *office application*, pemrograman visual *basic*, desain grafis, *video editing* dan *artificial intelligence*.

Fadhila Tangguh Admojo, M.Cs



Penulis lahir di Jakarta 12 Agustus 1983. Lulusan S2 Master of Computer Science dari Universitas Gadjah Mada. Memutuskan mengajar sejak tahun 2003 hingga sekarang masih menjadi dosen di bidang ilmu komputer. Saat ini sedang melanjutkan pendidikan ke jenjang yang lebih tinggi.

ALGORITMA DAN STRUKTUR DATA

Buku ini berisi tentang algoritma dan struktur data, dengan mempelajari buku ini anda dapat memahami dan mempelajari algoritma dan struktur data dari dasar. Mulai dari pengenalan Operator dan logika pemrograman yang mana setiap babnya penulis memberikan penjelasan tentang pemrograman dengan konsep oop serta penulis juga memberikan beberapa contoh *study kasus* yang mudah dipelajari dan dipahami. Didalam buku ini terdiri dari 8 bab yaitu:

1. Pengantar Algoritma
 - Pengertian Algoritma
 - Kegunaan dan contohnya
2. *Flowchart*
 - *Flowchart* aplikasi
 - *Flowchart* Sistem
3. Pemrograman Java
 - Bahasa Pemrograman Java
 - Struktur Program
 - Variabel
 - Tipe Data
 - Operator
 - Contoh
 - Latihan
4. Struktur Kontrol
 - Logika *If* (Tunggal, Majemuk dan *nested*)
 - Contoh
 - Latihan
 - *Switch* dan *Break*
 - Contoh
 - Latihan
5. *Looping* (Perulangan)
 - *For While*, *Dowhile*
 - *Break* dan *continue*
 - *Nested Loop*
 - Contoh
 - Latihan
6. *Array*
 - *Array* 1 dimensi dan Operasi
 - Definisi *Array* dan deklarasi
 - Kegunaan 1 *Array* dimensi
7. *Procedure* dan *Function*
 - Definisi *Procedure*
 - *Procedure*
 - Definisi *Function*
 - Sifat *Function*
 - *Function* (Non -void)
 - Parameter formal dan aktual
 - Contoh
 - Latihan
8. *Sorting, Searching Array*
 - *Bubble Sort*
 - *Selection Sort*
 - *Insertion Sort*
 - *Quike Sort*
 - *Searching* sekuensial
 - Contoh
 - Latihan