



Dasar-Dasar Pemrograman

Yuyun Khairunisa, M.Kom., Eri Mardiani, Nur Rahmansyah, S.Kom., M.Kom., Agus Ambarwari, S.Pd., M.Kom., Zuriati, S.Kom., M.Kom., Putri Ariatna Alia, S.ST MT., Drs. Purwadi Budi Santoso, MT., Nur Hayati, S.Si., MTI., Erik Yohan Kartiko, S.Pd., M.Kom., Johan Suryo Prayogo, S.Kom., MT., Yulifda Elin Yuspita, M.Kom., Agung Teguh Setyadi, S.Kom., M.Kom., Nurhafifah Matondang, S.Kom., MTI., Imanaji Hari Sayekti, S.Pd., M.Pd.

Editor: Sri Tria Siska, S.Kom., M.Kom





DASAR-DASAR PEMROGRAMAN

Yuyun Khairunisa, M.Kom — Eri Mardiani — Nur Rahmansyah., S.Kom., M.Kom. —
Agus Ambarwari, S.Pd., M.Kom. — Zuriati, S.Kom., M.Kom. —
Putri Ariatna Alia S.ST M.T — Drs. Purwadi Budi Santoso, MT — Nur Hayati, S.Si., MTI —
Erik Yohan Kartiko, S.Pd., M.Kom. — Johan Suryo Prayogo, S.Kom., M.T. —
Yulifda Elin Yuspita, M.Kom — Agung Teguh Setyadi, S.Kom., M.Kom. —
Nurhafifah Matondang, S.Kom.,M.T.I — Imanaji Hari Sayekti, S.Pd., M.Pd.



Dasar-Dasar Pemrograman

Copyright © PT Penamuda Media, 2023

Penulis:

Yuyun Khairunisa, M.Kom — Eri Mardiani — Nur Rahmansyah., S.Kom., M.Kom. —
Agus Ambarwari, S.Pd., M.Kom. — Zuriati, S.Kom., M.Kom. —
Putri Ariatna Alia S.ST M.T — Drs. Purwadi Budi Santoso, MT — Nur Hayati, S.Si., MTI —
Erik Yohan Kartiko, S.Pd., M.Kom. — Johan Suryo Prayogo, S.Kom., M.T. —
Yulifda Elin Yuspita, M.Kom — Agung Teguh Setyadi, S.Kom., M.Kom. —
Nurhafifah Matondang, S.Kom.,M.T.I — Imanaji Hari Sayekti, S.Pd., M.Pd.

ISBN:

9 786230 945908

Editor:

Sri Tria Siska, S.Kom., M.Kom

Penyunting dan Penata Letak:

Tim PT Penamuda Media

Desain Sampul:

Tim Desain PT Penamuda Media

Penerbit:

PT Penamuda Media

Redaksi:

Casa Sidoarum RT03 Ngentak, Sidoarum Godean Sleman Yogyakarta

Web : www.penamuda.com

E-mail : penamudamedia@gmail.com

Instagram : [@penamudamedia](https://www.instagram.com/penamudamedia)

WhatsApp : +6285700592256

Cetakan Pertama, Juli 2023

viii + 194 halaman; 15,5 x 23 cm

Hak cipta dilindungi undang-undang

Dilarang memperbanyak maupun mengedarkan buku dalam bentuk dan dengan cara apapun tanpa izin tertulis dari penerbit maupun penulis

Kata Pengantar

Puji Syukur Kita ucapkan Kehadiran Alloh SWT atas segala nikmat dan karunianya untuk kita semua, berkat rahmat Alloh SWT yang maha kuasa akhirnya kami bisa menyelesaikan penyusunan Buku Dasar – Dasar Pemograman ini. Buku ini membahas tentang bahsa pemograman tingkat menengah yang sangat mudah untuk dipelajari dan dipahami nantinya. Buku ini juga bisa membantu untuk mengenal dasar – dasar Algoritma Pemograman berserta dengan contoh kasus penyelesaian sehingga bagi para pemula dapat menguasai pemograman ini dengan cepat dan mudah. Buku ini juga sangat cocok untuk mahasiswa dan pemula yang baru mempelajari dan ingin lebih menguasai pemograman khususnya pada program Bahasa C/C++. Materi buku ini dibuat melalui beberapa tahap dan latihan untuk bisa mempercepat pemahaman bagi pengguna buku Dasar Pemograman ini.

Dalam kesempatan kali ini kami mengucapkan terima kasih kepada semua pihak yang telah membantu dan banyak memberikan motivasi kepada kami sehingga buku ini dapat kami selesaikan dengan baik. Ucapan terimakasih khususnya kami ucapkan kepada kedua Orang tua kami yang terus berkontribusi dalam duni pendidikan dan selalu memotivasi agar kami selalu berkarya yang bisa bermanfaat bagi semua

orang. Dan seluruh teman – teman yang tidak bisa kami sebutkan satu persatu. Akhirnya dengan segala kerendahan hati kami sangat mengharapkan kriti dan saran yang sifatnya membangun dari pembaca demi kesempurnaan buku ini sehingga lebih bermanfaat untuk kedepannya. Dan Semua pihak yang terkait yang telah banyak membantu akhirnya kami berharap semoga buku ini bisa dipergunakan bagi orang banyak dan bermanfaat bagi para pembaca Buku Dasar Pemrograman. Aamiin.

Payakumbuh, Juni 2023

Sri Tria Siska, S.Kom., M.Kom

Daftar Isi

KATA PENGANTAR.....	v
DAFTAR ISI	vii
BAB 1 PENGENALAN DASAR DASAR ALGORITMA PEMROGRAMAN	1
BAB 2 KONSEP DASAR PEMROGRAMAN	13
BAB 3 SEJARAH FLOWCHART	26
BAB 4 TIPE DATA, OPERATOR DAN EKSPRESI	50
BAB 5 OPERASI SELEKSI	81
BAB 6 OPERASI PERULANGAN	94
BAB 7 PENGENALAN BAHASA C/C++	102
BAB 8 PEMROGRAMAN MODULAR	113
BAB 9 ARRAY	123
BAB 10 STRUKTUR.....	131
BAB 11 PEMROGRAMAN ARRAY OF STRUCT.....	140
BAB 12 PEMROGRAMAN PENCARIAN	150
BAB 13 PEMROGRAMAN SORTING.....	159
BAB 14 PEMROGRAMAN BERBASIS OBJEK.....	170
TENTANG PENULIS.....	181
DAFTAR PUSTAKA.....	190





BAB 1

PENGENALAN DASAR DASAR ALGORITMA PEMROGRAMAN

Untuk membahas pengendalian dasar dasar algoritma pemrograman maka kita harus memahami mengenai komputer terlebih dahulu. Disini komputer adalah alat yang diberi serangkaian perintah oleh manusia sehingga dapat menyelesaikan permasalahan secara cepat, akurat. Dan sekumpulan instruksi yang merupakan penyelesaian masalah disebut dengan program. Supaya program dapat dilaksanakan oleh komputer, maka program harus ditulis dengan menggunakan bahasa yang dimengerti oleh komputer. Bahasa komputer yang digunakan dalam penulisan program dinamakan bahasa pemrograman. Dan urutan langkah-langkah yang sistematis untuk menyelesaikan sebuah masalah dinamakan algoritma

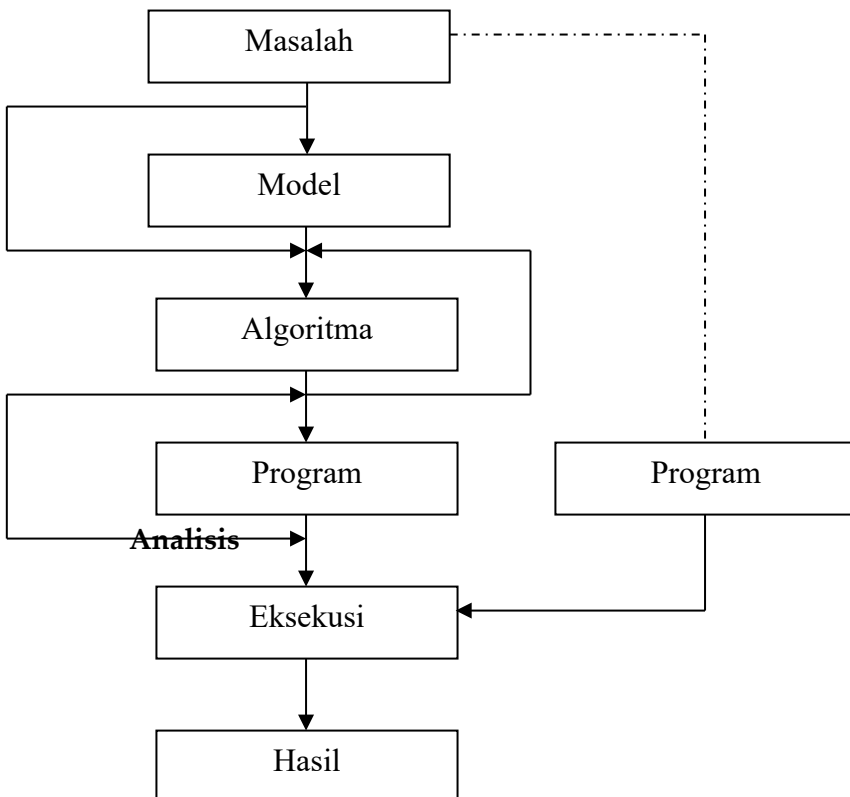
Disaat membahas mengenai algoritma di bidang pemrograman, maka yang dimaksud adalah solusi dari suatu masalah yang harus dipecahkan dengan menggunakan

komputer. Algoritma harus dibuat secara runut agar komputer mengerti dan mampu mengeksekusinya.

Sehingga untuk memahami algoritma maka bisa didefinisikan sebagai berikut

1. Tahapan atau Langkah-langkah yang dilakukan agar solusi masalah dapat diperoleh
2. Prosedur yang merupakan urutan langkah-langkah yang berintegrasi
3. Metode khusus yang digunakan untuk menyelesaikan suatu masalah yang nyata

Untuk Gambarnya adalah sebagai berikut



Pada gambar diatas bisa kita lihat Tahap pemecahan masalah adalah Proses dari masalah hingga terbentuk suatu algoritma. Tahap implementasi adalah proses penerapan algoritma hingga menghasilkan solusi. Solusi yang dimaksud adalah suatu program yang merupakan implementasi dari algoritma yang disusun

Ciri Algoritma:

- a. Algoritma memiliki logika perhitungan atau metode yang tepat dalam menyelesaikan masalah.
- b. Menghasilkan output yang tepat dan benar dalam waktu yang singkat.
- c. Algoritma harus ditulis dengan bahasa yang standar secara sistematis dan rapi sehingga tidak menimbulkan arti ganda
- d. Algoritma ditulis dengan format yang mudah dipahami dan mudah diimplementasikan ke dalam bahasa pemrograman.
- e. Semua operasi yang dibutuhkan terdefinisi dengan jelas.
- f. Semua proses dalam algoritma harus berakhir setelah sejumlah langkah dilakukan

Kriteria Pemilihan Dalam Algoritma

1. Harus Ada output

Suatu algoritma haruslah mempunyai output yang harus merupakan solusi dari masalah yang sedang diselesaikan.

2. Efektifitas dan Efisiensi

Efektif jika algoritma tersebut menghasilkan suatu solusi yang sesuai dengan masalah yang diselesaikan dalam arti algoritma harus tepat guna.

3. Jumlah langkahnya berhingga

Algoritma merupakan barisan instruksi yang dibuat harus dalam suatu urutan tertentu atau harus berhingga agar masalah yang dihadapi dapat diselesaikan dengan tidak memerlukan waktu relatif lama.

4. Harus ada Akhir

Dalam mencari penyelesaian suatu masalah maka harus berhenti dan berakhir dengan hasil akhir yang merupakan solusinya atau berupa informasi yang tidak diketemukan solusinya. Istilah lain dalam algoritma dikenal sebagai SEMI ALGORITMA, yaitu prosedur yang hanya akan berhenti jika mempunyai atau menghasilkan solusi, sedangkan jika tidak menghasilkan solusi, maka prosedur tersebut akan berjalan tanpa henti.

5. Terstruktur

Urutan barisan langkah-langkah yang digunakan harus disusun sedemikian rupa agar proses penyelesaian tidak berbelit-belit sedemikian sehingga bagian-bagian proses dapat dibedakan dengan jelas mana bagian input, proses

dan output sehingga memudahkan user melakukan pemeriksaan ulang.

Sehingga secara umum suatu algoritma harus menghasilkan output yang tepat guna atau efektif dalam waktu yang relatif singkat dan penggunaan memori yang relatif sedikit atau efisien dengan langkah yang berhingga dan prosedurnya berakhir baik dalam keadaan diperoleh suatu solusi ataupun tidak ada solusinya

Contoh untuk pembahasan algoritma yang mudah dimengerti adalah mengirim surat via email

1. Buka situs email
2. Masukkan Login dan password email
3. Klik Kirim Surat/Compose
4. Ketik di Kepada/To Alamat email yang dituju
5. Ketik Subject dari email
6. Ketik isi surat yang ingin disampaikan jika ada berkas klik Attach File/berkas
7. Jika sudah selesai klik Send/Kirim

Tahapan Analisa Algoritma

1. Bagaimana merencanakan suatu algoritma:

Menentukan beberapa model atau desain sebagai penyelesaian dari suatu masalah untuk mendapat sebuah

solusi yang mungkin. Dengan demikian, akan banyak terdapat variasi desain atau model yang dapat diambil yang terbaik.

2. **Bagaimana menyatakan suatu algoritma**

Menentukan model suatu algoritma yang digunakan sehingga dapat membuat barisan langkah secara berurutan guna mendapatkan solusi penyelesaian masalah. Menentukan model tersebut agar dapat digunakan dengan cara:

- **Dengan Bahasa semu(Pseudocode):** yaitu dengan menggunakan bahasa sehari-hari, tetapi harus jelas dan terstruktur, seperti telah penulis sebutkan pada contoh-contoh sebelumnya(Contoh prosedur berikirm surat)

Contoh:

1. Untuk mengitung Luas Lingkaran:
 2. Masukkan Jari-jari
 3. Masukkan Phi
 4. Hitung Luas = jari-jari * Phi
 5. Cetak Luas
- **Dengan diagram alur atau flowchart:** yaitu dengan membuat suatu penulisan atau penyajian algoritma berupa diagram yang menggambarkan susunan alur logika dari suatu permasalahan

Dengan Statement Program/Penggalan Program

Contoh:

1. Read Alas
2. Read Tinggi
3. $Luas=jari_jari*\phi$
4. Write(luas)

3. Bagaimana validitas suatu algoritma

Yakni jika penyelesaian memenuhi solusi yang sebenarnya, artinya solusi yang didapat merupakan penyelesaian suatu masalah dan bukannya membuat masalah baru.

4. Bagaimana menganalisa suatu algoritma

Caranya melihat running time atau waktu tempuh yang digunakan dalam menyelesaikan masalah serta jumlah memori yang digunakan dalam penyelesaian masalah tersebut.

5. Bagaimana menguji program dari suatu algoritma

Yaitu dengan cara menyajikannya dalam salah satu bahasa pemrogramana. Dalam proses, uji program oleh komputer akan melalui beberapa tahap yaitu:

1. **Fase Debugging**, yaitu fase dari suatu proses program eksekusi yang akan melakukan koreksi terhadap kesalahan program. Yang dimaksud disini

adalah error atau salah dalam penulisan program baik logika maupun sintaksnya.

2. **Fase Profilling**, yaitu fase yang akan bekerja jika program tersebut sudah benar atau telah melalui proses pada fase debugging. Fase ini bekerja untuk melihat dan mengukur waktu tempuh atau running time yang diperlukan serta jumlah memori/storage yang digunakan dalam menyelesaikan suatu algoritma.

Analisis Suatu Algoritma

(Untuk melihat faktor efisiensi & efektifitas dari algoritma tersebut), dapat dilakukan terhadap suatu algoritma dengan melihat pada:

- **Waktu tempu(Running Time) dari suatu algoritma:** adalah satuan waktu yang ditempuh atau diperlukan oleh suatu algoritma dalam menyelesaikan suatu masalah.

Hal-hal yang dapat mempengaruhi daripada waktu tempuh adalah:

1. **Banyaknya langkah:** Makin banyak langkah atau instruksi yang digunakan dalam menyelesaikan masalah, maka makin lama waktu tempuh yang dibutuhkan dalam proses tersebut

2. **Besar dan jenis input data:** Besar dan jenis input data pada suatu algoritma akan sangat berpengaruh pada proses perhitungan yang terjadi. Jika jenis data adalah tingkat ketelitian tunggal, maka waktu tempuh akan menjadi relatif lebih cepat dibandingkan dengan tingkat ketelitian ganda(double precision)
3. **Jenis operasi:** Waktu tempuh juga dipengaruhi oleh jenis operasi yang digunakan. Jenis operasi tersebut meliputi operasi matematika, nalar atau logika, atau yang lainnya. Sebagai contoh, operasi perkalian atau pembagian akan memakan waktu lebih lama dibandingkan operasi penjumlahan atau pengurangan.
4. **Komputer dan kompilator:** hal terakhir yang mempengaruhi waktu tempuh suatu proses algoritma adalah komputer dan kompilatornya, walaupun sebenarnya faktor ini diluar tahap rancangan atau tahap pembuatan algoritma yang efisien. Algoritma dibuat untuk mencapai waktu tempuh yang seefektif dan seefisien mungkin, tetapi kesemuanya itu akan sangat bergantung pada kemampuan komputer yang tentunya harus sesuai

dengan jumlah program atau langkah yang diperlukan oleh algoritma.

- **Jumlah Memori Yang digunakan:** banyaknya langkah yang digunakan dan jenis variabel data yang dipakai dalam suatu algoritma akan sangat mempengaruhi penggunaan memori. Dalam hal ini, diharapkan dapat memperkirakan seberapa banyak kebutuhan memori yang diperlukan selama proses berlangsung hingga proses selesai dikerjakan. Dengan demikian, dapat disiapkan kapasitas penyimpanan/storage yang memadai agar proses suatu algoritma berjalan tanpa ada hambatan atau kekurangan memori.

Sifat-Sifat Algoritma

- a. **Banyaknya langkah instruksi harus berhingga:** pelaksanaan sebuah algoritma yang terprogram haruslah dapat diakhiri atau diselesaikan melalui sejumlah langkah operasional yang berhingga. Jika tidak demikian, kita tidak akan dapat mengharapkan bahwa pelaksanaan algoritma tersebut dapat menghasilkan suatu solusi yang baik.
- b. **Langkah atau instruksi harus jelas:** artinya bahwa penulisan setiap langkah yang terdapat didalam sebuah

algoritma harus memiliki arti yang khusus atau spesifik sehingga dapat dibedakan antara penulisan langkah untuk komputer dengan penulisan langkah bagi manusia. Manusia akan lebih mudah memahami algoritma yang terdiri atas simbol-simbol (Contoh: pembuatan algoritma dengan diagram alur/flowchart) sedangkan komputer hanya membutuhkan sebuah penulisan algoritma dengan kode-kode yang dituangkan dalam bahasa yang dimengerti oleh komputer itu sendiri (bahasa pemrograman).

- c. **Proses harus jelas dan mempunyai batasan:** rangkaian suatu proses yang berisi langkah-langkah instruksi dari suatu algoritma yang akan dilaksanakan harus ditetapkan dengan jelas, baik dan pasti sebab sebuah algoritma harus memiliki instruksi dasar tertentu dimana setiap instruksi harus memiliki unsur pelaksana yang berfungsi sebagai pemroses data yang akan dimasukkan dalam sebuah komputer. Dengan demikian, sebuah algoritma harus ditulis dengan jelas tentang batasan-batasan proses yang akan dilaksanakan oleh komputer.
- d. **Input dan Output harus mempunyai batasan:** input merupakan data yang dimasukkan ke dalam algoritma yang untuk kemudian akan dilaksanakan oleh komputer. Dengan begitu, input yang diberikan harus sesuai dengan

jenis dari bahasa pemrograman yang digunakan, sedangkan output merupakan hasil yang diperoleh dari pekerjaan yang dilaksanakan komputer untuk kepentingan user yang merupakan pihak diluar komputer. Algoritma harus menghasilkan output karena merupakan solusi yang diharapkan dari suatu masalah yang timbul.

- e. **Efektifitas:** instruksi yang diberikan pada komputer agar hanya menjalankan atau melaksanakan proses yang mampu dilaksanakannya. Yang dimaksud mampu adalah bahwa suatu algoritma atau instruksi-instruksi dalam sebuah program hanya akan dapat dilaksanakan jika informasi yang diberikan oleh instruksi-instruksi tersebut lengkap, benar dan jelas.
- f. **Adanya batasan ruang lingkup,** sebuah algoritma yang baik adalah hanya ditujukan bagi suatu masalah tertentu saja. Susunana input harus ditentukan lebih dulu sebab susunan tersebut enentukan sifat umum dari algoritma yang bersangkutan.



BAB 2

KONSEP DASAR PEMROGRAMAN

2.1 Pengertian Algoritma

Algoritma merupakan istilah dari “algoritma” yang berasal dari matematikan Muslim bernama Ibnu Kharazmi di abad ke-19. Dalam konteks komputer, algoritma sering digunakan untuk merancang program komputer. Algoritma dapat ditulis menggunakan berbagai bahasa pemrograman dan memberikan petunjuk tentang apa yang harus dilakukan oleh komputer untuk menyelesaikan suatu tugas. Algoritma adalah urutan langkah-langkah yang terdefinisi dengan jelas dan logis untuk menyelesaikan suatu masalah atau mencapai suatu tujuan. Secara umum, algoritma adalah instruksi yang sistematis yang digunakan untuk memecahkan masalah atau menjalankan tugas tertentu. Algoritma sering digunakan dalam bidang ilmu komputer dan matematika, tetapi juga dapat diterapkan dalam berbagai bidang lainnya.

Algoritma haruslah jelas dan terstruktur sehingga dapat diikuti dan dimengerti oleh mesin atau manusia. Setiap langkah dalam algoritma haruslah jelas dan memiliki tujuan yang terdefinisi dengan baik. Algoritma juga harus mampu menghasilkan output yang diharapkan dalam kondisi masukan yang tepat.

Algoritma dapat berupa serangkaian instruksi dasar seperti pengulangan (looping), pemilihan (selection), pengambilan keputusan, atau pemrosesan data. Instruksi-instruksi tersebut biasanya dituliskan dalam bentuk pseudocode atau dalam bahasa pemrograman yang dapat dimengerti oleh komputer.

Algoritma memiliki peran penting dalam pengembangan perangkat lunak, pemrosesan data, kecerdasan buatan, dan banyak aplikasi lainnya. Dengan menggunakan algoritma yang tepat, kita dapat mencapai solusi yang efisien dan akurat untuk berbagai masalah yang kompleks.

Algoritme juga disebut sebagai kumpulan program yang mengimplementasikan atau mengekspresikan tahapan pemecahan masalah. Himpunan semua program dipartisi menjadi kelas-kelas yang setara. Dua program setara jika mereka pada dasarnya adalah program yang sama. Himpunan kelas kesetaraan membentuk kategori algoritma. Meskipun kumpulan program bahkan tidak membentuk kategori,

kumpulan algoritma membentuk kategori dengan struktur tambahan. Kondisi tersebut menggambarkan bahwa dua program yang pada dasarnya sama berubah menjadi hubungan koherensi yang memperkaya kategori algoritme dengan struktur ekstra sehingga properti universal dari kategori algoritma terbukti (Yanofsky, 2011).

Algoritma yang baik biasanya harus memenuhi beberapa kriteria, seperti akurat, efisien, dapat dipahami, dan dapat diimplementasikan.

2.2 Fungsi Algoritma

Kegunaan atau fungsi utama dari algoritma pemrograman yaitu untuk memecahkan permasalahan. Algoritma adalah sesuatu hal yang sangat penting dalam kegiatan menciptakan sebuah program. Berikut merupakan fungsi-fungsi dari algoritma pada pemrograman yaitu sebagai berikut:

1. Menyelesaikan masalah: Algoritma membantu dalam merancang langkah-langkah sistematis dan terstruktur untuk menyelesaikan masalah yang kompleks. Dengan menggunakan algoritma yang tepat, kita dapat memecahkan masalah dengan efisien dan akurat.
2. Mengoptimalkan kinerja: Algoritma dapat digunakan untuk mengoptimalkan kinerja suatu proses atau sistem. Dengan merancang algoritma yang efisien, kita dapat

mencapai hasil yang diinginkan dengan penggunaan sumber daya yang minimal.

3. Automatisasi: Algoritma digunakan untuk mengotomatisasi tugas-tugas yang berulang dan rutin. Dengan merancang algoritma yang tepat, kita dapat menginstruksikan komputer atau sistem lainnya untuk menjalankan tugas-tugas tersebut tanpa intervensi manusia secara terus-menerus.
4. Pengambilan keputusan: Algoritma juga digunakan untuk membantu dalam pengambilan keputusan. Dengan menggunakan data dan kriteria yang relevan, algoritma dapat memberikan petunjuk atau rekomendasi untuk membantu pengambilan keputusan yang lebih baik.
5. Pengamanan data: Algoritma kriptografi digunakan untuk melindungi data dan informasi penting dari akses yang tidak sah atau perubahan yang tidak diinginkan. Algoritma ini mengenkripsi data dengan menggunakan kunci kriptografi sehingga hanya pihak yang memiliki kunci yang tepat dapat mengakses dan membaca informasi tersebut.
6. Analisis data: Algoritma juga digunakan dalam analisis data untuk mengidentifikasi pola, tren, dan wawasan penting dari volume data yang besar. Algoritma analisis

data membantu dalam mengekstraksi informasi yang berharga dari data yang kompleks dan memungkinkan pengambilan keputusan yang didasarkan pada bukti dan fakta.

7. Fungsi-fungsi tersebut menunjukkan pentingnya algoritma dalam berbagai aspek kehidupan dan teknologi. Dengan menggunakan algoritma yang tepat, kita dapat mencapai efisiensi, keamanan, dan solusi yang diinginkan dalam berbagai konteks.

2.3 Struktur Algoritma

Struktur algoritma adalah kerangka atau pola yang digunakan untuk merancang algoritma. Berikut adalah beberapa struktur algoritma umum yang sering digunakan:

1. Struktur Sekuensial.

Struktur ini adalah struktur algoritma paling dasar. Urutan instruksi dieksekusi satu per satu secara berurutan dari awal hingga akhir. Contoh struktur sekuensial adalah sebagai berikut :

Algoritma menghitung luas persegi.

1. Mulai
2. Inisialisasi variabel panjang, lebar dan luas
3. Masukkan Nilai variabel panjang dan lebar
4. Hitung luas dengan rumus $\text{luas} = \text{panjang} * \text{lebar}$;
5. Tampilkan Nilai Luas
6. Selesai.

2. Struktur Percabangan (Conditional):

Struktur ini digunakan ketika kita perlu mengambil keputusan berdasarkan kondisi tertentu. Contohnya adalah pernyataan if-else, switch-case, atau pernyataan bersarang (nested statement).

Jika kondisi A terpenuhi, maka

Langkah 1: Melakukan tindakan 1

Langkah 2: Melakukan tindakan 2

Jika kondisi B terpenuhi, maka

Langkah 3: Melakukan tindakan 3

Jika tidak ada kondisi yang terpenuhi, maka

Langkah 4: Melakukan tindakan 4

Langkah5: Melanjutkan eksekusi program

3. Struktur Perulangan (Looping):

Digunakan ketika kita perlu mengulangi serangkaian instruksi. Contoh struktur perulangan termasuk for-loop, while-loop, do-while-loop, atau pernyataan bersarang (nested loop). Contoh algo

Ritma perulangan adalah sebagai berikut :

a. Perulangan dengan while

Inisialisasi variabel

while kondisi:

Langkah-langkah perulangan

Memperbarui variabel

Langkah setelah perulangan

b. Perulangan dengan for

for variabel in rentang:

Langkah-langkah perulangan

Langkah setelah perulangan

4. Struktur Pemanggilan Fungsi.

Dalam banyak bahasa pemrograman, fungsi digunakan untuk mengelompokkan serangkaian instruksi yang dapat dipanggil dari tempat lain dalam program. Struktur ini melibatkan pemanggilan fungsi dan pengembalian nilai. Contoh algoritma pemanggilan fungsi yaitu :

FungsiA(argumen):

Langkah-langkah di dalam fungsi A

Return nilai

FungsiB(argumen):

Langkah-langkah di dalam fungsi B

Return nilai

Langkah 1: Memanggil FungsiA(nilai_argumen)

Langkah 2: Menyimpan hasil pemanggilan FungsiA ke
dalam variabel hasil

Langkah 3: Memanggil FungsiB(hasil)

Langkah 4: Melakukan tindakan lain dengan
menggunakan hasil FungsiB

Dalam contoh diatas, terdapat dua fungsi yang disebutkan, yaitu FungsiA dan FungsiB. FungsiA dan FungsiB menerima argumen sebagai input, melakukan langkah-langkah di dalamnya, dan mengembalikan nilai. Langkah-langkah di dalam fungsi tersebut akan dieksekusi ketika fungsi dipanggil. Hasil pemanggilan fungsi dapat disimpan dalam variabel untuk digunakan di langkah-langkah berikutnya. Setelah pemanggilan fungsi, program dapat melanjutkan dengan langkah-langkah lainnya.

5. Struktur Rekursi.

Rekursi adalah ketika suatu fungsi memanggil dirinya sendiri. Ini digunakan untuk menyelesaikan masalah yang dapat dipecahkan dengan cara berulang atau terbagi menjadi submasalah yang lebih kecil. Contoh algoritma rekursi adalah sebagai berikut :

FungsiRekursif(argumen):

 Jika basis rekursi tercapai, maka

 Return nilai basis

 Jika tidak, maka

 Langkah-langkah rekursif

 Panggil kembali FungsiRekursif dengan argument
 berbeda

 Return hasil rekursif

Langkah 1: Memanggil FungsiRekursif(nilai_argumen)

Pada contoh diatas, terdapat fungsi rekursif yang disebut FungsiRekursif. Fungsi ini memerlukan argumen sebagai input. Pada setiap pemanggilan, fungsi memeriksa basis rekursi, yaitu kondisi di mana rekursi berhenti dan nilai basis dikembalikan. Jika basis rekursi belum tercapai, maka fungsi akan melakukan langkah-langkah rekursif dengan memanggil kembali dirinya sendiri dengan argumen yang berbeda.

6. Struktur Penanganan Kesalahan (Error Handling):

Digunakan untuk menangani situasi atau kondisi yang tidak diinginkan atau kesalahan dalam program. Ini termasuk penggunaan try-catch atau pernyataan lain untuk mengelola pengecualian (exceptions) dan menangani kesalahan. Contoh algoritma penanganan kesalahan adalah sebagai berikut :

Langkah 1: Mulai eksekusi program

Langkah 2: Coba

Langkah-langkah yang mungkin menimbulkan kesalahan

Jika terjadi kesalahan, lempar kesalahan

Jika tidak ada kesalahan, lanjutkan ke langkah berikutnya

Langkah 3: Tangkap kesalahan

Jika kesalahan terjadi, tangkap kesalahan yang dilempar

Langkah-langkah penanganan kesalahan

Langkah 4: Melanjutkan eksekusi program

Dalam algoritma ini, langkah-langkah yang dapat menimbulkan kesalahan dikelompokkan dalam blok "Coba". Jika kesalahan terjadi, kesalahan tersebut dilempar untuk ditangkap di blok "Tangkap kesalahan". Di dalam blok "Tangkap kesalahan", langkah-langkah penanganan kesalahan dilakukan, seperti mencetak pesan kesalahan atau melakukan tindakan pemulihan yang sesuai. Setelah penanganan kesalahan, eksekusi program dapat melanjutkan ke langkah-langkah berikutnya.

Berbagai struktur algoritma dapat dikombinasikan dan disusun sesuai dengan kebutuhan program yang sedang dirancang. Struktur algoritma dapat digabungkan dan disusun sesuai dengan kebutuhan dan kompleksitas masalah yang ingin diselesaikan. Penggunaan struktur yang tepat membantu dalam merancang algoritma yang efisien dan mudah dipahami.

2.4 Efisiensi Algoritma

Dalam ilmu komputer, efisiensi algoritmik adalah properti dari suatu algoritme yang berkaitan dengan jumlah sumber daya komputasi yang digunakan oleh algoritme tersebut. Suatu algoritma harus dianalisis untuk menentukan penggunaan sumber dayanya, dan efisiensi suatu algoritma dapat diukur berdasarkan penggunaan sumber daya yang berbeda. Efisiensi algoritmik dapat dianggap analog dengan produktivitas rekayasa untuk proses yang berulang atau berkelanjutan.

Untuk efisiensi maksimum, diinginkan untuk meminimalkan penggunaan sumber daya. Namun, sumber daya yang berbeda seperti kompleksitas waktu dan ruang tidak dapat dibandingkan secara langsung, sehingga mana dari dua algoritma yang dianggap lebih efisien seringkali bergantung pada ukuran efisiensi mana yang dianggap paling penting.

Dua hal yang sering digunakan untuk mengukur efisiennya algoritma :

- a. Waktu: berapa lama waktu yang diperlukan algoritme untuk menyelesaikannya?
- b. Ruang: berapa banyak memori kerja (biasanya RAM) yang dibutuhkan oleh algoritme?

Untuk komputer yang dayanya disuplai oleh baterai (mis. laptop dan smartphone), atau untuk perhitungan yang sangat panjang/besar (mis. superkomputer), pengukuran lainnya adalah :

- a. Konsumsi daya langsung: daya yang dibutuhkan secara langsung untuk mengoperasikan komputer.
- b. Konsumsi daya tidak langsung: daya yang dibutuhkan untuk pendinginan, penerangan, dll.

Ukuran efisiensi komputasi yang kurang umum mungkin juga relevan dalam beberapa kasus:

- a. Ukuran transmisi: bandwidth bisa menjadi faktor pembatas. Kompresi data dapat digunakan untuk mengurangi jumlah data yang akan dikirim. Menampilkan gambar atau gambar (misalnya logo Google) dapat mengakibatkan pengiriman puluhan ribu byte (dalam hal ini 48K) dibandingkan dengan pengiriman enam byte untuk teks "Google". Ini penting untuk tugas komputasi yang terikat I/O.
- b. Ruang eksternal: ruang yang dibutuhkan pada disk atau perangkat memori eksternal lainnya; ini bisa untuk penyimpanan sementara saat algoritme sedang dilakukan, atau bisa juga penyimpanan jangka panjang perlu dilakukan untuk referensi di masa mendatang.

- c. Waktu respons (latensi): ini sangat relevan dalam aplikasi waktu nyata ketika sistem komputer harus merespons dengan cepat beberapa peristiwa eksternal.
- d. Total biaya kepemilikan: terutama jika komputer didedikasikan untuk satu algoritme tertentu.



BAB 3

SEJARAH FLOWCHART

Konsep dasar dari flowchart pertama kali muncul pada tahun 1921 oleh seorang insinyur dan ilmuwan komputer bernama Frank Gilbreth Sr. dan istrinya, Lillian Gilbreth. Mereka mengembangkan simbol-simbol grafis untuk menggambarkan gerakan dan aktivitas dalam proses produksi.

Pada tahun 1940-an, flowchart mulai digunakan secara luas oleh industri untuk mendokumentasikan proses produksi dan alur kerja. Flowchart digunakan sebagai alat komunikasi visual untuk memperjelas langkah-langkah yang terlibat dalam suatu proses.

Pada tahun 1950-an, flowchart mulai digunakan dalam analisis dan pemrograman komputer. Metode ini membantu programmer dalam merencanakan dan memvisualisasikan langkah-langkah dalam algoritma dan program komputer.

Pada tahun 1960-an, American National Standards Institute (ANSI) dan International Organization for Standardization (ISO) mulai mengembangkan standar untuk simbol-simbol flowchart. Hal ini membantu menyatukan penggunaan simbol-simbol dalam dokumentasi dan komunikasi proses.

Seiring dengan perkembangan komputer dan perangkat lunak, flowchart menjadi lebih terintegrasi dalam alat bantu pemrograman. Perangkat lunak seperti Microsoft Visio, Lucidchart, dan beberapa alat pemrograman menyediakan fitur flowchart yang memungkinkan pembuatan dan dokumentasi proses dengan mudah.

Flowchart juga dapat digunakan dalam mendesign database perusahaan untuk membuat susunan / kumpulan data operasional lengkap dari suatu organisasi/perusahaan yang diorganisir / dikelola dan simpan secara terintegrasi dengan menggunakan metode tertentu, dengan menggunakan komputer, sehingga mampu menyediakan informasi yang optimal diperlukan pemakainya. (Eri Mardiani, 2020)

Hingga saat ini, flowchart tetap menjadi alat yang penting dalam analisis proses, perencanaan sistem, dokumentasi program, dan komunikasi visual. Flowchart terus digunakan untuk menggambarkan alur kerja, algoritma, dan prosedur

dalam berbagai bidang seperti bisnis, rekayasa, pemrograman komputer, dan manajemen proyek.

Pengertian Flowchart

Flowchart adalah sebuah diagram visual yang digunakan untuk menggambarkan urutan langkah-langkah atau proses dalam suatu sistem atau program. Flowchart menggambarkan alur kerja atau aliran informasi dari satu langkah ke langkah lainnya dengan menggunakan simbol-simbol grafis dan panah-panah yang menghubungkannya.

Tujuan Flowchart

Tujuan utama dari flowchart adalah untuk menggambarkan proses secara jelas dan mudah dimengerti oleh orang yang melihatnya. Flowchart dapat digunakan untuk menggambarkan berbagai jenis proses, seperti alur kerja bisnis, algoritma komputer, instruksi pemrograman, prosedur operasional, dan banyak lagi.

Dengan menggunakan flowchart, seseorang dapat dengan mudah memahami urutan langkah-langkah yang harus dilakukan dalam suatu proses dan mengidentifikasi bagian-bagian yang mungkin memerlukan perbaikan atau optimisasi. Flowchart juga berguna untuk mengkomunikasikan proses

kepada orang lain, seperti tim kerja, pengembang, atau pemangku kepentingan lainnya.

Flowchart dapat dibuat menggunakan berbagai jenis perangkat lunak, yang berbayar maupun opensource diantaranya : Draw.io Microsoft Visio, perangkat lunak pemrograman seperti Visual Studio, atau bahkan menggunakan alat tulis tradisional seperti kertas dan pena.

Jenis Flowchart

Berikut ini adalah beberapa jenis flowchart yang umum digunakan:

Flowchart Proses: Jenis flowchart ini digunakan untuk menggambarkan urutan langkah-langkah atau proses secara keseluruhan. Ia menunjukkan bagaimana suatu proses dimulai, langkah-langkah yang dilakukan di antara, dan bagaimana proses tersebut berakhir.

Flowchart Keputusan: Flowchart ini digunakan untuk menggambarkan cabang keputusan dalam suatu proses. Simbol yang digunakan adalah diamond (berbentuk seperti permata) untuk menunjukkan suatu keputusan. Panah-panah mengarah ke cabang yang berbeda tergantung pada keputusan yang diambil.

Flowchart Pengolahan Data: Flowchart ini digunakan untuk menggambarkan aliran pengolahan data dalam suatu

proses. Ia menunjukkan langkah-langkah pemrosesan data, seperti pengambilan data, manipulasi data, dan output hasil.

Flowchart Dokumen: Jenis flowchart ini digunakan untuk menggambarkan alur dokumen atau informasi dalam suatu proses. Ia menunjukkan bagaimana dokumen atau informasi berpindah dari satu tahap ke tahap berikutnya.

Flowchart Sistem: Flowchart ini digunakan untuk menggambarkan sistem secara keseluruhan, termasuk komponen-komponen utama dan interaksinya. Ia memberikan gambaran visual tentang bagaimana sistem bekerja secara keseluruhan.

Flowchart Algoritma: Flowchart ini digunakan dalam pemrograman komputer untuk menggambarkan langkah-langkah dalam suatu algoritma atau program. Ia menunjukkan aliran eksekusi dari satu langkah ke langkah berikutnya.

Flowchart Manajemen Proyek: Flowchart ini digunakan untuk menggambarkan alur kerja dalam manajemen proyek. Ia menunjukkan tugas-tugas, tanggung jawab, dan ketergantungan antara aktivitas-aktivitas dalam proyek.

Flowchart Arus Data: Flowchart ini digunakan untuk menggambarkan alur data dalam sistem atau proses. Ia menunjukkan bagaimana data masuk ke sistem, diproses, dan menghasilkan output.

Setiap jenis flowchart memiliki simbol-simbol khusus yang digunakan untuk menggambarkan langkah-langkah, keputusan, input/output, dan koneksi antar langkah. Simbol-simbol tersebut membantu dalam memahami dan menganalisis alur kerja atau proses yang dijelaskan oleh flowchart tersebut.

Pedoman membuat flowchart

1. Tentukan Tujuan: Pahami dengan jelas tujuan dari flowchart yang akan Anda buat. Apakah itu untuk mendokumentasikan proses, merencanakan algoritma, atau mengkomunikasikan langkah-langkah dalam suatu sistem? Memahami tujuan akan membantu Anda fokus pada informasi yang ingin disampaikan.
2. Identifikasi Langkah-langkah atau Proses: Identifikasi langkah-langkah atau proses yang ingin Anda gambarkan dalam flowchart. Tentukan urutan logis dari langkah-langkah tersebut dan pastikan untuk memperhatikan aliran data atau informasi antara langkah-langkah tersebut.
3. Pilih Simbol dan Notasi yang Sesuai: Gunakan simbol dan notasi yang sesuai dengan standar yang diterima umum atau yang sudah ditetapkan dalam organisasi atau proyek Anda. Pastikan simbol yang digunakan

mudah dipahami oleh semua orang yang melihat flowchart.

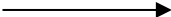
4. **Buat Alur yang Jelas:** Pastikan alur flowchart yang Anda buat mudah diikuti. Gunakan panah yang jelas untuk menghubungkan langkah-langkah, keputusan, dan aliran data. Hindari alur yang bertele-tele atau ambigu yang dapat menyebabkan kebingungan.
5. **Gunakan Keterangan dan Label yang Jelas:** Berikan keterangan atau label yang jelas pada setiap simbol dan panah dalam flowchart. Hal ini akan membantu orang lain memahami dengan mudah setiap langkah atau keputusan yang diwakili oleh simbol atau panah tersebut.
6. **Pertimbangkan Tingkat Rincian:** Sesuaikan tingkat rincian flowchart dengan kebutuhan dan konteksnya. Jika flowchart digunakan untuk dokumentasi tingkat tinggi, Anda mungkin ingin menjelaskan langkah-langkah secara umum. Namun, jika flowchart digunakan untuk pemrograman atau analisis yang lebih mendetail, Anda mungkin perlu memberikan langkah-langkah yang lebih spesifik.
7. **Review dan Uji Flowchart:** Setelah membuat flowchart, review kembali untuk memastikan kejelasan dan ketepatan informasi yang disajikan. Uji flowchart



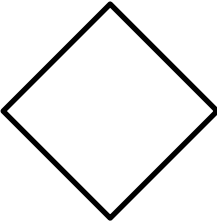
dengan melibatkan orang lain yang memiliki pengetahuan tentang proses yang digambarkan atau mereka yang akan menggunakan flowchart tersebut.



- Gunakan Perangkat Lunak atau Alat Bantu: Pertimbangkan menggunakan perangkat lunak atau alat bantu untuk membuat flowchart. Ada banyak perangkat lunak dan alat online yang menyediakan fitur flowchart yang memudahkan pembuatan, penyuntingan, dan pembagian flowchart.



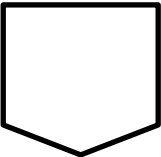
Simbol flowchart

Simbol-simbol ini digunakan secara kombinasi dalam flowchart untuk menggambarkan langkah-langkah, keputusan, input/output, aliran data, dan koneksi dalam alur proses atau program. Kombinasi dan penggunaan simbol-simbol ini membantu dalam menyusun flowchart yang jelas dan mudah dipahami.

Gambar	Nama	Keterangan
	Aliran Data / Flowline	Simbol Berbentuk Panah menunjukkan arah aliran data. Digunakan untuk menggambarkan aliran

		data dari satu langkah ke langkah lainnya.
	Terminal / Terminator	Simbol berbentuk oval atau persegi panjang dengan sudut bulat. Digunakan untuk menunjukkan awal dan akhir dari alur proses atau program.
	Proses / Process	Simbol berbentuk persegi panjang. Digunakan untuk menunjukkan langkah-langkah atau tindakan yang dilakukan dalam proses atau program.
	Keputusan / Decision	berbentuk diamond (permata). Digunakan untuk menunjukkan cabang keputusan dalam alur proses. Panah keluar dari simbol keputusan mengarah ke langkah berikutnya berdasarkan

		hasil keputusan yang diambil.
	Konektor	berbentuk lingkaran kecil yang terhubung dengan panah. Digunakan untuk menghubungkan langkah-langkah dalam alur proses yang terpisah secara fisik, tetapi saling terkait secara logis.
	Input/Output	berbentuk persegi panjang dengan sudut bulat di satu sisi. Digunakan untuk menunjukkan input data atau output hasil dari proses.
	Predefined Process	berbentuk persegi panjang dengan sudut dipotong. Digunakan untuk menunjukkan langkah-langkah yang sudah ditetapkan atau proses yang telah

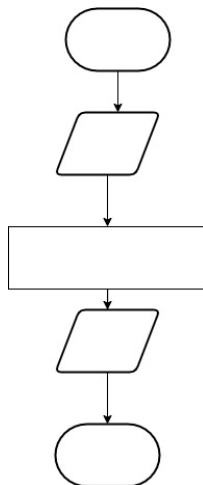
		didefinisikan sebelumnya.
	Dokumen	berbentuk persegi panjang dengan sudut bersudut. Digunakan untuk menunjukkan dokumen atau informasi yang diproses dalam alur kerja.
	Penggabungan (Merge) dan Pemisahan (Split)	berbentuk panah dengan label yang menunjukkan penggabungan atau pemisahan aliran proses.
	Off-page Connector	berbentuk lingkaran dengan huruf atau angka di dalamnya. Digunakan untuk menghubungkan langkah-langkah atau bagian-bagian flowchart yang berada di halaman yang berbeda.

Tabel 3.1 Diagram flowchart

Flowchart terdiri dari tiga struktur

1. Struktur squence / Struktur sederhana

Diagram ini untuk menggambarkan langkah-langkah yang dijalankan secara berurutan, satu per satu, tanpa adanya percabangan atau pengulangan. Langkah-langkah tersebut dieksekusi secara berurutan dari atas ke bawah.

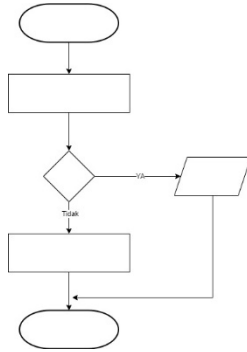


Gambar 3.1 Squence Diagram

2. Struktur Branching / Decision / Keputusan

Diagram ini digunakan untuk menggambarkan percabangan atau pemilihan langkah berdasarkan suatu kondisi atau kriteria tertentu. Biasanya, terdapat sebuah pertanyaan atau kondisi

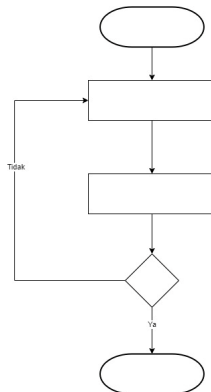
yang mengarahkan alur program ke langkah-langkah yang berbeda, tergantung pada hasil keputusan.



Gambar 3.2 Decision Diagram

3. Struktur Looping / Pengulangan

Diagram ini digunakan untuk menggambarkan langkah-langkah yang diulang secara berulang selama kondisi tertentu terpenuhi. Dengan menggunakan struktur pengulangan, langkah-langkah dapat dijalankan beberapa kali sampai kondisi yang ditentukan terpenuhi.



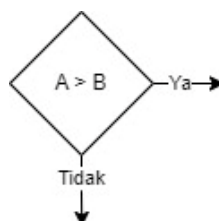
Gambar 3.3 Loop Diagram

4. Struktur Branching pada flowchart digunakan untuk menggambarkan percabangan atau pengambilan keputusan dalam alur program atau proses. Struktur ini memungkinkan program untuk melakukan langkah-langkah yang berbeda tergantung pada hasil dari kondisi atau pilihan yang ditentukan. Struktur dari percabangan yang bersyarat dan tidak bersyarat diantaranya :

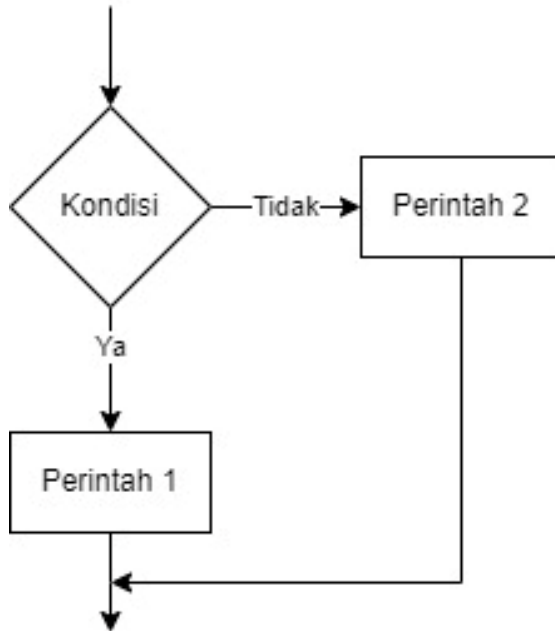
1. Bersyarat : **If... , If Else, Nested If atau If else Majemuk dan Switch ... Case**

2. Tidak Bersyarat : Go To

1. Struktur If..., dan If... Else: Struktur ini digunakan untuk membuat keputusan berdasarkan kondisi yang bernilai benar atau salah (true atau false). Jika kondisi terpenuhi (true), langkah-langkah dalam blok "if" akan dieksekusi. Jika kondisi tidak terpenuhi (false), langkah-langkah dalam blok "else" akan dieksekusi sebagai alternatif.

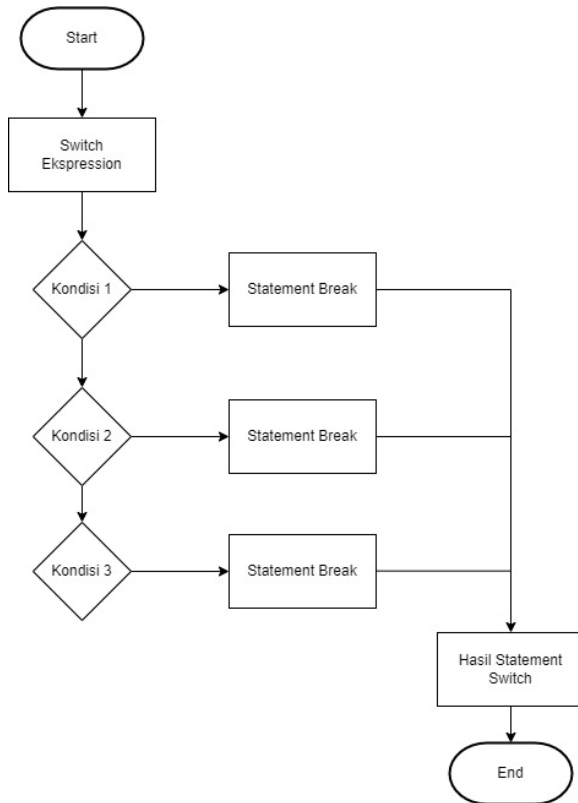


Gambar 3.4 Struktur IF



Gambar 3.5 Struktur If Else

2. Struktur Nested IF atau If Else Majemuk struktur ini digunakan untuk membuat keputusan berdasarkan beberapa kondisi yang berbeda. Setiap kondisi akan diuji secara berurutan, dan langkah-langkah dalam blok yang sesuai dengan kondisi yang terpenuhi akan dieksekusi. Jika tidak ada kondisi yang terpenuhi, langkah-langkah dalam blok "else" terakhir akan dieksekusi sebagai alternatif.



Gambar 3.6 Nested If / If Else Majemuk

3. Struktur switch ... case struktur ini digunakan untuk melakukan seleksi berdasarkan nilai tertentu. Nilai tersebut akan dibandingkan dengan beberapa kondisi yang mungkin terjadi, dan langkah-langkah yang sesuai dengan nilai tersebut akan dieksekusi.

4. Struktur Goto struktur ini digunakan untuk mengarahkan alur program atau proses ke langkah atau bagian tertentu dalam flowchart. Ini memungkinkan untuk melompati langkah-langkah di antara dan langsung menuju tujuan yang ditentukan.

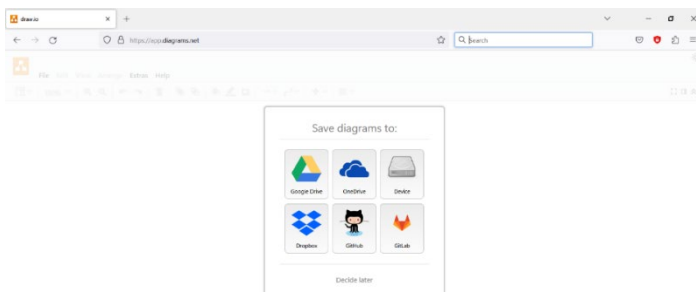
Aplikasi Pembuatan Diagram Flowchart

Pada materi ini saya menggunakan aplikasi draw.io, dapat diakses menggunakan kata kunci draw.io atau link website <https://app.diagrams.net/>.

Aplikasi ini memiliki 2 pilihan online dan offline memiliki banyak fitur dan kemudahan untuk dipergunakan, pada tulisan ini saya menggunakan sistem online.

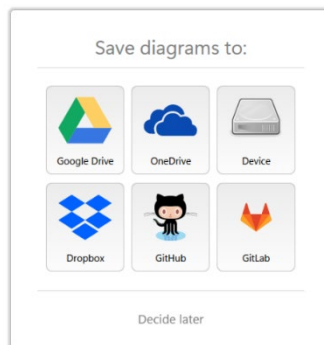
Jika anda ingin menggunakan aplikasi ini secara offline bisa diakses pada link berikut ini <https://sourceforge.net/projects/diagrams-net.mirror/>

Jika halaman sudah diakses maka akan tampil berikut ini



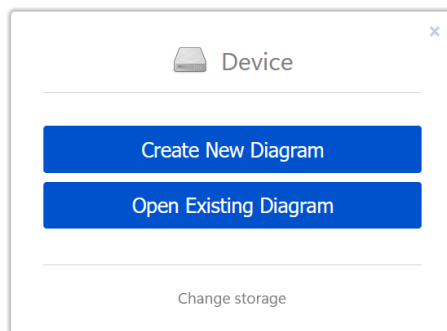
Gambar 3.7 Tampilan Website Diagram.Net

Pada bagian ini anda dapat memilih 6 tempat penyimpanan Google Drive, OneDrive, Device, Dropbox, Github dan Gitlab. Jika ingin menggunakan penyimpanan online maka dipersilahkan login ke website penyimpanan online salah satu dari kelima jasa penyedia layanan, jika tidak memiliki ID maka silahkan gunakan pilihan device pada menu yang tersedia.



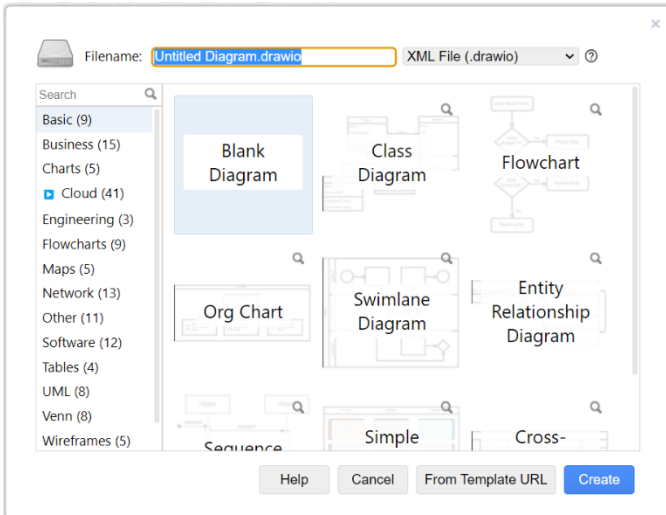
Gambar 3.8 Menu penyimpanan diagram

Disini penulis menggunakan device sebagai media penyimpanan data, jika sudah di klik akan muncul menu dibawah ini :



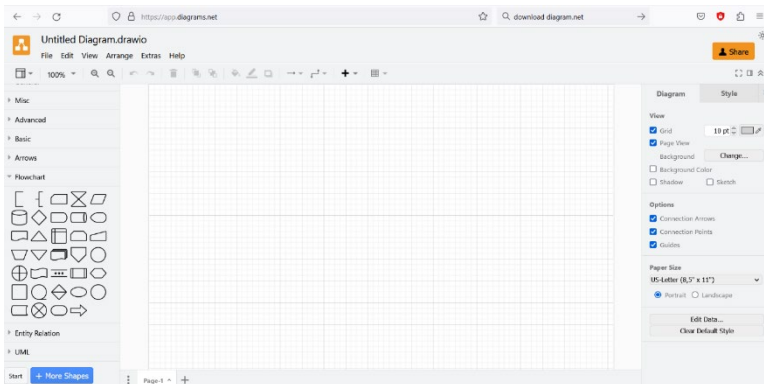
Gambar 3.9 Membuat Diagram Baru

Klik Create New Diagram untuk membuat diagram baru pada aplikasi ini tersedia beragam diagram.



Gambar 3.10 Pilihan Diagram Baru

Pada buku ini penulis menggunakan blank diagram, maka akan tampil berikut ini



Gambar 3.11 Tampilan Utama Aplikasi Draw.io

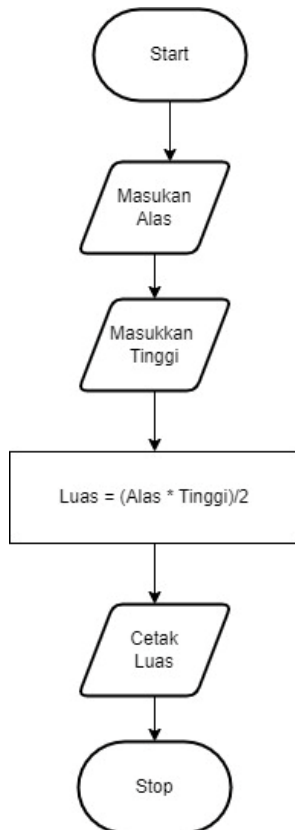
Keterangan menu :

1. Menu Bar : Menu untuk modifikasi diagram
2. Tool bar : Menu untuk mengkonfigurasi diagram
3. Diagram : Menu untuk memilih diagram
- 4 Tab Diagram dan Style : Menu untuk memodifikasi tulisan serta ukuran diagram
5. More Shape : Untuk Menambahkan bentuk lain yang belum tersedia
6. Tab : untuk menambahkan tab dalam pembuatan diagram.

Contoh Flowchart

Contoh 1 Membuat diagram flowchart menghitung Luas Segi tiga :

1. Masukan Nilai Alas
2. Masukan Nilai Tinggi
3. Hitung Luas $= (\text{Alas} * \text{Tinggi}) / 2$
4. Cetak Luas

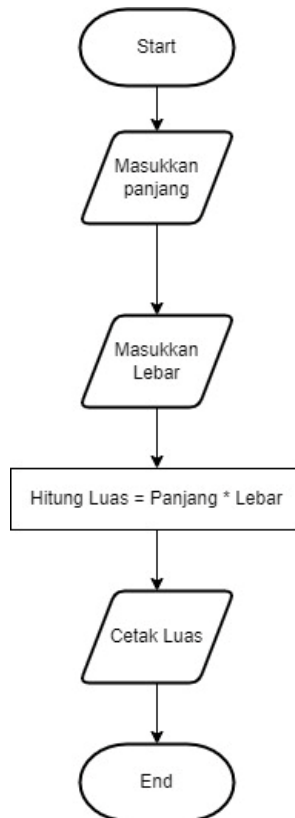


Gambar 3.12 Diagram Hitung Luas Segitiga

Contoh 2 Membuat diagram flowchart menghitung Luas Persegi Panjang :

1. Masukan Nilai Panjang
2. Masukan Nilai Lebar
3. Hitung Luas = Panjang * Lebar

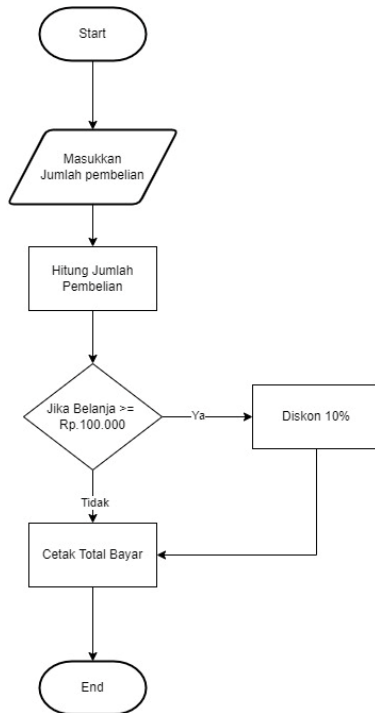
4. Cetak Luas



Gambar 3.13 Diagram Hitung Luas Persegi Panjang

Contoh 3 Membuat diagram flowchart hitung belanja barang

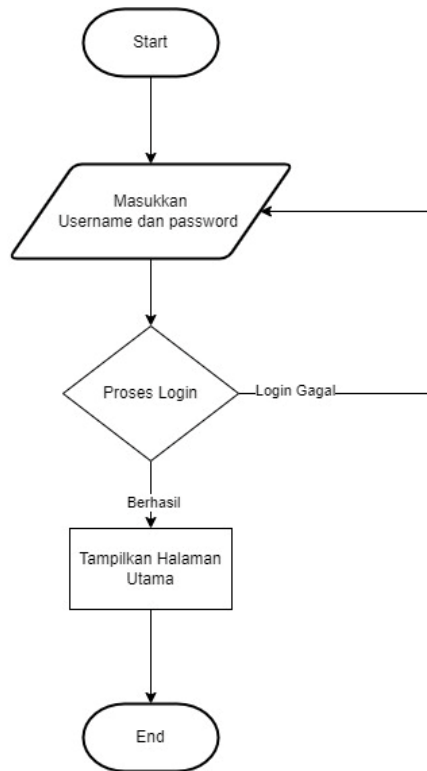
1. Masukan Jumlah barang pembelian
2. Hitung bayar = harga * Jumlah barang
3. Jika belanja ≥ 100.000 maka diberikan discount 10%, selain dari itu tidak mendapat discount
4. Cetak total bayar



Gambar 3.14 Diagram Hitung Belanja barang

Contoh 4 Membuat diagram flowchart login

1. Masukan User Name dan Password
2. Proses Login
3. Jika berhasil maka tampil halaman utama jika gagal maka kembali ke halaman login
4. tampilkan halaman utama



Gambar 3.14 Diagram Hitung Belanja barang



BAB 4

TIPE DATA, OPERATOR DAN EKSPRESI

Dalam dunia pemrograman, pemahaman yang kuat tentang tipe data, variabel, ekspresi, dan operator adalah landasan yang penting untuk menguasai suatu bahasa pemrograman. Program komputer biasanya bekerja dengan berbagai jenis atau tipe data dan membutuhkan cara untuk menyimpan nilai yang digunakan. Nilai-nilai ini dapat berupa angka atau karakter. Bahasa C memiliki dua cara untuk menyimpan nilai angka, yaitu variabel dan konstanta. Variabel adalah lokasi penyimpanan data yang memiliki nilai yang dapat berubah selama eksekusi program. Sebaliknya, konstanta memiliki nilai tetap yang tidak dapat berubah. Konstanta dan variabel merupakan elemen dasar dari setiap program.

Pada bab ini akan membahas secara rinci tentang berbagai tipe data yang tersedia dalam bahasa C, bagaimana mendeklarasikan variabel dan menginisialisasi-nya dengan nilai, serta cara menggunakan ekspresi dan operator untuk

melakukan operasi yang diperlukan. Selain itu, juga akan diberikan contoh-contoh praktis untuk membantu memperkuat pemahaman konsep-konsep ini. Akhirnya, dengan memahami dan menguasai tipe data, variabel, ekspresi, dan operator dalam bahasa C, Anda akan memiliki dasar yang kuat untuk membangun program yang lebih kompleks dan efektif.

4.1 Set Karakter dalam Bahasa C

Saat Anda menulis program, Anda mengekspresikan file program C sebagai baris teks yang berisi karakter dari set karakter. Saat program dijalankan, program menggunakan karakter dari set karakter. Set karakter ini terkait, tetapi tidak harus memiliki pengkodean yang sama atau semua anggota yang sama.

Setiap set karakter berisi nilai kode yang berbeda untuk setiap karakter dalam bahasa C. Sebuah set karakter juga dapat berisi karakter tambahan dengan nilai kode lain. Set karakter bahasa C berupa huruf, angka, dan karakter khusus seperti berikut ini:

- 1) Huruf, termasuk huruf kecil dan huruf besar (a-z dan A-Z)
- 2) Angka (0-9)
- 3) Karakter khusus, diantaranya:

;	:	{	,	'	"		}
>	<	/	\	~	_	[]
!	\$?	*	+	=	()
-	%	#	^	@	&	.	

4.2 Identifiers dan Keywords

Pengenal (identifiers) adalah nama yang diberikan kepada berbagai elemen program seperti konstanta, variabel, nama fungsi, array, dll. Setiap elemen dalam program memiliki nama yang berbeda, tetapi kita tidak dapat memilih nama apa pun kecuali jika nama tersebut sesuai dengan nama yang valid dalam bahasa C. Mari kita pelajari terlebih dahulu aturan untuk mendefinisikan nama atau pengenal.

4.2.1 Aturan untuk Membuat Identifiers

Identifiers atau pengenal didefinisikan menurut aturan berikut ini:

- 1) Terdiri atas huruf dan angka.
- 2) Karakter pertama harus berupa alfabet atau garis bawah.
- 3) Huruf besar dan kecil diperbolehkan. Teks yang sama dengan huruf yang berbeda tidak setara, misalnya: TEKS tidak sama dengan teks.
- 4) Kecuali karakter khusus garis bawah (`_`), tidak ada simbol khusus lainnya yang dapat digunakan.

Sebagai contoh, berikut adalah beberapa pengenal yang valid dan tidak valid:

Valid	Tidak Valid	
Y	123	Karakter pertama harus alfabet
X123	"X."	Titik (.) tidak diperbolehkan
_XI	error flag	Spasi tidak diperbolehkan
tax_rate	order-no	Tanda hubung tidak diperbolehkan
temp		

4.2.2 Keywords

Keywords atau kata kunci merupakan kata-kata yang dicadangkan, memiliki arti standar dan sudah ditentukan sebelumnya dalam bahasa C. Kata kunci tidak dapat digunakan sebagai pengenal.

Berikut ini adalah daftar kata kunci dalam bahasa C:

char	while	do	typedef	auto
int	if	else	switch	case
printf	double	struct	break	static
long	enum	register	extern	return
union	const	float	short	unsigned
continue	for	signed	void	default
goto	sizeof	volatile		

Catatan: Umumnya semua kata kunci menggunakan huruf kecil, meskipun huruf besar dari nama yang sama dapat digunakan sebagai pengenal.

4.3 Tipe Data

Untuk menyimpan data di dalam komputer, pertama-tama kita perlu mengidentifikasi jenis elemen data yang kita butuhkan dalam program kita. Ada beberapa jenis data yang berbeda, yang dapat direpresentasikan secara berbeda di dalam memori komputer. Tipe data menentukan dua hal:

- 1) Kisaran nilai yang diizinkan yang dapat disimpan.
- 2) Kebutuhan memori untuk menyimpan tipe data.

Bahasa C menyediakan empat tipe data dasar yaitu int, char, float, dan double. Dengan menggunakan tipe-tipe ini, kita dapat menyimpan data dengan cara yang sederhana sebagai elemen tunggal atau kita dapat mengelompokkannya dan menggunakan cara yang berbeda untuk menyimpannya sesuai kebutuhan. Keempat tipe data dasar tersebut dijelaskan oleh tabel 4.1.

Tabel 4.1: Tipe Data Dasar

Tipe	Deskripsi	Memori	Rentang Nilai
int	Bilangan bulat	2 atau 4 bytes	-32,768 s.d. 32,767
char	Karakter	1 byte	- 128 s.d. 128

float	Bilangan riil	4 bytes	3.4e-38 s.d. 3.4e+38
double	Bilangan riil dengan presisi yang lebih tinggi	8 bytes	1.7e-308 s.d. 1.7e+308

Kebutuhan memori atau ukuran data yang terkait dengan tipe data menunjukkan kisaran angka yang dapat disimpan dalam item data tipe tersebut. Dengan menggunakan kata kunci **sizeof**, kita dapat mengetahui ukuran tipe data atau variabel pada platform tertentu. Ekspresi **sizeof(type)** memberikan ukuran penyimpanan objek atau tipe dalam byte. Berikut adalah contoh untuk mendapatkan ukuran tipe **int** dan **float**, kemudian mencetak ruang penyimpanan yang digunakan oleh tipe data **float** serta nilai jangkauannya:

```
#include <stdio.h>
#include <float.h>

int main() {
    printf("Memori tipe int : %d \n", sizeof(int));
    printf("Memori tipe float : %d \n", sizeof(float));
    printf("Nilai float terkecil: %E\n", FLT_MIN );
    printf("Nilai float terbesar: %E\n", FLT_MAX );
    printf("Nilai presisi: %d\n", FLT_DIG );
}
```

```
return 0;
}
```

Hasil keluaran dari kode program di atas pada sistem Linux adalah sebagai berikut:

```
Memori tipe int : 4
Memori tipe float : 4
Nilai float terkecil: 1.175494E-38
Nilai float terbesar: 3.402823E+38
Nilai presisi: 6
```

4.3.1 Kualifikasi Tipe Data

Short, long, signed, unsigned disebut sebagai kualifikasi tipe data dan dapat digunakan dengan tipe data apa pun. Sebuah **short int** membutuhkan memori yang lebih kecil daripada **int** dan **long int** mungkin membutuhkan memori yang lebih besar daripada **int**. Jika **int** dan **short int** membutuhkan 2 bytes, maka **long int** membutuhkan 4 bytes. Ukuran memori dan rentang nilai tipe data dengan kualifikasi ditunjukkan pada tabel 4.2.

Tabel 4.2: Kualifikasi Tipe Data

Tipe Data	Memori	Rentang Nilai
short int atau int	2 bytes	-32768 s.d. 32,767
long int	4 bytes	-2147483648 s.d. 2147483647

signed int	2 bytes	-32768 s.d. 32767
unsigned int	2 bytes	0 s.d. 65535
signed char	1 byte	-128 s.d. 127
unsigned char	1 byte	0 s.d. 255

4.4 Variabel

Variabel adalah pengenal yang nilainya berubah dari waktu ke waktu selama eksekusi. Variabel adalah lokasi penyimpanan data yang diberi nama di memori komputer. Dengan menggunakan nama variabel dalam program Anda, pada dasarnya Anda mengacu pada data yang tersimpan di sana. Setiap variabel dalam bahasa C memiliki tipe tertentu, yang menentukan ukuran dan tata letak memori variabel; rentang nilai yang dapat disimpan dalam memori tersebut; dan serangkaian operasi yang dapat diterapkan pada variabel.

Perhatikan! Bahwa sebuah nilai harus ditetapkan ke variabel pada suatu titik waktu dalam program yang disebut sebagai pernyataan penugasan. Variabel tersebut kemudian dapat diakses di kemudian hari di dalam program. Jika variabel diakses sebelum diberi nilai, maka variabel tersebut akan memberikan nilai kosong. Tipe data dari sebuah variabel tidak berubah sedangkan nilai yang diberikan dapat berubah. Semua variabel memiliki tiga atribut penting, yaitu (1) nama; (2) nilai; dan (3) memori, di mana nilai tersebut disimpan.

4.4.1 Deklarasi Variabel

Sebelum data dapat disimpan dalam memori, kita harus memberikan nama pada lokasi memori tersebut. Untuk itu kita membuat deklarasi. Deklarasi mengasosiasikan sekelompok pengenalan dengan tipe data tertentu. Semuanya harus dideklarasikan sebelum muncul dalam pernyataan program, jika tidak, maka pengaksesan variabel akan menghasilkan nilai kosong atau kesalahan diagnostik.

Berikut adalah sintaks untuk mendeklarasikan variabel.

```
tipe_data nama_variabel;
```

Sebagai contoh,

```
int x, y, z;  
short int a;  
char ch1, ch2;  
float f1, upah;  
double phi;
```

4.4.2 Inisialisasi Variabel

Inisialisasi variabel berarti memberikan nilai pada variabel. Nilai awal dapat diberikan kepada variabel dengan dua cara:

- a) Di dalam Deklarasi Tipe. Nilai ditetapkan pada waktu deklarasi.

Sebagai contoh,

```
int x = 10;
float y = 0.4e-5;
char z = 'z';
```

- b) Menggunakan Pernyataan Penugasan. Nilai-nilai tersebut ditetapkan tepat setelah deklarasi dibuat.

Sebagai contoh,

```
int x;
float y;
char z;

x = 10;
y = 0.4e-5;
z = 'z';
```

4.4.3 Contoh Program

Perhatikan kode program berikut ini. Di sini, variabel dideklarasikan di bagian atas, kemudian juga didefinisikan dan diinisialisasi di fungsi utama.

```
#include <stdio.h>

// Definisi variabel:
```

```
extern int x, y;
extern int z;
extern float res;

int main () {
    // Definisi variabel:
    int x, y;
    int z;
    float res;

    // Inisialisasi yang sebenarnya
    x = 10;
    y = 20;
    z = x + y;

    printf("nilai z : %d \n", z);

    res = 70.0/3.0;
    printf("nilai res : %f \n", res);

    return 0;
}
```

Berikut adalah hasil keluaran dari kode program di atas.

```
nilai z : 30
nilai res : 23.333334
```

4.5 Konstanta

Konstanta adalah pengenal yang nilainya tidak dapat diubah selama eksekusi program, sedangkan nilai variabel terus berubah. Nilai-nilai yang tetap ini juga disebut literal. Konstanta dapat berupa salah satu tipe data dasar, seperti bilangan bulat (integer); bilangan riil (floating-point); karakter (character); atau string.

4.5.1 Konstanta Integer

Konstanta integer (bilangan bulat) dapat berupa desimal, oktal, atau heksadesimal. Awalan menentukan basis dari konstanta: untuk desimal tidak ada awalan, oktal diawali 0, dan heksadesimal diawali 0X atau 0x. Konstanta tipe ini juga dapat memiliki akhiran yang merupakan kombinasi dari U (unsigned) dan L (long). Selain itu, akhiran dapat berupa huruf kecil atau huruf besar dan dapat dalam urutan apa pun.

Berikut ini beberapa contoh konstanta integer.

```
212 /* Legal */
215u /* Legal */
0xFeeL /* Legal */
078 /* Ilegal: 8 bukan merupakan digit oktal */
032UU /* Ilegal: tidak dapat mengulang akhiran */
```

Contoh lainnya adalah sebagai berikut.

```
85 /* desimal */  
0213 /* oktal */  
0x4b /* heksadesimal */  
30 /* int */  
30u /* unsigned int */  
30l /* long */  
30ul /* unsigned long */
```

4.5.2 Konstanta Floating-Point

Konstanta floating-point (bilangan riil) dapat berupa integer, titik desimal, pecahan, dan eksponensial. Konstanta floating-point dapat direpresentasikan dalam bentuk desimal ataupun eksponensial. Saat merepresentasikan dalam bentuk desimal, titik desimal/eksponen/keduanya harus disertakan. Saat merepresentasikan dalam bentuk eksponensial, bagian bilangan bulat/pecahan/keduanya harus disertakan. Tanda eksponensial disimbolkan dengan e atau E.

Berikut ini beberapa contoh konstanta floating-point.

```
3.14159 /* Legal */  
314159E-5L /* Legal */  
510E /* Illegal: eksponen tidak lengkap */  
210f /* Illegal: tidak ada desimal/eksponen */  
.e55 /* Illegal: ada bilangan yang hilang */
```

4.5.3 Konstanta Karakter (Character)

Literal karakter diapit oleh tanda kutip tunggal, misalnya 'c', dan dapat disimpan dalam variabel sederhana bertipe char. Karakter literal dapat berupa karakter biasa (misalnya, 'c'), escape sequence (misalnya, '\t'), atau karakter universal (misalnya, '\u02C0').

Terdapat beberapa karakter tertentu dalam bahasa C yang memiliki arti khusus apabila didahului dengan garis miring, dan digunakan untuk mewakili karakter seperti baris baru (\n) atau tab (\t). Berikut daftar beberapa karakter escape sequence dalam bahasa C.

Escape Sequence	Maknanya
\\	Garis miring terbalik (\)
\'	Tanda petik satu (')
\"	Tanda petik ganda (")
\?	Tanda tanya (?)
\a	Peringatan atau bel
\b	Menghapus spasi
\f	Bentuk umpan
\n	Baris baru

<code>\r</code>	Pengembalian
<code>\t</code>	Tab horisontal
<code>\v</code>	Tab vertikal
<code>\ooo</code>	Bilangan oktal satu hingga tiga digit
<code>\xhh...</code>	Angka heksadesimal yang terdiri dari satu digit atau lebih

Contoh penggunaan karakter escape sequence dalam program.

```
#include <stdio.h>

int main() {
    printf("Hello\tWorld\n\n");

    return 0;
}
```

Berikut adalah hasil keluaran dari kode program di atas:

```
Hello World
```

4.5.4 Konstanta String

Konstanta string diapit oleh tanda kutip ganda (""). String mirip dengan karakter (char), perbedaannya string terdiri atas

beberapa karakter. String dapat berisi karakter biasa, escape sequence, dan karakter universal. Anda dapat memecah baris panjang menjadi beberapa baris menggunakan string dan memisahkannya menggunakan spasi.

Berikut adalah contoh tiga string literal yang identik.

```
"halo, sayang"  
"halo, \  
sayang"  
"halo, " "s" "ayang"
```

4.5.5 Pendefinisian Konstanta

Ada dua cara untuk mendefinisikan konstanta dalam bahasa C.

a) #define preprocessor

Berikut ini adalah cara mendefinisikan suatu konstanta menggunakan #define preprocessor.

```
#define pengenalan nilai
```

Secara rinci, ditunjukkan oleh contoh kode program berikut.

```
#include <stdio.h>  
  
#define PANJANG 10  
#define LEBAR 5  
#define BARISBARU '\n'
```

```
int main() {  
    int luas;  
  
    luas = PANJANG * LEBAR;  
    printf("luas area : %d", luas);  
    printf("%c", BARISBARU);  
  
    return 0;  
}
```

Berikut adalah hasil keluaran dari kode program di atas.

```
luas area : 50
```

b) Kata kunci const

Berikut ini adalah cara mendeklarasikan suatu konstanta menggunakan kata kunci const.

```
const tipe_data variabel = nilai;
```

Secara rinci, ditunjukkan oleh contoh kode program berikut.

```
#include <stdio.h>  
  
int main() {  
    const int PANJANG = 10;  
    const int LEBAR = 5;  
    const char BARISBARU = '\n';  
    int luas;  
  
    luas = PANJANG * LEBAR;  
    printf("luas area : %d", luas);  
}
```

```
printf("%c", BARISBARU);  
  
return 0;  
}
```

Berikut adalah hasil keluaran dari kode program di atas.

```
luas area : 50
```

Sebagai catatan, sebaiknya konstanta didefinisikan dalam KAPITAL.

4.6 Ekspresi dan Operator

Pada bagian sebelumnya kita telah mempelajari variabel, konstanta, tipe data, dan cara mendeklarasikannya dalam pemrograman C. Langkah selanjutnya adalah menggunakan variabel-variabel tersebut dalam ekspresi. Untuk menulis sebuah ekspresi, kita membutuhkan operator dan variabel. Ekspresi adalah urutan operator dan operan yang melakukan salah satu atau kombinasi dari berikut ini:

- menentukan penghitungan suatu nilai
- menunjuk sebuah objek atau fungsi
- menghasilkan pengaruh

Operator merupakan sebuah simbol yang memberitahu kompiler untuk melakukan manipulasi matematis atau logis tertentu. Sebuah operator melakukan sebuah operasi (evaluasi) pada satu atau lebih operan. Operan adalah sub-ekspresi yang menjadi tempat operator bekerja. Bahasa C kaya akan operator bawaan dan menyediakan beberapa jenis operator, antara lain:

- 1) Penugasan (assignment)
- 2) Aritmatika
- 3) Relasional
- 4) Logika
- 5) Bitwise

Dalam sub-bab ini akan dijelaskan satu per satu jenis operator yang tersedia di bahasa C termasuk sintaks dan penggunaan setiap operator dan bagaimana mereka digunakan dalam bahasa C.

4.6.1 Operator Penugasan

Sebelumnya, kita telah melihat bahwa variabel pada dasarnya adalah lokasi memori dan dapat menyimpan nilai tertentu. Tapi, bagaimana cara memberikan nilai ke variabel? Bahasa C menyediakan operator penugasan untuk tujuan ini. Fungsi dari operator ini adalah untuk menetapkan nilai atau menetapkan nilai dalam variabel di sisi kanan ekspresi ke variabel di sisi kiri.

Sintaks dari ekspresi penugasan adalah sebagai berikut.

```
variabel = konstanta / variabel / ekspresi;
```

Tipe data dari variabel di sisi kiri harus sama dengan tipe data dari konstanta/variabel/ekspresi di sisi kanan dengan beberapa pengecualian, di mana konversi tipe secara otomatis dapat dilakukan.

Berikut ini adalah beberapa contoh pernyataan penugasan.

```
x = y ; /* x diberi nilai y */  
x = 5 ; /* x diberi nilai 5*/  
x = y+5; /* x diberi nilai ekspresi y+5 */
```

Ekspresi di sisi kanan pernyataan penugasan dapat berupa ekspresi aritmatika, relasional, logika, atau campuran. Ekspresi-ekspresi tersebut berbeda dalam hal jenis operator yang menghubungkan variabel dan konstanta di sisi kanan variabel.

Sebagai contoh,

```
int j;  
float p, q, avg, L;  
avg = (p+q) / 2; /* ekspresi aritmatika */  
j = p && q; /* ekspresi logika */  
j = (p+q) && (p<q); /* ekspresi campuran */
```

4.6.2 Operator Aritmatika

Operator aritmatika yang didukung oleh bahasa C ditunjukkan oleh Tabel 4.3. Asumsikan variabel A menampung nilai 10 dan variabel B menampung nilai 20.

Tabel 4.3: Operator Aritmatika

Operator	Deskripsi	Contoh
+	Menambahkan dua operan	$A + B = 30$
-	Mengurangkan operan kedua dari operan pertama	$A - B = -10$
*	Mengalikan kedua operan	$A * B = 200$
/	Membagi pembilang dengan penyebut	$B / A = 2$
%	Operator modulus dan sisa dari setelah pembagian bilangan bulat	$B \% A = 0$
++	Operator increment, meningkatkan nilai bilangan bulat sebesar satu	$A++ = 11$
--	Operator decrement, mengurangi nilai bilangan bulat sebesar satu	$A-- = 9$

Cobalah contoh kode program C berikut untuk memahami semua operator aritmatika.

```
#include <stdio.h>

main() {
    int x = 21;
    int y = 10;
    int z;

    z = x + y;
    printf("1) Nilai z adalah %d\n", z);
    z = x - y;
    printf("2) Nilai z adalah %d\n", z);
    z = x * y;
    printf("3) Nilai z adalah %d\n", z);
    z = x / y;
    printf("4) Nilai z adalah %d\n", z);
    z = x % y;
    printf("5) Nilai z adalah %d\n", z);
    z = x++;
    printf("6) Nilai z adalah %d\n", z);
    z = x--;
    printf("7) Nilai z adalah %d\n", z);
}
```

Berikut adalah hasil keluaran dari kode program di atas.

- 1) Nilai z adalah 31
- 2) Nilai z adalah 11
- 3) Nilai z adalah 210
- 4) Nilai z adalah 2

- 5) Nilai z adalah 1
- 6) Nilai z adalah 21
- 7) Nilai z adalah 22

4.6.3 Operator Relasional

Operator relasional yang didukung oleh bahasa C ditunjukkan oleh Tabel 4.4. Asumsikan variabel A menampung nilai 10 dan variabel B menampung nilai 20.

Tabel 4.4: Operator Relasional

Operator	Deskripsi	Contoh
==	Memeriksa apakah nilai dari dua operan sama atau tidak, jika ya maka kondisi menjadi benar	(A == B) is not true
!=	Memeriksa apakah nilai dari dua operan sama atau tidak, jika nilainya tidak sama maka kondisi menjadi true	(A != B) is true
>	Memeriksa apakah nilai operan kiri lebih besar dari nilai operan kanan, jika ya maka kondisi menjadi benar	(A > B) is not true

<	Memeriksa apakah nilai operan kiri kurang dari nilai operan kanan, jika ya maka kondisi menjadi true	(A < B) is true
>=	Memeriksa apakah nilai operan kiri lebih besar dari atau sama dengan nilai operan kanan, jika ya maka kondisi menjadi benar	(A >= B) is not true
<=	Memeriksa apakah nilai operan kiri kurang dari atau sama dengan nilai operan kanan, jika ya maka kondisi menjadi benar	(A <= B) is true

Cobalah contoh kode program C berikut untuk memahami semua operator relasional.

```
#include <stdio.h>

main() {
    int x = 21;
    int y = 10;
    int z;

    if( x == y ) {
        printf("1) x sama dengan y\n");
    } else {
```

```
printf("1) x tidak sama dengan y\n" );
}
if ( x < y ) {
    printf("2) x kurang dari y\n" );
} else {
    printf("2) x tidak kurang dari y\n" );
}
if ( x > y ) {
    printf("3) x lebih besar dari y\n" );
} else {
    printf("3) x tidak lebih besar dari y\n" );
}

/* Nilai x dan y diubah */
x = 5;
y = 20;
if ( x <= y ) {
    printf("4) x lebih kecil atau sama dengan y\n" );
}
if ( y >= x ) {
    printf("5) y lebih besar atau sama dengan x \n" );
}
}
```

Berikut adalah hasil keluaran dari kode program di atas.

- 1) x tidak sama dengan y
- 2) x tidak kurang dari y
- 3) x lebih besar dari y

- 4) x lebih kecil atau sama dengan y
- 5) y lebih besar atau sama dengan x

4.6.4 Operator Logika

Operator logika yang didukung oleh bahasa C ditunjukkan oleh Tabel 4.5. Asumsikan variabel A bernilai 1 dan variabel B bernilai 0.

Tabel 4.5: Operator Relasional

Operator	Deskripsi	Contoh
&&	Disebut operator logika AND. Jika kedua operan bukan nol, maka kondisi menjadi benar.	(A && B) is false
	Disebut operator logika OR. Jika salah satu dari dua operan bukan nol, maka kondisi menjadi benar.	(A B) is true
!	Disebut operator logika NOT. Digunakan untuk membalikkan kondisi logika dari operan. Jika suatu kondisi bernilai benar, maka operator logika NOT akan menjadikannya salah.	!(A && B) is true

Cobalah contoh kode program C berikut untuk memahami semua operator logika.

```
#include <stdio.h>

main() {
    int x = 5;
    int y = 20;
    int z;

    if ( x && y ) {
        printf("1) Kondisi ini benar \n" );
    }
    if ( x || y ) {
        printf("2) Kondisi ini benar \n" );
    }
    /* Nilai x dan y diubah */
    x = 0;
    y = 10;
    if ( x && y ) {
        printf("3) Kondisi ini benar \n" );
    } else {
        printf("3) Kondisi ini tidak benar \n" );
    }
    if ( !(x && y) ) {
        printf("4) Kondisi ini benar \n" );
    }
}
```

Berikut adalah hasil keluaran dari kode program di atas.

- 1) Kondisi ini benar
- 2) Kondisi ini benar
- 3) Kondisi ini tidak benar
- 4) Kondisi ini benar

4.6.5 Operator Bitwise

Operator bitwise bekerja pada bit dan melakukan operasi bit demi bit. Berikut ini adalah tabel kebenaran untuk $\&$, $|$, dan \wedge .

p	q	p & q	p q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Asumsikan variabel $A = 60$; dan variabel $B = 13$; kedua nilai tersebut dalam format biner adalah sebagai berikut.

A = 0011 1100

B = 0000 1101

A & B = 0000 1100

A | B = 0011 1101

A ^ B = 0011 0001

~A = 1100 0011

Operator Bitwise yang didukung oleh bahasa C tercantum dalam tabel 4.6. Asumsikan variabel A menampung 60 dan variabel B menampung 13:

Tabel 4.6: Operator Bitwise

Operator	Deskripsi	Contoh
&	Operator Biner AND menyalin bit jika bit tersebut ada di kedua operan.	$(A \& B) = 12$, yaitu 0000 1100
	Operator Biner OR menyalin bit jika bit tersebut ada di salah satu operan.	$(A B) = 61$, yaitu 0011 1101
^	Operator Biner XOR menyalin bit jika bit tersebut ada pada salah satu operan, tetapi tidak pada keduanya.	$(A \wedge B) = 49$, yaitu 0011 0001
~	Operator Komplemen Biner adalah unary dan memiliki efek 'membalik' bit.	$(\sim A) = -60$, yaitu 1100 0011

<<	Operator Pergeseran Biner ke Kiri. Nilai operan kiri digeser ke kiri dengan jumlah bit yang ditentukan oleh operan kanan.	A << 2 = 240, yaitu 1111 0000
>>	Operator Pergeseran Biner ke Kanan. Nilai operan kiri digeser ke kanan dengan jumlah bit yang ditentukan oleh operan kanan.	A >> 2 = 15, yaitu 0000 1111

Cobalah contoh kode program C berikut untuk memahami semua operator bitwise.

```
#include <stdio.h>

main() {
    unsigned int A = 60; /* 60 = 0011 1100 */
    unsigned int B = 13; /* 13 = 0000 1101 */
    int C = 0;

    C = A & B; /* 12 = 0000 1100 */
    printf("1) Nilai C adalah %d\n", C);
    C = A | B; /* 61 = 0011 1101 */
    printf("2) Nilai C adalah %d\n", C);
    C = A ^ B; /* 49 = 0011 0001 */
    printf("3) Nilai C adalah %d\n", C);
}
```

```
C = ~A; /*-61 = 1100 0011 */  
printf("4) Nilai C adalah %d\n", C);  
C = A << 2; /* 240 = 1111 0000 */  
printf("5) Nilai C adalah %d\n", C);  
C = A >> 2; /* 15 = 0000 1111 */  
printf("6) Nilai C adalah %d\n", C);  
}
```

Berikut adalah hasil keluaran dari kode program di atas.

- 1) Nilai C adalah 12
- 2) Nilai C adalah 61
- 3) Nilai C adalah 49
- 4) Nilai C adalah -61
- 5) Nilai C adalah 240
- 6) Nilai C adalah 15

BAB 5

OPERASI SELEKSI

5.1 Pengertian Operasi Seleksi

Operasi seleksi adalah operasi yang dilakukan untuk menyeleksi suatu kondisi. Jawaban atas kondisi yang diberikan, digunakan untuk menentukan langkah berikutnya yang akan dilakukan. Statemen untuk operasi seleksi kondisi adalah IF-THEN dan SWITCH-CASE.

5.2 Statemen IF

Statemen IF digunakan untuk menyeleksi sebuah pernyataan dari beberapa pernyataan yang tersedia. Melalui statemen IF dapat ditentukan apakah sebuah pernyataan akan dikerjakan atau tidak, tergantung dari jawaban atas kondisi yang diberikan. Jika ada beberapa pernyataan maka awal pernyataan ditandai dengan { dan diakhiri tanda }. Struktur statemen IF adalah:

if (kondisi)

```
{  
    pernyataan1;  
    ...  
    pernyataanX;  
}
```

Struktur tersebut berarti: jika kondisi benar maka pernyataan yang berada diantara { dan } akan dikerjakan, jika kondisi bernilai salah maka program berhenti.

Contoh 1:

Program untuk menentukan seseorang lulus ujian. Jika $\text{score} \geq 70$ dinyatakan lulus. Jika $\text{score} < 70$ tidak ada output.

Jawab:

```
# include <stdio.h>  
  
main()  
{  
int score_anda;  
printf(" Masukkan Score Ujian = ");  
scanf("%d", &score_anda);  
if (score_anda >= 70)  
    printf("Selamat Anda Lulus");  
}
```

Hasil eksekusi program jika dimasukkan nilai ujian 80.

Masukkan Score Ujian = 80

Selamat Anda Lulus

Silahkan eksekusi program untuk nilai ujian kecil dari 70.

5.2.1 STATEMEN IF-ELSE

Struktur statemen IF-ELSE adalah:

if (kondisi)

```
{  
    pernyataan1;  
    ...  
    pernyataanX;  
}
```

else

```
{  
    pernyataan2;  
    ...  
    pernyataanY;  
}
```

Struktur tersebut berarti: jika kondisi benar maka pernyataan yang berada diantara { dan } akan dikerjakan, yaitu pernyataan1 hingga pernyataanX. Jika kondisi bernilai salah, maka yang akan dikerjakan adalah pernyataan setelah else, yaitu pernyataan2 hingga pernyataanY.

Contoh 2:

Program menentukan seseorang lulus ujian atau tidak. Jika $\text{score} \geq 70$ dinyatakan lulus. Jika $\text{score} < 70$ maka dinyatakan belum lulus.

Jawab:

```
# include <stdio.h>

main()
{
    int nilai_anda;
    printf(" Masukkan Score Ujian = ");
    scanf(" %d", &nilai_anda);
    if (nilai_anda >= 70)
        printf("Selamat Anda Lulus");
    else
        printf ("Anda Belum Lulus");
}
```

Hasil eksekusi program jika dimasukkan nilai ujian 80.

Masukkan Score Ujian = 80

Selamat Anda Lulus

Silahkan eksekusi program untuk nilai ujian kecil dari 70.

Catatan:

Perhatikan perbedaan output program Contoh 1 dan Contoh 2 ketika diinputkan nilai ujian kecil dari 70.

5.2.2 STATEMEN IF–ELSE-IF

Statemen IF–ELSE-IF merupakan statemen yang memiliki kondisi dan pernyataan majemuk. Struktur statemen IF–ELSE-IF adalah:

```
if (kondisi1)
    {
        pernyataan1;
        ...
        pernyataanX;
    }
else
if (kondisi2)
    {
        pernyataan2;
        ...
        pernyataanY;
    }
else
if (kondisi3)
    {
        pernyataan3;
        ...
        pernyataanZ;
    }
```

Struktur tersebut berarti: jika kondisi1 benar maka pernyataan selanjutnya yang berada diantara { dan } akan dikerjakan, yaitu pernyataan1 hingga pernyataanX. Jika kondisi1 bernilai salah maka pernyataan setelah else yang akan dikerjakan, yaitu kondisi2. Jika kondisi2 bernilai benar maka pernyataan yang berada diantara { dan } akan dikerjakan, yaitu pernyataan2 hingga pernyataanY. Jika kondisi2 bernilai salah maka pernyataan setelah else yang akan dikerjakan, yaitu kondisi3. Jika kondisi3 bernilai benar maka pernyataan yang berada diantara { dan } akan dikerjakan, yaitu pernyataan3 hingga pernyataanZ. Jika kondisi3 bernilai salah maka program akan berhenti.

Contoh 3:

Program menentukan sebuah bilangan termasuk bilangan positif, negatif atau nol.

Jawab:

```
# include <stdio.h>

main()
{
    int bil_bul;

    printf("Masukkan sebuah bilangan bulat = ");
    scanf(" %d",&bil_bul);

    if (bil_bul>0)
        printf(" %d adalah bilangan positif ", bil_bul);
```

```
else if (bil_bul<0)
    printf(" %d adalah bilangan negatif ", bil_bul);
else
    printf (" %d adalah bilangan nol ", bil_bul );
}
```

Hasil eksekusi program jika dimasukkan bilangan 8.

Masukkan sebuah bilangan bulat = 8

8 adalah bilangan positif

Silahkan eksekusi program untuk bilangan yang lain.

5.2.3 STATEMEN NESTED IF

Sebuah statemen IF dapat berada didalam statemen IF lainnya, ini disebut NESTED-IF. Struktur statemen NESTED-IF adalah:

```
if (kondisi1)
    pernyataan1;
else if (kondisi2)
    pernyataan2;
else if (kondisi3)
    pernyataan3;
else if (kondisi4)
    pernyataan4;
```

Contoh 4:

Program untuk menentukan seorang mahasiswa lulus ujian atau tidak. Terdapat 2 ketentuan:

Ketentuan 1:

jika nilai ≥ 50 maka dinyatakan lulus, jika < 50 maka dinyatakan tidak lulus.

Ketentuan 2:

jika nilai ≥ 90 mendapatkan nilai A, jika nilai ≥ 75 mendapatkan nilai B, jika nilai ≥ 65 mendapatkan nilai C, jika nilai ≥ 50 mendapatkan nilai D, jika nilai < 50 mendapatkan nilai E.

Jawab:

```
# include <stdio.h>

int main()
{
int nilai_ujian;
printf("Masukkan Score Ujian =");
scanf("%d", &nilai_ujian);
if(nilai_ujian >= 50)
{
printf("Selamat Anda Lulus \n");
if(nilai_ujian >= 90)
{
printf(" Nilai Anda A \n");
}
}
}
```



```
else if(nilai_ujian >=75)
{
printf("Nilai Anda B \n");
}
else if(nilai_ujian >=65)
{
printf("Nilai Anda C \n");
}
else if(nilai_ujian >=50)
{
printf("Nilai Anda D \n");
}
}
else
{
printf(" Sayang Sekali Anda Belum Lulus");
printf("Nilai Anda E \n");
}
}
```

Hasil eksekusi program jika dimasukkan nilai 95.

Masukkan Score Ujian = 95

Selamat Anda Lulus

Nilai Anda A

Silahkan eksekusi program untuk score yang lain.

5.3 Statemen SWITCH-CASE

5.3.1 Statemen SWITCH

Statemen SWITCH-CASE digunakan untuk menentukan keputusan yang akan diambil berdasarkan sejumlah pilihan penyelesaian. Statemen SWITCH-CASE memiliki fungsi yang sama seperti IF-ELSE bertingkat, tetapi pemilihan pernyataannya menggunakan data bertipe integer atau karakter. Struktur statemen SWITCH-CASE adalah:

switch (ekspresi integer atau karakter)

```
{  
    case konstanta1 :  
        pernyataan1;  
        break;  
    case konstanta2 :  
        pernyataan2;  
        break;  
    case konstanta3 :  
        pernyataan3;  
        break;  
    case konstanta4 :  
        pernyataanX;  
    default :  
        pernyataanY;  
}
```

Contoh 5:

Program menampilkan nama-nama hari dalam seminggu berdasarkan kode hari yang diinputkan. Jika kode hari=1 maka output adalah hari Senin, jika kode hari=2 maka output adalah hari Selasa, jika kode hari=3 maka output adalah hari Rabu, jika kode hari=4 maka output adalah hari Kamis, jika kode hari=5 maka output adalah hari Jumat, jika kode hari=6 maka output adalah hari Sabtu, jika kode hari=7 maka output adalah hari Minggu, jika kode hari yang diinputkan selain angka 1 s.d 7 maka tampilkan pesan kesalahan memasukkan kode hari.

Jawab:

```
# include<stdio.h>

main()
{
int kode_hr;
printf("Masukkan kode hari, pilih angka 1 s.d 7 = ");
scanf ("%d",&kode_hr);
switch (kode_hr)
{ case 1:
    printf("Anda memilih angka 1\n ");
    printf("Angka 1 adalah kode hari Senin \n ");
    break;
case 2:
    printf(" Anda memilih angka 2 \n ");
```

```
printf(" Angka 2 adalah kode hari Selasa \n ");  
break;
```

case 3:

```
printf(" Anda memilih angka 3 \n " );  
printf(" Angka 3 adalah kode hari Rabu \n ");  
break;
```

case 4:

```
printf(" Anda memilih angka 4 \n " );  
printf(" Angka 4 adalah kode hari Kamis \n ");  
break;
```

case 5:

```
printf(" Anda memilih angka 5 \n");  
printf(" Angka 5 adalah kode hari Jumat \n ");  
break;
```

case 6:

```
printf(" Anda memilih angka 6 \n " );  
printf(" Angka 6 adalah kode hari Sabtu \n ");  
break;
```

case 7:

```
printf(" Anda memilih angka 7 \n " );  
printf(" Angka 7 adalah kode hari Minggu \n ");  
break;
```

default:

```
printf(" Anda salah input, kode hari adalah angka 1 s.d  
7 \n ");  
}  
}
```

Hasil eksekusi program jika dimasukkan kode hari =1.

Masukkan kode hari, pilih angka 1 s.d 7 = 1

Anda memilih angka 1

Angka 1 adalah kode hari Senin

Silahkan eksekusi program untuk kode hari yang lain.



BAB 6

OPERASI PERULANGAN

Operasi perulangan pada pemrograman dasar merupakan proses yang di lakukan lebih dari satu kali, hal tersebut dapat membuang waktu apabila tidak disederhanakan, oleh karena itu proses looping dapat mempermudah hal tersebut dengan cara memberi batas waktu yang telah ditentukan.

Proses looping ada 2 diantaranya penambahan dan pengurangan, dimana penambahan atau *increment* merupakan terjadi proses penambahan jumlah lebih dari satu kali, sedangkan proses pengurangan atau *decrement* merupakan proses pengurangan jumlah lebih dari satukali Dalam bahasa C, terdapat beberapa macam perintah saat proses looping digunakan diantaranya : for, do while, while.

Dalam pemaparan perulangan terdapat 3 komponen saat proses perulangan berlangsung diantaranya :

1. Awal perulangan
2. Perulangan berlangsung
3. Perulangan berhenti

FOR

Perintah *for* digunakan untuk menjalankan perintah dengan jumlah perulangan yang telah diketahui.

Sintaks dari *for* adalah

```
For (ex1; ex2; ex3) {  
  statement (s);  
}
```

Keterangan :

Ex1 : expresi untuk penentuan awal

Ex2 : expresi untuk kondisi

Ex3 : expresi untuk penambahan (*increament*) dan pengurangan (*decreament*)

Statement : merupakan kode program.

Penjelasan dari penulisan alur instruksi *for* adalah

1. Ex1, merupakan nilai awal untuk dieksekusi pertama kali. Tahap ini digunakan untuk deklarasi awal dan sebagai variabel control.
2. Ex2, merupakan batas perulangan proses yang diberikan, serta sebagai proses evaluasi jika kondisi benar maka steatement atau perintah akan di jalankan, akan tetapi apabila kondisi bernilai salah, maka steatement atau perintah akan berhenti dan proses perulangan tersebut berhenti. Contoh : $i < 5$, maka selama nilai variabel (i) berisi angka yang

kurang dari 5, maka proses perulangan akan terus berlangsung.

3. Ex3, merupakan bagian yang digunakan untuk memproses variabel. Pada ekspresi 3 ini akan ada pilhan dua proses diantaranya proses penambahan (*increment*) $i = i+1$ dan proses pengurangan (*decrement*) $i = i-1$.

Untuk lebih memahami teori diatas maka kita belajar lebih detail dalam bentuk program sederhana

```
#include <stdio.h>

int main() {
    int i;
    /*for*/
    for (i=1; i<=10; i=i+1) {
        printf("Hello Word %d\n", i);
    }
    return 0;
}
```

Pada program diatas diketahui bahwa untuk membuat program dengan proses perulangan variabel *i* menggunakan tipe data integer.

Variabel ini nantinya akan dipakai sebagai variabel counter, yakni variabel yang menentukan berapa jumlah perulangan yang di lakukan.

Pada saat penulisan rumus for diatas dengan perintah `For(i=1;i<=10;i=i+1)`expresi pertama merupakan nilai awal untuk menentukan angka dimulainya perintah yaitu `i=1`, expresi kedua menentukan batas proses perulangan di lakukan, dalam hal ini perulangan dilakukan 10 kali `i<=10`, expresi ketiga merupakan penentuan terjadi penambahan yaitu `i=i+1`.

Pada perintah selanjutnya `printf("Hello Word %d\n",i)` perintah `printf` di tujukan untuk menampilkan kalimat pada compiler yaitu "Hello word") serta `%d` untuk menampilkan tipe data int, dimana nilai tersebut menampilkan nilai variabel `i` sehingga output dari program tersebut pada compiler adalah seperti gambar di bawah

```
Hello Word 1
Hello Word 2
Hello Word 3
Hello Word 4
Hello Word 5
Hello Word 6
Hello Word 7
Hello Word 8
Hello Word 9
Hello Word 10
|
```

WHILE

Instruksi while digunakan apabila syarat terpenuhi, maka perulangan baru dapat terjadi, maksud dari penjelasan tersebut mari kita lihat dari struktur perulangan dengan perintah while pada halaman selanjutnya

```
While(kondisi)
{
//Blok pertanyaan atau statement
}
```

Pada kode diatas berarti pernyataan akan terjadi perulangan apabila kondisi pada perintah while bernilai benar, jika kondisi bernilai salah maka proses perulangan tidak akan dijalankan atau berlangsung. Apabila anda masih kesulitan memahami, mari kita bersama melihat contoh program di bawah ini

```
#include <iostream>
using namespace std;
int main()
{
    int a = 1;
    while(a <= 10)
    {
        cout << "Looping ke -- " <<
a << endl;
        a++;
    }
    return 0;
}
```

Penjelasan dari program diatas, adalah variabel di deklarasikan sebagai a, dengan dimulai dengan angka 1, jika variabel tersebut bernilai benar, maka masuk ke perintah while dengan perulangan sebanyak 5 kali, karena pada kondisi dalam perintah while tertulis while (a <= 5) di dalam tersebut ada perintah cetak kalimat, perulangan yang ke - , sesuai dengan perulangan yang tertulis dalam kondisi while.

Variabel a bernilai 1, program sedang diproses pada baris while,

DO WHILE

Instruksi do..while digunakan untuk mengulang suatu perintah atau statement selama kondisi tersebut masih bernilai benar, dimana perintah tersebut dibuat sesuai dengan keinginan pembuat program. Do..while berfungsi mengulang sub statement yang ada.

```
do{  
statement(s);  
} while(condition);
```

Pada syntax diatas dilakukan proses mencari kebenaran dalam perintah do, sedang nilai lainnya apabila tidak bernilai benar maka diterjemahkan oleh perintah while, sehingga jika

kondisi benar, maka perintah akan dijalankan kembali dan proses perulangan akan dijalankan terus menerus sampai kondisi salah. Untuk lebih jelasnya akan di jabarkan pada penjelasan berikut dengan bantuan contoh program.

```
int main() {
    /*do-while*/
    int i = 1;
    do{
        printf("urutan angka 1-10 :
%d\n",i);
        i=i+1;
    }
    while(i<=10);
    getchar();
    return 0;
}
```

Pada contoh diatas akan di buat sebuah program untuk menampilkan urutan angka 1 – 10 dengan perintah syntax do - while, diketahui bahwa penentuan nilai variabel awal ditulis sebelum perintah do yaitu deklarasi variabel i dengan tipe data integer dimana penulisannya adalah `int i = 1`, pada program tersebut di mulai dari 1.

Didalam perintah do terdapat perintah untuk penulisan perintah yang tampil pada compiler dengan perintah `printf("urutan angka 1-10 : %d\n",i);` sehingga pada compiler akan muncul urutan angka 1-10 dan menampilkan tipe data integer yang sesuai dengan penambahan nilai yang ada pada

proses increment di dalam perintah do yaitu $i = i + 1$. Pada perintah while dimasukkan batas perulangan terjadi $\text{while}(i \leq 10)$, maka output yang keluar sebanyak 10 kali perulangan. Output dari program tersebut adalah

```
urutan angka 1-10 : 1
urutan angka 1-10 : 2
urutan angka 1-10 : 3
urutan angka 1-10 : 4
urutan angka 1-10 : 5
urutan angka 1-10 : 6
urutan angka 1-10 : 7
urutan angka 1-10 : 8
urutan angka 1-10 : 9
urutan angka 1-10 : 10
```



BAB 7

PENGENALAN BAHASA C/C++

Sekilas C/C++

Bahasa C pada awalnya dikembangkan oleh Dennis M. Ritchie dan Brian W. Kernighan pada awal tahun 1970. Bahasa C ini hanya berkembang di lingkungan sistem UNIX saja ($\pm 90\%$ sistem operasi UNIX ditulis dalam bahasa C). Namun seiring perkembangannya pada tahun 1986, dikembangkan Bahasa C kompatibel yaitu superset C dengan kemampuan yang lebih baik yaitu dapat dipakai untuk pemrograman berorientasi objek. Bahasa C inilah yang selanjutnya dikenal dengan C++ atau CPP dan dikembangkan oleh Bjarne Stroustrup.

Compiler C/C++

- GCC (GNU's Compiler Collection) yang dapat berjalan di Linux
- Microsoft C/Microsoft QuickC dan sekarang menjadi Visual Studio

- Turbo C/Borland C++ dan sekarang lebih dikenal dengan Embarcadero Rad Studio
- Dev-C yang dikembangkan dengan memakai Delphi. Saat ini memakai compiler C yang dikembangkan oleh Embarcadero sehingga Namanya menjadi Embarcadero Dev CPP

Manfaat/kegunaan

Bahasa C banyak dipakai untuk mengembangkan aplikasi. Karena Bahasa C ini selain bisa digunakan pada banyak sistem operasi (kompatibel), juga pemrograman-nya lebih efisien bila dibandingkan dengan bahasa assembly atau bahasa mesin. Bahasa C/C++ ini bisa dipakai untuk sembarang sistem operasi. Sedangkan bahasa yang lainnya ada ketergantungan dengan sistem operasi. Seperti contohnya adalah Visual Basic, Visual C yang sekarang bernama Visual Studio dan dikembangkan oleh Microsoft. Visual Basic atau Visual Studio hanya dapat dipasang di Microsoft Windows, meskipun hasilnya dapat dijalankan di sistem operasi yang lain tetapi Sistemnya hanya berjalan di Microsoft Windows. Itupun bergantung juga dengan Microsoft Windows versi yang mana, apakah Windows 9x, Windows XP, Windows 7, Windows 8, Windows 10. Beberapa manfaat dari Bahasa C/C++ ini di antaranya:

1. membuat sistem operasi dan program-program sistem,
2. pemrograman yang "dekat" ke perangkat keras (misalnya untuk kontrol peralatan),
3. membuat *tool kit (utility)*,
4. menulis program aplikasi
5. membuat bahasa pemrograman lain

Struktur Program C

Bentuk procedure	Bentuk fungsi
Header file void main() { //ini komentar isi program }	Header file int main() { //isi program return 0; }

Keterangan :

Header file

file yang berisi fungsi atau procedure yang sudah dibuat.

Misalnya `stdio.h` , `stdlib.h`, `iostream` dan sebagainya

void main() atau **int main()**

merupakan awal program Bahasa C

Tanda “{” adalah awal program dan diakhiri dengan “}”. Sedangkan komentar, biasanya menggunakan bahasa si pembuat program, ditulis dengan diawali tanda “//” untuk 1 baris kalimat. Bila lebih dari 1 baris bisa dengan diawali “(“ dan diakhiri dengan “*)” dan tidak akan diproses oleh *compiler*.

Contoh : menampilkan tulisan **selamat datang di C**.

Bentuk procedure

```
#include <stdio.h>

void main()
{
//isi program
printf("selamat datang di C");
}
```

Bentuk fungsi

```
#include <stdio.h>

int main()
{
//isi program
printf("selamat datang di C");
return 0;
}
```

Struktur Program C++

Bentuk Lama	Bentuk Modern
<pre>#include <iostream.h> int main() { //isi program return 0; }</pre>	<pre>#include <iostream> int main() { //isi program return 0; }</pre>

Contoh :

Menampilkan selamat datang di C++

Bentuk Lama

```
#include <iostream.h>
int main()
{
//isi program
cout<<"selamat datang di C";
return 0;
}
```

Bentuk Modern

```
#include <iostream>
int main()
{
//isi program
```

```
std::cout<<"selamat datang di C++";  
return 0;  
}
```

Atau

```
#include <iostream>  
using namespace std;  
int main()  
{  
//isi program  
cout<<"selamat datang di C++";  
return 0;  
}
```

Pada contoh program C terdapat kalimat

```
printf("selamat datang di C");
```

Yang berfungsi untuk menampilkan tulisan selamat datang di C dengan memakai perintah printf. Perintah tersebut ada di stdio.h amaka header yang dipakai adalah stdio.h

Sedangkan pada contoh CPP yang dipakai adalah perintah cout . Perintah ini ada di header file iostream.h yang dalam bentuk CPP modern ada di iostream. Dan penulisannya bisa dengan std::cout. Agar penulisannya cukup dengan cout saja maka perlu ditambahkan perintah

```
using namesapace std;
```

Perintah tersebut yang mengasumsikan semua perintah yang ada memakai std.

Cara membuat program

Bagaimana cara membuat program yang sebenarnya ?

Untuk menjawab pertanyaan tersebut maka akan dibuat suatu studi kasus yang sederhana yang sering kita jumpai di kehidupan sehari-hari sebagai berikut

Ibu membeli gula sebanyak 1 karung gula pasir dari toko grosir dan ingin menjualnya dalam kemasan 0.5 kg. Untuk keperluan tersebut ibu menyediakan kantung ukuran 0.5 kg sebanyak yang diperlukan, Ibu tersebut kebingungan Ketika ternyata dalam 1 karung tersebut setelah ditimbang berat rata-rata lebih dari 100 kg. Biasanya sekitar 100,25 kg sampai dengan 100,35 kg. Bagaimana bentuk program yang sesuai untuk menyelesaikan persoalan ibu tersebut ?

Sebelum menyelesaikannya secara program maka kita harus mengetahui cara menyelesaikan persoalan ibu tersebut. Sebenarnya kita cukup dengan membagi 1 karung dengan 0.5 kg. Misalnya jika 1 karung setara dengan 100,25 kg maka cara menyelesaikannya adalah:

$100,25 \div 0,5$ dan hasilnya adalah 200 dan sisa 0,25

Secara program bisa menggunakan perintah div dan mod menjadi :

$100,25 \text{ div } 0,5 = 200$ dan

$100,25 \text{ mod } 0,5 = 0,25$

Kemudian dalam Bahasa C dicari fungsi yang bisa dipakai untuk mewujudkannya. Ternyata **div** setara dengan operator `"/` dan **mod** bisa digantikan dengan memakai operator `"%"`. Namun kenyataan yang diperoleh ternyata operator tersebut hanya sesuai bila bilangan yang dibagi maupun pembaginya adalah tipe integer (tanpa tanda desimal) sedangkan bilangan yang dipakai adalah bilangan dengan tanda desimal (ada tanda desimalnya). Jadi program C yang dibust tidak bisa memakai operator `"/` dan `"%"`. Namun ada cara yang bisa digunakan yaitu dengan proses pengurangan.

Contoh $5 : 2 = 2$ sisa 1 dengan cara

$5 - 2 = 3$ pengurangan 1

$3 - 2 = 1$ pengurangan 2

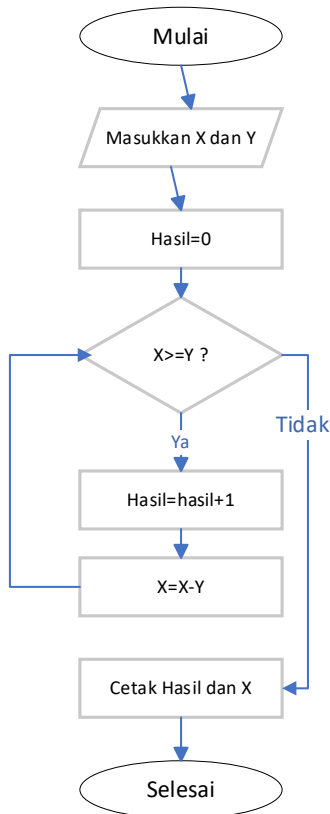
Terlihat pada pengurangan yang ke-2 diperoleh sisa yaitu 1 yang nilainya kurang dari pembaginya yaitu 2. Jadi hasil akhir adalah 5 dibagi 2 adalah 2 (banyak pengurangan-nya) dan sisa 1 (nilai akhir yang kurang dari 2)

Secara algoritma

1. Masukkan nilai yang akan dibagi, sebut saja x (pada contoh 5).

2. Masukkan nilai pembaginya sebut y (pada contoh 2)
3. Kurangi x dengan y dan catat prosesnya (berapa kali dan hasilnya)
4. Apakah hasilnya lebih dari pembagi ?
5. Jika no. 4 ya, maka lakukan no 3
6. Jika no. 4 tidak, maka hasil akhir diperoleh (berapa kali dan sisanya yaitu hasilnya Pada contoh dengan nilai $x=5$ dan $y=2$ diperoleh 2 kali dan sisanya 1)

Bentuk Flowchart



Bentuk program C

```
#include <stdio.h>

int main()
{
float X,Y;
int Hasil;

//masukkan X dn Y
printf("Yang dibagi ? "); scanf("%f",&X);
printf("Pembaginya ? "); scanf("%f",&Y);
//Hasil awal bernilai 0
Hasil=0;

//Proses pengecekan dan pembagian
while (X>=Y) { //selama X>=Y
    Hasil=Hasil+1; //Hasil+1
    X=X-Y; //X=X-Y
} //selesai X<Y
printf("Hasilnya = %i dan sisa %f ",Hasil,X);
return 0;
}
```

Bentuk program CPP

```
#include <iostream>

using namespace std;

int main()
```

```
{
float X,Y;
int Hasil;
//masukkan X dn Y
cout<<"Yang dibagi ? "; cin>>X;
cout<<"Pembaginya ? "; cin>>Y;
//Hasil awal bernilai 0
Hasil=0;
//Proses pengecekan dan pembagian
while (X>=Y) { //selama X>=Y
    Hasil=Hasil+1; //Hasil+1
    X=X-Y; //X=X-Y
} //selesai, keluar while, X<Y
cout<<"Hasilnya ="<<Hasil<<" sisa "<<X;
return 0;
}
```




BAB 8

PEMROGRAMAN MODULAR

1. Pendahuluan

Pembelajaran mengenai bahasa pemrograman biasanya diawali dengan permasalahan yang sederhana dan tidak memerlukan program yang panjang. Hal ini tentu tidak akan bermasalah ketika kita menyelesaikan program tersebut. Lain halnya jika permasalahan menjadi kompleks dan butuh alur penyelesaian yang cukup panjang, pasti akan timbul beberapa permasalahan dalam pemrograman. Beberapa kesulitan yang sering kita temui diantaranya adalah:

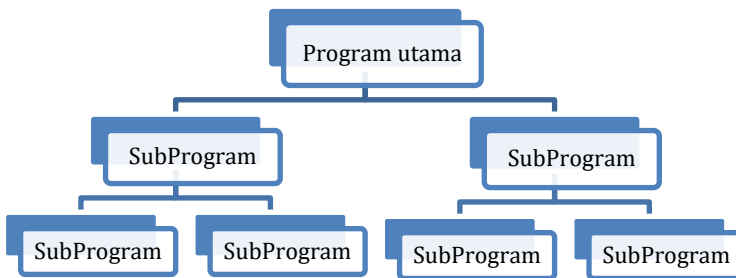
- Kesulitan dalam hal mencari dan mengingat variabel-variabel yang sudah dideklarasikan sebelumnya.
- Kesulitan dalam melakukan dokumentasi
- Kesulitan dalam mencari kesalahan dalam program
- Kesulitan dalam melihat efisiensi algoritma

- Source code biasanya ditulis secara berulang-ulang untuk permasalahan yang mengandung solusi yang sama

Berdasarkan hal tersebut, maka muncullah pemrograman modular. Paradigma pemrograman modular diperkenalkan pertama kali oleh Information & System Institute, Inc. pada acara the National Symposium on Modular Programming pada tahun 1968 oleh salah satu tokoh pionir dalam pemrograman modular yaitu Larry Constantine.

Pemrograman modular adalah pemrograman yang dilakukan dengan cara membuat subprogram (modul) di luar program utama.

Skema pemrograman modular adalah sebagai berikut:



Keuntungan dengan membuat modul dari program yang cukup besar diantaranya adalah:

- Menghindari penulisan teks program yang sama secara berulang-ulang

- Memudahkan dalam menemukan kesalahan ketika program dieksekusi
- Memudahkan proses dokumentasi program

2. Pemrograman Modular Pada C

Sejak awal pemrograman C sudah menerapkan sistem modul, dimana pada program utama terdapat bagian-bagian yang dinamakan header, program utama, dan subprogram. Teknik pemrograman modular dalam program C disebut dengan Function (Fungsi).

Pada prosesnya, terkadang suatu fungsi memerlukan nilai awal (parameter), akan tetapi tidak semua fungsi memerlukannya. Keuntungan menggunakan fungsi diantaranya adalah:

- Mempermudah pada saat penelusuran kesalahan dalam sebuah program (top-down).
- Membagi program utama menjadi beberapa program kecil disesuaikan dengan tujuannya masing-masing sehingga dapat menghindari penulisan source code yang sama secara berulang-ulang (divide-and-conquer).
- Kesalahan dan modifikasi dapat dilokalisasi pada modul tertentu.
- Memudahkan proses dokumentasi.
- Reusability.

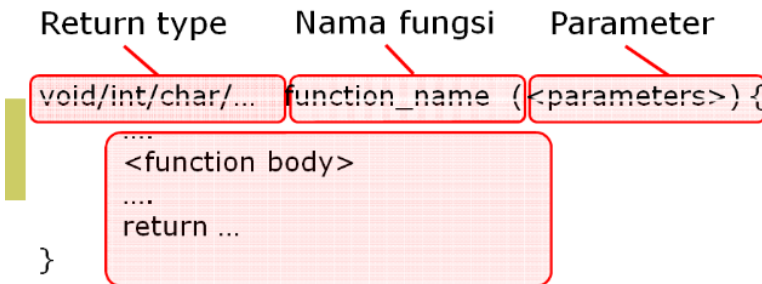
Sebuah fungsi dikatakan baik jika memiliki kriteria berikut:

- Bernilai fan-in yang tinggi (intensitas pemanggilan modul)
- Bernilai fan-out rendah (bersifat spesifik)
- Bernilai self-contained tinggi.

Pada proses perancangan fungsi, perlu diperhatikan hal-hal berikut ini:

- Input data
- Tujuan dari fungsi
- Algoritma pengolahan data
- Hasil output

Bentuk umum:



Contoh fungsi:

```
[*]function.cpp  
1 #include<stdio.h>  
2 int hitung(int A, int B);  
3  
4 main(){  
5     int A, B, T;  
6     A=5; B=2; T=0;  
7     T=hitung(A,B);  
8     printf("\n %i", T);  
9 }  
10  
11 int hitung(int A, int B){  
12     int T;  
13     A=A*2;  
14     B=B*2;  
15     T=A+B;  
16     return(T);  
17 }
```

main program memanggil fungsi

fungsi

3. Struktur Fungsi

Struktur fungsi pada C terdiri dari:

- Deklarasi fungsi

Terdiri dari judul fungsi, tipe data void yang akan dikembalikan, tidak ada kode implementasi fungsi tersebut.

Bentuk umum:

```
tipe_data | void nama_fungsi([argumen 1, argumen 2, ...]);
```

- Definisi fungsi

Terdiri dari fungsi prototipe yang disertai dengan kode implementasi dari fungsi tersebut, berisikan instruksi yang akan melakukan tugas sesuai dengan tujuan dari fungsi tersebut.

4. Jenis-jenis Fungsi C

Pada pemrograman C terdapat dua jenis fungsi yaitu:

- Fungsi yang tidak mengembalikan nilai (void)

Fungsi void sendiri sering disebut sebagai prosedur. Fungsi ini disebut void karena fungsi tersebut tidak mengembalikan suatu nilai keluaran yang didapat dari hasil proses fungsi tersebut. Ciri-ciri dari fungsi void adalah tidak ada keyword "return", tidak ada tipe data di dalam deklarasi fungsi, menggunakan keyword "void", tidak

dapat langsung ditampilkan hasil outputnya, dan tidak memiliki nilai kembalian fungsi.

Contoh:

```
void tampilkan_jml(int a,int b){
    int jml;
    jml = a + b;
    printf("%d",jml);
}
```

Keyword "void" juga dapat digunakan jika suatu fungsi tidak mengandung parameter apapun.

```
void print_error(void){
    printf("Error : unexpected error occurred!");
}
```

- Fungsi yang mengembalikan nilai (non-void)

Fungsi non-void sendiri sering disebut function. Fungsi ini disebut non-void karena mengembalikan nilai kembalian yang berasal dari keluarann hasil proses function tersebut. Ciri-ciri dari fungsi non-void adalah adanya keyword "return", ada tipe data yang mewakili deklarasi fungsi, tidak ada keyword "void", dapat dianalogikan sebagai variabel yang memiliki tipe data tertentu sehingga dapat

langsung ditampilkan hasil outputnya, dan memiliki nilai kembalian fungsi.

Contoh:

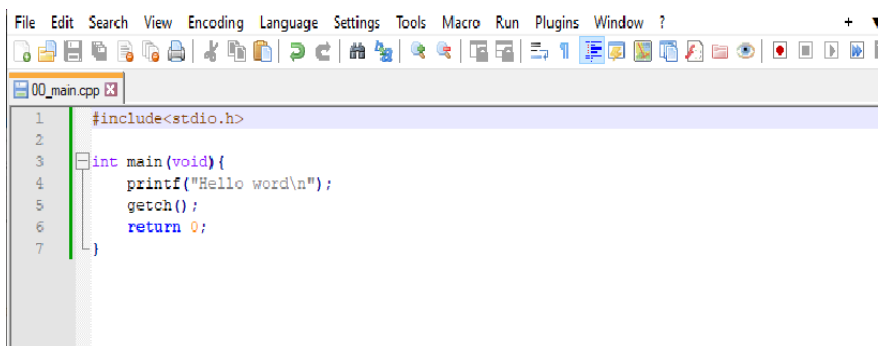
```
int jumlah(int a, int b){
    int jml;
    jml = a + b;
    return jml;
}
```

5. Latihan

Disini kita akan melihat perbedaan program C yang dieksekusi langsung dari program utama dan yang menggunakan modular

— Hello Word

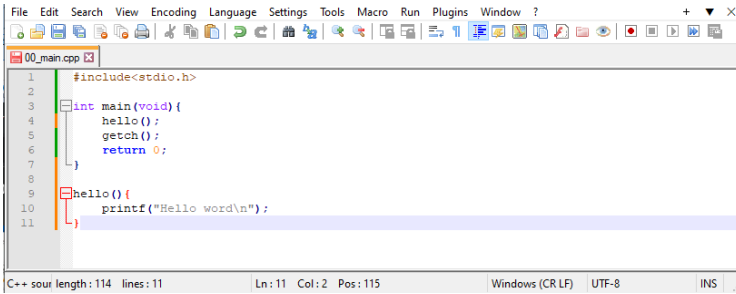
Ilustrasi 1: (tanpa fungsi)



The image shows a screenshot of a code editor window titled '00_main.cpp'. The editor contains the following C code:

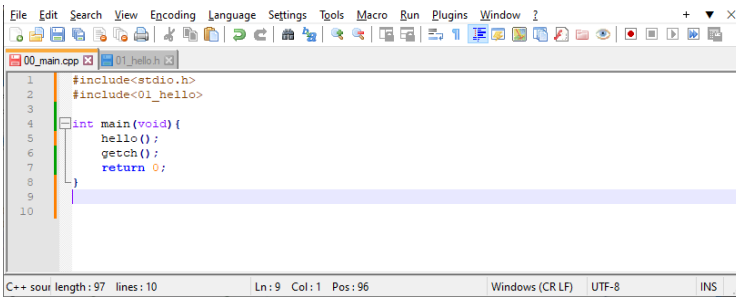
```
1  #include<stdio.h>
2
3  int main(void){
4      printf("Hello word\n");
5      getch();
6      return 0;
7  }
```

Ilustrasi 2: (dengan fungsi pada satu file)

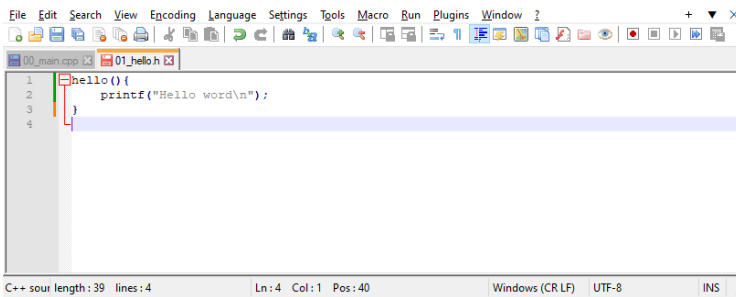


```
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
00_main.cpp
1 #include<stdio.h>
2
3 int main(void) {
4     hello();
5     getch();
6     return 0;
7 }
8
9 hello() {
10     printf("Hello word\n");
11 }
C++ sour length: 114 lines: 11 Ln: 11 Col: 2 Pos: 115 Windows (CR LF) UTF-8 INS
```

Ilustrasi 3: (dengan fungsi pada file yang berbeda)



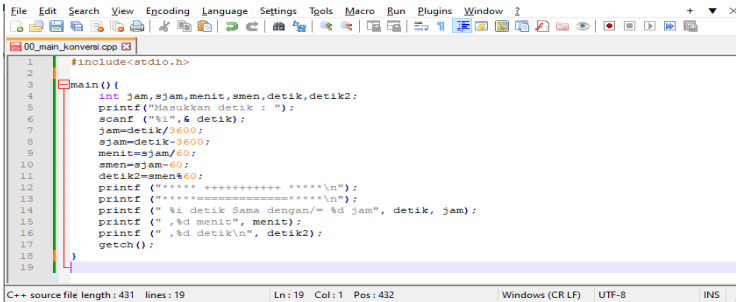
```
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
00_main.cpp 01_hello.h
1 #include<stdio.h>
2 #include<01_hello.h>
3
4 int main(void) {
5     hello();
6     getch();
7     return 0;
8 }
9
10
C++ sour length: 97 lines: 10 Ln: 9 Col: 1 Pos: 96 Windows (CR LF) UTF-8 INS
```



```
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
00_main.cpp 01_hello.h
1 hello() {
2     printf("Hello word\n");
3 }
4
C++ sour length: 39 lines: 4 Ln: 4 Col: 1 Pos: 40 Windows (CR LF) UTF-8 INS
```


— Konversi detik ke Jam, menit dan detik

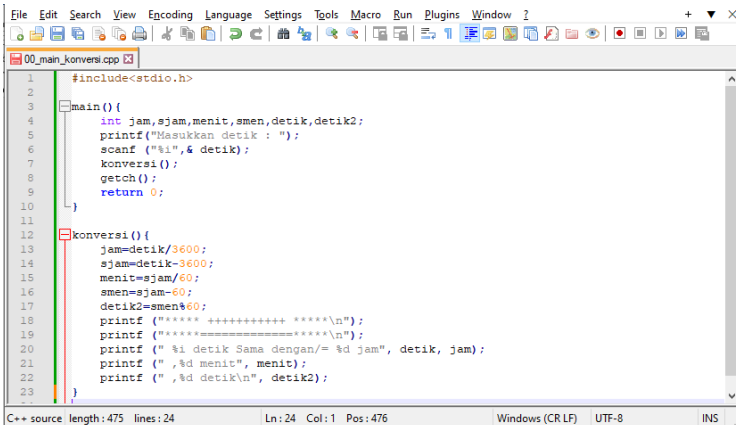
Ilustrasi 1: (tanpa fungsi)



```
1 #include<stdio.h>
2
3 main() {
4     int jam,sjam,menit,smen,detik,detik2;
5     printf("Masukkan detik : ");
6     scanf ("%i",& detik);
7     jam=detik/3600;
8     sjam=detik-3600;
9     menit=sjam/60;
10    smen=sjam-60;
11    detik2=smen%60;
12    printf ("***** \n");
13    printf ("***** \n");
14    printf (" %i detik Sama dengan/= %d jam", detik, jam);
15    printf (" ,%d menit", menit);
16    printf (" ,%d detik\n", detik2);
17    getch();
18 }
19
```

C++ source file length: 431 lines: 19 Ln: 19 Col: 1 Pos: 432 Windows (CR LF) UTF-8 INS

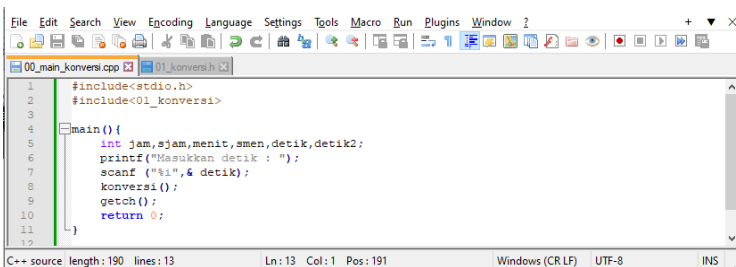
Ilustrasi 2: (dengan fungsi pada satu file)



```
1 #include<stdio.h>
2
3 main() {
4     int jam,sjam,menit,smen,detik,detik2;
5     printf("Masukkan detik : ");
6     scanf ("%i",& detik);
7     konversi();
8     getch();
9     return 0;
10 }
11
12 konversi() {
13     jam=detik/3600;
14     sjam=detik-3600;
15     menit=sjam/60;
16     smen=sjam-60;
17     detik2=smen%60;
18     printf ("***** \n");
19     printf ("***** \n");
20     printf (" %i detik Sama dengan/= %d jam", detik, jam);
21     printf (" ,%d menit", menit);
22     printf (" ,%d detik\n", detik2);
23 }
```

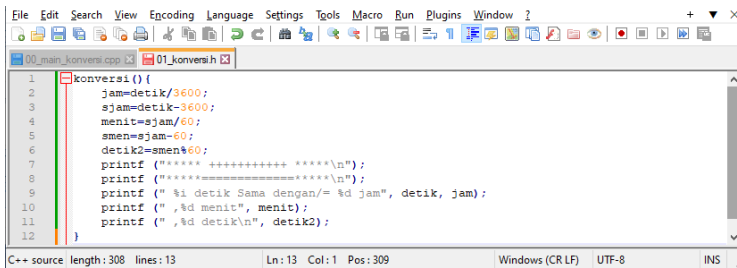
C++ source length: 475 lines: 24 Ln: 24 Col: 1 Pos: 476 Windows (CR LF) UTF-8 INS

Ilustrasi 3: (dengan fungsi pada file yang berbeda)



```
1 #include<stdio.h>
2 #include<01_konversi>
3
4 main() {
5     int jam,sjam,menit,smen,detik,detik2;
6     printf("Masukkan detik : ");
7     scanf ("%i",& detik);
8     konversi();
9     getch();
10    return 0;
11 }
12
```

C++ source length: 190 lines: 13 Ln: 13 Col: 1 Pos: 191 Windows (CR LF) UTF-8 INS



```
1 konversi() {
2     jam=detik/3600;
3     sjam=detik-3600;
4     menit=sjam/60;
5     smen=sjam-60;
6     detik2=smen*60;
7     printf ("***** ++++++\n");
8     printf ("*****\n");
9     printf (" %i detik Sama dengan/= %d jam", detik, jam);
10    printf (" ,%d menit", menit);
11    printf (" ,%d detik\n", detik2);
12 }
```

6. Soal Latihan

Buatlah program modular untuk mengkonversi nilai mahasiswa ke dalam nilai mutu dimana input data yang dimasukkan adalah jumlah kehadiran, nilai tugas, nilai UTS, dan nilai UAS.

Ketentuan:

nilai akhir = ((jumlah kehadiran/14)*10%) + (nilai tugas*15%) + (nilai UTS*35%)+(nilai UAS*40%)

Nilai mutu:

A = 80-100

B = 65-79

C = 55-64

D = 45-54

E = \leq 44



BAB 9. ARRAY

Pengenalan Array

Array merupakan tempat penyimpanan yang dapat menyimpan banyak data dengan menggunakan nama yang sama. Data yang tersimpan dalam Array dinamakan elemen array, data ini harus memiliki tipe data yang serupa. Tiap data yang disimpan dalam array memiliki subscript atau indeks untuk menunjukkan letak penyimpanannya. Pada awal pendeklarasian array, programmer dapat menentukan berapa banyak memori yang nantinya akan digunakan. Berikut contoh dari penyimpanan array berisi umur yang tersimpan dalam 3 indeks.

Indeks 0	Indeks 1	Indeks 2
50	75	30

Deklarasi Array

Dalam mendeklarasikan array, harus diawali dengan pemilihan tipe data yang sesuai dengan jenis elemen array. Deklarasi array dilanjutkan dengan pembuatan nama variabel dan ukuran dari array. Struktur dalam pendeklarasian array dapat dilihat pada contoh berikut :

```
tipe_data nama_array [ukuran];
```

Pendeklarasian array dalam bahasa C dapat dilihat pada contoh berikut :

```
int no_hp [50];
```

Inisialisasi Array

Inisialisasi array merupakan proses pemberian nilai awal pada elemen-elemen array pada saat dideklarasikan. Dalam bahasa C, terdapat beberapa cara untuk melakukan inisialisasi array.

1. Inisialisasi Awal

Array dapat diinisialisasi dengan melakukan pemberian nilai di awal deklarasi, berikut contohnya dalam Bahasa C:

```
int angka[5] = {1, 5, 7, 11, 17};
```

Dalam contoh di atas, array di buat dengan nama variabel “angka” dan memiliki data “int” dengan 5 elemen. Nilai awal pada array tersebut di isikan dengan menggunakan kurung kurawal {}. Elemen-elemen array akan di masukkan dalam indeks secara berurutan.

2. Inisialisasi Sebagian

Array juga dapat diberi nilai default. Misalnya, jika dalam inisialisasi array hanya di lakukan pada Sebagian data saja, maka sisa dari elemen array adalah 0. Berikut contoh penggunaannya :

```
int angka[5] = {1, 2, 3};
```

Dalam contoh di atas, elemen-elemen array dengan indeks 0, 1, dan 2 akan diinisialisasi dengan nilai 1, 2 dan 3. Elemen dengan indeks 3 dan 4 akan memiliki nilai default yaitu 0.

3. Inisialisasi Tanpa Ukuran

Dalam beberapa kasus, array dapat dapat diinisialisasi tanpa menyebutkan ukurannya secara tepat. Compiler akan menentukan ukuran array berdasarkan jumlah elemen yang diinisialisasi. Berikut adalah contoh penggunaan inisialisasi tanpa ukuran:

```
int angka[] = {1, 2, 3, 4, 5};
```

Dalam contoh di atas, array angka akan memiliki ukuran 5 karena terdapat 5 elemen yang diinisialisasi.

4. Inisialisasi Karakter

Array yang memiliki tipe data berupa karakter (string), dapat di inisialisasi dengan tanda kutip ganda ("). Berikut adalah contoh penggunaan inisialisasi karakter:

```
char nama[] = "Erik";
```

Dalam contoh di atas, array nama akan berisi karakter-karakter 'E', 'r', 'i', 'k', dan karakter null (\0) sebagai penanda akhir dari string.

Menampilkan isi dalam array

Untuk menampilkan hasil dari array, dapat menggunakan perintah berikut :

```
printf("%c", nama_array[indeks]);
```

Jika terdapat banyak data dalam array, maka untuk menampilkan isinya dapat menggunakan perulangan seperti pada contoh berikut ini :

```
for (int i = 0; i < 5; i++) {
    printf("Nilai array ke %d adalah %c\n", i, nama_array[i]);
}
```

Memasukkan nilai pada array

Nilai dalam array yang telah diinisialisasi dapat diganti dengan cara memasukkan nilai yang lain. Contoh penggantinya dapat dilihat pada coding berikut :

```
#include <stdio.h>
int main() {
    // isi awal array
    char huruf[5] = {'a', 'b', 'c', 'd', 'e'};
    // mengubah isi data array
    huruf[2] = 'z';
    // mencetak isi array
    printf("Huruf: %c\n", huruf[2]);
    return 0;
}
```

Array 2 Dimensi

Array 2 dimensi merupakan struktur data berupa matriks yang terdiri dari baris dan kolom. Tiap elemen dalam array 2 dimensi dapat diakses dengan menggunakan indeks yang berisikan baris dan kolom. Berikut contoh dari penyimpanan array berisi umur yang tersimpan dalam 3 baris dan 3 kolom indeks.

Indeks	Kolom 0	Kolom 1	Kolom 2
Baris 0	30	50	70
Baris 1	28	40	50
Baris 2	10	30	53

Deklarasi Array 2 Dimensi

Struktur dalam pendeklarasian array 2 dimensi dapat dilihat pada contoh berikut :

```
tipe_data nama_array[baris][kolom];
```

Untuk mendeklarasikan array 2 dimensi dalam bahasa C, sintaks yang digunakan adalah sebagai berikut:

```
int umur[3][3];
```

Inisialisasi Array 2 Dimensi

Array 2 dimensi dapat diinisialisasi saat deklarasi atau setelahnya dengan menggunakan loop. Contohnya sebagai berikut :

```
int umur[3][3] = {{30, 50, 70}, {28, 40, 50}, {10, 30, 53}};
```

Menampilkan Elemen Array 2 Dimensi

Untuk mengakses nilai pada elemen array 2 dimensi, dapat menggunakan indeks baris dan kolomnya. Akses isi dalam elemen array dapat dilakukan dengan pemanggilan 1 nilai maupun keseluruhan isi array. Contoh :

```
printf("array indeks [0][0] = %d\n", umur[0][0]);
```

Hasil yang akan ditampilkan dari coding tersebut adalah nilai elemen yang terdapat pada indeks baris 0 dan kolom 0, yaitu angka 30.

Menampilkan Elemen Array 2 Dimensi dengan Looping

Untuk menampilkan isian dalam array 2 dimensi dengan menggunakan looping / perulangan, dibutuhkan adanya perulangan yang bersarang. Perulangan yang pertama akan memberikan umpan balik pada bagian kolom, perulangan yang kedua akan memberikan umpan balik pada bagian baris. Contoh penggunaannya dalam Bahasa C dapat dilihat pada gambar berikut :

```
for (int i = 0; i < baris; i++)
{
    for (int j = 0; j < kolom; j++)
    {
        printf("array indeks [%d][%d] = %d\n", i, j, umur[i][j]);
    }
}
```

Memasukkan nilai pada array

Untuk memberikan input / masukkan dalam array 2 dimensi, dapat dilakukan dengan menggunakan pemanggilan indeks baris dan kolom pada array. Contoh penggunaannya dalam Bahasa C adalah sebagai berikut :

```
#include <stdio.h>

int main() {
    int baris = 3;
    int kolom = 3;
    int umur[3][3];

    for (int i = 0; i < baris; i++) {
        for (int j = 0; j < kolom; j++) {
            scanf("%d", &umur[i][j]);
            printf("\n");
        }
    }

    return 0;
}
```

Pembuatan coding array 1 dan 2 dimensi

Berikut contoh pembuatan array 1 dimensi dengan studi kasus penyimpanan umur. Umur yang disimpan memiliki tipe data interger. Jumlah indeks penyimpanan array yang disediakan adalah 3.

```
#include <stdio.h>

int main() {
    int batas = 3;
    int umur[3];

    for (int i = 0; i < batas; i++)
    {
        scanf("%d", &umur[i]);
        printf("\n");
    }

    for (int i = 0; i < batas; i++)
    {
        printf("array indeks [%d] = %d\n", i, umur[i]);
    }

    return 0;
}
```

Berikut contoh pembuatan array 2 dimensi dengan studi kasus penyimpanan umur. Umur yang disimpan memiliki tipe data interger. Jumlah indeks penyimpanan array yang disediakan adalah 9.

```
#include <stdio.h>

int main() {
    int baris = 3;
    int kolom = 3;
    int umur[3][3];

    for (int i = 0; i < baris; i++) {
        for (int j = 0; j < kolom; j++) {
            scanf("%d", &umur[i][j]);
            printf("\n");
        }
    }

    for (int i = 0; i < baris; i++) {
        for (int j = 0; j < kolom; j++) {
            printf("array indeks [%d][%d] = %d\n", i, j, umur[i][j]);
        }
    }

    return 0;
}
```



BAB 10 STRUKTUR

Struktur adalah tipe data yang menyimpan koleksi dari variabel yang sifat dari setiap variabel memiliki tipe yang berbeda. Perbedaan utama antara struktur dan array adalah bahwa array hanya bisa menyimpan informasi dari tipe data yang sama, sedangkan untuk struktur bisa menyimpan informasi dari tipe data yang berbeda.

Deklarasi Struktur

Struktur dapat dideklarasikan dengan menggunakan kata kunci `struct` diikuti dengan nama struktur. Struktur juga dapat memiliki fungsi dalam bahasa pemrograman `c++`. Deklarasi struktur adalah sebagai berikut :

```
struct nama_struktur{  
    tipe_data nama_variabel;  
    tipe_data nama_variabel;  
    .....  
};
```

atau

```

Struct nama_struktur{
    tipe_data nama_variabel;
    tipe_data nama_variabel;
    .....
    tipeData namaFungsi(DaftarParameter){
        //badan fungsi
    }
    tipeData namaFungsi(DaftarParameter){
        //badan fungsi
    }
    .....
};
    
```

Struktur menjadi tipe data yang ditentukan pengguna, setiap nama variabel yang dideklarasikan dalam struktur disebut anggota struktur. Misalnya jika mendefinisikan struktur untuk mahasiswa maka informasi terkait untuk mahasiswa adalah : nim, nama, prodi, biaya. Struktur mahasiswa dapat dinyatakan sebagai :

```

struct mahasiswa{
    int nim;
    string nama;
    string prodi;
    int biaya;
};
    
```

Dalam bahasa pemrograman c++, struktur dan kelas mempunyai banyak kemiripan. Perbedaan struktur dan kelas terdapat pada tingkat akses *default* -nya. Deklarasi struktur jika tidak disertai dengan tingkat aksesnya, maka akan bersifat *public*. Dalam kelas, jika tidak disertai dengan tingkat aksesnya, maka akan bersifat *private*.

```
struct persegi{  
    int panjang;    //panjang dan lebar bersifat public  
    int lebar;  
};
```

Penulisan agar anggota struktur menjadi *private*, dengan menuliskan kata kunci *private* dan diikuti dengan titik dua (:).

```
struct persegi{  
private :  
    int panjang;    //panjang dan lebar bersifat private  
    int lebar;  
};
```

Inisialisasi Struktur

Inisialisasi struktur sama dengan inisialisasi tipe data lainnya, yaitu menugaskan beberapa konstanta ke anggota

struktur. Ketika pengguna tidak eksplisit menginisialisasi struktur, maka C secara otomatis akan melakukannya. Untuk anggota *integer* dan *float* nilainya diinisialisasi ke nol, anggota *char* dan *string* nilainya diinisialisasi ke '\0' secara *default*. Penulisan inisialisasi diapit oleh '{ }' kurung kurawal dan dipisahkan dengan ',' (koma). Sintaks penulisan inisialisasi variabel struktur adalah sebagai berikut :

```
struct nama_struct{
    tipe_data nama_variabel1;
    tipe_data nama_variabel2;
    tipe_data nama_variabel3;
    .....
} struct_var = { constant1, constant2, constant3,..... };
```

Atau

```
struct nama_struct{
    tipe_data nama_variabel1;
    tipe_data nama_variabel2;
    tipe_data nama_variabel3;
    .....
};
struct nama_struct struct_var = { constant1, constant2, constant3,.....
};
```

Contoh penulisan inisialisasi struktur yang bernama mahasiswa yang mempunyai member struktur nim dan biaya yang bertipe *integer* dan *string* adalah :

```
struct mahasiswa{
    int nim;
    string nama;
    string prodi;
    int biaya;
} mhs1 = {6024199, "Serena Nusaibah", "Sistem Informasi",
3500000};
```

atau penulisannya :

```
struct mahasiswa mhs1 = {6024199, "Serena Nusaibah", "Sistem
Informasi", 3500000};
```

Ilustrasi inisialisasi pada struktur :

```
struct mahasiswa mhs1 = { 6024199, "Serena Nusaibah", "Sistem
Informasi", 3500000};
```

6024199	Serena Nusaibah	Sistem Informasi	3500000
NIM	Nama	Prodi	Biaya

```
struct mahasiswa mhs2 = {6024199, "Serena Nusaibah"};
```

6024199	Serena Nusaibah	\0	0
NIM	Nama	Prodi	Biaya

Inisialisasi Parsial adalah kondisi dimana semua anggota struktur tidak diinisialisasi. Untuk anggota struktur yang tidak diinisialisasi maka akan diberikan nilai *default*.

Mengakses member Struktur

Variabel struktur umumnya diakses dengan menggunakan operator '.' (titik). Sintaks penulisan untuk mengakses member struktur :

```
struct_var.nama_member
```

Operator titik digunakan untuk memilih anggota tertentu dari struktur. Sebagai contoh, penulisan untuk menugaskan nilai-nilai ke anggota struktur :

```
mhs1.nim = 6024199;  
mhs1.nama = "Serena Nusaibah";  
mhs1.prodi = "Sistem Informasi";
```

Penulisan untuk memasukkan nilai pada anggota data dari variabel struktur adalah :

bahasa c :

```
scanf("%d", &mhs1.nim);
```


bahasa c++:

```
cin>>mhs1.nama;
```

penulisan untuk mencetak nilai dari struktur adalah :

bahasa c:

```
printf("%f", mhs1.biaya);
```

bahasa c++:

```
cout<<mhs1.biaya;
```

contoh penulisan program *struct* dalam c++

Membuat program untuk menampilkan data informasi mahasiswa :

```
#include<iostream>
```

```
#include<string>
```

```
using namespace std;
```

```
struct mahasiswa{
```

```
    int nim;
```

```
    string nama;
```

```
    string prodi;
```

```
    int biaya;
```

```
}mhs1;
```

```
Int main(){  
    mhs1.nim = 6024199;  
    mhs1.nama = "Serena Nusaibah";  
    mhs1.prodi = "Sistem Informasi";  
    mhs1.biaya = 3500000;  
  
    cout<<"-----Data Mahasiswa-----";  
    cout<<"NIM = "<<mhs1.nim<<endl;  
    cout<<"Nama = "<<mhs1.nama<<endl;  
    cout<<"Prodi = "<<mhs1.prodi<<endl;  
    cout<<"Biaya = "<<mhs1.biaya<<endl;  
}
```

Output :

```
-----Data Mahasiswa-----  
NIM = 6024199  
Nama = Serena Nusaibah Isfiehan  
Prodi = Sistem Informasi  
Biaya = 3500000
```

Menyalin dan Membandingkan Struktur

Di dalam struktur kita dapat menetapkan struktur ke struktur lain yang mempunyai tipe yang sama, sebagai contoh jika kita mempunyai dua variabel struktur yaitu *mhs1* dan *mhs2* bertipe *struct* mahasiswa :

```
struct mahasiswa mhs1 = {6024199, "Serena Nusaibah", "Sistem  
Informasi", 3500000};
```

```
struct mahasiswa mhs2;
```

Kemudian penulisan untuk menetapkan satu variabel struktur ke struktur variabel lainnya adalah :

```
mhs2 = mhs1;
```

Pernyataan diatas adalah menginisialisasi anggota *mhs2* dengan nilai-nilai dari anggota *mhs1*. Untuk melakukan perbandingan nilai, dalam bahasa C tidak mengizinkan perbandingan satu variabel struktur dengan yang lain. Namun yang diperbolehkan adalah membandingkan anggota individu dari struktur dengan anggota dari struktur lainnya. Ketika kita membandingkan anggota individu dengan anggota individu dari struktur lainnya, perbandingannya akan berperilaku seperti perbandingan variabel biasa lainnya. Sebagai contoh jika kita mencoba untuk membandingkan biaya dua mahasiswa, contoh penulisannya :

```
If (mhs1.biaya > mhs2.biaya) // melakukan pengecekan apakah biaya  
dari mhs1 lebih besar dari biaya mhs2
```



BAB 11

PEMROGRAMAN ARRAY OF STRUCT

1. Pendahuluan Pemrograman Array of Struk

Pemograman yang melibatkan Array of Struct mengacu kepada penggunaan Struktur data yang mengandung beberapa elemen dengan jenis jenis data yang berbeda dan ditempatkan dalam sebuah Array. Array of Struct memungkinkan untuk membuat tipe data khusus yang terdiri dari variabel variabel dengan jenis data yang berbeda, sebelum pembahasan lebih mendalam perlu diketahui apa itu Array dan Struct .

Aray merupakan Struktur data yang digunakan untuk mengelola atau menyimpan kumpulan variabel atau elemen jenis data yang sama. Elemen elemen diindeks secara berurutan dengan menggunakan indeks yang dimulai dari indek 0. Aray memungkinkan penyimpanan data urutan teratur dan memberikan akses cepat ke elemen elemen tersebut. Array dibagi kedalam beberapa dimensi yaitu Array 1 dimensi, Array 2 diemensi, sampai dengan Array n-dimensi.

a. Pendeklarasian Array

type-data nama var [ukuran];

type_data : type dari data elemen dalam sebuah Array

(varchar, int, char, float)

nama var : menunjukkan sebuah variabel yang akan digunakan

ukuran : menunjukkan batas maksimal atau tertinggi suatu elemen dari sebuah Array

contoh sebuah array : a ukuran [8];

b. Inisialisasi

Inisialisasi Array melibatkan pemberian nilai awal pada saat pendefinisian array tersebut. Contohnya jika memiliki array yang diberi nama "nilai" dengan Panjang ukuran 8 dapat menggunakan atau menginisialisasikan dengan memberikan nilai awal sebagai berikut

Int nilai [8] = {1, 2, 5, 7, 8, 9, 11, 12 };

Namun masih perlu diperhatikan dari contoh diatas ukuran array yang difenisikan jumlah elemennya harus sesuai dengan ukuran array yang telah didefinisikan.

c. Pengaksesan

Pengisian dan pengambilan nilai pada indeks tertentu dapat dilakukan dengan mengeset nilai atau menampilkan nilai pada

indeks yang dimaksud. Pengaksesan elemen array dapat dilakukan berurutan atau random berdasarkan indeks tertentu secara langsung.

```
#include <iostream>

Int main () {
Int angka [] = {1, 3, 6, 8, 12};
Int n;
For (n=0; n<3; n++)
{
cout<<"cetak data perulangan : <<angka [n];
cout<<"cetak nilai indeks 0 : "<<angka[0]; }
```

Dalam pemograman, Array sangat penting untuk diterapkan karena memungkinkan menyimpan dalam jumlah data yang besar dengan cara yang efisien. Setiap elemen dalam array memiliki posisi yang disebut indeks. Indeks digunakan untuk mengakses atau memanipulasi nilai nilai didalam Array.

2. Struktur data dalam Pemograman

Struktur (struct) adalah cara untuk mengelompokkan beberapa variabel dengan jenis data yang berbeda kedalam satu entitas tunggal. Dalam struct setiap anggota memiliki nama yang unik, ini memungkinkan supaya dapat membuat tipe data yang khusus. Struktur digunakan untuk membuat tipe data khusus yang mencerminkan objek atau konsep dalam program.

Struct digunakan ketika data yang akan dikelompokkan memiliki tipe data yang berbeda, format data struct yang dapat menyimpan variabel variabel dengan nama yang sama, namun memiliki tipe data yang berbeda . variabel variabel tersebut memiliki keterkaitan satu dengan yang lainnya.

a. Deklarasi

```
Struct pegawai  
Char nip [15];  
Char nama [30];  
Float Indeks_kinerja;
```

b. Array Of struct

```
#include <iostream>  
  
Struct pegawai {char nip [15];  
Char nip[15]  
Char nama [30]  
Float indeks_kinerja ;  
};  
  
Int main () {
```

Dalam pemograman Struct memungkinkan untuk menggabungkan beberapa tipe data yang berbeda menjadi satu entitas tunggal, dalam Struct juga bisa mendefinisikan field dengan tipe data yang berbeda yang nantinya dapat digunakan

untuk membuat sebuah objek yang berisikan data tersebut kemudian dalam mendefinisikan struct tersebut diperlukan nama variabel beserta tipe data masing masing.

3. Array dalam Pemograman

Indeks Array dimulai dari 0 untuk elemen pertama , kemudian 1 untuk elemen kedua, dan seterusnya. Oleh karena itu array digunakan untuk merepresentasikan jenis record yang sama yang kemudian disimpan secara terus menerus atau berulang ulang berdasarkan indeksinya. Jika ada array dengan ukuran N , indeksinya akan berkisaran dari 0 sampai N-1. Jika memiliki Array denga 5 elemen, indeksinya adalah dari 0 sampai 4.

Dalam pemograman untuk melakukan akses Array bisa menggunakan sintaks yang mencakup nama Arrya diikuti oleh tanda kurung siku yang berisi inks dengan yang ingin diakses. Misalkan jika memiliki Array yang diberi nama “Bilangan Ganjil” yang berisi [1,3,5,7,9]. Elemen elemen dari array tersebut dapat diakses menggunakan angka yang ada dalam kurung siku, indeks[0] untuk mengakses elemen pertma [1], indeks [1] untuk mengakses elemen kedua [3] dan begitu seterusnya.

Contoh Array dalam Algoritma

a: Array [1...5] of interger

Terbentuk adanya 5 variabel dengan indeks 0 sampai 4 bertipe interger.

Dalam Bahasa pemograman C++ ditulis **int a[5]**

4. Array Of Struct Dalam Pemograman

Array of Struct digunakan dalam menyimpan sejumlah elemen data yang kompleks dan terstruktur dalam satu variabel. Setiap elemen dalam array tersebut bisa memiliki beberapa atribut atau field yang berbeda beda. Misalnya dalam menyimpan informasi tentang sejumlah Mahasiswa, menggunakan array of Struct setiap elemen struct mewakili satu mahasiswa dan atribut atributnya berisi Nim, Nama, Kelas, dan lainnya.

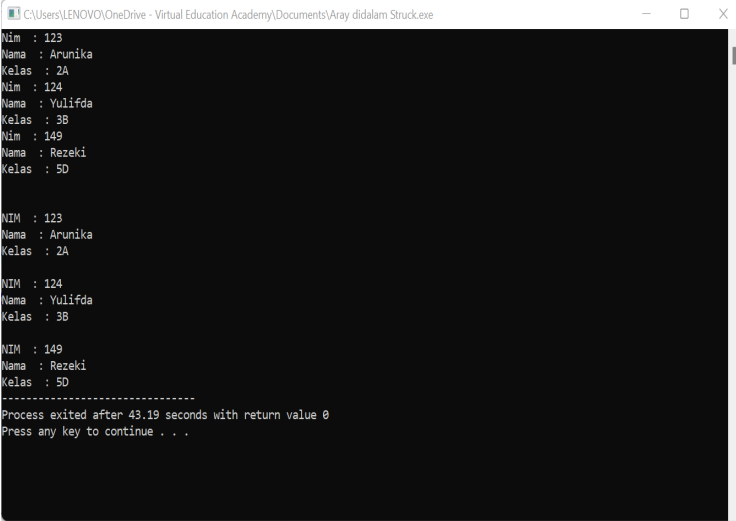
Langkah Langkah dalam membuat array of struct

1. Tentukan Struct yang akan digunakan dalam Array
2. Deklarasikan array of struct
3. Isi data kedalam array of struct
4. Gunakan array of struct

Contoh penggunaan Array of Struct dalam Bahasa C++ memakai aplikasi Dev C++ 6.3

```
#include<iostream>
using namespace std;
struct Mahasiswa
```

```
{
char Nim[7];
char Nama[20];
char Kelas[10];
};
main()
{
Mahasiswa[5];
int i;
for (i=0; i<3; i++)
{
cout<<"Nim : ";
cin>>Mahasiswa[i].Nim;
cout<<"Nama : ";
cin>>Mahasiswa[i].Nama;
cout<<"Kelas : ";
cin>>Mahasiswa[i].Kelas;
}
for(i=0; i<3; i++)
{
cout<<"\n\nNIM : "<<Mahasiswa[i].Nim;
cout<<"\nNama : "<<Mahasiswa[i].Nama;
cout<<"\nKelas : "<<Mahasiswa[i].Kelas;
}
}
```



```
C:\Users\LENOVO\OneDrive - Virtual Education Academy\Documents\Aray didalam Struct.exe
Nim : 123
Nama : Arunika
Kelas : 2A
Nim : 124
Nama : Yulifda
Kelas : 3B
Nim : 149
Nama : Rezeki
Kelas : 5D

NIM : 123
Nama : Arunika
Kelas : 2A

NIM : 124
Nama : Yulifda
Kelas : 3B

NIM : 149
Nama : Rezeki
Kelas : 5D
-----
Process exited after 43.19 seconds with return value 0
Press any key to continue . . .
```

Gambar 1 Tampilan Gambar Hasil Excekuksi Program

Dalam codingan diatas didefinisikan sebuah Struct tentang Mahasiswa yang mempunyai 3 variabel nim, nama dan kelas yang masing masing merupakan Array dengan Panjang karakter tertentu. Dengan menggunakan Array of Struct program ini mengelola data mahasiswa dalam bentuk yang terStruct tur. Informasi setiap data mahasiswa disimpan dalam elemen Array of Struct sehingga dapat memudahkan dalam mengakses dan memanipulasi data tersebut.

Contoh program Data mahasiswa

```
#include <iostream>
using namespace std;
```

```

struct mahasiswa
{
    char nm_mhs[25], nim[12],jur[5];
    int jml_sks, program_studi;
};
struct mahasiswa bayar[5];
main(){
    int bts,var,tetap;
    for(int i=0; i<2; i++)
    {
        //data yang dimasukan
        cout<<"===== ";
        cout<<"\nNama Mahasiswa    = ";cin>>bayar[i].nm_mhs;
        cout<<"NIM            = ";cin>>bayar[i].nim;
        cout<<"Jurusan    = ";cin>>bayar[i].jur;
        input:
        cout<<"Program[1=S2, 2=S3] = ";
        cin>>bayar[i].program;

        if(bayar[i].program<0 || bayar[i].program>2)
        {
            cout<<"Program tidak sesuai\n";
            goto input;
        }
        cout<<"Jumlah sks        = "; cin>>bayar[i].sks;
        if(bayar[i].program==1)
        {
            tetap=500000;
            var=bayar[i].sks*25000;

```

```

}else if(bayar[i].program==2)
{
    tetap=750000;
    var=bayar[i].sks*50000;
}cout<<endl;

```

//Output data

```

cout<<"\n\n-----\n";
cout<<" Output ";
cout<<"\n-----\n";
cout<<"\nNama Mahasiswa = "<<bayar[i].nama;
cout<<"\nNIM      = "<<bayar[i].nim;
cout<<"\nProgram   = "<<bayar[i].program;
cout<<"\nJumlah sks = "<<bayar[i].sks;
cout<<"\nSPP tetap  = "<<tetap;
cout<<"\nSPP variabel = "<<var;
cout<<endl<<endl;
}
}

```

Berikut adalah Hasil Runing dari Program

```

-----
Nama mhs      = Arunika
NIM           = 1234
Jurusan       = PTIDK
Program(1=0, z=51) = 1
Jumlah sks    = 43
-----
Output
-----
Nama mhs      = Arunika
NIM           = 1234
Program       = 1
Jumlah sks    = 43
SPP tetap     = 500000
SPP variabel   = 1075000
-----
Nama mhs      =

```

Gambar 2 Hasil runing Program

BAB 12

PEMROGRAMAN PENCARIAN

Pemrograman pencarian adalah sebuah proses pembuatan program yang bertujuan untuk mencari suatu data dalam sebuah kumpulan data atau struktur data. Sebelum membahas dan membuat pemrograman pencarian, kita perlu mengerti dahulu definisi dan pengertian algoritma pencarian, istilah-istilah dalam algoritma pencarian, karakteristik algoritma pencarian, dan jenis-jenis algoritma pencarian. Berikut pembahasannya.

A. Pengertian Algoritma Pencarian

Dalam bidang ilmu komputer, algoritma pencarian merujuk pada algoritma yang diciptakan dengan tujuan memecahkan permasalahan yang berkaitan dengan pencarian. Baik berupa pencarian elemen dari sebuah himpunan, pencarian node dari tree, pencarian graf yang mencari rute terpendek dari sebuah graf.

Algoritma pencarian merupakan serangkaian langkah prosedural yang digunakan untuk mencari data spesifik dalam kumpulan data yang ada. Algoritma ini dapat dianggap sebagai salah satu fondasi dalam operasi komputasi. Ketika sebuah sistem melakukan pencarian data, perbedaan antara aplikasi yang cepat dan lambat sering kali tergantung pada penggunaan algoritma pencarian yang tepat. Dengan memilih dan menerapkan algoritma pencarian yang efisien, sistem dapat menemukan data yang dicari dengan lebih cepat dan efektif.

B. Istilah-Istilah dalam Algoritma Pencarian

Pencarian adalah suatu prosedur yang dilakukan langkah demi langkah untuk memecahkan masalah pencarian dalam suatu ruang pencarian (search space) yang telah ditentukan. Masalah pencarian umumnya melibatkan tiga faktor utama sebagai berikut:

1. Search space (ruang pencarian): Merupakan representasi dari kumpulan solusi yang mungkin, yang tersedia bagi sistem. Ruang pencarian ini dapat berupa struktur data seperti graf, pohon, atau himpunan nilai yang mungkin.
2. Start state (keadaan awal): Merupakan keadaan atau kondisi awal yang ditemui oleh agen saat memulai proses pencarian. Pencarian dimulai dari keadaan awal

ini dan bertujuan untuk mencapai keadaan tujuan yang diinginkan.

3. Goal test (pengujian tujuan): Merupakan suatu fungsi yang mengevaluasi keadaan saat ini dan menentukan apakah keadaan tersebut merupakan keadaan tujuan yang diinginkan atau bukan. Fungsi ini mengamati keadaan saat ini dan memberikan nilai kembalian berdasarkan apakah tujuan sudah tercapai atau belum.

Dengan memanfaatkan prosedur pencarian, sistem dapat menjelajahi ruang pencarian secara sistematis, memulai dari keadaan awal, dan menggunakan goal test untuk menentukan apakah keadaan tujuan telah tercapai atau masih perlu dilakukan langkah pencarian tambahan.

C. Karakteristik Algoritma Pencarian

Berikut adalah empat sifat penting dari algoritma pencarian untuk membandingkan efisiensi dari sebuah algoritma:

1. Keoptimalan (Optimality): Algoritma pencarian dikatakan optimal jika dapat menemukan solusi terbaik atau solusi optimal dalam ruang pencarian yang diberikan. Solusi optimal adalah solusi yang memenuhi

kriteria yang diinginkan dengan biaya minimum atau hasil yang maksimum.

2. Kelengkapan (Completeness): Algoritma pencarian dikatakan lengkap jika dapat menemukan solusi jika ada solusi yang ada dalam ruang pencarian yang diberikan. Artinya, algoritma tersebut tidak akan mengalami kegagalan dalam menemukan solusi yang ada jika solusi tersebut memang ada.
3. Efisiensi Waktu (Time Efficiency): Efisiensi waktu mengacu pada seberapa cepat algoritma pencarian dapat menemukan solusi. Algoritma yang lebih efisien waktu akan dapat menemukan solusi dalam waktu yang lebih singkat.
4. Efisiensi Ruang (Space Efficiency): Efisiensi ruang merujuk pada seberapa sedikit memori atau ruang penyimpanan yang diperlukan oleh algoritma pencarian untuk menjalankan operasinya. Algoritma dengan efisiensi ruang yang tinggi membutuhkan sedikit ruang penyimpanan tambahan.

Dengan membandingkan algoritma pencarian berdasarkan sifat-sifat ini, kita dapat mengevaluasi dan memilih algoritma yang paling sesuai dengan kebutuhan dan batasan yang ada dalam konteks pencarian yang sedang dilakukan.

D. Jenis-Jenis Algoritma Pencarian

Ada banyak jenis algoritma pencarian dan dikelompokkan ke dalam beberapa kategori berdasarkan pendekatan dan strategi yang digunakan. Beberapa kategori umum dari algoritma pencarian meliputi:

1. Pencarian Linier: Algoritma pencarian linier atau sequential search adalah pendekatan sederhana yang melibatkan pemeriksaan setiap elemen dalam urutan secara berurutan hingga menemukan elemen yang dicari atau mencapai akhir urutan.
2. Pencarian Biner: Algoritma pencarian biner atau binary search digunakan pada urutan yang sudah terurut secara terus-menerus. Algoritma ini membagi urutan menjadi dua bagian setiap kali dan memeriksa apakah elemen yang dicari berada di bagian kiri atau kanan. Dengan demikian, mencari dapat dilakukan secara efisien dalam logaritma basis 2 dari jumlah elemen.
3. Pencarian Hash: Algoritma pencarian hash melibatkan penggunaan fungsi hash untuk mengamankan data dalam struktur data seperti tabel hash. Ini memungkinkan pencarian dengan kompleksitas waktu yang konstan, asalkan tidak ada konflik dalam nilai hash.
4. Pencarian Graf: Algoritma pencarian graf, seperti Depth-First Search (DFS) dan Breadth-First Search (BFS), digunakan untuk mencari node atau jalur dalam

struktur data graf. DFS mengeksplorasi setiap cabang dari suatu node sebelum kembali, sementara BFS melintasi node dalam urutan tingkat.

5. Pencarian Heuristik: Algoritma pencarian heuristik, seperti Algoritma A* (A-star), menggunakan pendekatan yang menggabungkan pencarian informasi dan pencarian heuristik untuk mencapai solusi secara efisien. Ini sering digunakan dalam masalah optimisasi dan pencarian jalan terpendek.

Jenis-jenis diatas merupakan kategori umum dari algoritma pencarian, masih ada banyak jenis algoritma pencarian lain yang dikembangkan untuk mengatasi masalah pencarian yang lebih kompleks dan spesifik.

Dalam bab ini akan membahas secara mendetail tentang algoritma pencarian elemen dari sebuah himpunan. Algoritma pencarian yang digunakan adalah jenis algoritma pencarian linier dan algoritma pencarian biner.

Namun untuk memahami penerapan algoritma pencarian diperlukan pemahaman tentang array yang telah diterangkan dalam bab sebelumnya. Berikut penjelasan dan penerapan algoritma pencarian linier dan algoritma pencarian biner beserta penerapannya kedalam bahasa C.

E. Pencarian Linier

Pencarian elemen array secara linear (linear search) adalah metode sederhana untuk mencari nilai tertentu dalam suatu

array. Metode ini bekerja dengan cara melakukan pengecekan pada setiap elemen array secara berurutan. Pencarian akan berhenti saat menemukan nilai yang dibutuhkan. Algoritma sederhana dalam pencarian linier dapat dilihat pada contoh berikut :

- 1) Mulai pencarian dari nilai elemen pertama array.
- 2) Periksa setiap nilai elemen array secara berurutan.
- 3) Jika nilai elemen yang dicari ditemukan, keluarkan tempat indeks penyimpanannya.
- 4) Jika seluruh elemen array telah diperiksa nilai tidak ditemukan, keluarkan tulisan "elemen tidak ditemukan".

Penerapan dalam Bahasa C :

```
#include <stdio.h>

int linearSearch(int arr[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (arr[i] == target) {
            return i; // Elemen ditemukan, kembalikan indeksnya
        }
    }
    return -1; // Elemen tidak ditemukan
}

int main() {
    int arr[] = {5, 2, 7, 1, 9};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 7;

    int result = linearSearch(arr, size, target);
    if (result == -1) {
        printf("Elemen tidak ditemukan\n");
    } else {
        printf("Elemen ditemukan pada indeks %d\n", result);
    }

    return 0;
}
```

F. Pencarian Biner

Pencarian elemen array cara bagi dua (binary search) adalah teknik efisien untuk mencari nilai elemen tertentu dalam array jika dalam keadaan berurutan. Metode ini bekerja dengan membagi array menjadi dua bagian. Pemeriksaan yang dilakukan dalam metode ini adalah memeriksa apakah elemen yang dicari berada di setengah bagian yang relevan. Proses ini berlanjut hingga elemen ditemukan atau setengah bagian yang relevan tidak lagi dapat dibagi. Algoritma sederhana dalam pencarian bagi dua dapat dilihat pada contoh berikut :

- 1) Tentukan elemen pertama (low) dan elemen terakhir (high) dalam array.
- 2) Hitung elemen tengah (mid) dengan rumus $mid = (low + high) / 2$.
- 3) Bandingkan elemen tengah dengan elemen yang dicari:
- 4) Jika elemen tengah sama dengan elemen yang dicari, kembalikan indeksnya.
- 5) Jika elemen tengah lebih kecil dari elemen yang dicari, cari di setengah bagian kanan array dengan mengubah $low = mid + 1$.

- 6) Jika elemen tengah lebih besar dari elemen yang dicari, cari di setengah bagian kiri array dengan mengubah $high = mid - 1$.
- 7) Ulangi langkah 2 dan 3 hingga elemen ditemukan atau low lebih besar dari high.
- 8) Jika elemen tidak ditemukan setelah langkah 4, kembalikan nilai yang menunjukkan bahwa elemen tidak ada dalam array

Penerapan dalam Bahasa C :

```
#include <stdio.h>

int binarySearch(int arr[], int size, int target) {
    int low = 0;
    int high = size - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (arr[mid] == target) {
            return mid; // Elemen ditemukan, kembalikan indeksnya
        } else if (arr[mid] < target) {
            low = mid + 1; // Cari di setengah bagian kanan
        } else {
            high = mid - 1; // Cari di setengah bagian kiri
        }
    }

    return -1; // Elemen tidak ditemukan
}

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 7;

    int result = binarySearch(arr, size, target);

    if (result == -1) {
        printf("Elemen tidak ditemukan\n");
    } else {
        printf("Elemen ditemukan pada indeks %d\n", result);
    }

    return 0;
}
```



BAB 13

PEMROGRAMAN SORTING

1. Tujuan mempelajari Algoritma Sorting

- a. Mampu melakukan perancangan aplikasi menggunakan Struktur Sorting (pengurutan)
- b. Mampu melakukan membuat, mendeklarasikan serta analisis pada struktur algoritma Sorting
- c. Mampu mengimplementasikan algoritma Sorting pada sebuah aplikasi secara tepat dan efisien

2. Pendahuluan

Pengurutan data dalam struktur data sangat penting terutama untuk data yang beripe data numerik ataupun karakter. Pengurutan dapat dilakukan secara ascending (urut naik) dan descending (urut turun). Pengurutan (Sorting) adalah proses pengurutan sekumpulan data yang sebelumnya disusun secara acak sehingga tersusun secara teratur menurut aturan tertentu.

Contoh:

Data Acak : 5 6 8 1 3 25 10

Ascending : 1 3 5 6 8 10 25

Descending : 25 10 8 6 5 3 1

3. Manfaat Metode Sorting

1. Data yang terurut mudah untuk di cari dan diperiksa jika terjadi kesalahan
2. Data yang terurut dapat dihapus jika sewaktu-waktu sudah tidak digunakan lagi
3. Semakin mudah dalam mengurutkan data sehingga semakin mudah menyisipkan data dan penggabungan data
4. Mempersingkat waktu membuat program

4. Jenis Metode Sorting

a. Bubble Sort

Diberi nama "Bubble" karena proses pengurutan secara berangsur-angsur bergerak/berpindah ke posisi yang tepat , seperti gelembung yang keluar dari sebuah gelas bersoda. Bubble sort mengurutkan data dengan cara membandingkan elemen sekarang dengan elemen berikutnya. Jika elemen sekarang lebih besar dari elemen

berikutnya maka elemen tersebut ditukar (untuk pengurutan ascending).

Algoritma ini seolah olah menggeser satu per satu elemen dari kanan ke kiri atau kiri ke kanan, tergantung jenis pengurutannya. Ketika suatu tahap proses telah selesai, maka bubble sort akan mengalami proses selanjutnya, demikian seterusnya. Bubble sort berhenti jika seluruh array telah diperiksa dan tidak ada pertukaran lagi yang bisa dilakukan,serta tercapai pengurutan yang telah diinginkan

Contoh : Dari data 22 10 15 3 8 2

Proses 1 :

22	10	15	3	8	2
10	22	15	3	8	2
10	15	22	3	8	2
10	15	3	22	8	2
10	15	3	8	22	2
10	15	3	8	2	22

Pengecekan dimulai dari data yang paling awal, kemudian dibandingkan dengan data di depannya,jika data didepannya lebih besar maka akan di tukar.

Proses 2:

10	15	3	8	2	22
10	15	3	8	2	22
10	3	15	8	2	22

Dasar-Dasar Pemrograman

10	3	8	15	2	22
10	3	8	2	15	22

Pengecekan dilakukan sampai dengan data yang kedua terakhir karena data terakhir pasti sudah paling besar.

Proses 3 :

10	3	8	2	15	22
3	10	8	2	15	22
3	8	10	2	15	22
3	8	2	10	15	22

Proses 4 :

3	8	2	10	15	22
3	8	2	10	15	22
3	2	8	10	15	22

Proses 5 :

3	2	8	10	15	22
2	3	8	10	15	22

Algoritma Bubble Sort Pada C

```
// Bubble sort in C

#include <stdio.h>

// perform the bubble sort
```

```

void bubbleSort(int array[], int size) {

    // loop to access each array element
    for (int step = 0; step < size - 1; ++step) {

        // loop to compare array elements
        for (int i = 0; i < size - step - 1; ++i) {

            // compare two adjacent elements
            // change > to < to sort in descending order
            if (array[i] > array[i + 1]) {

                // swapping occurs if elements
                // are not in the intended order
                int temp = array[i];
                array[i] = array[i + 1];
                array[i + 1] = temp;
            }
        }
    }

    // print array
    void printArray(int array[], int size) {

```

```

for (int i = 0; i < size; ++i) {
    printf("%d ", array[i]);
}
printf("\n");
}

int main() {
    int data[] = {-2, 45, 0, 11, -9};

    // find the array's length
    int size = sizeof(data) / sizeof(data[0]);

    bubbleSort(data, size);

    printf("Sorted Array in Ascending Order: \n");
    printArray(data, size);
}

```

2. Selection Sort

Cara kerja metode ini didasarkan pada pencarian elemen dengan nilai terkecil. kemudian dilakukan penukaran dengan elemen ke-I. Secara singkat metode ini bisa dijelaskan sebagai berikut. Pada langkah pertama, dicari data yang terkecil dari data pertama sampai terakhir.

Kemudian data tersebut kita tukar dari data pertama. Dengan demikian, data pertama sekarang mempunyai nilai paling kecil dibanding dengan data lain. Pada langkah kedua, data terkecil kita cari mulai dari data kedua sampai data terakhir. Data terkecil yang kita peroleh kita tukar dengan data kedua. Demikian seterusnya sampai seluruh data terurut.

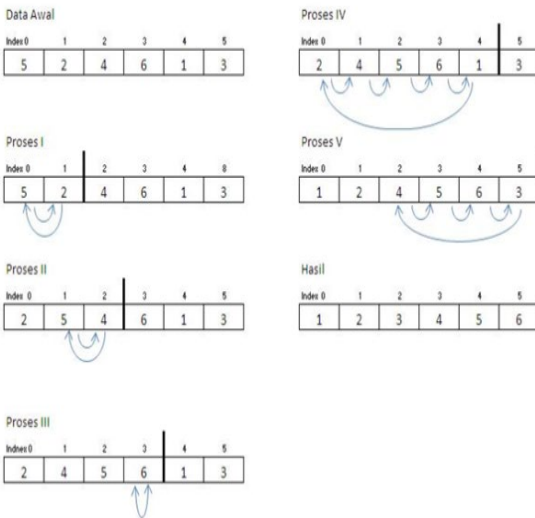
44	55	12	42	94	18	06	67	Data Awal
06	55	12	42	94	18	44	67	Tukarkan data ke 1 dengan data ke 7
06	12	55	42	94	18	44	67	Tukarkan data ke 2 dengan data ke 3
06	12	18	42	94	55	44	67	Tukarkan data ke 3 dengan data ke 6
06	12	18	42	94	55	44	67	Data ke 4 tidak ditukarkan
06	12	18	42	44	55	94	67	Data ke 5 ditukarkan dengan data ke 7
06	12	18	42	44	55	94	67	Data ke 6 tidak ditukarkan
06	12	18	42	44	55	67	94	Data ke 7 ditukarkan dengan data ke 8
06	12	18	42	44	55	67	94	Data setelah terurut

3. Insertion Sort

- a. Insertion Sort merupakan algoritma yang efisien untuk mengurutkan angka yang mempunyai jumlah elemen sedikit. Dimana:- Input : deretan angka sejumlah n buah – Output : permutasi (pengurutan) sejumlah n angka dari input yang sudah terurut secara ascending maupun descending
- b. Metode penyisipan (Insertion sort) bertujuan untuk menjadikan bagian sisi kiri array terurutkan sampai dengan seluruh array berhasil diurutkan.

- c. Metode ini mengurutkan bilangan-bilangan yang telah dibaca; dan berikutnya secara berulang akan menyisipkan bilangan-bilangan dalam array yang belum terbaca ke sisi kiri array yang telah terurut.
- d. Insertion Sort bekerja seperti banyak orang yang sedang mengurutkan kartu di tangan. Dimulai dengan tangan kiri yang kosong dan kartunya tertumpuk di meja. Selanjutnya kita ambil satu persatu kartu di meja dan diletakkan di tangan kiri dengan posisi yang benar (terurut). Untuk menemukan posisi yang banar, maka kita harus membandingkan satu persatu kartu yang ada (di tangan kiri) secara berurutan.

Contoh insert sort :



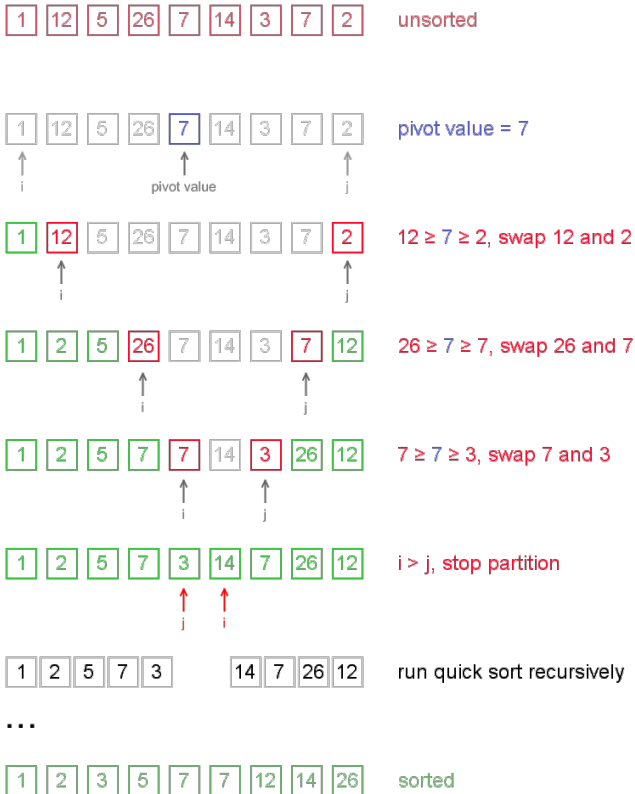
4. Shell Sort

Metode ini dikembangkan oleh Donald L. Shell pada tahun 1959. Dalam metode ini jarak antara dua elemen yang dibandingkan dan ditukarkan tertentu. Secara singkat metode ini dijelaskan sebagai berikut. Pada langkah pertama, kita ambil elemen pertama dan kita bandingkan dan kita bandingkan dengan elemen pada jarak tertentu dari elemen pertama tersebut. Kemudian elemen kedua kita bandingkan dengan elemen lain dengan jarak yang sama seperti jarak yang sama seperti diatas. Demikian seterusnya sampai seluruh elemen dibandingkan. Pada langkah kedua proses diulang dengan langkah yang lebih kecil, pada langkah ketiga jarak tersebut diperkecil lagi seluruh proses dihentikan jika jarak sudah sama dengan satu.

5. Quick Sort

Metode ini dikembangkan oleh CAR Hoare. Secara garis besar metode ini dijelaskan sebagai berikut. Misalnya kita ingin mengurutkan data A yang membunyai N elemen. Kita pilih sembarang elemen dari data tersebut, biasanya elemen pertama, misalnya X. Kemudian semua elemen tersebut disusun dengan menempatkan X pada posisi J sedemikian rupa shingga elemen ke 1 sampai ke J-1

mempunyai nilai lebih besar dari X. Sampai berikutnya diulang untuk setiap sub data.



6. Merge Sort

Merge sort merupakan algoritma pengurutan dalam ilmu komputer yang dirancang untuk memenuhi kebutuhan pengurutan atas suatu rangkaian data yang tidak memungkinkan untuk ditampung dalam memori komputer karena jumlahnya yang terlalu besar. Algoritma ini ditemukan oleh John von Neumann pada tahun 1945.

Algoritma pengurutan data merge sort dilakukan dengan menggunakan cara divide and conquer yaitu dengan memecah kemudian menyelesaikan setiap bagian kemudian menggabungkannya kembali. Pertama data dipecah menjadi 2 bagian dimana bagian pertama merupakan setengah (jika data genap) atau setengah minus satu (jika data ganjil) dari seluruh data, kemudian dilakukan pemecahan kembali untuk masing-masing blok sampai hanya terdiri dari satu data tiap blok.

Setelah itu digabungkan kembali dengan membandingkan pada blok yang sama apakah data pertama lebih besar daripada data ke-tengah+1, jika ya maka data ke-tengah+1 dipindah sebagai data pertama, kemudian data ke-pertama sampai ke-tengah digeser menjadi data ke-dua sampai ke-tengah+1, demikian seterusnya sampai menjadi satu blok utuh seperti awalnya. Sehingga metode merge sort merupakan metode yang membutuhkan fungsi rekursi untuk penyelesaiannya.

```

Merge Sort
10 2 5 6 23 5 6 7 2 1
n = 10

SPLIT (berdasarkan hasil n div 2)
(10 2 5 6 23) (5 6 7 2 1)
n = 5                n = 5
(10 2) (5 6 23) (5 6) (7 2 1)
n = 2  n = 3      n = 2  n = 3
{ 10 } { 2 } { 5 } { 6 23 } { 5 } { 6 } { 7 } { 2 1 }
n=1 n=1 n=1 n=2    n=1 n=1 n=1 n=2
{ 10 } { 2 } { 5 } { 6 } { 23 } { 5 } { 6 } { 7 } { 2 } { 1 }

MERGE (Berdasarkan urutan split)
{ 10 } { 2 } { 5 } { 6 23 } { 5 } { 6 } { 7 } { 1 2 }
{ 2 10 } { 5 6 23 } { 5 6 } { 1 2 7 }
{ 2 5 6 10 23 } { 1 2 5 6 7 }
{ 1 2 2 5 5 6 6 7 10 23 }
    
```



BAB 14

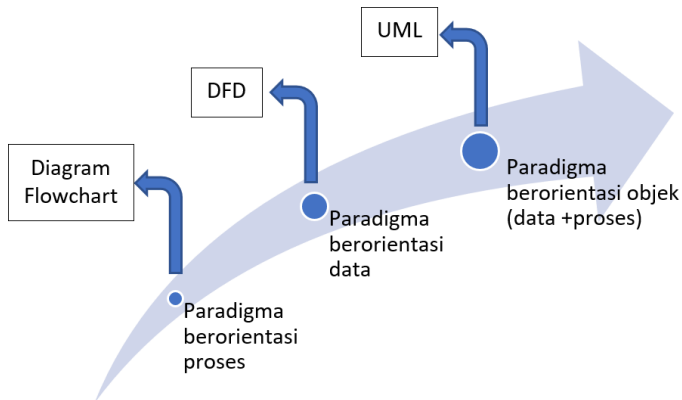
PEMROGRAMAN BERBASIS OBJEK

Pada buku bagian ini akan membahas tentang pemrograman berbasis objek dimulai dari bagaimana proses pendekatan untuk mempermudah pembuatan perangkat lunak melalui cara kerja sistematis dengan teknik dan konvensi notasi yang terdefinisi. Kemudian dilanjutkan dari manakah proses UML itu dimulai dan sampai manakah akhir dari pemodelan UML dalam pengembangan perangkat lunak?. Diagram mana sajakah yang seharusnya digunakan dan seperti apa urutannya?. Serta bagaimana penerapan prinsip-prinsip dasar pemrograman berbasis objek pada salah satu bahasa pemrograman yang mudah? Berikut ini kita akan jabarkan jawaban dari masing-masing pertanyaan tersebut dalam sebuah *bookchapter*.

A. Pendekatan Berorientasi Objek

Pendekatan berorientasi objek adalah sebuah proses untuk menghasilkan perangkat lunak yang terorganisir

melalui suatu pendekatan dengan melihat permasalahan melalui sejumlah teknik dan konvensi notasi yang terdefinisi (Gomaa, 2011). Pendekatan ini memerlukan suatu cara kerja yang sistematis sehingga memudahkan pembuatan perangkat lunak yang sesuai tujuan. Semakin berkembangnya teknologi munculah bermacam-macam paradigma terkait bagaimana bisa menyelesaikan permasalahan secara efektif dan efisien untuk mengembangkan perangkat lunak.



Gambar 1. Evolusi paradigma berorientasi objek

Pada awalnya untuk mengetahui alur kerja sebuah sistem, seseorang menggambarkannya dengan *flowchart*. Kemudian mulai ramai penggunaan *database*, munculah DFD (*Data Flow Diagram*) untuk mengetahui aliran datanya. Baru pada era 90an atau generasi ketiga dicarilah cara menggambarkan alur kerja dan aliran data ke dalam sebuah penggambaran. Di era inilah UML (*Unified Modeling*

Language) lahir. *Activity diagram* menggantikan *flowchart diagram*. Ada juga *sequence diagram* yang menggantikan DFD. Sementara untuk *leveling level 0* pada DFD sudah terwakilkan dengan *use case diagram* di UML.

B. Mengapa menggunakan UML?

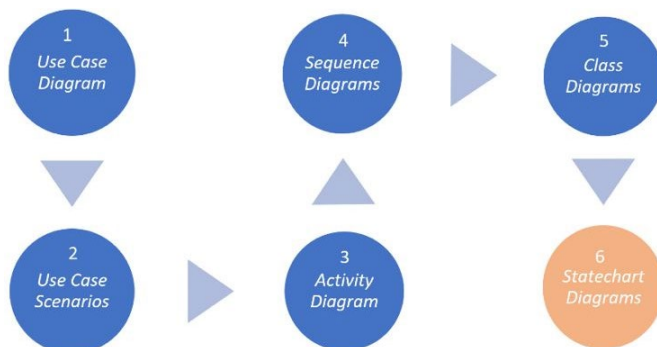
Jika bahasa pemrograman yang digunakan berorientasi objek maka adalah pilihan tepat untuk memilih UML sebagai *tools* untuk memodelkan secara visual perangkat lunak tanpa terbatas pada bahasa pemrograman tertentu. Untuk mengetahui bahasa pemrograman yang digunakan mendukung konsep objek, maka bisa dikenali dari prinsip-prinsip yang digunakan pada penulisan bahasa pemrograman tersebut.

1. UML (*Unified Modeling Language*)

UML adalah standar pemodelan, bukan metodologi untuk pengembangan *software* (Podeswa, 2010), jadi dari manakah proses UML itu dimulai dan sampai manakah akhir dari pemodelan UML dalam pengembangan *software*?. Diagram mana sajakah yang seharusnya kita gunakan dan seperti apa urutannya?.

Berdasarkan (sparxsystems, 2023) yang merupakan salah satu *tools* UML terkenal menjabarkan bahwa yang pertama kali harus dilakukan adalah

gambarkan batasan-batasan dan fungsi-fungsi pemodelan dengan menggunakan *use case diagram* yang berisikan *use case* itu sendiri dan perilaku dari aktor yang ada. Selanjutnya modelkan proses bisnis organisasi menggunakan *activity diagram*. Realisasi yang dibuat dari *use case* kemudian dilustrasikan dalam bentuk *sequence diagrams*. Sementara struktur statis dari sistem digambarkan dengan menggunakan *class diagrams*. Kemudian bagaimana cara seorang pengembang *deploy* atau menjadikan suatu aplikasi dapat berfungsi pada perangkat target, dibuatlah *deployment diagrams*. Ada pula pendapat lainnya terkait urutan dari diagram-diagram yang ada di UML. Seperti (Kendall, 2020) juga ikut menjabarkan urutannya seperti pada Gambar 2 berikut:



Gambar 2. Urutan diagram dalam UML

Analisis sistem diawali dengan *use case diagrams*. Setiap *use case* yang ada di *use case diagram* menghasilkan satu *activity diagram*. Jadi satu proses bisnis satu *activity diagram*. Setiap *sequence diagram* juga sama, berasal dari satu *use case*, hanya saja *sequence diagram* bisa jadi lebih dari satu untuk setiap *use case* yang sudah ada. Di *sequence diagrams* sudah tampak *method* dan parameter-parameter yang bisa dibaca oleh seorang *programmer*. Jadi *use case diagrams* dan *activity diagram* digunakan untuk berkomunikasi dengan klien, sementara *sequence diagrams* adalah alat untuk berkomunikasi antar tim, karena dengan *sequence diagrams* inilah nanti akan menghasilkan *class diagrams*.

Dari urutan yang diberikan oleh (Kendall, 2020) sayangnya ada 1 tahap yang kurang cocok jika diterapkan pada sistem perangkat lunak sederhana yang di dalamnya tidak terdapat otomatisasi seperti memerlukan pendefinisian *state* apa yang sedang berjalan. Perlu membuat *statechart diagrams* jika sistem yang dibuat mampu berjalan secara mandiri seperti robot, sementara peran manusia hanya menyalakan dan mematikan robot.

2. Use Case

Use case merupakan ringkasan dari sistem yang isinya adalah apa yang dilakukan *user* di dalam sistem, bukan apa yang sistem lakukan. Contohnya

mengotorisasi login itu dilakukan oleh sistem bukan manusia, maka ini tidak masuk di *use case diagrams*. Apa yang sistem lakukan tergambarkan melalui *activity diagram*. Aktor yang ada di *use case* boleh manusia boleh juga sistem lain yang berinteraksi.

Ada sebuah contoh kasus *use case* dari sistem ATM yang diberi nama “melakukan login”. Di lain sisi bisa juga *use case* ini dipecah menjadi “memasukan kartu” dan “memasukan pin”. Jika kasusnya dipecah maka *use case* “memasukan pin” harus menggunakan relasi *include* karena memasukan pin itu wajib dalam proses melakukan *login*. Sedangkan suatu relasi yang bersifat opsional maka digunakanlah relasi *extend*. Contoh *extend* adalah misal melakukan transaksi. Melakukan transaksi bisa jadi melakukan transfer, mengambil uang, dan lain sebagainya. Hal ini tidak wajib dilakukan oleh seorang aktor dalam sistem ATM. Arah panah dari relasi *include* adalah dari *use case* utamanya menuju ke *including use case*-nya (bapak ke anak). Sementara untuk *extend* arah panahnya dari *extended*-nya ke *home/use case* utama (anak ke bapak). Sementara hubungan generalisasi arah panahnya dari khusus ke umum (Sukamto and Shalahuddin, 2016).

3. *Activity Diagram*

Activity diagram adalah diagram yang digunakan dalam pemodelan sistem atau proses bisnis untuk

menggambarkan urutan tindakan yang terjadi (Hariyanto, 2004). *Activity diagram* dapat memodelkan sistem secara lebih luas dibandingkan *flowchart* yang hanya spesifik untuk pemrograman. *Activity diagram* memiliki tingkat abstraksi lebih tinggi sementara *flowchart* cenderung lebih menyajikan detail langkah-langkah dalam proses atau algoritma. *Activity diagram* secara eksplisit juga mendukung aktivitas paralel dan sinkronisasi aktivitas-aktivitas yang ada.

4. *Sequence Diagram*

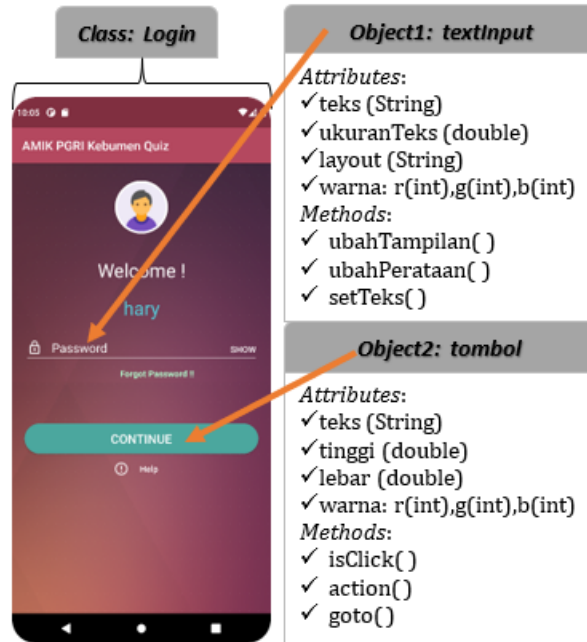
Diagram ini memvisualisasikan bagaimana antar objek berkomunikasi dan berinteraksi satu sama lain sepanjang waktu (Hariyanto, 2004). Dalam diagram ini sudah terlihat kapan pesan dikirim dan bagaimana pesan itu dikirim, serta respon apa yang diberikan oleh objek penerima pesan. Pada diagram ini, waktu berjalan dari atas ke bawah, sehingga urutan pesan dapat terlihat dengan jelas.

C. Pemrograman Berorientasi Objek dengan JAVA

JAVA adalah salah satu bahasa pemrograman yang mendukung pemrograman berbasis objek dan digunakan secara luas baik untuk pemrograman web, *mobile*, maupun desktop. Bahasa pemrograman JAVA juga tidak terbatas

pada sistem operasi tertentu. JAVA memiliki karakteristik yang lebih sederhana bagi para pemula yang baru pertama belajar pemrograman.

Kelas dan Objek di JAVA



Gambar 3. Orientasi Objek di JAVA

Kelas adalah sekelompok objek yang memiliki karakteristik dan perilaku serupa. Sedangkan objek adalah kelompok dari banyak atribut (variabel) dan metode (fungsi) yang terkait dalam satu kesatuan. Pada gambar 3 dicontohkan terdapat sebuah tampilan login. Kelas dari tampilan tersebut diberi nama kelas Login. Kemudian di kelas tersebut ada objek tombol dan textInput. Masing-

masing objek tersebut memiliki atribut dan *method* masing-masing. Ketika halaman login pertama kali diakses, *method* `setTeks()` akan dijalankan untuk menampilkan tulisan "Password" di `textInput`. Kemudian jika tombol "Continue" diklik *method* `isClick()` akan dijalankan sehingga nantinya ada pengecekan otorisasi login yang kemudian bisa jadi nanti dilanjutkan ke *method* `goto()`.

D. Prinsip-Prinsip Orientasi Objek

1. Pengapsulan (Encapsulation)

Encapsulation adalah suatu cara untuk menyembunyikan implementasi detail dari suatu kelas. Kita dapat menyembunyikan informasi dari suatu kelas sehingga anggota-anggota kelas tersebut tidak dapat diakses dari luar. Terdapat 4 macam tipe akses di JAVA, yaitu : *public*, *private*, *protected* dan *default*. Tiga tipe akses pertama tertulis secara eksplisit pada kode untuk mengindikasikan tipe akses, sedangkan *default* tidak diperlukan penulisan keyword dalam kode program.

2. Pewarisan (Inheritance)

Seperti halnya manusia pada umumnya terdapat orang tua dan anak. Pada PBO juga dikenal *parent class* atau *base class* dan ada pula *subclass* atau *child class*. *Child class* mewarisi semua data yang ada di *parent class* atau

dapat disimpulkan juga bahwa *child class* adalah perluasan dari *parent class*. *Child class* ditandai dengan adanya *keyword extends* setelah nama *child class* itu sendiri.

3. Banyak Bentuk (Polymorphism)

Polimorfisme adalah saat sebuah kelas memiliki banyak objek, kemudian salah satu objek itu memiliki kemampuan mengambil beberapa bentuk yang berbeda agar tidak terjadi duplikasi objek. Polimorfisme statis diimplementasikan dengan *method overloading*, sedangkan polimorfisme dinamis dengan *method overriding*.

4. Abstraksi

Pada sebuah struktur kode program kompleksitasnya dapat diatur dengan menerapkan paradigma abstraksi. Pengembang program dapat menangani atau mengabaikan detail yang tidak relevan dan menampilkan data relevan sesuai kebutuhannya saja. Untuk membuat sebuah tombol maka menggunakan perintah yang sudah terdefinisi sebagai *abstract class*. Class ini bersifat abstrak dan tidak dapat diinstansiasi, hanya berisikan variabel umum dan deklarasi *method* tanpa detail penggunaannya.

E. Implementasi Pemrograman Berbasis Objek

```
public abstract class Shape {
    String color;
    void setColor(String color){
        this.color = color;
    }
    String getColor(){
        return this.color;
    }
    abstract float getArea();
}
```

Abstraksi

```
public class Triangle extends Shape {
    public float alas;
    protected float tinggi;
    public Triangle(){
        System.out.println("Constructor:Triangle");
    }
    public Triangle(int alas, int tinggi) {
        this.alas = alas;
        this.tinggi = tinggi;
    }
    @Override
    float getArea() {
        return 0.5f * alas * tinggi;
    }
}
```

Inheritance
Encapsulation

```
public class KonsepDasarPBO {
    public static void main(String[] args) {
        // membuat objek dari class Triangle
        Shape segitiga = new Triangle(4, 5);
        // membuat objek dari class Circle
        Shape lingkaran = new Circle(10);
        //Banyak bentuk
        Triangle hexagon1 = new Hexagon();
        //Pemanggilan bentuk lain
        Hexagon hexagon2 = new Hexagon();
        System.out.println("Hexagon?: "+ hexagon1.getArea());
        System.out.println("Hexagon2: "+ hexagon2.getArea());
        System.out.println("Luas Segitiga: " +segitiga.getArea());
        System.out.println("Luas Lingkaran: " +lingkaran.getArea());
    }
}
```

Polymorphism

Tentang Penulis



Eri Mardiani, S1 Sistem Informasi di UPN Veteran Jakarta, S2 Ilmu Komputer di STMIK Nusa Mandiri, cukup aktif menulis dan beberapa karya tulisnya telah termuat di salah satu di media massa di Jakarta dan saat ini sebagai dosen di beberapa Perguruan Tinggi, Dosen di Universitas Nasional, saat ini menjabat sebagai Sekretaris ICT Research Center FTKI di Universitas Nasional serta entrepreneur **shop and travel**, silahkan subscribe youtube chanel saya <https://youtu.be/rGLfvsUbgTs>

Buku Yang Telah Kami Buat

No	Judul	Penerbit
1	Aplikasi Penjualan dengan Program Java Netbeans, Xampp, dan iReport	Elex Media Komputindo
2	Aplikasi Penjualan Menggunakan VB.Net	Elex Media Komputindo
3	Aplikasi Inventory dengan menggunakan JAVA NETBEANS, XAMPP dan iReport	Elex Media Komputindo
4	Kumpulan Latihan Visual Basic	Elex Media Komputindo
5	Aplikasi Penjualan dengan Visual Basic, Xampp dan Data Report	Elex Media Komputindo

6	Aplikasi Penggajian dengan Visual Basic, MySQL dan Data Report	Elex Media Komputindo
7	Kumpulan Latihan SQL	Elex Media Komputindo
8	Membuat Aplikasi Penjualan Menggunakan Java, Mysql dan iReport	Elex Media Komputindo
9	Panduan Khusus VB6 Bagi Pemula	Elex Media Komputindo
10	Aplikasi Komputer	Mitra Wacana Media
11	Membuat Aplikasi Inventory Menggunakan Java, Mysql dan iReport	Elex Media Komputindo
12	Kumpulan Latihan PHP	Elex Media Komputindo
13	Kumpulan Latihan Vb.Net	Elex Media Komputindo
14	Kewirausahaan	Pena Muda



Yuyun Khairunisa, M.Kom merupakan Dosen pengajar pada program studi Teknologi Permainan Politeknik Negeri Media Kreatif. Aktifitas sehari-hari selain mengajar dan menulis, juga aktif melakukan penelitian dan mengelola jurnal ilmiah Journal Of Multimedia and IT (Jommit).



Nur Rahmansyah., S.Kom., M.Kom., Menempuh Pendidikan S1 Sistem Informasi di STMIK Nusa Mandiri Jakarta, kemudian melanjutkan S2 Ilmu Komputer di Universitas Budi Luhur. Aktif menulis sejak tahun 2011 beberapa karya telah dihasilkan

dalam bentuk buku & jurnal yang telah terbit diantaranya penerbit Gramedia / Elex Media Komputindo dan di beberapa jurnal nasional serta internasional. Selain menulis saat ini bekerja sebagai Dosen di Politeknik Negeri Media Kreatif Jakarta mengajar Animasi, Design, Video Editing dan beberapa mata kuliah yang berkaitan dengan Ilmu Komputer. Penulis Juga memiliki channel youtube jangan lupa subscribe ya: <https://youtube.com/@udanurman>



Agus Ambarwari lahir di Nganjuk–Jawa Timur, pada 8 Agustus 1992. Penulis mendapatkan gelar sarjana Pendidikan Teknik Informatika dari Universitas Negeri Malang pada tahun 2014. Gelar Magister Ilmu Komputer diperoleh dari Institut

Pertanian Bogor pada tahun 2017. Saat ini, penulis berprofesi sebagai dosen di Politeknik Negeri Lampung. Bidang penelitian yang ditekuni adalah *Image Processing*, *Internet of Things*, dan *Machine Learning*.



Lahir di kota Duri Provinsi Riau pada 3 Juni 1970. Meraih gelar S1 dari Departemen Teknik Komputer Universitas Gunadarma pada 1994. Pada tahun 2012, meraih gelar S2 dari Departemen Ilmu Komputer Institut Pertanian Bogor. Saat ini menjadi dosen di Politeknik Negeri Lampung pada program studi Teknologi Rekayasa Internet. Penulis dapat dihubungi melalui email : zuriati_mi@polinela.ac.id.



Putri Ariatna Alia lahir pada hari selasa tanggal 5 juli 1994 di Surabaya provinsi Jawa Timur. Penulis telah menyelesaikan pendidikan S2 Teknik Elektro dengan Konsentrasi Teknik Telematika di Institut Teknologi Sepuluh Nopember dan D4 Teknik Elektro konsentrasi Teknik Telekomunikasi di Politeknik Elektronika Negeri Surabaya – Institut Teknologi Sepuluh Nopember.



Purwadi Budi Santoso, Lahir di kota Magelang pada tanggal 16 Desember 1968. Menyelesaikan studi S1 di Jurusan Matematika program Studi Ilmu Komputer UGM Yogyakarta tahun 1992. Pada tahun 2000 menyelesaikan studi Pasca Sarjana program Magister Informatika bidang Rekayasa Perangkat Lunak di ITB Bandung. Menjadi dosen jurusan Teknik Informatika di Sekolah Tinggi Teknologi Mandala (STT Mandala) Bandung sejak tahun 1993 sampai sekarang.

Dosen Program Studi Informatika

Fakultas Teknologi Komunikasi dan Informatika, Universitas Nasioanal



Penulis lahir di Jakarta tanggal 16 Juni 1984. Penulis menyelesaikan pendidikan S1 pada jurusan Matematika, Universitas Nasional dan S2 pada jurusan Magister Teknologi Informasi, Universitas Indonesia. Penulis melanjutkan studi S3 pada jurusan Ilmu Komputer, Institut Pertanian Bogor.

Penulis mulai mempelajari pemrograman sejak kuliah S1 mulai dari pemrograman Basic, Pascal, C/C++, C#, VB.Net, Java, Python, dan lain-lain. Beberapa penelitian terkait dengan implementasi pemrograman sudah dipublikasikan dalam bentuk jurnal.



Erik Yohan Kartiko, lahir di kota Malang pada tanggal 29 Desember 1995. Menyelesaikan studi Sarjana Pendidikan Teknik Informatika di Universitas Negeri Malang pada tahun 2018. Pada tahun 2022 menyelesaikan studi Magister Ilmu Komputer di Universitas Brawijaya. Bekerja dalam dunia Pendidikan mulai tahun 2018.



Johan Suryo Prayogo, S.Kom., M.T., seorang Dosen prodi Sistem Informasi di Universitas Anwar Medika. Mengambil studi Strata 1 Teknik Informatika di Universitas Surabaya (UBAYA) serta melanjutkan program Magister Teknik Informatika di Universitas Atma Jaya Yogyakarta (UAJY). Penulis yang memiliki kegemaran membaca dan mempelajari sesuatu yang baru khususnya bidang *UI/UX* dan *web development* tersebut saat ini memiliki sepasang putra dan putri dari seorang istri bernama Sofie Ayuningtyas. Besar harapan penulis dengan terbitnya buku ini menjadi sarana bagi mahasiswa dan masyarakat umumnya untuk mempelajari pemrograman menggunakan bahasa C/C++ dengan lebih menyenangkan.

Yulifda Elin Yuspita, M.Kom.

yulifdaelinyuspita@uinbukittinggi.ac.id

Dosen Ilmu Komputer



Penulis lahir di Gumarang, 22 Desember 1991. Penulis menyelesaikan pendidikan S1 pada Jurusan Sistem Informasi pada Universitas Putra Indonesia “YPTK” Padang. kemudian menyelesaikan studi S2 pada Jurusan Ilmu Komputer di Universitas Putra Indonesia “YPTK Padang”. Penulis adalah Dosen Tetap pada Program Studi Pendidikan Teknik Informatika dan Komputer Fakultas Tarbiyah dan Ilmu Keguruan UIN Sjech M. Djamil Djambek Bukittinggi. Penulis aktif melakukan penelitian yang berkaitan tentang Ilmu Komputer diantaranya Pengaruh Desain Media Pembelajaran dengan Program Adobe Flash CS 6 untuk Belajar Berhitung (2021), Selection Of Internet Provider To Improve Quality Of Service And Learning Using Decision Support System (2022), Decision-Making System for KIP IAIN Bukittinggi Scholarship Recipients Using the SAW and TOPSIS Methods (2022), Sistem Informasi Pusat Pengaduan Pelayanan Masyarakat Menggunakan Work System Framework (2022), Sistem Informasi Consultation Online Menggunakan Bahasa Pemograman PHP dan Database MySQL (2023).



Agung Teguh Setyadi, S.Kom., M.Kom. lahir pada tanggal 26 September 1995 di Malang. Penulis telah menyelesaikan studi sarjana pada jurusan S1 Teknik Informatika ITS (Institut Teknologi Sepuluh Nopember) Surabaya (2013-2018) dan magister pada jurusan S2 Teknik Informatika ITS (Institut Teknologi Sepuluh Nopember) Surabaya (2019-2021) dengan mengambil konsentrasi bidang ilmu data science. Saat ini penulis berkerja sebagai dosen di jurusan S1 Rekayasa Perangkat Lunak Universitas Anwar Medika. Penulis dapat dihubungi melalui email: agung.teguh.setyadi@gmail.com.



Nurhafifah Matondang merupakan dosen tetap Program Studi D3 Sistem Informasi Fakultas Ilmu Komputer UPN Veteran Jakarta. Penulis mengampu mata kuliah Analisis Perancangan Sistem, Big Data Analytics, Sistem Basis Data, Struktur Data, Matematika Diskrit, Machine Learning dan Technopreneurship. Telah menempuh pendidikan S-1 Teknik Informatika di UPN Veteran Jakarta dan S-2 Manajemen di UPN Veteran Jakarta, S-2 Teknik Informatika di Universitas Bina Nusantara. Penulis juga aktif melakukan penelitian dibidang machine learning dan sistem informasi dan pengabdian pemasaran menggunakan media digital.



Imanaji Hari Sayekti, S.Pd., M.Pd. sejak 2013 mulai berkarir di dunia pendidikan dan industri *software development*. Ia menyelesaikan program sarjananya di Universitas Negeri Yogyakarta pada tahun 2013. Kemudian pada tahun 2014 melanjutkan pendidikannya di kampus yang sama yaitu Universitas Negeri Yogyakarta dan selesai pada tahun 2016. Selama menjalani pendidikan magisternya sampai dengan buku ini ditulis ia terjun langsung di dunia pendidikan maupun industri *software house*. Dalam kesehariannya ia bekerja sebagai staf pengajar di suatu kampus di Kebumen sekaligus sebagai team *programmer* pada instansi pemerintah di Kebumen. Ia memiliki kemampuan yang ideal dalam hal pendidikan maupun industri karena memiliki latar belakang dan pengalaman sebagai pengajar sekaligus sebagai *programmer* selama kurang lebih 10 tahun terakhir. Bagi pembaca yang ingin bertukar pikiran dengan penulis bisa menghubunginya melalui email di imanajihari@gmail.com.

DAFTAR PUSTAKA

- Yanofsky, N. S. (2011). Towards a definition of an algorithm. *Journal of Logic and Computation*, 21(2), 253-286.
- Jussupow, E., Benbasat, I., & Heinzl, A. (2020). Why are we averse towards algorithms? A comprehensive literature review on algorithm aversion.
- Putri, F. M. (2021). Konsep Dasar Dalam Mempelajari Mata Kuliah Algoritma Pemrograman.
- Mardiani, E, Rahmansyah,N, Ningsih, Sari (2022). PKM Meningkatkan Penjualan UMKM Dengan E-Commerce Disaat Pandemi Covid 19. *Jurnal Mindabaharu: Jurnal Pengabdian Masyarakat Volume 6, No 2 Desember, 2022.* 234-243.
- Mardiani, E., Rahmansyah, N., Wahyudi, N. M., Wijaya, Y. F., & Al Rizky, F. (2021). *Kumpulan Latihan PHP*. Elex Media Komputindo
- Mardiani, E., Nur Hayati, Muhamad Iqbal Wasta Purnama, Sari Ningsih, Ucuk Darusalam, Tri Esthi Handayani, Nur Rahmansyah. (20213). *Kumpulan Latihan VB.Net*. Elex Media Komputindo
- Mardiani, E, Rahmansyah,N, Al Rizky,F (2020). Perilaku Konsumen Terhadap E-commerce disaat pandemi covid 19 di Shop and Travel. *Jurnal Informatik: Jurnal Ilmu Komputer 16(3), 212-217.*

- Mardiani, E, Rahmansyah, N., Kurniawan, H., Sensuse, D.I., dan Jayanta. 2016. *Kumpulan Latihan SQL*, Elex Media Komputindo. Jakarta
- Mardiani, E, Nur Rahmansyah, Hendra Kurniawan, Anita Muliawati, Dwi Sidik Permana. 2017. *Membuat Aplikasi Penjualan Menggunakan Java Netbeans, Mysql, dan iReport*, Jakarta : Elex Media Komputindo
- Mardiani, E,, Nur Rahmansyah, Satriawan Desmana,Ahmad Rifqi (2023). Analysis of Buyer's Trust in E-Commerce Shop And Travel Web. *Jurnal SITEKIN: Jurnal Sains, Teknologi dan Industri* , Vol. 20, No. 2, June 2023, pp.850 – 857
- Mardiani, E,, Nurjayati, Satriawan Desmana,Ahmad Rifqi (2023). Analysis of Buyer's Trust in E-Commerce Shop And Travel Web. *Jurnal SITEKIN: Jurnal Sains, Teknologi dan Industri* , Vol. 20, No. 2, June 2023, pp.850 – 857
- Rahmansyah,Nur., Mulyani, Deta., Mardiani, Eri., Rahman, Adityo. Perancangan Sistem Transaksi Berbasis Web Pada UKM Pangkas Rambut Tasik. *Jurnal JUNSIBI: Jurnal Sistem informasi Bisnis* April 2022, 22
- Anonymous (2020) 'Unit 2-Data Types, Operators and Expressions', in *Block-1 Introduction to The C Programming Language*. New Delhi: Indira Gandhi National Open University, pp. 23–36. Available at: <http://egyankosh.ac.in//handle/123456789/64124> (Accessed: 29 May 2023).

- Kernighan, B.W. and Ritchie, D.M. (2002) *The C Programming Language*. Second Edition. Englewood Cliffs, New Jersey: Prentice Hall. Available at: <https://kremlin.cc/k&r.pdf> (Accessed: 29 May 2023).
- tutorialspoint.com (2013) *C Programming Tutorial*. Tutorialspoint. Available at: https://www.unf.edu/~wkloster/2220/ppts/cprogramming_tutorial.pdf (Accessed: 29 May 2023).
- Didit N. Utama & Riya Widayanti (2005). *Algoritma & Pemrograman dengan Borland C++*. Yogyakarta: Penerbit Graha Ilmu.
- Kadir, A., dan Heriyanto. (2010). *Algoritma Pemrograman Menggunakan C++*. Yogyakarta: Penerbit Graha Ilmu
- Auli Damayanti dkk. (2020). *Modul Praktikum Algoritma dan Pemrograman*. Surabaya: Penerbit MIPA Universitas Airlangga
- Raharjo, B. (2010). *Pemrograman C++*. Penerbit Informatika.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.
- Sedgewick, R., & Wayne, K. (2011). *Algorithms*. Addison-Wesley Professional.
- Kernighan, B. W., & Ritchie, D. M. (1988). *The C Programming Language*. Prentice Hall.
- Deitel, P. J., & Deitel, H. M. (2016). *C How to Program*. Pearson. <https://www.programiz.com/dsa/sorting-algorithm>
<http://web.if.unila.ac.id/umarroso/2015/11/15/struktur-data-metode-sorting/>

- Gomaa, H., 2011. Software modeling and design : UML, use cases, patterns, and software architectures. Cambridge University Press, Cambridge.
- Hariyanto, B., 2004. Rekayasa Sistem Berorientasi Objek. Informatika, Bandung.
- Kendall, K.E., Kendall, J.E., 2020. Systems Analysis and Design, 10th Edition, 10th ed. Pearson Education Limited, United Kingdom.
- Podeswa, H., 2010. UML for the IT Business Analyst, Second Edition: A Practical Guide to Requirements Gathering Using the Unified Modeling Language, 2nd ed. Course Technology PTR, Boston.
- sparxsystems, 2023. Sparx Systems Enterprise Architect [WWW Document]. URL <https://sparxsystems.com/> (accessed 5.24.23).
- Sukamto, R.A., Shalahuddin, M., 2016. Rekayasa Perangkat Lunak: Terstruktur dan Berorientasi Objek, 4th ed. Informatika, Bandung.
- 9 786230 945908**



Dasar-Dasar Pemrograman

Buku ini terdiri dari beberapa bab, Bab pertama Pengenalan Dasar – Dasar Algoritma dan Pemograman, bab kedua Konsep Dasar Pemograman, bab ketiga Sejarah Flowchart, bab keempat Tipe Data, Operator dan Ekspresi, bab kelima Operasi Seleksi, bab keenam Operasi Perulangan, bab ketujuh Pengenalan Bahasa C/C++, bab kedelapan Pemograman Modular, bab kesembilan ARRAY, bab kesepuluh Struktur, bab kesebelas Pemograman Array Of Struct, bab kedua belas Pemograman Pencarian, bab ketiga belas Pemograman Sorting, bab terakhir Pemograman Berbasis Objek.

Buku ini sangat membantu mahasiswa dan pemula untuk memahami pembuatan program dalam bahasa pemrograman C/C++. Selain itu, Buku ini juga membahas tentang konsep dasar pemrograman, sehingga memudahkan mahasiswa dan pemula membuat program sederhana. Buku ini dilengkapi dengan penjelasan tentang Algoritma dan Flowchart sebagai materi dasar pembuatan program, sehingga program yang dibuat lebih terstruktur dan terarah sesuai dengan kebutuhan. Setelah membaca buku ini, kami mengharapkan pembaca lebih memahami flowchart dan algoritma untuk pembuatan program sederhana, dan dapat mempraktekkan soal-soal latihan yang ada.

ISBN 978-623-09-4590-8



PT Penerbit Penamuda Media
Godean, Yogyakarta
085700592256
@penamuda_media
penamuda.com