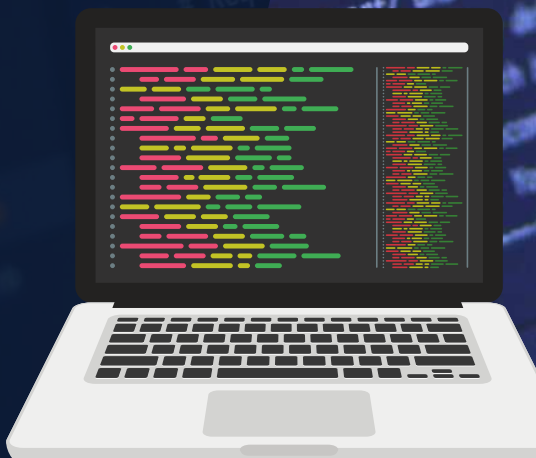
DASAR-DASAR PEMPROGRAMAN

- Irmawati
- Dani Anggoro
- Zumhur Alamin
- Rudiansyah
- Sugiarto Cokrowibowo
- Muhammad Akbar Yasin
- Paray Theo Lonando
- Wirawan Istiono
- Wulan Sri Lestari
- Hendra Nusa Putra
- Ade Wisandra



DASAR-DASAR PEMPROGRAMAN

Irmawati
Dani Anggoro
Zumhur Alamin
Rudiansyah
Sugiarto Cokrowibowo
Muhammad Akbar Yasin
Paray Theo Lonando
Wirawan Istiono
Wulan Sri Lestari
Hendra Nusa Putra
Ade Wisandra

GET PRESS INDONESIA

DASAR-DASAR PEMPROGRAMAN

Penulis :
Irmawati
Dani Anggoro
Zumhur Alamin
Rudiansyah
Sugiarto Cokrowibowo
Muhammad Akbar Yasin
Paray Theo Lonando
Wirawan Istiono
Wulan Sri Lestari
Hendra Nusa Putra
Ade Wisandra

ISBN : 978-623-125-074-2

Editor : Diana Purnama Sari., S.E., M.E
Penyunting : Ari Yanto., M.Pd
Desain Sampul dan Tata Letak : Atyka Trianisa, S.Pd

Penerbit : GET PRESS INDONESIA
Anggota IKAPI No. 033/SBA/2022

Redaksi :
Jln. Palarik Air Pacah No 26 Kel. Air Pacah
Kec. Koto Tangah Kota Padang Sumatera Barat
Website : www. getpress.co.id
Email : adm.getpress@gmail.com

Cetakan pertama, Februari 2024

KATA PENGANTAR

Segala Puji dan syukur atas kehadirat Allah SWT dalam segala kesempatan. Sholawat beriring salam dan doa kita sampaikan kepada Nabi Muhammad SAW. Alhamdulillah atas Rahmat dan Karunia-Nya penulis telah menyelesaikan Buku Dasar-Dasar Pemprograman ini.

Buku ini membahas Pengantar Pemrograman, Bahasa Pemrograman C++, Deklarasi I/O, Percabangan, Perulangan, File I/O, Array, Pengurutan (Sorting), Struct, Fungsi Rekursif, *Structural Query Language* (SQL) 2 .

Proses penulisan buku ini berhasil diselesaikan atas kerjasama tim penulis. Demi kualitas yang lebih baik dan kepuasan para pembaca, saran dan masukan yang membangun dari pembaca sangat kami harapkan.

Penulis ucapkan terima kasih kepada semua pihak yang telah mendukung dalam penyelesaian buku ini. Terutama pihak yang telah membantu terbitnya buku ini dan telah mempercayakan mendorong, dan menginisiasi terbitnya buku ini. Semoga buku ini dapat bermanfaat bagi masyarakat Indonesia.

Padang, Februari 2024

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
PENGANTAR PEMROGRAMAN

Oleh Irmawati

1.1 Pendahuluan

Pemrograman merupakan proses pembuatan serangkaian instruksi yang dapat dijalankan oleh komputer untuk mencapai tujuan tertentu. Melibatkan penulisan kode menggunakan bahasa pemrograman, yang kemudian diinterpretasikan atau dikompilasi menjadi bentuk yang dapat dimengerti oleh mesin. Pemrograman merupakan unsur kunci dalam pengembangan perangkat lunak dan teknologi informasi (Martin, 2011).

Peran Pemrograman dalam Teknologi:

1. Pengembangan Perangkat Lunak: Pemrograman digunakan untuk menciptakan berbagai jenis perangkat lunak, termasuk aplikasi bisnis dan permainan komputer.
2. Pemrograman Web: Untuk mengembangkan situs web dan aplikasi web, bahasa pemrograman seperti HTML, CSS, dan JavaScript digunakan untuk membangun antarmuka pengguna yang interaktif.
3. Pemrograman Perangkat Keras (*Hardware*): Pemrograman diperlukan dalam pengembangan perangkat keras komputer dan perangkat embedded seperti mikrokontroler dan FPGA.
4. Kecerdasan Buatan (AI): Dalam pengembangan kecerdasan buatan, pemrograman digunakan untuk membuat algoritma yang memungkinkan mesin belajar dari data dan membuat keputusan tanpa intervensi manusia.
5. Pemrograman Mobile: Aplikasi mobile membutuhkan pemrograman untuk membangun aplikasi yang dapat dijalankan pada perangkat seluler, baik Android maupun iOS.
6. Analisis Data: Pemrograman digunakan untuk mengolah dan menganalisis data, dengan bahasa pemrograman seperti Python yang luas digunakan dalam solusi analitik dan ilmiah.

Algoritma adalah serangkaian langkah-langkah terstruktur yang dirancang untuk menyelesaikan suatu masalah atau tugas tertentu. Algoritma menyediakan panduan atau resep yang jelas untuk mencapai tujuan tersebut. Sifat-sifat seperti kejelasan, keberlanjutan, dan keefektifan menjadi kriteria penting dalam merancang algoritma. Dalam pemrograman, implementasi algoritma menjadi dasar untuk menulis kode yang efisien dan efektif (Cormen *et al.*, 2022).

Logika pemrograman melibatkan cara berpikir analitis dalam merancang algoritma dan membuat keputusan yang tepat selama pengembangan perangkat lunak. Pemahaman terhadap aliran eksekusi program dan kontrolnya adalah aspek kunci dari logika pemrograman. Prinsip-prinsip logika digunakan untuk memastikan kebenaran dan efisiensi dari suatu solusi program (Pratt, Zelkowitz and Gopal, 1984).

1.2 Pengembangan Pemrograman

Software Development Kit (SDK) adalah kumpulan alat, perpustakaan, dan dokumen yang memungkinkan pengembang perangkat lunak untuk membuat, menguji, dan mengintegrasikan aplikasi perangkat lunak pada suatu platform atau lingkungan pengembangan tertentu. Peran SDK dalam pengembangan pemrograman sangat penting karena memberikan alat dan sumber daya yang dibutuhkan pengembang untuk menciptakan aplikasi yang kompatibel dan efisien pada suatu platform atau sistem (Nong *et al.*, 2022).

Berikut adalah beberapa peran utama dari SDK dalam pengembangan pemrograman:

1. Menghadirkan Alat Pengembangan:
   SDK menyediakan seperangkat alat pengembangan yang membantu pengembang dalam proses pembuatan perangkat lunak. Termasuk kompiler, debugger, profiler, dan alat lain yang diperlukan untuk mengembangkan dan menguji kode.
2. Perpustakaan dan API:
   SDK biasanya berisi perpustakaan (libraries) dan antarmuka pemrograman aplikasi (API) yang menyediakan fungsi-fungsi umum dan layanan yang dapat digunakan oleh

pengembang. Membantu mengurangi pekerjaan yang harus dilakukan dari awal, mempercepat proses pengembangan, dan memastikan konsistensi aplikasi.

3. Dokumentasi:
   SDK dilengkapi dengan dokumentasi yang lengkap, yang memberikan informasi tentang cara menggunakan setiap fitur SDK dengan benar. Dokumentasi ini sangat penting untuk memahami fungsionalitas SDK dan mempercepat kurva pembelajaran pengembang.
4. Integrasi dengan Perangkat Keras dan Perangkat Lunak:
   SDK memfasilitasi integrasi aplikasi dengan perangkat keras dan perangkat lunak yang mendasarinya. Memungkinkan pengembang untuk memaksimalkan fitur-fitur yang ditawarkan oleh platform atau sistem yang mereka targetkan.
5. Pemeliharaan Keseragaman:
   SDK membantu dalam memelihara keseragaman antara aplikasi yang dikembangkan untuk suatu platform tertentu. Membantu dalam menciptakan pengalaman pengguna yang konsisten di seluruh aplikasi yang berjalan pada platform tersebut.
6. Pembaruan dan Dukungan:
   Pengembang biasanya menerima pembaruan dan dukungan teknis dari penyedia SDK. Memastikan bahwa aplikasi yang dikembangkan selalu dapat diintegrasikan dengan perangkat keras atau perangkat lunak yang diperbarui, dan masalah teknis dapat diselesaikan dengan cepat.

Text Editor dan *Integrated Development Environment* (IDE) merupakan dua jenis perangkat lunak yang digunakan oleh para pengembang perangkat lunak untuk menulis dan mengelola kode. Meskipun keduanya bertujuan membantu dalam proses pengembangan perangkat lunak, terdapat perbedaan signifikan dalam fungsionalitas, kustomisasi, ketersediaan sumber terbuka, kecepatan, dan kinerja. Berikut adalah perbandingan antara keduanya:

1. Fungsi Utama:
   a. Text Editor:
      Digunakan khusus untuk menulis dan mengedit kode.
      Menyediakan fungsi dasar seperti pemformatan teks dan penyorotan sintaks.
   b. IDE:
      Menyediakan lingkungan pengembangan lengkap.
      Melibatkan fitur-fitur seperti penyusunan kode, debugging, dan manajemen proyek.
2. Fungsionalitas:
   a. Text Editor:
      Fungsionalitas terbatas pada penyuntingan teks.
      Cocok untuk pekerjaan cepat dan pengembangan kecil.
   b. IDE:
      Menyediakan alat lengkap untuk pengembangan perangkat lunak.
      Termasuk debugger, kompiler, analisis kode, dan integrasi dengan manajemen versi.
3. Kustomisasi:
   a. Text Editor:
      Biasanya lebih ringan dan dapat disesuaikan.
      Pengguna dapat menambahkan plugin sesuai kebutuhan.
   b. IDE:
      Cenderung lebih berat dan mungkin memiliki kurva belajar yang lebih tinggi.
      Menyediakan pengaturan yang kaya dan alat kustomisasi untuk kebutuhan pengembangan yang lebih kompleks.
4. Ketersediaan Sumber Terbuka:
   a. Text Editor:
      Banyak text editor bersifat open source (contoh: VSCode, Atom).
   b. IDE:
      Beberapa IDE bersifat open source (contoh: Eclipse, Visual Studio Code), tetapi banyak juga yang berbayar.

5. Kecepatan dan Kinerja:
    a. Text Editor:
       Biasanya lebih cepat dan memiliki jejak memori yang lebih kecil.
    b. IDE:
       Memiliki kecenderungan untuk menggunakan lebih banyak sumber daya sistem.

1.3 Struktur Dasar Program

Struktur dasar pemrograman adalah kerangka atau format umum yang digunakan dalam menulis program komputer. Struktur ini membantu dalam mengorganisir kode program sehingga dapat dijalankan dan dipahami dengan lebih baik. Meskipun struktur ini dapat sedikit bervariasi tergantung pada bahasa pemrograman yang digunakan, umumnya terdiri dari beberapa elemen utama. Berikut adalah komponen-komponen utama dari struktur dasar pemrograman (Stroustrup, 2014):

1. Deklarasi dan Inisialisasi:
   Program dimulai dengan deklarasi variabel dan inisialisasi nilai awal jika diperlukan. Variabel adalah tempat untuk menyimpan data, dan deklarasi memberi tahu program tentang tipe data dan nama variabel yang akan digunakan.
2. Input:
   Penerimaan input dari pengguna atau sumber lainnya adalah komponen penting dalam banyak program. Memungkinkan program berinteraksi dengan pengguna atau mendapatkan data eksternal.
3. Proses dan Operasi:
   Bagian utama dari program diisi dengan operasi dan proses yang harus dilakukan. Termasuk perhitungan, manipulasi data, dan logika kontrol alur program.
4. Output:
   Program menampilkan hasil atau memberikan keluaran kepada pengguna atau sistem lainnya. Output dapat berupa teks, nilai numerik, atau bentuk visual lainnya.

5. Struktur Kontrol Alur (Percabangan dan Perulangan):
   Penggunaan percabangan (*if-else*) dan perulangan (*for, while*) membentuk struktur kontrol alur program. Memungkinkan program untuk membuat keputusan atau mengulang tindakan tertentu berdasarkan kondisi tertentu.
6. Fungsi atau Prosedur:
   Pemisahan kode ke dalam fungsi atau prosedur membantu dalam pengorganisasian dan pengelolaan kode yang lebih baik. Fungsi adalah blok kode yang dapat dipanggil dari bagian lain program.
7. Kommentar:
   Penggunaan komentar membantu menjelaskan atau memberikan dokumentasi pada bagian-bagian tertentu dari kode. Meningkatkan keterbacaan dan pemahaman kode.

1.4 Paradigma Pemrograman

Paradigma pemrograman merujuk pada suatu kerangka pemikiran atau pendekatan yang digunakan dalam pengembangan perangkat lunak. Hal ini mencakup seperangkat aturan, konsep, dan metode yang memandu cara seorang programmer merancang dan mengimplementasikan solusi untuk masalah yang dihadapi (Mitchell, 2003). Berikut adalah beberapa paradigma pemrograman utama:

1. Imperatif:
   Paradigma ini berfokus pada penentuan langkah-langkah yang harus diambil untuk menyelesaikan suatu tugas. Programmer menginstruksikan komputer bagaimana cara melaksanakan serangkaian perintah.
   Contoh bahasa pemrograman: C, Fortran.
2. Deklaratif:
   Paradigma deklaratif fokus pada yang ingin dicapai tanpa memberikan langkah-langkah rinci. Memungkinkan programmer untuk mendeskripsikan hasil yang diinginkan.
   Contoh bahasa pemrograman: SQL, Haskell.
3. Orientasi Objek:
   Berbasis konsep objek, paradigma ini menggabungkan data dan metode (fungsi) yang beroperasi pada data tersebut.

Objek dapat berinteraksi satu sama lain untuk mencapai tujuan tertentu.
Contoh bahasa pemrograman: Java, Python.

4. Fungsional:
   Paradigma fungsional menekankan penggunaan fungsi matematika dan menghindari perubahan keadaan atau mutasi data. Fokus utamanya adalah pada evaluasi fungsi.
   Contoh bahasa pemrograman: Haskell, Lisp.
5. Logika:
   Menggunakan aturan logika formal untuk menentukan solusi dari suatu masalah. Program ditulis sebagai kumpulan aturan logika dan fakta.
   Contoh bahasa pemrograman: Prolog.
6. Berbasis Agen:
   Paradigma ini menangani entitas independen yang disebut agen yang berinteraksi satu sama lain. Agen memiliki kemampuan untuk memproses informasi dan mengambil keputusan.
   Contoh bahasa pemrograman: AgentSpeak.
7. Berbasis Konkurensi:
   Berfokus pada pengelolaan banyak proses atau tugas yang berjalan bersamaan untuk meningkatkan efisiensi dan kinerja sistem.
   Contoh bahasa pemrograman: Erlang, Go.
8. Berorientasi Layanan:
   Memisahkan aplikasi menjadi layanan independen yang berkomunikasi satu sama lain melalui antarmuka. Ini mendukung fleksibilitas dan skalabilitas.
   Contoh arsitektur: Arsitektur Mikro.

1.4.1 Pemrograman Prosedural

Pemrograman prosedural adalah paradigma pemrograman yang fokus pada prosedur atau langkah-langkah yang harus dilakukan dalam menyelesaikan suatu tugas atau masalah. Dalam pemrograman prosedural, program dibangun menggunakan serangkaian prosedur atau fungsi yang saling berinteraksi untuk mencapai tujuan tertentu. Struktur programnya didasarkan pada

urutan langkah-langkah yang dieksekusi satu per satu (Kernighan and Ritchie, 2002).

Beberapa karakteristik utama dari pemrograman prosedural meliputi:

1. Prosedur atau Fungsi: Program dibagi menjadi serangkaian prosedur atau fungsi. Setiap prosedur memiliki tugas tertentu dan dapat dipanggil dari bagian lain dalam program.
2. Variabel: Data disimpan dalam variabel, dan variabel ini dapat diakses oleh prosedur-prosedur yang ada dalam program. Namun, biasanya ada batasan akses untuk mendorong konsep enkapsulasi.
3. Struktur Kontrol: Pemrograman prosedural menggunakan struktur kontrol seperti pengulangan (*loop*) dan percabangan (*branching*) untuk mengontrol alur eksekusi program.
4. Top-down Design: Pendekatan perancangan "top-down" digunakan, di mana program direncanakan mulai dari tingkat tinggi hingga rincian terkecil.
5. Modularitas: Program dibagi menjadi modul atau unit yang independen secara fungsional. Setiap modul menangani tugas tertentu dan dapat digunakan kembali.

Contoh bahasa pemrograman yang mendukung paradigma pemrograman prosedural meliputi C, Pascal, dan Fortran.

Sementara ini pemrograman prosedural masih relevan, paradigma pemrograman lain seperti objek-berorientasi dan fungsional juga memiliki peran penting dalam pengembangan perangkat lunak modern. Pemilihan paradigma pemrograman harus disesuaikan dengan kebutuhan dan karakteristik proyek yang sedang dikerjakan.

1.4.2 Pemrograman Berorientasi Objek

Pemrograman Berorientasi Objek (OOP) adalah suatu paradigma pemrograman yang merancang perangkat lunak dengan memanfaatkan konsep objek sebagai elemen dasar. Dalam OOP, suatu objek dianggap sebagai entitas yang memiliki karakteristik (atribut) dan perilaku (metode) yang terkait. Paradigma ini

bertujuan untuk meningkatkan modularitas, reusabilitas kode, dan manajemen kompleksitas dalam pengembangan perangkat lunak (Helm *et al.*, 2000).

Konsep utama dalam Pemrograman Berorientasi Objek mencakup:

1. Kelas (Class): Sebagai blueprint atau cetak biru, kelas mendefinisikan atribut dan metode yang akan dimiliki oleh objek. Ini menciptakan suatu tipe data baru.
2. Objek: Merupakan instance konkret dari suatu kelas. Setiap objek memiliki atribut unik sesuai dengan definisi kelasnya.
3. Enkapsulasi: Konsep ini melibatkan penyembunyian detail implementasi internal objek dan hanya mengekspos fungsionalitas yang diperlukan. Atribut biasanya didefinisikan sebagai "private" dengan metode akses yang disediakan.
4. Warisan (*Inheritance*): Mekanisme yang memungkinkan suatu kelas mewarisi atribut dan metode dari kelas lain. Ini memfasilitasi reusabilitas kode dan memungkinkan pembuatan hierarki kelas.
5. Polimorfisme: Kemampuan suatu objek untuk merespons secara berbeda terhadap pemanggilan metode yang sama. Polimorfisme dapat dicapai melalui konsep overloading dan overriding metode.

Beberapa bahasa pemrograman yang mendukung Pemrograman Berorientasi Objek termasuk Java, Python, C++, dan C#.

DAFTAR PUSTAKA

Cormen, T.H. *et al.* 2022. *Introduction to algorithms*. MIT press.

Helm, R. *et al.* 2000. *Design patterns: Elements of reusable object-oriented software*. Braille Jymico Incorporated Quebec.

Kernighan, B.W. and Ritchie, D.M. 2002. 'The C programming language'.

Martin, R.C. 2011. *The clean coder: a code of conduct for professional programmers*. Pearson Education.

Mitchell, J.C. 2003. *Concepts in programming languages*. Cambridge University Press.

Nong, Y. *et al.* 2022. 'Generating realistic vulnerabilities via neural code editing: an empirical study', in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1097–1109.

Pratt, T.W., Zelkowitz, M. V and Gopal, T. V. 1984. *Programming languages: design and implementation*. Prentice-Hall Englewood Cliffs, NJ.

Stroustrup, B. 2014. *Programming: principles and practice using C++*. Pearson Education.

BAB 2
BAHASA PEMOGRAMAN C++

Oleh Dani Anggoro

2.1 Pendahuluan

2.1.1 Sejarah Perkembangan C++

C++ adalah sebuah bahasa pemrograman tingkat tinggi yang dikembangkan dari bahasa pemrograman C oleh Bjarne Stroustrup pada awal tahun 1980-an di Bell LAbs. Menurutnya, C++ adalah bahasa pemrograman yang dirancang untuk memberikan efisiensi dan fleksibilitas dari bahasa tingkat rendah seperti C. Bahasa pemrograman ini telah mengalami beberapa perbaikan dengan standarisasi yang dilakukan oleh *International Organization for Standardization* (ISO) dan *International Electrotechnical Commission* (IEC). Perbaikan meliputi sisi bahasa maupun library.

2.1.2 Keunggulan C++ dibanding dengan bahasa pemograman lainnya

C++ adalah bahasa pemrograman yang populer dan berkualitas tinggi. Berdasarkan penelitian yang dilakukan oleh TIOBE (www.tiobe.com) , C++ selalu masuk tiga besar bahasa pemrograman yang paling banyak digunakan oleh programmer profesional di seluruh dunia selain C dan JAVA.

Tabel 2.1. Bahasa Pemrogram Terpopuler menurut tiobe.com

Programming Language	2024	2019	2014	2009	2004
Python	1	4	8	6	11
C	2	2	1	2	2
C++	3	3	4	3	3
Java	4	1	2	1	1

Bahasa ini digunakan untuk mengembangkan beraneka ragam program dari embedded system, telekomunikasi, bisnis sampai hiburan atau game. C++ juga dikembangkan untuk pengembangan aplikasi berbasis web. Menurut Andrei Alexandrescu, menyoroti fitur-fitur seperti template dan konsep yang membuat C++ menjadi pilihan yang kuat untuk ekspresi pemrograman generik.

2.2 Editor

Sebelum menggunakan C++, perlu menyiapkan software pendukung baik sebagai editor ataupun compiler.

2.2.1 Editor Teks

Editor teks digunakan sebagai media untuk menuliskan kode program. Editor yang dapat digunakan untuk menuliskan kode C++ diantaranya: Notepad, OS Edit Command, Brief, Epsilon, EMACS atau Vim. File yang dibuat pada editor disebut source file dan disimpan dengan ektensi file .cpp.

2.2.2 Compiler

Compiler digunakan untuk mengkompile kode program yang telah ditulis di editor teks. Compiler yang sering digunakan adalah GNU C/C++ yang dapat digunakan secara gratis. Selain compiler tersebut, dapat juga menggunakan compler lain seperti Eclipe, NetBeans atau lainnya.

2.3 Sintaks Dasar

2.3.1 Pemograman Berorientasi Objek dalam Bahasa C++

Sebuah program C++ dapat diartikan sebagai kumpulan objek yang berkomunikasi dengan method-method yang ada. Berikut penjelasan singkat tentang kelas, objek, method dan variabel instance pada pemograman C++:

1. Kelas
   Kelas adalah blueprint atau cetak biru untuk menciptakan objek. Kelas mendefinisikan struktur dan perilaku objek yang akan dibuat berdasarkan kelas. Selain itu, kelas juga mendefinisikan atribut (variabel anggota) dan metode (fungsi anggota) yang dimiliki oleh objek.

```
class Manusia {
  // atribut
  public:
    string nama;
    int umur;

  // metode
  void sapa() {
    cout << "Halo, nama saya " << nama << "!" << endl;
  }
};
```

Gambar 2.1. Penulisan Kode Class menggunakan C++

2. Objek
   Objek adalah instance dari kelas, yaitu realisasi konkret dari blueprint kelas. Objek dibuat menggunakan konstruktor kelas.

```
Manusia person; // Membuat objek "person" berdasarkan kelas Manusia
person.nama = "John";
person.umur = 25;
person.sapa(); // Memanggil metode dari objek
```

Gambar 2.2. Penulisan Kode Objek menggunakan C++

3. Method
   Metode adalah fungsi anggota dari kelas yang mendefinisikan perilaku objek. Dapat memanipulasi variabel anggota dan melakukan operasi tertentu.

```
class Kalkulator {
public:
   int tambah(int a, int b) {
      return a + b;
   }
};

Kalkulator calc;
int hasil = calc.tambah(3, 5); // Memanggil metode "tambah"
```

Gambar 2.3. Penulisan Kode Method menggunakan C++

4. Variabel Instance
   Variabel instance adalah variabel yang dimiliki oleh setiap objek individu. Setiap objek memiliki salinan independen dari variabel instance kelasnya.

```
class Mobil {
public:
   string merk;
   int tahunProduksi;
};

Mobil mobil1;
mobil1.merk = "Toyota";
mobil1.tahunProduksi = 2020;

Mobil mobil2;
mobil2.merk = "Honda";
mobil2.tahunProduksi = 2019;
```

Gambar 2.4. Penulisan Kode Variabel Instance menggunakan C++

2.3.2 Struktur Program C++

Berikut adalah contoh pentulisan kode sederhana menggunakan bahasa pemrograman C++

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

int main() {
   cout << "Hello, World!";
   return 0;
}
```

Gambar 2.5. Penulisan Kode Variabel Instance menggunakan C++

Penjelasan kode program diatas adalah sebagai berikut:

1. #include <iostream>: Mendeklarasikan penggunaan pustaka iostream yang berisi fungsi-fungsi input/output standar (seperti cout).
2. Using namespace std;: Menggunakan namespace std agar tidak perlu menulis std:: setiap kali menggunakan elemen dari namespace tersebut. Ini membuat penggunaan cout menjadi lebih sederhana.
3. Int main(): Fungsi utama dari program, eksekusi dimulai dari sini.
4. Cout << "Hello, World!" << endl;: Menampilkan teks "Hello, World!" ke layar menggunakan objek cout dari pustaka iostream. Endl digunakan untuk memberi karakter baru (new line) setelah menampilkan teks.
5. Return 0;: Mengindikasikan bahwa program telah berakhir dengan sukses.

2.3.3 Titik-koma dan Blok dalam C++

Dalam C++ titik koma merupakan terminator. Setiap statmen kode program harus diakhiri dengan titik koma.

```
x = y;
y = y+1;
add(x, y);
```

Gambar 2.6. Penulisan Titik Koma di Akhir Statment

Blok merupakan kumpulan statmen yang beruhubungan logis yang diawali dan diakhiri

```
{
   cout << "Hello, World!";
   return 0;
}
```

Gambar 2.7. Penulisan Blok

2.3.4 Identifier

Identifier C++ adalah nama yang digunakan untuk mengidentifikasi variabel, fungsi, kelas, modul atatau lainnya. Identifier diawali dengan A-Z atau a-z atau underscore _ yang diikuti oleh huruf dan atau angka 0-9.

C++ bersifat case-sensitive identifier dengan nama Hello dan hello merupakan 2 identifier yang berbeda.

2.3.5 Kata Kunci

Berikut ini daftar kata kunci yang ada pada C++. Kata kunci ini tidak dapat digunakan sebagai nama identifier.

Tabel 2.2. Daftar Kata Kunci yang Tidak Bisa Digunakan Sebagai Identifier

auto	register	continue	static	operator
double	typedef	for	while	this
int	char	signed	dynamic_cast	true
struct	extern	void	namespace	typeid
break	return	default	static_cast	and
else	union	goto	bool	bitor
long	const	sizeof	explicit	compl
switch	float	volatile	new	not
case	short	do	catch	or
enum	unsigned	if	false	xor

2.3.6 Komentar pada C++

Dalam C++, komentar digunakan untuk memberikan penjelasan atau dokumentasi pada kode program. Ada dua jenis komentar utama di C++: komentar satu baris dan komentar multibaris. Berikut adalah contoh keduanya:

1. Komentar Satu Baris

```
#include <iostream>

int main() {
  // Ini adalah komentar satu baris
  std::cout << "Hello, World!" << std::endl; // Menampilkan teks ke layar

  return 0; // Mengindikasikan bahwa program berakhir dengan sukses
}
```

Gambar 2.8. Penulisan Komentar Satu Baris

2. Komentar Multibaris

```
#include <iostream>

/*
  Ini adalah komentar multibaris.
  C++ akan mengabaikan semua teks di antara /* dan
*/.
  Komentar ini digunakan untuk memberikan
penjelasan lebih panjang.
*/
```

Gambar 2.9. Penulisan Komentar Multibaris

2.3.7 Variabel

Variabel dalam C++ adalah sebuah nama yang digunakan untuk menyimpan nilai di dalam memori. Setiap variabel harus dideklarasikan dengan tipe datanya sebelum digunakan. Setelah dideklarasikan, variabel dapat diisi dengan nilai dan nilai tersebut dapat diubah selama jalannya program. Contoh penggunaan variabel dalam C++:

```
#include <iostream>

int main() {
   // Mendeklarasikan variabel dengan tipe data int
   int angka;

   // Memberi nilai pada variabel
   angka = 5;

   // Menampilkan nilai variabel ke layar
   std::cout << "Nilai dari variabel 'angka' adalah: " <<
angka << std::endl;

   // Mengganti nilai variabel
   angka = 10;
```

```
    // Menampilkan nilai variabel yang sudah diubah
    std::cout << "Nilai baru dari variabel 'angka' adalah: " << angka << std::endl;

    return 0;
}
```

Gambar 2.10. Penulisan Variabel

Dalam contoh di atas:
angka adalah variabel bertipe data int yang digunakan untuk menyimpan nilai bilangan bulat.

1. Pertama, variabel angka dideklarasikan dan kemudian diisi dengan nilai 5.
2. Kemudian, nilai dari variabel angka ditampilkan ke layar.
3. Nilai variabel angka kemudian diubah menjadi 10.
4. Nilai yang baru diubah dari variabel angka ditampilkan ke layar.

Variabel dapat memiliki berbagai tipe data, seperti int untuk bilangan bulat, double untuk bilangan pecahan, char untuk karakter, dan sebagainya

2.4 Kondisi dan Perulangan

2.4.1 Statment if

Statment if terdiri dari sebuah ekpresi Boolean yang diikuti dengan satu stement atau lebih. Berikut ini contoh penulisan kode untuk statment if.

```
#include <iostream>

int main() {
   // Mendeklarasikan variabel
   int nilai = 75;

   // Menggunakan statement if untuk melakukan
pengecekan
   if (nilai >= 70) {
      std::cout << "Selamat, Anda lulus!" << std::endl;
   }
   return 0;
}
```

Gambar 2.11. Penulisan Kondisi If

Penjelasan singkat:

1. int nilai = 75;: Mendeklarasikan variabel nilai dan memberinya nilai 75.
2. if (nilai >= 70) { ... }: Statement if digunakan untuk melakukan pengecekan. Jika ekspresi dalam kurung kurawal (dalam hal ini, nilai >= 70) bernilai true, maka blok kode di dalam kurung kurawal akan dieksekusi. Jika tidak, blok kode tersebut dilewati.
3. std::cout << "Selamat, Anda lulus!" << std::endl;: Bagian dalam kurung kurawal merupakan blok kode yang akan dieksekusi jika kondisi dalam statement if bernilai true. Dalam contoh ini, pesan "Selamat, Anda lulus!" akan ditampilkan jika nilai lebih besar atau sama dengan 70.

2.4.2 Statment if... else

Statment if bisa diikuti dengan stement else, yang dieksekusi jika operasi if bernilai salah

```
#include <iostream>

int main() {
  // Mendeklarasikan variabel
  int nilai = 65;

  // Menggunakan statement if-else untuk melakukan pengecekan
  if (nilai >= 70) {
    std::cout << "Selamat, Anda lulus!" << std::endl;
  } else {
    std::cout << "Maaf, Anda belum lulus." << std::endl;
  }

  return 0;
}
```

Gambar 2.12. Penulisan Kondisi If...else

Penjelasan singkat:

1. int nilai = 65;: Mendeklarasikan variabel nilai dan memberinya nilai 65.
2. if (nilai >= 70) { ... } else { ... }: Statement if-else digunakan untuk melakukan pengecekan. Jika kondisi dalam kurung kurawal if bernilai true, maka blok kode di dalamnya akan dieksekusi. Jika tidak, blok kode dalam kurung kurawal else akan dieksekusi.
3. Dalam contoh ini, jika nilai lebih besar atau sama dengan 70, maka pesan "Selamat, Anda lulus!" akan ditampilkan; jika tidak, pesan "Maaf, Anda belum lulus." akan ditampilkan.

2.4.3 Statment if... else Bersarang

Statment if..else bersarang adalah kondisi dimana didalam kondisi statment if terdapat kondisi didalamnya. Contoh penggunaan kondisi ini sebagai berikut:

```
#include <iostream>
int main() {
  // Mendeklarasikan variabel
  int nilai = 80;

  // Menggunakan statement if..else bersarang
  if (nilai >= 70) {
    std::cout << "Anda lulus dalam ujian pertama." << std::endl;

    if (nilai >= 80) {
      std::cout << "Dan Anda juga mendapatkan nilai tinggi!" << std::endl;
    } else {
      std::cout << "Namun, nilai Anda tidak mencapai tingkat yang tinggi." << std::endl;
    }
  } else {
    std::cout << "Anda belum lulus dalam ujian pertama." << std::endl;
  }
  return 0;
}
```

Gambar 2.13. Penulisan Kondisi If...else

Penjelasan singkat:

1. int nilai = 80;: Mendeklarasikan variabel nilai dan memberinya nilai 80.
2. Statement if (nilai >= 70) {...} else {...}: Pengecekan pertama. Jika nilai lebih besar atau sama dengan 70, maka blok kode dalam if pertama akan dieksekusi; jika tidak, blok kode dalam else akan dieksekusi.

3. Di dalam blok kode if, terdapat statement if (nilai >= 80) {...} else {...} yang merupakan statement if..else bersarang. Jika nilai lebih besar atau sama dengan 80, maka blok kode dalam if bersarang akan dieksekusi; jika tidak, blok kode dalam else bersarang akan dieksekusi.
4. Pesan-pesan yang sesuai akan ditampilkan berdasarkan hasil kondisi yang terpenuhi.

2.4.4 Switch

Statment switch memungkinkan sebuah variabel diperiksa kesamaannya dengan daftar nilai yang ada. Contoh penggunaan switch sebagai berikut:

```
#include <iostream>

int main() {
  // Mendeklarasikan variabel
  int pilihan;

  // Menampilkan menu
  std::cout << "Pilih sebuah angka (1-3): ";
  std::cin >> pilihan;

  // Menggunakan statement switch untuk kondisi berbeda
  switch (pilihan) {
    case 1:
      std::cout << "Anda memilih angka satu." << std::endl;
      break;

    case 2:
      std::cout << "Anda memilih angka dua." << std::endl;
      break;

    case 3:
```

```
            std::cout << "Anda memilih angka tiga." <<
        std::endl;
            break;

        default:
            std::cout << "Pilihan tidak valid." << std::endl;
            break;
    }

    return 0;
}
```

Gambar 2.14. Penulisan Kondisi If...else

Penjelasan singkat:

1. int pilihan;: Mendeklarasikan variabel pilihan untuk menyimpan input dari pengguna.
2. std::cout << "Pilih sebuah angka (1-3): "; std::cin >> pilihan;: Menampilkan pesan untuk meminta input dari pengguna dan menyimpan input ke dalam variabel pilihan.
3. switch (pilihan) { ... }: Statement switch digunakan untuk memilih blok kode yang akan dieksekusi berdasarkan nilai variabel pilihan.
4. case 1:, case 2:, dan case 3:: Masing-masing adalah label kasus yang sesuai dengan nilai yang mungkin diinputkan oleh pengguna.
5. break;: Digunakan untuk keluar dari blok switch setelah salah satu kondisi terpenuhi.
6. default:: Merupakan kasus yang akan dieksekusi jika nilai tidak cocok dengan salah satu case.

2.4.5 Perulangan While

Perulangan while adalah struktur kontrol yang memungkinkan untuk mengulangi suatu proses dengan jumlah perulangan tertentu. Selama kondisi bernilai benar, maka statment while akan terus diesekusi. Berikut contohnya :

```
#include <iostream>

int main() {
   // Mendeklarasikan variabel
   int hitungan = 1;

   // Menggunakan perulangan while
   while (hitungan <= 5) {
      std::cout << "Iterasi ke-" << hitungan << 
std::endl;
      hitungan++;
   }

   return 0;
}
```

Gambar 2.15. Penulisan Perulangan while

Penjelasan singkat:

1. Int hitungan = 1;: Mendeklarasikan variabel hitungan dan memberikan nilai awal 1.
2. While (hitungan <= 5) { ... }: Perulangan while akan terus berjalan selama kondisi dalam kurung kurawal while bernilai true. Dalam hal ini, perulangan akan berlanjut selama nilai hitungan kurang dari atau sama dengan 5.
3. Std::cout << "Iterasi ke-" << hitungan << std::endl;: Baris ini menampilkan pesan yang menunjukkan iterasi ke berapa sedang berlangsung.
4. Hitungan++;: Menambahkan nilai variabel hitungan setiap kali perulangan dilakukan untuk menghindari perulangan tak terbatas.
5. Program akan keluar dari perulangan saat hitungan melebihi

2.4.6 Perulangan do...while

Perulangan ini mempunyai tujuan yang sama dengan perulangan while, tetapi perulangan do.. while pasti akan diesekusi minimal satu kali

```
#include <iostream>

int main() {
   // Mendeklarasikan variabel
   int hitungan = 1;

   // Menggunakan perulangan do...while
   do {
      std::cout << "Iterasi ke-" << hitungan << 
std::endl;
      hitungan++;
   } while (hitungan <= 5);

   return 0;
}
```

Gambar 2.16. Penulisan Perulangan do...while

Penjelasan singkat:

1. Int hitungan = 1;: Mendeklarasikan variabel hitungan dan memberikan nilai awal 1.
2. Do { ... } while (hitungan <= 5);: Perulangan do...while akan selalu dieksekusi setidaknya sekali, dan akan terus berjalan selama kondisi dalam kurung kurawal while bernilai true. Dalam hal ini, perulangan akan terus berlanjut selama nilai hitungan kurang dari atau sama dengan 5.
3. Std::cout << "Iterasi ke-" << hitungan << std::endl;: Baris ini menampilkan pesan yang menunjukkan iterasi ke berapa sedang berlangsung.
4. Hitungan++;: Menambahkan nilai variabel hitungan setiap kali perulangan dilakukan untuk menghindari perulangan tak terbatas.
5. Program akan keluar dari perulangan saat hitungan melebihi 5.
6. Hasil dari program ini akan mencetak iterasi ke-1 hingga ke-5 di layar. Perbedaannya dengan while biasa adalah bahwa

do...while memastikan setidaknya satu iterasi dieksekusi sebelum mengevaluasi kondisi.

2.4.7 Perulangan for

Perulangan for adalah struktur kontrol repetitif yang memungkinkan menjalankan proses dengan jumlah perulangan tertentu.

```
#include <iostream>

int main() {
   // Menggunakan perulangan for
   for (int hitungan = 1; hitungan <= 5; hitungan++) {
      std::cout << "Iterasi ke-" << hitungan << std::endl;
   }

   return 0;
}
```

Gambar 2.17. Penulisan Perulangan for

Penjelasan singkat:

1. For (int hitungan = 1; hitungan <= 5; hitungan++) { ... }: perulangan for memiliki tiga bagian:
2. Bagian pertama (int hitungan = 1;): mendeklarasikan dan menginisialisasi variabel hitungan dengan nilai awal 1.
3. Bagian kedua (hitungan <= 5;): kondisi yang harus terpenuhi agar perulangan terus berjalan. Dalam hal ini, perulangan akan berlanjut selama nilai hitungan kurang dari atau sama dengan 5.
4. Bagian ketiga (hitungan++): menambahkan nilai variabel hitungan setiap kali perulangan dilakukan.
5. Std::cout << "iterasi ke-" << hitungan << std::endl;: baris ini menampilkan pesan yang menunjukkan iterasi ke berapa sedang berlangsung.
6. Program akan keluar dari perulangan setelah iterasi ke-5.

7. Hasil dari program ini akan mencetak iterasi ke-1 hingga ke-5 di layar, mirip dengan contoh sebelumnya namun dengan struktur perulangan yang sedikit berbeda.

2.4.8 Statment Break

Statment break digunakan untuk menghentikan esekusi perulangan

```
#include <iostream>

int main() {
  // Menggunakan perulangan for dengan statement break
  for (int hitungan = 1; hitungan <= 5; hitungan++) {
    std::cout << "Iterasi ke-" << hitungan << std::endl;

    // Menambahkan kondisi untuk keluar dari perulangan
    if (hitungan == 3) {
      std::cout << "Kondisi terpenuhi. Keluar dari perulangan." << std::endl;
      break; // Keluar dari perulangan jika hitungan mencapai 3
    }
  }

  return 0;
}
```

Gambar 2.18. Penulisan Statment Break

Penjelasan singkat:

1. For (int hitungan = 1; hitungan <= 5; hitungan++) { ... }: perulangan for dengan tiga bagian seperti sebelumnya.
2. Std::cout << "iterasi ke-" << hitungan << std::endl;: menampilkan pesan yang menunjukkan iterasi ke berapa sedang berlangsung.

3. If (hitungan == 3) { ... }: menambahkan kondisi di dalam perulangan. Jika nilai hitungan mencapai 3, maka blok kode dalam kurung kurawal akan dieksekusi.
4. Std::cout << "kondisi terpenuhi. Keluar dari perulangan." << std::endl;: menampilkan pesan bahwa kondisi terpenuhi.
5. Break;: statement break digunakan untuk keluar dari perulangan, sehingga program akan melanjutkan eksekusi setelah perulangan.

2.4.9 Kata kunci countinue

Kata kunci countinue digunakan untuk melompat pada iterasi selanjunya pada perulangan.

```
#include <iostream>

int main() {
  // Menggunakan perulangan for dengan kata kunci continue
  for (int hitungan = 1; hitungan <= 5; hitungan++) {
    // Menambahkan kondisi untuk menggunakan continue
    if (hitungan == 3) {
      std::cout << "Kondisi terpenuhi. Melanjutkan ke iterasi berikutnya." << std::endl;
      continue; // Melanjutkan ke iterasi berikutnya jika hitungan mencapai 3
    }

    std::cout << "Iterasi ke-" << hitungan << std::endl;
  }

  return 0;
}
```

Gambar 2.19. Penulisan Statment Countinue

Penjelasan singkat:

1. For (int hitungan = 1; hitungan <= 5; hitungan++) { ... }: perulangan for dengan tiga bagian seperti sebelumnya.
2. If (hitungan == 3) { ... }: menambahkan kondisi di dalam perulangan. Jika nilai hitungan mencapai 3, maka blok kode dalam kurung kurawal akan dieksekusi.
3. Std::cout << "kondisi terpenuhi. Melanjutkan ke iterasi berikutnya." << std::endl;: menampilkan pesan bahwa kondisi terpenuhi dan kata kunci continue akan menjalankan iterasi berikutnya tanpa mengeksekusi kode di bawahnya.
4. Continue;: statement continue digunakan untuk melanjutkan ke iterasi berikutnya dalam perulangan.

2.5 Fungsi

Fungsi adalah sekumpulan statment yang secara bersama-sama melalukan suatu tugas. Pada program yang dibuat menggunakan C++ memiliki minimal satu fungsi main(). Fungsi sering disebut juga method, sub, prosedur dan sebagainya

Berikut contoh kode fungsi pada C++:

```
#include <iostream>
// Deklarasi fungsi
int tambahkan(int a, int b);
int main() {
   // Memanggil fungsi
   int hasil = tambahkan(3, 5);
   // Menampilkan hasil fungsi ke layar
   std::cout << "Hasil penambahan: " << hasil <<
std::endl;
   return 0;
}

// Definisi fungsi
int tambahkan(int a, int b) {
   return a + b;
}
```

Gambar 2.20. Penulisan suatu Fungsi

1. Deklarasi Fungsi:
   int tambahkan(int a, int b); adalah deklarasi fungsi yang memberi tahu kompiler bahwa ada suatu fungsi bernama tambahkan yang menerima dua parameter bertipe int dan mengembalikan nilai bertipe int. Deklarasi ini umumnya ditempatkan di bagian atas file atau dalam header file.
2. Definisi Fungsi:
   int tambahkan(int a, int b) {...} adalah definisi fungsi yang berisi implementasi sebenarnya dari fungsi tersebut. Di sini, fungsi tambahkan menerima dua parameter a dan b, menjumlahkannya, dan mengembalikan hasil penjumlahan.
3. Memanggil Fungsi:
   int hasil = tambahkan(3, 5); adalah pemanggilan fungsi tambahkan dengan memberikan nilai 3 dan 5 sebagai argumen. Hasilnya disimpan dalam variabel hasil.
4. Menampilkan Hasil:
   std::cout << "Hasil penambahan: " << hasil << std::endl; digunakan untuk menampilkan hasil penambahan ke layar menggunakan objek cout dari pustaka iostream.

2.6 Input dan Output Stream

Dalam C++, "stream" adalah aliran data yang menghubungkan program C++ dengan sumber atau tujuan data. Objek stream digunakan untuk melakukan operasi input dan output pada program.

2.6.1 Input

Digunakan untuk membaca data dari sumber eksternal ke dalam program. std::cin adalah contoh dari objek input stream yang terkait dengan standard input (keyboard).

2.6.2 Output

Digunakan untuk menulis data dari program ke tujuan eksternal. std::cout adalah contoh dari objek output stream yang terkait dengan standard output (layar).

Contoh penggunaan stream:

```
#include <iostream>
#include <fstream> // Untuk file stream

int main() {
   int angka;

   // Menggunakan input stream untuk membaca
dari keyboard
   std::cout << "Masukkan sebuah angka: ";
   std::cin >> angka;

   // Menggunakan output stream untuk menulis ke
layar
   std::cout << "Anda memasukkan angka: " << angka
<< std::endl;

   // Menggunakan output stream untuk menulis ke
file
   std::ofstream fileOutput("output.txt");
   fileOutput << "Angka yang dimasukkan: " << angka
<< std::endl;
   fileOutput.close();

   return 0;
}
```

Gambar 2.21. Penulisan suatu Fungsi

Dalam contoh di atas, std::cin dan std::cout adalah objek stream yang digunakan untuk operasi input dan output ke dan dari standard input/output. Objek stream juga dapat digunakan untuk membaca dan menulis dari/ke file, socket, atau perangkat lainnya. Dalam hal ini, std::ofstream digunakan untuk menulis ke file dengan nama "output.txt".

DAFTAR PUSTAKA

Jubilee Enterprise. 2015, Cepat Menguasai C++. Jakarta: PT Elex Media Komputindo

Fatchiyatur Rohmah, Mimin & Eko Hayono, Rokhmad. 2022. Teori dan Aplikasi Algoritma Pemograman dengan Bahasa C dan C++. Pekalongan: PT. Nasya Expanding Management

T. Henny Febrianan Harumy, dkk. 2016. Belajar Dasar Algoritma dan Pemograman C++. Yokyakarta: Deepublish

BAB 3
DEKLARASI I/O

Oleh Zumhur Alamin

3.1 Pendahuluan

3.1.1 Definisi dan Pentingnya

Langkah-langkah untuk memasukkan data ke dalam program (*input*) dan menampilkan hasil keluaran dari program (*output*) dikenal dengan istilah deklarasi I/O. Dalam konteks pemrograman, *input* sering kali berasal dari pengguna atau sumber eksternal, sedangkan *output* adalah cara program memberikan informasi atau hasil kepada pengguna atau sistem lainnya. Dengan kata lain Deklarasi I/O adalah proses mendefinisikan dan mengonfigurasi variabel atau sumber daya yang akan digunakan untuk masukan (*input*) dan keluaran (*output*) dalam suatu program.

Deklarasi I/O menjadi lebih relevan dalam aplikasi yang membutuhkan komunikasi dua arah antara program dan pengguna. Misalnya, dalam aplikasi *video game*, pengguna memberikan *input* melalui keyboard atau mouse, sementara program merespon dengan menampilkan perubahan pada layar (Harmayani et al., 2021). Oleh karena itu, memahami cara mendeklarasikan dan mengelola *input*/*output* memungkinkan programmer untuk menciptakan aplikasi yang responsif dan interaktif .

Pentingnya deklarasi I/O juga terletak pada kemampuannya untuk membuat program lebih dinamis. Dengan *input* yang bervariasi, program dapat menyesuaikan perilakunya, memberikan pengalaman yang lebih banyak kepada pengguna. Begitu juga, *output* yang diformat dengan baik memastikan bahwa informasi yang disajikan oleh program mudah dimengerti, meningkatkan kegunaan aplikasi secara keseluruhan.

Deklarasi I/O memainkan peran penting dalam mengelola data. Melalui *input*, program menerima data yang kemudian dapat diolah untuk menghasilkan *output* yang relevan. Pemahaman tentang cara deklarasi I/O berinteraksi dengan struktur data dan

variabel sangat diperlukan untuk memanfaatkan potensi penuh dari data yang dimanipulasi oleh program.

3.1.2 Peran Deklarasi I/O dalam Pemrograman

Pentingnya deklarasi I/O dalam pemrograman tak hanya terletak pada proses teknis penerimaan dan penyampaian data. Secara konseptual, deklarasi I/O memungkinkan programmer untuk membawa program mereka dari dunia algoritmik menuju interaksi yang lebih manusiawi (Nandita & Yahfizham, 2023). Dalam sebuah aplikasi, bagaimana program berkomunikasi dengan pengguna atau sistem lain dapat menjadi faktor penentu kesuksesan aplikasi itu sendiri.

Deklarasi I/O menciptakan jembatan antara dunia digital komputer dan pengguna nyata. Pada saat program dijalankan, pengguna memasukkan informasi melalui *input*, seperti mengisi formulir atau memberikan perintah tertentu. Program kemudian memproses *input* ini dan menghasilkan *output* yang bermakna. Oleh karena itu, deklarasi I/O bukan sekadar teknik, tetapi sebuah seni dalam membawa aplikasi ke tingkat interaksi yang lebih tinggi.

Salah satu peran kunci deklarasi I/O adalah memberikan kontrol kepada pengguna. Dengan *input*, pengguna dapat memberikan arahan atau memberikan data yang program butuhkan untuk menjalankan tugasnya. Sebaliknya, *output* memberikan umpan balik kepada pengguna, memastikan bahwa interaksi tersebut memenuhi tujuan dan ekspektasi pengguna. Dengan memahami dan menerapkan prinsip-prinsip deklarasi I/O dengan baik, programmer dapat menciptakan program yang responsif, ramah pengguna, dan dapat diandalkan. Dalam bab ini, kita akan menjelajahi lebih lanjut tentang bagaimana deklarasi I/O dapat diterapkan dalam berbagai konteks pemrograman.

3.2 Tipe Data dan Variabel

3.2.1 Pengertian Tipe Data

Tipe data adalah konsep dasar dalam pemrograman yang mengacu pada jenis nilai yang dapat disimpan dan dioperasikan oleh suatu variabel atau ekspresi. Dalam konteks deklarasi I/O, pemahaman tipe data menjadi penting karena berhubungan erat

dengan jenis data yang dapat diterima atau ditampilkan oleh program. Tipe data dapat berupa bilangan bulat, pecahan, karakter, atau struktur data yang lebih kompleks (Chaniago & Sastradipraja, 2022). Dalam deklarasi I/O, pemahaman yang mendalam tentang tipe data berperan untuk memastikan bahwa program dapat berinteraksi dengan *input* dan *output* sesuai dengan harapan.

Memilih tipe data yang tepat adalah langkah pertama menuju efektivitas program, memastikan data dapat direpresentasikan dengan benar dan diolah sesuai kebutuhan. Tipe data dapat dibayangkan sebagai aturan yang memberitahu komputer bagaimana menginterpretasikan dan memanipulasi nilai-nilai. Sebagai contoh, ketika kita meminta pengguna memasukkan usia mereka, kita mungkin ingin menyimpan nilai tersebut dalam tipe data bilangan bulat karena usia tidak dapat memiliki nilai pecahan.

Pemahaman tentang tipe data juga membantu dalam menghindari kesalahan yang mungkin muncul akibat inkonsistensi dalam pengelolaan nilai. Misalnya, jika program mengharapkan angka bulat tetapi menerima *input* berupa karakter, hasilnya dapat tidak terduga. Dengan demikian, kita akan menggali lebih dalam tentang berbagai tipe data yang sering digunakan dalam konteks deklarasi I/O, serta cara memilih yang sesuai dengan skenario penggunaannya.

3.2.2 Macam-Macam Tipe Data

Pemrogram memiliki akses ke berbagai macam tipe data dengan karakteristik dan kegunaan yang unik. Tipe data dasar seperti integer, float, dan char, serta tipe data majemuk yaitu array atau struct (Sitorus, 2015). Kita akan membahas tipe data yang paling umum digunakan dalam konteks deklarasi I/O, menyediakan dasar bagi programmer untuk memilih tipe data yang sesuai dengan kebutuhan program. Berikut ini merupakan contoh dari beberapa tipe data tersebut.

1. Integer (int)
   Digunakan untuk merepresentasikan bilangan bulat tanpa nilai pecahan. Misalnya, usia seseorang atau jumlah item dalam stok barang.

2. Float/*Double*
   Mengacu pada tipe data yang digunakan untuk menyimpan nilai numerik yang terdapat bagian pecahan. Misalnya, ketika menghitung hasil perhitungan matematika yang melibatkan desimal.
3. Char (*Character*)
   Digunakan untuk menyimpan karakter tunggal, seperti huruf atau simbol. Misalnya menyimpan jenis kelamin, dimana 'L' = laki-laki atau 'P' = perempuan.
4. Array
   Merupakan kumpulan elemen-elemen dengan tipe data yang sama. Array memungkinkan pengelompokan data terstruktur, seperti daftar nilai suhu harian selama seminggu.
5. Struct (*Structure*)
   Memungkinkan programmer untuk membuat tipe data khusus yang menggabungkan beberapa variabel dengan tipe data berbeda ke dalam satu kesatuan. Sebagai contoh, kita dapat membuat struktur data untuk merepresentasikan informasi tentang seorang siswa, termasuk nama, usia, dan nilai-nilai ujian.

Pemilihan tipe data yang sesuai dapat meningkatkan efisiensi dan kejelasan kode, memastikan bahwa data dapat direpresentasikan dan diolah dengan cara yang paling efektif serta sesuai dengan kebutuhan program.

3.2.3 Pendeklarasian Variabel

Pendeklarasian variabel adalah langkah penting dalam membangun struktur data dalam program. Sedangakan variabel sendiri merupakan wadah penyimpanan yang diberi nama untuk menampung nilai tertentu sesuai dengan tipe datanya (Pratiwi, 2020). Proses ini memungkinkan programmer untuk memberi tahu program tentang jenis data yang akan disimpan dan bagaimana data tersebut akan diakses dan dimanipulasi selama eksekusi program.

Saat mendeklarasikan variabel, kita memberikan pengenalan kepada komputer tentang apa yang diharapkan dari ruang penyimpanan yang diberikan. Misalnya, jika kita membuat variabel dengan nama "umur" dan menetapkannya sebagai integer, kita

memberitahu program bahwa variabel tersebut akan menyimpan nilai bilangan bulat. Pemilihan nama variabel yang deskriptif dan mendeklarasikannya dengan tipe data yang sesuai adalah praktik terbaik yang memudahkan pemahaman kode.

Variabel tidak hanya menyimpan nilai, tetapi juga menyediakan sarana untuk berinteraksi dengan dunia luar. Dalam konteks deklarasi I/O, variabel sering digunakan untuk menyimpan dan memproses data masukan dari pengguna atau menampilkan hasil keluaran dari program. Misalnya, ketika kita meminta pengguna memasukkan nama, kita dapat menggunakan variabel untuk menyimpan data tersebut dan kemudian menampilkannya dalam output. Selain itu, variabel dapat diinisialisasi dengan nilai awal atau diubah nilainya selama eksekusi program. Ini memberikan fleksibilitas dan dinamisme dalam memanipulasi data sesuai kebutuhan program.

Pemahaman yang kuat tentang tipe data dan variabel merupakan dasar penting dalam pengembangan program. Tipe data menentukan jenis nilai yang dapat disimpan dalam variabel, sementara variabel digunakan untuk menyimpan dan mengelola data dalam program. Dalam bahasa pemrograman C++, terdapat berbagai macam tipe data yang dapat digunakan sesuai dengan jenis nilai yang akan disimpan. Berikut adalah beberapa contoh penggunaan tipe data dan pendeklarasian variabel dalam C++.

Tabel 3.1. Penggunaan tipe data dan deklarasi variabel bahasa C++

No.	Tipe Data	Deklarasi
1.	**Bilangan Bulat (Integer)** Menyimpan bilangan bulat positif/negatif. Contoh : int	int angka = 24;
2.	**Desimal (Floating Point)** Menyimpan nilai desimal. Contoh : float dan double	float nilaif = 3.14f; double nilaid = 2.71828;
3.	**Karakter (Character)** Menyimpan satu karakter. Contoh : char	char karakter = 'A';
4.	**String**	#include <string>

	Menyimpan serangkaian karakter. Contoh : std::string	std::string kata = "Contoh kata";
5.	**Boolean** Menyimpan nilai kebenaran (true/false). Contoh: bool	bool periksa = true;
6.	**Array** Menyimpan sejumlah nilai bertipe data yang sama. Contoh: int[], double[], char[]	int array[] = {1, 2, 3, 4, 5};
7.	**Pointer** Menyimpan alamat memori dari variabel. Contoh: int*, double*	int nilai = 10; int* pointer = &nilai;
9.	**Enumeration (Enum)** Menyimpan satu nilai dari sekumpulan nilai yang telah ditentukan sebelumnya. Contoh: enum	enum Warna { MERAH, HIJAU, BIRU }; Warna warnaPilihan = MERAH;

Sumber : diolah penulis

Melalui contoh-contoh pada tabel 3.1, kita dapat melihat cara tipe data dan variabel digunakan untuk merepresentasikan berbagai jenis nilai dalam program C++. Setiap tipe data memiliki ukuran dan batasan tertentu. Pemilihan tipe data yang sesuai dengan jenis nilai yang akan disimpan sangat penting untuk memastikan efisiensi dan kejelasan dalam pengembangan program.

3.3 Operator dalam Deklarasi I/O

3.3.1 Penggunaan Operator dalam Operasi I/O

Operator adalah elemen kunci dalam pemrograman yang memungkinkan program melakukan operasi matematika, perbandingan, dan manipulasi data (Miftachurohmah et al., 2023). Dalam konteks deklarasi I/O, operator digunakan untuk

memodifikasi atau menggabungkan nilai-nilai yang terlibat dalam proses input dan output.

Operator dalam operasi I/O dapat memodifikasi data masukan atau mengatur format keluaran. Sebagai contoh, operator aritmatika dapat digunakan untuk menghitung total belanja dari sejumlah item yang dimasukkan pengguna. Dalam situasi ini, variabel yang mewakili harga barang dapat dijumlahkan dengan menggunakan operator penambahan (+), menciptakan hasil yang merepresentasikan total belanja.

Tidak hanya itu, operator juga dapat digunakan untuk menggabungkan atau memanipulasi string dalam konteks I/O. Misalnya, operator penggabungan string (+) dapat digunakan untuk menyatukan beberapa string menjadi satu kesatuan, yang mungkin berguna saat menyusun pesan keluaran yang informatif atau menarik.

3.3.2 Operator Aritmatika dalam Konteks I/O

Operator aritmatika berfungsi untuk melakukan perhitungan matematis pada data dalam operasi deklarasi I/O. Dalam kebanyakan kasus, kita menghadapi tugas-tugas seperti menghitung total belanja, rata-rata nilai, atau perhitungan lainnya. Operator aritmatika, seperti penambahan (+), pengurangan (-), perkalian (*), dan pembagian (/), memungkinkan kita untuk mengekspresikan perhitungan tersebut dengan jelas dan efisien. Misalnya, saat memproses pembayaran di program kasir, kita dapat menggunakan operator penambahan (+) untuk mengakumulasi total belanjaan dari beberapa item yang dimasukkan pengguna.

Selain itu, operator modulus (%) dapat bermanfaat dalam kasus tertentu, seperti memeriksa apakah suatu bilangan adalah bilangan genap atau ganjil. Penggunaan operator ini dapat memberikan fleksibilitas tambahan dalam mengelola data numerik yang terlibat dalam operasi deklarasi I/O (Huda et al., 2021). Melalui contoh konkret, pembaca akan mendapatkan wawasan tentang cara menggunakan operator aritmatika untuk mencapai perhitungan matematis yang dibutuhkan dalam berbagai skenario pemrograman.

3.3.3 Operator Perbandingan dan Logika untuk Kontrol Alur Program

Selain operator aritmatika, operator perbandingan dan logika juga berperan dalam mengendalikan alur program terutama dalam konteks deklarasi I/O. Operator perbandingan, seperti sama dengan '**==**', tidak sama dengan '**!=**', kurang dari '<', dan lebih dari '>', memungkinkan program membuat keputusan berdasarkan perbandingan antara nilai-nilai tertentu. Hal ini sering digunakan dalam pengambilan keputusan terkait dengan data masukan, seperti memeriksa apakah suatu nilai lebih besar dari batas tertentu atau apakah dua nilai sama.

Selain itu, operator logika, seperti && (dan), || (atau), dan ! (bukan/negasi), memungkinkan program menggabungkan kondisi-kondisi perbandingan untuk mengontrol jalannya eksekusi program (Hanief et al., 2020). Misalnya, dalam skenario deklarasi I/O, kita mungkin ingin membuat suatu kondisi yang memeriksa apakah dua nilai input memenuhi kriteria tertentu sebelum melanjutkan eksekusi program.

Hal ini memberikan fleksibilitas dalam mengatasi skenario kompleks dan memastikan bahwa program dapat menangani berbagai kondisi dengan efektif. Melalui skenario-skenario yang umum dalam pemrograman, sekiranya dipahami bagaimana operator-operator ini dapat membentuk struktur keputusan yang memandu program dalam mengelola data masukan dan menghasilkan output yang sesuai.

3.3.4 Operator String untuk Manipulasi Teks

Dalam operasi deklarasi I/O, manipulasi teks seringkali diperlukan untuk menyajikan informasi atau mengolah data dalam bentuk string. Operator string memungkinkan *programmer* untuk melakukan berbagai tugas, mulai dari penggabungan string hingga manipulasi karakter individual.

1. Operator Penggabungan String '+'
   Operator '+' memungkinkan penggabungan dua string atau lebih menjadi satu kesatuan. Misalnya, dalam menampilkan pesan kepada pengguna, kita dapat menggunakan operator

ini untuk menggabungkan teks statis dengan nilai variabel atau hasil perhitungan tertentu.

2. Operator Indeks [ ]
   Operator indeks dapat digunakan untuk mengakses karakter individual dalam sebuah string. Ini memungkinkan *programmer* untuk mengambil atau memanipulasi karakter pada posisi tertentu dalam string. Sebagai contoh, kita dapat menggunakan operator ini untuk mengganti atau menambahkan karakter pada posisi tertentu dalam sebuah kata.
3. Operasi Pemotongan atau Slicing String
   Operator ini memungkinkan untuk mengambil sebagian dari string. Misalnya, jika kita memiliki string tanggal dalam format tertentu, kita dapat memisahkan tanggal, bulan, dan tahun menjadi variabel-variabel terpisah.

Penerapan operator string ini dalam operasi deklarasi I/O dapat meningkatkan fleksibilitas program dalam menangani dan menyajikan informasi dalam bentuk teks. Melalui contoh penerapan praktis, bab ini akan memberikan wawasan kepada pembaca tentang cara efektif menggunakan operator string dalam konteks pemrograman.

3.3.5 Operator Penugasan untuk Manipulasi Variabel

Operator penugasan, seperti **'='**, memiliki peran penting dalam mengelola variabel dan memanipulasi nilai-nilai yang terlibat dalam operasi deklarasi I/O. Melalui operator ini, *programmer* dapat menginstruksikan program untuk memberikan nilai tertentu ke dalam variabel, menginisialisasi variabel, atau mengubah nilai variabel yang sudah ada. Berikut contoh dari penjelasan terkait manipulasi variabel.

1. Pendeklarasian dan Inisialisasi Variabel
   Operator penugasan sering digunakan saat pertama kali mendeklarasikan variabel dan memberikannya nilai awal. Misalnya, dalam menyimpan nilai masukan dari pengguna, variabel dapat dideklarasikan dan diinisialisasi menggunakan operator penugasan.

```
int umur = 25; //Deklarasi dan inisialisasi variabel umur dengan nilai 25.
```

2. Perubahan Nilai Variabel
   Operator penugasan juga berguna untuk mengubah nilai variabel selama eksekusi program. Misalnya, setelah menghitung total belanja, nilai variabel total dapat diubah menggunakan operator penugasan.

```
total = total + harga_barang; //Menambahkan harga_barang ke dalam nilai variabel total.
```

3. Penerapan Operator Penugasan pada String
   Dalam konteks string, operator penugasan dapat digunakan untuk mengganti nilai dari suatu variabel string atau menginisialisasi variabel string dengan nilai teks tertentu.

```
string nama = "Zumhur"; //Inisialisasi variabel nama".
nama = nama + " Alamin"; //Gabungkan string untuk menambahkan nama belakang.
```

Pemahaman yang baik tentang penggunaan operator penugasan meningkatkan efisiensi dan kejelasan kode dalam operasi deklarasi I/O. Bab ini akan mengilustrasikan berbagai cara operator penugasan dapat diterapkan dalam mengelola variabel.

3.3.6 Contoh Penggunaan Operator dalam Program I/O

Untuk memberikan gambaran nyata tentang bagaimana operator dapat diterapkan dalam operasi deklarasi I/O, berikut beberapa contoh penggunaan operator dalam berbagai skenario pemrograman.

Contoh 1: Operator Aritmatika dalam Perhitungan Total Belanja

```
int harga_barang = 50;
int jumlah_barang = 3;
int total = harga_barang * jumlah_barang; //Operator perkalian digunakan untuk menghitung total belanja.
```

Dalam contoh tersebut, operator perkalian digunakan untuk menghitung total belanja berdasarkan harga dan jumlah barang yang dimasukkan pengguna.

Contoh 2: Operator Perbandingan untuk Pengecekan Kondisi

```
int umur_pengguna = 17;
if (umur_pengguna >= 17) {
      cout << "Anda berhak untuk memiliki KTP." << endl;
} else {
     cout << "Anda harus berusia 17 atau lebih untuk memiliki KTP.";
     cout << endl;
}
```

Pada contoh tersebut, operator perbandingan digunakan untuk memeriksa apakah umur pengguna mencukupi syarat untuk memperoleh KTP (Kartu Tanda Penduduk).

Contoh 3: Operator String untuk Penggabungan Teks

```
string nama_depan = "Zumhur";
string nama_belakang = "Alamin";
string nama_lengkap = nama_depan + " " + nama_belakang;
//Operator penggabungan string untuk menyatukan nama depan
dan belakang.
```

Operator penggabungan string digunakan untuk menyatukan nama depan dan belakang dalam contoh tersebut.

Contoh 4: Operator Logika untuk Pengendalian Alur Program

```
int nilai_ujian = 75;
if (nilai_ujian >= 60 && nilai_ujian <= 100) {
   cout << "Anda lulus ujian." << endl;
} else {
   cout << "Anda tidak lulus ujian." << endl;
}
```

Dalam contoh tersebut, operator logika digunakan untuk mengendalikan alur program berdasarkan rentang nilai ujian.

Melalui contoh-contoh ini, pembaca dapat melihat bagaimana operator-operator tersebut berfungsi mengelola data masukan dan

keluaran dalam operasi deklarasi I/O. Penerapan yang cermat dan bijak terhadap operator ini memungkinkan *programmer* untuk menciptakan program yang efisien, responsif, dan sesuai dengan kebutuhan fungsionalitas yang diinginkan.

3.4 Operasi Input (I)

3.4.1 Pengenalan Operasi Input

Dalam konteks deklarasi I/O, pemahaman yang baik tentang cara melakukan operasi input menjadi kunci untuk memproses informasi yang diterima dari pengguna dengan benar. Operasi input merujuk pada mekanisme program untuk menerima data dari pengguna. Contohnya Dalam bahasa C++, operasi input umumnya melibatkan penggunaan baris kode khusus, seperti **std::cin** yang memungkinkan program membaca nilai atau data yang dimasukkan oleh pengguna.

Proses pengenalan operasi input melibatkan pemahaman tentang bagaimana variabel dideklarasikan dan diinisialisasi untuk menampung data yang akan dimasukkan (Buana et al., 2022). Selain itu, konsep validasi input juga menjadi hal penting untuk memastikan data yang dimasukkan sesuai dengan yang diharapkan oleh program. Pemahaman detil mengenai aliran eksekusi program dalam konteks operasi input juga perlu diperhatikan, termasuk urutan eksekusi instruksi dan bagaimana data disimpan dalam variabel.

Pembaca akan dibimbing untuk memahami signifikansi operasi input dalam konteks yang lebih umum, yakni bagaimana informasi yang diperoleh dari pengguna dapat berpengaruh pada jalannya program. Selain itu, perhatian khusus akan diberikan pada validasi input atau penanganan kesalahan input guna meningkatkan pemahaman pembaca terhadap operasi input.

3.4.2 Menerima Input dari Pengguna

Operasi input seringkali melibatkan penerimaan data dari pengguna melalui terminal atau antarmuka pengguna. Contoh sederhana dari operasi input khususnya untuk bahasa C++ adalah menggunakan **std::cin** untuk membaca nilai yang dimasukkan oleh pengguna. Dengan memahami bagaimana cara menerima input dari

pengguna, seorang *programmer* dapat menciptakan program yang responsif dan mampu beradaptasi dengan berbagai jenis masukan.

Dalam C++, **std::cin** digunakan untuk membaca data dari aliran masukan standar, yang biasanya terhubung dengan keyboard. Pada contoh berikut, program meminta pengguna untuk memasukkan suatu angka dan kemudian menampilkan nilai tersebut:

```cpp
#include <iostream>

int main() {
    int angka;
    std::cout << "Masukkan suatu angka: ";
    std::cin >> angka;
    std::cout << "Anda memasukkan angka: " << angka << std::endl;

    return 0;
}
```

Gambar 3.1a. Kode Program *input* Angka dengan C++
(Sumber : Diolah Penulis)

Kode program pada gambar 3.1a dapat disederhanakan dengan menggunakan "**using namespace std"** seperti pada gambar 3.1b berikut :

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

int main() {
    int angka;
    cout << "Masukkan suatu angka: ";
    cin >> angka;
    cout << "Anda memasukkan angka: " << angka << endl;

    return 0;
}
```

Gambar 3.1b. Kode Program *input* Angka disederhanakan
(Sumber : Diolah Penulis)

Dalam gambar 3.1a dan 3.1b, program meminta pengguna untuk memasukkan suatu angka, dan nilai tersebut kemudian disimpan dalam variabel angka menggunakan

operator >> pada std::cin. Nilai yang diterima kemudian dapat digunakan dalam berbagai operasi dalam program.

3.4.3 Menangani Tipe Data dan Kesalahan Input

Penting untuk memastikan bahwa input yang dimasukkan sesuai dengan tipe data yang diharapkan oleh program. Jika pengguna memasukkan tipe data yang tidak sesuai, hal ini dapat menyebabkan kesalahan atau perilaku yang tidak diinginkan. Oleh karena itu, sebaiknya program dilengkapi dengan penanganan kesalahan input, seperti pada contoh berikut:

```cpp
#include <iostream>
#include <limits> // Untuk std::numeric_limits
using namespace std;

int main() {
    int angka;

    cout << "Masukkan suatu angka: ";
    while (!(cin >> angka) || cin.fail()) {
        // Mengatur status input stream kembali ke normal
        cin.clear();
        // Membersihkan input buffer
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Input tidak valid. \nSilakan coba lagi: ";
    }

    cout << "Anda memasukkan angka: " << angka << endl;

    return 0;
}
```

Gambar 3.2. Kode Program validasi *input* Angka dengan C++
(Sumber : Diolah Penulis)

Pada contoh yang terdapat dalam gambar 3.2, program menggunakan loop while untuk terus meminta input sampai pengguna memasukkan nilai yang sesuai dengan tipe data yang diharapkan. Jika pengguna memasukkan nilai yang tidak valid, program akan memberikan pesan kesalahan dan meminta input ulang. Memahami teknik validasi dalam menerima input dari pengguna membantu menciptakan program yang tanggap dan dapat menangani berbagai skenario masukan dengan baik.

3.4.4 Input String dari Pengguna

Pemrosesan string dalam operasi input menjadi penting ketika kita perlu mengelola teks atau kata-kata yang dimasukkan oleh pengguna. Dalam bahasa pemrograman C++, string dapat dibaca menggunakan std::getline untuk menerima satu baris teks secara lengkap. Fungsi std::getline memungkinkan kita untuk membaca keseluruhan baris teks, termasuk spasi. Berikut ini adalah contoh penggunaan **std::**getline dalam bahasa C++ :

```cpp
#include <iostream>
#include <string>

int main() {
    std::string nama;

    std::cout << "Masukkan nama Anda: ";
    std::getline(std::cin, nama);

    std::cout << "Nama anda adalah : " << nama << std::endl;
    return 0;
}
```

Gambar 3.3. Kode Program *input* String dari Pengguna dengan C++ (Sumber : Diolah Penulis)

Pada gambar 3.3, std::getline (std::cin, teks) berfungsi membaca seluruh baris teks yang dimasukkan oleh pengguna dan menyimpannya dalam variabel string nama.

3.4.5 Validasi Input String

Pada saat yang sama, penting untuk memvalidasi input string agar sesuai dengan ekspektasi. Misalnya, memastikan bahwa string tidak kosong dan memiliki panjang tertentu.

```cpp
#include <iostream>
#include <string>

int main() {
    std::string nama;

    do {
        std::cout << "Masukkan nama Anda: ";
        std::getline(std::cin, nama);
    } while (nama.empty() || nama.length() > 50);

    std::cout << std::endl;
    std::cout << "Nama yang anda adalah : " << nama << std::endl;
    return 0;
}
```

Gambar 3.4. Validasi *input* String dari Pengguna dengan C++
Sumber : Diolah Penulis)

Dalam contoh yang terlihat pada gambar 3.4, program akan terus meminta input nama hingga pengguna memasukkan string yang valid atau sesuai dengan ketentuannya. Pemahaman tentang cara memproses input string dan memvalidasinya sangat berguna ketika mengembangkan program yang memerlukan pengelolaan teks atau kalimat dari pengguna.

Operasi input dalam deklarasi I/O dapat disimpulkan merupakan tahap penting dalam interaksi program dengan pengguna. Penerimaan input yang tepat dan validasi yang baik adalah langkah-langkah dasar untuk memastikan program dapat berfungsi dengan benar. Dengan memahami penggunaan std::cin, teknik pengamanan input, dan cara menerima serta memproses string dari pengguna, *programmer* dapat membangun program yang interaktif dan responsif.

3.5 Operasi Output (O)

3.5.1 Pengenalan Operasi Output

Operasi output adalah sebuah proses dalam pemrograman yang memungkinkan program untuk menampilkan informasi kepada pengguna atau menyimpan data ke berbagai sumber eksternal. Dalam konteks deklarasi I/O, operasi output berkaitan dengan proses menampilkan hasil atau data dari program ke luar.

Salah satu contoh operasi output dalam bahasa pemrograman C++ yaitu menggunakan perintah std::cout yang berfungsi untuk menampilkan output ke layar. Dengan std::cout, program dapat menampilkan teks, nilai variabel, atau hasil perhitungan ke terminal ataupun tampilan lainnya. Pemahaman operasi output menjadi penting dalam menampilkan informasi yang jelas dan terstruktur kepada pengguna.

3.5.2 Menampilkan Output ke Layar

Seperti yang dijelaskan sebelumnya, perintah std::cout berfungsi untuk menampilkan teks atau nilai ke layar dalam bahasa pemrograman C++. Penggunaan std::cout untuk menampilkan teks sederhana adalah hal yang umum. Contoh berikut menunjukkan cara menampilkan pesan sederhana ke layar:

```cpp
#include <iostream>

int main() {
    std::cout << "Selamat datang di program C++!" << std::endl;
    std::cout << "Saatnya eksplorasi dunia pemrograman bersama.";
    std::cout << std::endl;

    return 0;
}
```

Gambar 3.5. Hasil output sederhana ke layar dengan C++
(Sumber : Diolah Penulis)

Dalam contoh yang terlihat pada gambar 3.5, std::cout digunakan untuk menampilkan dua pesan berturut-turut ke layar, dipisahkan oleh << (operator penggabungan).

3.5.3 Menampilkan Output dengan Presisi

Pada saat menampilkan nilai desimal, pengaturan presisi sangat penting. std::cout memungkinkan kita untuk mengatur presisi desimal menggunakan manipulator **std::setprecision** dari header **<iomanip>**. Contohnya seperti berikut :

```cpp
#include <iostream>
#include <iomanip>

int main() {
    double nilaiDesimal = 3.1415926535;

    std::cout << "Nilai desimal default: " << nilaiDesimal;
    std::cout << std::endl;
    std::cout << std::setprecision(4);
    std::cout << "Nilai desimal dengan presisi 4: ";
    std::cout << nilaiDesimal << std::endl;

    return 0;
}
```

Gambar 3.6. Output dengan presisi menggunakan C++
(Sumber : Diolah Penulis)

Pada gambar 3.6 di atas, std::setprecision(4) digunakan untuk mengatur presisi desimal menjadi 4 digit.

3.5.4 Menampilkan Nilai Variabel

Salah satu kelebihan **std::cout** terletak pada kemampuannya menampilkan nilai variabel. Contoh berikut menunjukkan cara menampilkan nilai variabel ke layar:

```cpp
#include <iostream>

int main() {
    int angka = 42;
    double nilaiDesimal = 3.14;

    std::cout << "Nilai variabel angka: " << angka << std::endl;
    std::cout << "Nilai variabel nilaiDesimal: " << nilaiDesimal;
    std::cout << std::endl;

    return 0;
}
```

Gambar 3.7. Output nilai variabel dengan C++
(Sumber : Diolah Penulis)

Dalam contoh pada gambar 3.7, std::cout digunakan untuk menampilkan nilai variabel angka dan nilaiDesimal ke layar.

3.5.5 Menyusun Output Lebih Kompleks

Perintah **std::cout** dapat digunakan untuk menyusun output yang lebih kompleks dengan menggabungkan teks dan nilai variabel. Contohnya sebagai berikut :

```
#include <iostream>
#include <string>

int main() {
    std::string nama = "John";
    int usia = 25;

    std::cout << "Nama: " << nama << ", Usia: " << usia << " tahun." << std::endl;

    return 0;
}
```

Gambar 3.8. Output teks dan nilai variabel dengan C++
(Sumber : Diolah Penulis)

Dalam contoh yang terlihat pada gambar 3.8, std::cout digunakan untuk menyusun output yang mencakup teks dan nilai variabel nama dan usia.

Mengoptimalkan penyajian output dalam program C++ dapat dicapai dengan menggunakan std::cout, yang memungkinkan kita menghasilkan tampilan yang jelas dan informatif. Namun, untuk meraih tingkat keterbacaan yang lebih tinggi dan meningkatkan pengalaman pengguna, kita perlu memahami dan menerapkan pengaturan tampilan dan format output secara lebih mendalam.

Dalam bahasa C++, terdapat dua perintah yang berperan penting dalam meningkatkan estetika dan keterbacaan output, yaitu std::endl dan std::setw. Kedua perintah ini dapat digunakan bersamaan dengan std::cout untuk mengontrol tata letak dan format output. Pemahaman yang baik mengenai cara mengaplikasikan std::endl dan std::setw membantu mencapai hasil yang lebih terstruktur dan informatif pada layar.

3.5.6 Penggunaan std::endl untuk Pindah Baris

Perintah **std::endl** adalah kode dalam bahasa C++ yang digunakan untuk menambahkan baris baru ke output, serupa dengan menggunakan "**\n**". Ini membantu dalam merapikan tata letak output. **std::endl** digunakan untuk memberikan karakter pindah baris pada output. Hal ini berguna untuk membuat output lebih terstruktur, terutama ketika mencetak pesan-pesan berurutan. Contohnya sebagai berikut :

```
1 #include <iostream>
2
3 int main() {
4     std::cout << "Baris pertama" << std::endl;
5     std::cout << "Baris kedua" << std::endl;
6     std::cout << "Baris ketiga" << std::endl;
7
8     return 0;
9 }
```

Gambar 3.9. Pindah baris pada output dengan C++
(Sumber : Diolah Penulis)

Dalam contoh pada gambar 3.9, std::endl digunakan untuk menambahkan karakter pindah baris setelah setiap baris teks.

3.5.7 Penggunaan std::setw untuk Menetapkan Lebar Output

Perintah **std::setw** dari header **<iomanip>** merupakan kode dalam bahasa C++ yang digunakan untuk menetapkan lebar minimum untuk output saat menampilkan nilai variabel. Ini bermanfaat dalam menata letak output dengan rapi. Perintah ini berguna untuk menyusun output secara teratur, terutama saat menampilkan beberapa nilai dalam baris yang sama. Penggunaan perintah **std::setw** seperti contoh berikut :

```cpp
#include <iostream>
#include <iomanip>

int main() {
    int angka1 = 123;
    int angka2 = 456;

    std::cout << std::setw(8) << "Angka 1" << std::setw(8) << "Angka 2" << std::endl;
    std::cout << std::setw(8) << angka1 << std::setw(8) << angka2 << std::endl;

    return 0;
}
```

Gambar 3.10. Mengatur tata letak output dengan C++
(Sumber : Diolah Penulis)

Dalam contoh pada gambar 3.10, std::setw(8) digunakan untuk menetapkan lebar output menjadi 8 karakter untuk setiap nilai yang ditampilkan. Penerapan std::endl dan std::setw dapat membantu menciptakan tampilan output yang rapi.

3.5.8 Menampilkan Output Berdasarkan Input

Operasi output seringkali berhubungan erat dengan operasi input. Misalnya, menampilkan pesan berdasarkan input pengguna. Penggunaan std::cin dan std::cout memungkinkan program untuk menerima input dari pengguna dan segera memberikan output yang sesuai. Contoh berikut menunjukkan cara meminta pengguna untuk memasukkan suatu nilai, kemudian menampilkan nilai tersebut kembali:

```cpp
#include <iostream>

int main() {
    int angka;

    std::cout << "Masukkan suatu angka: ";
    std::cin >> angka;

    std::cout << "Anda memasukkan angka: " << angka << std::endl;

    return 0;
}
```

Gambar 3.11. Output berdasarkan Input dengan C++
(Sumber : Diolah Penulis)

Dalam contoh pada gambar 3.11, std::cin digunakan untuk membaca nilai yang dimasukkan oleh pengguna, dan std::cout digunakan untuk menampilkan nilai.

3.5.9 Memanfaatkan Loop dan Kondisi Bersama-sama

Penggunaan std::cin dan std::cout juga efektif dalam kombinasi dengan loop dan kondisi untuk menciptakan program yang dapat berinteraksi dengan pengguna. Berikut contoh menggunakan loop untuk meminta beberapa input:

```cpp
#include <iostream>

int main() {
    int totalAngka = 0;
    int angka;

    std::cout << "Masukkan beberapa angka (akhiri dengan 0): ";

    while (true) {
        std::cin >> angka;

        if (angka == 0) {
            break;
        }

        totalAngka += angka;
    }

    std::cout << "Total angka yang dimasukkan: " << totalAngka << std::endl;

    return 0;
}
```

Gambar 3.12. Kombinasi loop dan kondisi dengan C++
(Sumber : Diolah Penulis)

Dalam contoh pada gambar 3.12, program meminta pengguna untuk memasukkan beberapa angka hingga pengguna memasukkan angka 0. Program menggunakan loop while dan kondisi untuk terus membaca input pengguna. Penggunaan std::cin dan std::cout bersama-sama menciptakan kemungkinan untuk mengembangkan program yang dapat berkomunikasi secara interaktif dengan pengguna.

Operasi output adalah langkah penting dalam berkomunikasi dengan pengguna. Dengan memahami penggunaan std::cout,

std::endl, dan std::setw, *programmer* dapat menciptakan output yang jelas dan terstruktur. Melibatkan penggunaan output bersamaan dengan input memungkinkan program untuk memberikan respons yang dinamis dan sesuai dengan kebutuhan. Keseluruhan, pemahaman yang kuat tentang operasi output membantu membangun program yang interaktif, informatif, dan ramah pengguna.

DAFTAR PUSTAKA

Buana, I. K. S., Setiawan, H., & Putro, P. A. W. 2022. *Pemrograman Terstruktur*. Syiah Kuala University Press.

Chaniago, M. B., & Sastradipraja, C. K. 2022. *Buku Ajar Algoritma dan Struktur Data*. Kaizen Media Publishing.

Hanief, S., Jepriana, I. W., & Kom, S. 2020. *Konsep Algoritme dan Aplikasinya dalam Bahasa Pemrograman C++*. Penerbit Andi.

Harmayani, Apdilah, D., Mapilindo, Oktopanda, & Hutahaean, J. 2021. *Aplikasi Komputer*. Yayasan Kita Menulis.

Huda, A., Ardi, N., & Muabi, A. 2021. *Pengantar Coding Berbasis C/C++*. UNP PRESS.

Miftachurohmah, N., Erkamim, M., Purnawati, N. W., Halim, A., Bahana, R., & Irmawati, I. 2023. *BUKU AJAR PEMEROGRAMAN WEB II*. PT. Sonpedia Publishing Indonesia.

Nandita, E., & Yahfizham. 2023. Komparasi Stabilitas dan Efektifitas Phyton dengan C ++ Sebagai Algoritma Pemrograman Pemecahan Masalah pada Programmer Pemula. *Jurnal Arjuna: Publikasi Ilmu Pendidikan, Bahasa dan Matematika*, *1*(6).

Pratiwi, E. L. 2020. *Konsep Dasar Algoritma Dan Pemrograman Dengan Bahasa Java*. Poliban Press.

Sitorus, L. 2015. *Algoritma dan pemrograman*. Penerbit Andi.

BAB 4
PERCABANGAN

Oleh Rudiansyah

4.1 Pendahuluan

Dalam Pemrograman istilah percabangan merupakan alur program yang bercabang (Iqbal, 2023). Percabangan juga dikenal dengan istilah penyeleksian, *control flow, decision,* struktur kondisi, dan struktur if. Didalam percabangan, program memiliki satu bahkan lebih dari satu kondisi yang didalamnya ada sebuah instruksi yang dilaksanakan jika kondisi tertentu dapat terpenuhi. Hal ini memungkinkan program untuk membuat keputusan dan melakukan tindakan yang berbeda tergantung pada kondisi yang diberikan. Sebagai contoh, apabila suatu kondisi tersebut terpenuhi (*true*) maka program akan menjalankan blok kode program, sebaliknya apabila kondisi tersebut tidak terpenuhi (*false*) maka program tidak akan menjalankan blok kode program. Pernyataan penyeleksian yang mencari kebenaran disebut *conditional expression* atau harus berupa bilangan Boolean.

4.2 Jenis Percabangan

Jenis percabangan yang umum digunakan dalam pemrograman, diantaranya ;

4.2.1 Kondisi IF

Pernyataan "if " digunakan untuk mengevaluasi kondisi tertentu dan menjalankan blok kode hanya jika kondisi tersebut terpenuhi. Jika kondisi bernilai benar (*true*), blok kode dalam pernyataan "if" akan diekseksusi.

4.2.2 Kondisi "if-else"

Pernyataan "if-else" digunakan untuk mengevaluasi kondisi tertentu. Jika kondisi bernilai benar (*true*), blok kode dalam pernyataan "if" akan dieksekusi. Jika kondisi bernilai salah(*false*), blok kode dalam pernyataan "else" akan dieksekusi

4.2.3 Kondisi "if-else-if"

Pernyataan ini digunakan untuk mengevaluasi beberapa kondisi secara berurutan. Setiap kondisi dievaluasi secara berurutan dan blok kode yang terkait dengan kondisi yang cocok akan dieksekusi, jika tidak ada kondisi yang cocok, blok kode dalam bagian "else" akan dieksekusi (jika ada).

4.2.4 Kondisi "switch case"

Pernyataan "switch case" digunakan untuk memilih satu dari beberapa blok kode yang akan dieksekusi berdasarkan nilai suatu ekspresi. Program akan memeriksa nilai ekspresi dan menjalankan blok kode yang cocok dengan nilai tersebut. Percabangan memungkinkan program untuk mengambil keputusan dan melakukan tindakan yang berbeda tergantung pada kondisi yang ada. Ini sangat penting dalam mengendalikan alur program dan membuat program yang telah dinamis dan responsif terhadap input dan kondisi yang berbeda.

4.3 Percabangan Kondisi IF

Pernyataan if adalah bentuk pernyataan percabangan yang paling sederhana. Bentuk percabangan ini digunakan untuk memutuskan apakah suatu pernyataan atau blok pernyataan tertentu akan dieksekusi atau tidak, jika kondisi tertentu benar maka blok pernyataan dieksekusi, namun jika salah maka tidak akan dieksekusi. if hanya memiliki 1 blok pilihan yang akan dieksekusi jika kondisi bernilai benar.

```
if (kondisi) {
    // statement yang akan dieksekusi
    // apabila kondisi bernilai true
}
```

Bagian kondisi berperan sebagai penentu dari struktur percabangan. Jika kondisi terpenuhi (menghasilkan nilai *boolean true*), blok kode program akan dijalankan. Jika konidisi tidak terpenuhi (menghasilkan nilai boolean false), blok kode program

tidak akan dijalankan. Blok kode program yang di maksud adalah semua kode yang berada di antara tanda kurung kurawal "{" dan "}" (body).

Berikut ini adalah contoh kode program percabangan if dalam pemrograman C++:

```
#include <iostream>

using namespace std;

int main()
{
  int a = 4;
  int b = 5;

  if (a < b) {
    cout << "Nilai variabel a lebih kecil dari variabel b" << endl;
  }

  return 0;
}
```

4.4 Percabangan Kondisi IF-ELSE

Dalam pemrograman, struktur if-else adalah suatu mekanisme yang memungkinkan program untuk melakukan pemilihan kondisional. Dengan menggunakan struktur if-else, program dapat memutuskan untuk menjalankan satu blok kode jika suatu kondisi terpenuhi (*true*), dan menjalankan blok kode lainnya jika kondisi tersebut tidak terpenuhi (*false*).

```
if (kondisi) {
  // Kode program yang akan dijalankan jika kondisi berisi nilai True
}
else {
  // Kode program yang akan dijalankan jika kondisi berisi nilai False
}
```

Secara umum, struktur if-else terdiri dari dua blok kode yang berbeda, blok if dan blok else. Blok if berisi pernyataan atau serangkaian pernyataan yang akan dieksekusi jika kondisi yang diberikan benar atau terpenuhi. Jika kondisi pada blok if tidak

terpenuhi, maka program akan melanjutkan ke blok else (jika ada) dan menjalankan statement yang ada di dalamnya.

Berikut adalah contoh penggunaan struktur if-else dalam bahasa pemrograman C++:

```cpp
#include <iostream>

using namespace std;

int main() {
    int a;

    cout << "Masukkan sebuah angka : ";
    cin >> a;

    if (a > 0) {
        cout << "Angka yang Anda masukkan adalah positif." << endl;
    } else {
        cout << "Angka yang Anda masukkan bukanlah positif." << endl;
    }

    return 0;
}
```

Dalam contoh diatas, program akan meminta pengguna untuk memasukkan sebuah angka. Kemudian, menggunakan struktur if-else, program akan memeriksa apakah angka tersebut lebih besar dari 0. Jika iya, maka program akan mencetak "Angka yang Anda masukkan adalah positif." Jika tidak, program akan mencetak "Angka yang Anda masukkan bukanlah positif."

4.5 Percabangan Kondisi IF-ELSE-IF

Struktur if-else if adalah perluasan dari struktur if-else yang memungkinkan untuk mengevaluasi beberapa kondisi secara berurutan. Jika kondisi pertama tidak terpenuhi, program akan memeriksa kondisi kedua, dan seterusnya. Jika satu kondisi terpenuhi, blok kode yang sesuai akan dieksekusi, dan program akan keluar dari struktur if-else if tanpa mengevaluasi kondisi-kondisi berikutnya.

```
if (kondisi_1) {
  // Kode program yang dijalankan jika kondisi_1 berisi nilai True
} else if (kondisi_2) {
  // Kode program yang dijalankan jika kondisi_2 berisi nilai True
} else if (kondisi_3) {
  // Kode program yang dijalankan jika kondisi_3 berisi nilai True
} else  {
  // Kode program yang dijalankan jika semua kondisi tidak terpenuhi
}
```

Berikut adalah contoh penggunaan struktur if-else if dalam bahasa pemrograman C++:

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main() {
6       int a;
7
8       cout << "Masukkan nilai: ";
9       cin >> a;
10
11      if (a > 90) {
12          cout << "Nilai Anda A" << endl;
13      } else if (a > 80) {
14          cout << "Nilai Anda B" << endl;
15      } else if (a > 70) {
16          cout << "Nilai Anda C" << endl;
17      } else if (a > 60) {
18          cout << "Nilai Anda D" << endl;
19      } else {
20          cout << "Nilai Anda E" << endl;
21      }
22
23      return 0;
24  }
```

Dalam contoh di atas, program akan meminta pengguna untuk memasukkan sebuah nilai. Kemudian, menggunakan struktur if-else if, program akan mengevaluasi nilai tersebut berdasarkan kondisi yang disediakan. Jika nilai lebih besar dari 90, program akan mencetak "Nilai Anda A". Jika tidak, program akan memeriksa kondisi berikutnya, yaitu jika nilai lebih besar dari 80, dan seterusnya. Jika

tidak ada kondisi yang terpenuhi, program akan menjalankan blok else dan mencetak "Nilai Anda E". Dengan menggunakan struktur if-else if, pengguna dapat memeriksa beberapa kondisi secara berurutan dan menjalankan blok kode yang sesuai dengan kondisi yang terpenuhi pertama kali.

4.6 Percabangan SWITCH CASE

Switch case adalah struktur kontrol dalam pemrograman yang digunakan untuk memilih salah satu dari beberapa blok kode untuk dieksekusi berdasarkan nilai ekspresi yang diberikan. Dalam switch case, ekspresi tersebut dievaluasi, dan kemudian program akan mencocokkan nilai ekspresi dengan beberapa kemungkinan nilai yang mungkin terjadi.

Kondisi switch case terdiri dari 2 bagian, yakni perintah switch dimana terdapat nama variabel yang akan diperiksa, serta 1 atau lebih perintah case untuk setiap nilai yang akan diperiksa.

```
switch (nama_variabel) {
  case 'nilai_1':
    // Kode program yang dijalankan jika nama_variabel == nilai_1
    break;
  case 'nilai_2':
    // Kode program yang dijalankan jika nama_variabel == nilai_2
    break;
  case 'nilai_3':
    // Kode program yang dijalankan jika nama_variabel == nilai_3
    break;
  ...
  ...
  default:
    // Kode program yang dijalankan jika tidak ada kondisi yang terpenuhi
}
```

Di awal kode program, terdapat perintah switch untuk menginput variabel yang akan diperiksa. Kemudian terdapat beberapa perintah case yang diikuti dengan sebuah nilai. Jika isi dari nama_variabel sama dengan salah satu nilai ini, maka blok kode program akan dijalankan. Jika ternyata tidak ada kondisi case yang dipenuhi, blok default di baris paling bawah-lah yang akan dijalankan. Didalam setiap block case, diakhiri dengan perintah break agar struktur case langsung berhenti begitu kondisi terpenuhi.

Berikut adalah contoh penggunaan switch case dalam bahasa pemrograman C++:

```
1   #include <iostream>
2
3   using namespace std;
4
5   int main()
6   {
7     char a;
8
9     cout << "Input Nilai Anda (A - E): ";
10    cin >> a;
11
12    switch (a) {
13    case 'A':
14      cout << "Sangat Baik!" << endl;
15      break;
16    case 'B':
17      cout << "Baik" << endl;
18      break;
19    case 'C':
20      cout << "Cukup" << endl;
21      break;
22    case 'D':
23      cout << "Kurang / Tidak Lulus" << endl;
24      break;
25    case 'E':
26      cout << "Gagal / Tidak Lulus" << endl;
27      break;
28    default:
29      cout << "Format input tidak sesuai" << endl;
30    }
31
32    return 0;
33  }
```

Di baris 10 terdapat perintah untuk meminta user menginput salah satu huruf antara 'A' – 'E'. Nilai huruf ini kemudian disimpan ke dalam variabel nilai yang sudah di set ber tipe data char. Kondisi switch case dimulai pada baris 12. Di sini terdapat perintah switch (nilai) yang artinya kita ingin memeriksa isi dari variabel nilai. Seluruh block switch berada di dalam tanda kurung kurawal yang dimulai dari baris 12 sampai 30.

Di baris 13 terdapat perintah case 'A': Ini artinya jika variabel nilai berisi karakter 'A', maka jalankan isi dari block case, yakni

perintah cout << “Sangat Baik” << endl. Lalu terdapat perintah break di baris 15 agar struktur case lain tidak perlu di proses lagi. Setiap kasus di dalam switch case biasanya diakhiri dengan pernyataan break yang digunakan untuk keluar dari struktur switch dan mencegah program mengeksekusi kasus-kasus berikutnya. Jika break tidak ada, maka program akan melanjutkan mengeksekusi kasus-kasus berikutnya setelah kasus yang cocok ditemukan.

Di baris 16 terdapat perintah case kedua, yakni case ‘B’:. Sama seperti sebelumnya, blok ini akan dijalankan jika variabel nilai berisi huruf ‘B‘. Demikian seterusnya sampai case ‘E’ : di baris 25. Jika ternyata tidak ada nilai yang sesuai, maka block default di baris 28 yang akan di eksekusi.

Struktur switch case ini terlihat lebih rapi dan efisien daripada struktur if-else if. Namun switch case juga memiliki batasan, dimana tidak bisa dipakai untuk kondisi yang lebih kompleks seperti perbandingan dengan tanda lebih besar dari ” > “, maupun penggabungan kondisi. Struktur switch case ini hanya cocok untuk operasi perbandingan sederhana, dimana nilai yang diperiksa hanya terdiri dari nilai yang tetap.

DAFTAR PUSTAKA

Iqbal Ramadhani Mukhlis. 2023. *Buku Ajar Logika & Algoritma.* Jambi: PT Sonpedia Publishing Indonesia

BAB 5
PERULANGAN

Oleh Sugiarto Cokrowibowo

5.1 Perulangan

Decision atau pernyataan kondisional menggunakan *if, if-else, if-else if-else* dan *switch-case* yang telah dipelajari sebelumnya merupakan bagian mendasar dari sebuah program, namun masih terdapat fitur dasar lain dalam pemrograman yaitu perulangan(Davis, 2010). Istilah lain yang juga digunakan untuk merujuk proses perulangan pada pemrograman ini adalah: *loop, iteration* (iterasi), dan *repetition* (repetisi). Perulangan sebagaimana dapat dipahami dari namanya merupakan cara atau metode untuk mengeksekusi program secara berulang kali. Hal ini akan memungkinkan untuk membuat operasi yang lebih komplek dalam pemrograman (Quickly, 2020). Perulangan akan mengeksekusi operasi-operasi secara terus menerus selama nilai ekspresi pengendalinya (syarat kondisi perulangan) bernilai benar (Tselikis, 2023). Pada bab ini akan dibahas mekanisme perulangan pada bahasa pemrograman C++ menggunakan **for**, **while**, **do-while**, dan **range-base for**. Akan dibahas pula penggunaan perulangan bersarang (*nested loop*), perulangan tak hingga (*infinite loop*) dan pernyataan pengendali perulangan: **break**, **continue**, dan **goto**.

Tidak selamanya program yang ditulis akan bersifat unik dan berdiri sendiri. Sangat mungkin bahwa kita akan membutuhkan blok program yang akan dikerjakan secara berulang hingga pada kondisi tertentu terpenuhi. Itulah kegunaan perulangan dalam pemrograman (Malhotra & Malhotra, 2023). Mari kita perhatikan contoh program untuk mencetak tanggal pada bulan tertentu misalnya Januari yang terdiri dari 31 hari dari tanggal 1 sampai tanggal 31 berikut:

```
#include <iostream>
using namespace std;
int main( ){
    cout<<"Januari"<<endl;
    cout<<1<<" ";
    cout<<2<<" ";
    cout<<3<<" ";
    cout<<4<<" ";
    cout<<5<<" ";
    cout<<6<<" ";
    cout<<7<<" ";
    cout<<8<<" ";
    cout<<9<<" ";
    cout<<10<<" ";
    cout<<11<<" ";
    cout<<12<<" ";
    cout<<13<<" ";
    cout<<14<<" ";
    cout<<15<<" ";
    cout<<16<<" ";
    cout<<17<<" ";
    cout<<18<<" ";
    cout<<19<<" ";
    cout<<20<<" ";
    cout<<21<<" ";
    cout<<22<<" ";
    cout<<23<<" ";
    cout<<24<<" ";
    cout<<25<<" ";
    cout<<26<<" ";
    cout<<27<<" ";
    cout<<28<<" ";
    cout<<29<<" ";
    cout<<30<<" ";
    cout<<31<<" ";
    return 0;
}
```

Output dari program di atas adalah:

```
Januari
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
27 28 29 30 31
```

Program di atas melakukan operasi menampilkan tanggal beberapa kali dengan nilai tanggal dari 1 sampai 31. Operasi menampilkan tanggal dikerjakan secara berulang dengan kondisi tertentunya berupa nilai awal tanggal sama dengan 1 sampai tanggal sama dengan 31. Operasi berulang ini dapat dituliskan menjadi lebih singkat menggunakan perulangan (*loop*) sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    cout<<"Januari"<<endl;
    for(int i=1;i<=31;i++){
        cout<<i<<" ";
    }
    return 0;
}
```

Dengan output program yang sama penggunaan perulangan hanya memerlukan sedikit baris program. Mekanisme perulangan yang digunakan pada program diatas adalah perulangan menggunakan **for**. Selain menggunakan perulangan **for** pada C++ juga tersedia mekanisme perulangan menggunakan **while** dan **do-while** berikut:

Perulangan menggunakan **while**

```
#include <iostream>
using namespace std;
int main( ){
    cout<<"Januari"<<endl;
    int i = 1;
    while (i<=31){
        cout<<i++<<" ";
    }
```

```
    return 0;
}
```

Perulangan menggunakan **do-while**

```
#include <iostream>
using namespace std;
int main( ){
    cout<<"Januari"<<endl;
    int i = 1;
    do{
        cout<<i++<<" ";
    }while (i<=31);
    return 0;
}
```

Pada ketiga program di atas kita telah berhasil mempersingkat program untuk menuliskan tanggal di bulan Januari dari tanggal 1 sampai dengan tanggal 31. Dari program-program tersebut nampak pula bahwa perulangan **for**, **while**, dan **do-while** dapat saling menggantikan, namun pada penerapan lebih lanjut dibutuhkan intuisi programer untuk menentukan mekanisme perulangan yang mana yang akan dipilih dan digunakan untuk menyelesaikan program. Berikut ini beberapa pertimbangan yang mungkin digunakan untuk memilih mekanisme perulangan yang akan digunakan:

Tabel 4.1. Memilih mekanisme perulangan

No	Perulangan	Deskripsi	Situasi
1.	**for**	Perulangan **for** digunakan untuk mengeksekusi operasi berurutan secara berulang kali yang dikendalikan melalui *counter*	Perulangan **for** digunakan ketika kita memiliki variabel penghitung (*counter variable*) dan kita menginginkan untuk melakukan operasi perulangan berdasarkan penambahan/pengurangan nilai variabel pengitung tersebut pada rentang

No	Perulangan	Deskripsi	Situasi
		variable (*loop variable*)	tertentu, atau dengan kata lain perulangan **for** ini dipilih ketika kita tahu dengan pasti berapa kali operasi di dalam perulangan tersebut akan dieksekusi.
2.	**while**	Perulangan **while** digunakan untuk mengeksekusi operasi secara berulang selama syarat kondisi perulangannya terpenuhi. Syarat kondisi perulangan akan diperiksa di awal perulangan	Perulangan **while** dipilih ketika kita mengetahui syarat batas perulangan tetapi tidak mengetahui dengan pasti berapa kali operasi di dalam perulangan tersebut akan dieksekusi. Perulangan **while** akan melakukan pemeriksaan di bagian awal perulangan sehingga dapat digunakan untuk memastikan jika syarat kondisi perulangan tidak terpenuhi maka operasi tidak akan pernah dikerjakan.
3.	**do-while**	Perulangan **do-while** digunakan untuk mengeksekusi operasi secara berulang selama syarat kondisi perulangannya terpenuhi. Syarat kondisi	Perulangan **do-while** dipilih ketika kita mengetahui syarat batas perulangan tetapi tidak mengetahui dengan pasti berapa kali operasi di dalam perulangan tersebut akan dieksekusi dan kita hendak memastikan bahwa operasi akan dikerjakan paling sedikit satu kali meskipun syarat batas

No	Perulangan	Deskripsi	Situasi
		perulangan akan diperiksa di akhir perulangan, sehingga operasi akan dikerjakan paling sedikit satu kali meskipun syarat kondisi perulangannya tidak terpenuhi	perulangannya tidak terpenuhi.
4.	**range-base for**	**Range-based for** adalah mekanisme perulangan yang baru ditambahkan pada C++ 11. Perulangan ini memungkinkan kita untuk melakukan iterasi pada urutan data yang berada di dalam container data tanpa harus menentukan kondisi awal dan *increment*-nya	Perulangan **range-base for** dipilih jika kita akan mengakses urutan nilai yang ada pada kumpula data atau kontainer data seperti array dan vector.

Sumber: (Mueller, 2021)(Point, 2014)(Lei, 2021)

Selain keempat mekanisme perulangan di atas (**for**, **while**, **do-while**, dan **range-based for**) , pada C++ juga terdapat pernyataan pengontrol perulangan yaitu: **break**, **continue**, dan **goto**. Pengontrol perulangan ini memungkinkan program untuk melompat ataupun berhenti dari operasi yang sedang dikerjakan di dalam perulangan dengan menambahkan syarat kondisi tambahan.

5.2 Perulangan For

Kita telah sedikit membahas penggunaan perulangan **for** pada pengantar bab ini. Perulangan **for** dibuat menggunakan *keyword* **for** yang diikuti dengan tiga buah ekpresi di dalam tanda kurung yang berisi informasi tentang berapakali perulangan akan mengeksekusi operasi. Kita dapat penuliskan deklarasinya seperti berikut ini:

```
for(ekspresi1; ekspresi2; ekspresi1){ blok operasi }
```

dimana:

1. **Ekspresi1** sering juga disebut **Initializer** atau **initial expression**. Ekspresi ini hanya dieksekusi satu kali saja yaitu pada awal perulangan. Ekpresi1 ini digunakan untuk memberikan nilai awal pada *counter variable* (*loop variable*). *Counter variable* ini bertipe bilangan bulat (*int*) yang biasanya diberi nama menggunakan sebuah huruf kecil misalnya: i, j, m, n, x, y meskipun tidak menutup kemungkinan untuk menggunakan nama variabel lainnya.
2. **Ekspresi2** disebut juga dengan **Condition**. Bagian ini digunakan untuk memeriksa apakah perulangan akan diteruskan atau dihentikan. Jika kondisi pada ekpresi2 ini masih terpenuhi maka perulangan akan terus mengeksekusi operasi. gdsg
3. **Ekspresi3** disebut juga dengan **Finalizer** atau **Loop Expression**. Bagian ini berisi operasi untuk perubahan nilai *counter variable* yang dapat berupa operasi penambahan (*increment*) maupun pengurangan (*decrement*).
4. Blok operasi diletakkan diapit oleh tanda kurung kurawal {}. Blok operasi berisi operasi-operasi yang akan dieksekusi di

dalam perulangan. Jika hanya berisi satu baris operasi, maka kita dapat menuliskan operasi tersebut tanpa diapit menggunakan tanda kurung kurawal { } (Tselikis, 2023)(Mueller, 2021)(Grimes, 2017)(Kirch-Prinz & Prinz, 2002).

Selanjutnya kita akan berlatih menggunakan berbagai variasi penggunaan perulangan **for** untuk menambah pengalaman kita dalam membuat perulangan.

1. Menampilkan *counter variable* dari 0 sampai 9

   Ada banyak cara yang dapat dilakukan untuk menampilkan nilai dari *counter variable* dan berikut ini cara pertama

```
#include <iostream>
using namespace std;
int main( ){
    for(int i=0;i<10;i++){
        cout<<i<<" ";
    }
    return 0;
}
```

   Output dari program di atas adalah:

```
0 1 2 3 4 5 6 7 8 9
```

   Jika mengingat kembali deklarasi perulangan **for** pada program di atas maka dapat kita tuliskan menjadi sebagai berikut:

   Initializer: int i = 0 ini berarti bahwa kita membuat variabel **i** sebagai*counter variable* dan nilai awal untuk *counter variable* adalah sama dengan 0. Ini yang menyebabkan nilai **i** (*counter variable*) yang pertama dicetak adalah 0.

   Condition: i < 10 ini berarti bahwa selama nilai *counter variable* bernilai lebih kecil dari 10 maka perulangan akan terus dijalankan. Kondisi ini pula yang menyebabkan nilai *counter variable* yang dicetak hanya dari 0 sampai 9 dan tidak mencetak nilai 10, karena nilai 10 tidak lebih kecil dari 10 (tidak memenuhi syarat i < 10). Nah bagaimana jika kita hendak mencetak nilai *counter variable* dari 0 sampai 10?. Tenang, akan kita bahas di contoh selanjutnya.

   Finalizer: i++ bagian ini menyatakan perubahan nilai *counter variable* bahwa nilai *counter variable* kita berikutnya akan

bertambah 1 dari nilai *counter variable* saat ini (bertambah-tambah 1). Bagaimana jika kita hendak membuat perubahan nilai *counter variable* ini bertambah-tambah dua atau berkurang-kurang 3? Tenang, akan kita bahas pada contoh berikutnya

2. Menampilkan nilai *counter variable* dari 0 sampai 10
   Dengan melakukan sedikit perubahan pada bagian **condition** di contoh 1 maka kita dapat menyelesaian tugas ini sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    for(int i=0;i<=10;i++){
       cout<<i<<" ";
    }
    return 0;
}
```

   Output dari program di atas adalah:

```
0 1 2 3 4 5 6 7 8 9 10
```

   Yang kita lakukan adalah mengubah **condition** dari i<10 menjadi i<=10. Dengan condition i <= 10 ini akan mengakibatkan nilai i = 10 juga akan ikut tercetak.

3. Menampilkan nilai *counter variable* dari 0 sampai 10
   Masih menampilkan nilai 0 sampai 10 apa tidak salah?. Iya sudah seharusnya kita mempelajari dan mengetahui bukan hanya satu cara. Untuk menampilkan nilai *counter variable* dari 0 sampai 10 juga dapat kita lakukan dengan kode program berikut:

```
#include <iostream>
using namespace std;
int main( ){
    for(int i=0;i<11;i++){
       cout<<i<<" ";
    }
    return 0;
}
```

   Output dari program di atas adalah:

```
0 1 2 3 4 5 6 7 8 9 10
```

4. Menampilkan nilai *counter variable* dari 0 sampai 10
   Sampai disini keterampilan dan kesabaran kita benar-benar diuji. Masih adakah cara lain untuk menampilkan nilai dari 0 sampai 10?. Masih ada, bahkan masih ada banyak cara yang mungkin dapat anda gunakan.

```
#include <iostream>
using namespace std;
int main( ){
    for(int i=1;i<=11;i++){
       cout<<i-1<<" ";
    }
    return 0;
}
```

   Output dari program di atas adalah:

```
0 1 2 3 4 5 6 7 8 9 10
```

   Pada program di atas kita melakukan perubahan pada **initializer**, **condition** dan operasi yang sebelumnya mencetak nilai i menjadi mencetak nilai i-1 yaitu cout<<i-1<<" ";

5. Menampilkan nilai 0 2 4 6 8 10 12 14 16 18 20
   Kali ini kita akan menampilkan nilai dengan nilai awal sama dengan 0 dan nilai tersebut bertambah-tambah 2 sampai pada nilai sama dengan 20. Tugas ini dapat diselesaikan menggunakan kode program berikut:

```
#include <iostream>
using namespace std;
int main( ){
    for(int i=0;i<=20;i+=2){
       cout<<i<<" ";
    }
    return 0;
}
```

   Output dari program di atas adalah:

```
0 2 4 6 8 10 12 14 16 18 20
```

   Perhatikan bahwa pada bagian **finalizer** kita menggunakan operasi i+=2. Operasi ini akan mengakibatkan nilai *counter*

variable akan bertambah-tambah 2. Dengan analogi yang sama jika kita hendak membuat *counter variable* bertambah-tambah 3 maka dapat kita tuliskan dengan i+=3. Jika kita hendak membuat nilai *counter variable* bertambah-tambah 1 maka dapat kita tulis dengan i+=1 alih-alih menggunakan i++. Selain menggunakan operasi i+=2 operasi untuk menambah nilai *counter variable* dengan nilai 2 ini juga dapat ditulis dengan i = i+2. Seperti pada contoh program 6. Dengan analogi yang sama pula kita dapat membuat nilai *counter variable* berkurang satu atau berkurang dua dan seterusnya menggunakan operasi finalizer berikut: i--, i-=1, i-=2, i-=3, i-=4, i-=5, i=i-2, i=i-5, dll.

6. Menampilkan nilai 0 2 4 6 8 10 12 14 16 18 20
   Dengan operasi **finalizer** yang sedikit berbeda kita dapat menyelesaikan tugas ini dengan kode program berikut:

```
#include <iostream>
using namespace std;
int main( ){
    for(int i=0;i<=20;i=i+2){
       cout<<i<<" ";
    }
    return 0;
}
```

   Output dari program di atas adalah:

```
0 2 4 6 8 10 12 14 16 18 20
```

7. Menampilkan barisan 1 6 11 16 21 26 31 36 41 46 51
   Barisan pada tugas ini adalah barisan yang dimulai dengan nilai awal sama dengan 1 dan terus bertambah-tambah 5 di setiap nilai berikutnya dan berakhir di nilai 51. Maka kode programnya dapat dituliskan sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    for(int x=1;x<=51;x+=5){
       cout<<x<<" ";
    }
```

```
    return 0;
}
```

Output dari program di atas adalah:

```
1 6 11 16 21 26 31 36 41 46 51
```

Perhatikan bahwa pada kode program nomor 7 ini kita membuat *counter variable* **x** ini memberikan contoh kepada kita bahwa *counter variable* di dalam perulangan tidak selamanya harus dibuat dengan variabel **i**. Selain itu, *counter variable* ini juga dapat dideklarasikan sebelum memasuki perulangan seperti pada contoh program 8 berikut.

8. Menampilkan barisan 1 6 11 16 21 26 31 36 41 46 51

```
#include <iostream>
using namespace std;
int main( ){
    int x;
    for(x=1;x<=51;x+=5){
        cout<<x<<" ";
    }
    return 0;
}
```

Output dari program di atas adalah:

```
1 6 11 16 21 26 31 36 41 46 51
```

9. Menampilkan nilai 10 9 8 7 6 5 4 3 2 1

Pada tugas ini nilai *counter variable* yang dicetak dimulai dari nilai yang besar ke nilai yang kecil dari nilai sama dengan 10 sampai 1. Nilai yang ditampilkan berkurang-kurang 1. Tugas ini dapat diselesaikan dengan kode program berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int x;
    for(x=10;x>=1;x--){
        cout<<x<<" ";
    }
    return 0;
```

```
}
```

Output dari program di atas adalah:

```
10 9 8 7 6 5 4 3 2 1
```

10. Menampilkan nilai 10 9 8 7 6 5 4 3 2 1

Masih adakah cara lain untuk menampilkan nilai 10 9 8 7 6 5 4 3 2 1. Ya, masih ada banyak cara yang dapat dilakukan. Kita juga dapat menampilkan nilai 10 9 8 7 6 5 4 3 2 1 dengan cara berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int x;
    for(x=10;x>0;x--){
       cout<<x<<" ";
    }
    return 0;
}
```

Atau dapat juga ditampilkan dengan cara berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int x;
    for(x=10;x>0;x-=1){
       cout<<x<<" ";
    }
    return 0;
}
```

Atau dengan cara berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int x;
    for(x=10;x>0;x=x-1){
       cout<<x<<" ";
    }
    return 0;
}
```

11. Menampilkan barisan 21 18 15 12 9 6 3

    Nilai pada barisan ini dimulai dari nilai 21 sehingga kita dapat menggunakan **initializer** = 21, selanjutnya nilai terakhir yang ditampilkan adalah 3 dan semua nilai yang ditampilkan tersebut lebih besar atau sama dengan 3, maka kita dapat menggunakan **condition** >= 3. Terakhir untuk bagian **finalizer**, karena nilai tersebut berkurang-kurang 3 maka dapat kita gunakan **finalizer** i-=3 atau x-=3 tergantung dari nama *counter variable* yang kita gunakan.

```
#include <iostream>
using namespace std;
int main( ){
    int x;
    for(x=21;x>=3;x-=3){
       cout<<x<<" ";
    }
    return 0;
}
```

    Output dari program di atas adalah:

```
21 18 15 12 9 6 3
```

12. Menampilkan nilai 4 5 6 7 8 9 10 11 12 13 14 15

    Sebuah perulangan tidak harus selalu di mulai dari 0 atau 1 dan berakhir di nilai 10. Dari latihan yang telah kita kerjakan sebelumnya, asalkan kita dapat mendefenisikan **initializer**, **condition**, dan **finalizer** yang dibutuhkan, maka kita dapat membuat perulangannya. Disinilah intuisi programer dilatih. Untuk menyelesaikan tugas nomor 12 ini kita akan menggunakan initializer y=4, condition y<=15 dan finalizer y++. Maka source codenya akan menjadi seperti berikut ini:

```
#include <iostream>
using namespace std;
int main( ){
    for(int y=4;y<=15;y++){
       cout<<y<<" ";
    }
```

```
    return 0;
}
```

Output dari program di atas adalah:

```
4 5 6 7 8 9 10 11 12 13 14 15
```

Sampai pada latihan ini kita seharusnya sudah dapat memahami bahwa bagian terpenting pada perulangan **for** adalah kemampuan untuk menentukan **initializer**, **condition**, dan **finalizer**.

13. Menampilkan elemen-elemen array menggunakan perulangan **for**.

    Misalkan kita memiliki sebuah array berisi lima elemen berikut 3, 20, 8, 15, 10 kemudian kita diminta untuk menampilkan elemen-lemen array tersebut menggunakan perulangan for, maka kita dapat melakukannnya menggunakan kode program berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int arr[5] = { 3, 20, 8, 15, 10 };
    for (int i = 0; i < 5; i++) {
        cout << "arr[" << i << "] = " << arr[i] << '\n';
    }
    return 0;
}
```

Output dari program di atas adalah:

```
arr[0] = 3
arr[1] = 20
arr[2] = 8
arr[3] = 15
arr[4] = 10
```

penjelasan program: disini kita mendefinisikan array yang terdiri dari lima elemen. Sebagaimana kita ketahui bahwa indeks elemen array dimulai dari 0 dan indeks elemen terakhirnya adalah sebesar panjang array dikurang satu, jadi indeks elemen terakhir array kita ini adalah 4 berasal dari 5-1 (panjang array-1). Elemen array yang pertama bernilai 3 dan memiliki indeks elemen sama dengan 0 dan elemen array terakhir bernilai 10 dan memiliki

indeks elemen sama dengan 4. Nilai elemen-elemen array bisa berupa nilai apa saja asalkan berada dalam domain tipe data yang sesuai dengan defenisi array sehingga nilai ini bukanlah nilai yang teratur maka tidak dapat diwakili oleh *counter variable*. Akan tetapi, pada array, selain memiliki nilai elemen, array juga memiliki nilai indeks dan nilai ini teratur mulai dari 0 sampai panjang array-1 dalam kasus kita ini dari 0 sampai 4. Nilai indeks ini bertambah-tambah satu, maka nilai indeks ini dapat diwakili oleh *counter variable* dan kita formulasikan ke dalam perulangan for menjadi: for(int i=0;i<5;i++) atau dapat juga kita lakukan dengan for(int i=0;i<=4;i++). Pada tugas ini kita telah menampilkan indeks elemen array dan nilai dari masing-masing elemen array pada indeks yang bersesuaian (Dmitrović, 2023).

14. Menghitung hasil dari 1 + 2 + 3 + 4 + 5 + 6 menggunakan perulangan **for**.
    Untuk menghitung hasil penjumlahan seperti ini kita akan membutuhkan sebuah variabel untuk menyimpan hasil penjumlahan kita dapat memberi nama variabel tersebut dengan nama misalnya variabel **jum**. Variabel **jum** ini mula-mula kita beri nilai awal sama dengan 0. Kemudian kita melakukan perulangan untuk memperbaharui nilai variabel **jum** ini yaitu dengan cara menambahkannya nilai variabel **jum** sebelumnya dengan nilai *counter variable*. Operasi ini ditulis menjadi **jum = jum + i**. Nilai *counter variable* **i** dimulai dari 1 sampai 6 sesuai dengan tugas yang diberikan. Setelah perulangan selesai memperbaharui nilai variabel **jum** maka kita dapat mencetak hasilnya.

```
#include <iostream>
using namespace std;
int main( ){
    int jum = 0;
    for (int i = 1; i <= 6; i++) {
        jum = jum + i;
    }
    cout << jum;
    return 0;
}
```

Output dari program di atas adalah:

```
21
```

Sebagai informasi tambahan bahwa kita dapat mengganti operasi **jum = jum + i** menjadi **jum += i** untuk memperoleh hasil yang sama.

```
#include <iostream>
using namespace std;
int main( ){
    int jum = 0;
    for (int i = 1; i <= 6; i++) {
        jum += i;
    }
    cout << jum;
    return 0;
}
```

Dengan cara yang hampir sama sebenarnya dari sini kita dapat membuat operasi penjumlahan dari 1 + 2 + 3 + ... sampai +n

15. Menghitung hasil dari 1 + 2 + 3 + 4 + ... + n menggunakan perulangan **for**.
    Dengan cara yang hampir serupa dengan tugas nomor 14 kita cukup menambahkan sebuah variabel n bertipe **int** yang nilainya dapat kita atur misalnya n = 20 dan programnya menjadi sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int n;
    n = 20;
    int jum = 0;
    for (int i = 1; i <= n; i++) {
        jum += i;
    }
    cout << jum;
    return 0;
}
```

Kita juga dapat mengatur input nilai **n** dari input keyboard/console menggunakan perintah **cin** menjadi sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int n;
    cin >> n;
    int jum = 0;
    for (int i = 1; i <= n; i++) {
        jum += i;
    }
    cout << jum;
    return 0;
}
```

Saat dijalankan program ini membutuhkan input. Misalkan kita berikan input n = 76, maka output programnya akan menjadi seperti berikut ini:

```
76
2926
```

Artinya 1 + 2 + 3 + 4 + ... + 76 = 2926

16. Menghitung nilai n! (dibaca n faktorial)
    Nilai n! di dapatkan dengan menghitung perkalian 1 x 2 x 3 x 4 x 5 x ... x n yaitu perkalian dari satu sampai n. n! = 1 x 2 x ... x n. dengan operasi yang hampir serupa dengan tugas nomor 15 hanya saja disini kita menggunakan operasi perkalian. Untuk melakukan operasi perkalian semacam ini kita akan membutuhkan sebuah variabel untuk menyimpan hasil perkalian misalnya kita beri nama variabel **kal**. Kita akan memberi nilai awal untuk variabel **kal** ini mula-mulai bernilai sama dengan 1 (**kal = 1**). Ini berbeda dengan variabel jum kita tadi yang mula-mula kita beri nilai sama dengan nol, karena nilai identitas (nilai netral) dari operasi perkalian itu adalah 1 sedangkan nilai identitas untuk operasi penjumlahan adalah 0. Nilai apapun yang dikalikan dengan nilai 1 hasilnya adalah nilai itu sendiri, demikian pula pada penjumlahan nilai apapun yang ditambah dengan nilai 0 maka hasilnya adalah

nilai itu sendiri. Baik, dengan analagi yang sama maka kita dapat menyelesaikan tugas nomor 16 ini dengan kode program berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int n;
    cin >> n;
    int kal = 1;
    for (int i = 1; i <= n; i++) {
        kal *= i;
    }
    cout << kal;
    return 0;
}
```

Program ini membutuhkan nilai variabel **n** saat dijalankan. Misalkan kita memasukkan nilai variabel **n = 9** maka output programnya adalah sebagai berikut:

```
9
362880
```

17. Menghitung jumlah nilai elemen-elemen di dalam array menggunakan perulangan **for**.
    Misalkan kita memiliki sebuah array berisi lima elemen berikut 3, 20, 8, 15, 10 kemudian kita diminta untuk menghitung jumlah nilai elemen-lemen array tersebut menggunakan perulangan for. Operasi ini mirip dengan operasi menggunakan fungsi **sum( )** di excel. Dengan menggabungkan berbagai informasi yang telah kita peroleh dari tugas-tugas sebelumnya hanya saja disini yang dijumlahkan bukan lagi indeks atau nilai *counter variable* tetapi nilai yang tersimpan di setiap elemen elemen array, dengan kata lain yang sebenarnya kita cari adalah hasil penjumlahan 3 + 20 + 8 + 15 + 10. Pertama kita akan membutuhkan sebuah variabel misalnya kita beri nama variabel **sum**. Variabel **sum** ini akan kita gunakan untuk menyimpan hasil penjumlahan nilai elemen-elemen array. Selanjutnya kita akan memperbaharui nilai variabel **sum** ini di dalam perulangan **for** menggunakan operasi **sum += arr[i]**.

```
#include <iostream>
using namespace std;
int main( ){
    int arr[5] = { 3, 20, 8, 15, 10 };
    int sum = 0;
    for (int i = 0; i < 5; i++) {
        sum += arr[i];
    }
    cout << sum;
    return 0;
}
```

Output dari program di atas adalah:

```
56
```

18. Menampilkan bilangan genap dari 1 sampai 100.

    Bilangan genap adalah bilangan kelipatan 2 atau bulangan yang habis dibagi 2, atau dengan kata lain bilangan genap adalah bilangan yang jika dibagi dua sisanya sama dengan 0. Sebenarnya kita dapat menyelesaikan tugas ini menggunakan karakteristik dari perulangan **for** yaitu memulai dengan **initializer** = 2 dan selanjutnya menggunakan **finalizer** += 2, sedangkan untuk **condition** kita tetapkan **i <= 100** agar dapat menampilkan nilai sampai nilai ke 100.

```
#include <iostream>
using namespace std;
int main( ){
    for (int i = 2; i <=100; i+=2) {
        cout<< i <<" ";
    }
    return 0;
}
```

Dengan cara ini program akan menampilkan bilangan genap mulai dari 2 karena bilangan genap yang pertama (diantara 1 sampai 100) adalah 2 dan bilangan genap terakhir yang ditampilkan adalah 100.

```
2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46
48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88
90 92 94 96 98 100
```

Selain menggunakan cara di atas, kita dapat menyelesaikan tugas ini dengan menggunakan operasi pengendali (conditional) di dalam perulangan **for**, yaitu jika bilangan tersebut dibagi 2 sisanya adalah 0 maka bilangan tersebut adalah bilangan genap. Jika bilangan tersebut adalah bilangan genap maka kita tampilkan nilanya ke layar. Operasi sisa hasil bagi ini dilakukan menggunakan operator **modulo** yang ditulis dengan simbol **%** sebagai berikut (**i % 2 == 0**) dan kode programnya menjadi:

```
#include <iostream>
using namespace std;
int main( ){
    for (int i = 1; i <=100; i++) {
        if(i%2==0){
            cout<< i <<" ";
         }
    }
    return 0;
}
```

Perhatikan baik baik bagian **initializer**, **condition**, dan **finalizer** yang digunakan.

19. Memberikan label bilangan GENAP dan bilangan ganjil nilai dari 1 sampai 10 seperti berikut ini:

```
1 adalah bilangan GANJIL
2 adalah bilangan GENAP
3 adalah bilangan GANJIL
4 adalah bilangan GENAP
5 adalah bilangan GANJIL
6 adalah bilangan GENAP
7 adalah bilangan GANJIL
8 adalah bilangan GENAP
9 adalah bilangan GANJIL
10 adalah bilangan GENAP
```

Kita dapat memodifikasi jawaban tugas nomor 18 menggunakan operasi decison **if else** untuk menyelesaikan tugas ini menjadi sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    for (int i = 1; i <=10; i++) {
        if(i%2==0){
            cout<< i <<" adalah bilangan GENAP\n";
        }else{
            cout<< i <<" adalah bilangan GANJIL\n";
        }
    }
}
```

20. Buatlah program untuk memberikan label **FIZZ** dan **BUZZ** pada bilangan-bilangan dari 1 sampai n (nilai n dimasukkan melalui keyboard) dengan ketentuan jika bilangan tersebut adalah bilangan kelipatan 3 maka tambahkan label **FIZZ**, jika bilangan tersebut adalah bilangan kelipatan 5 maka tambahkan label **BUZZ**.

Tugas ini dapat diselesaikan dengan menambahkan sedikit modifikasi pada jawaban tugas nomor 19. Disini kita akan menggunakan operasi modulo 3 (**i%3 == 0**) dan operasi modulo 5 (**i%5 == 0**) sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int n;
    cin >> n;
    for (int i = 1; i <=n; i++) {
        cout<<i<<" ";
        if(i%3==0){
            cout<<"FIZZ";
        }
        if(i%5==0){
```

```
            cout<<"BUZZ";
        }
        cout<<endl;
    }
    return 0;
}
```

Perhatikan baik-baik penggunaan operasi conditional **if** di dalam perulangan ini. Disini kita tidak menggunakan **if else**. Output dari program di atas jika diberi input n = 20 adalah sebagai berikut:

```
20
1
2
3 FIZZ
4
5 BUZZ
6 FIZZ
7
8
9 FIZZ
10 BUZZ
11
12 FIZZ
13
14
15 FIZZBUZZ
16
17
18 FIZZ
19
20 BUZZ
```

5.3 Perulangan While

Kita telah berlatih menggunakan perulangan **for** di pembahasan sebelumnya. Kali ini kita akan berlatih menggunakan perulangan **while**. Perulangan **while** dideklarasikan menggunakan sebuah ekspersi **condition** sebagai berikut:

```
while(ekspresi) { blok operasi }
```

exspresi di dalam perulangan **while** ini selanjutnya kita sebut sebagai **condition** yang merupakan ekspresi logika ataupun nilai yang jika bernilai benar (**true**) maka perulangan **while** akan terus dijalankan (Naga, 2023)(Davis, 2014)(Gregoire, 2021). Bagian yang juga sangat penting dalam perulangan **while** dan **do-while** adalah memperbaharui nilai yang ada di ekpresi **condition**. Hal ini sangat penting dan membutuhkan kehati-hatian karena jika kita kurang teliti maka akan terjadi **infinite loop** atau kondisi dimana perulangan terus-menerus dijalankan tanpa henti karena ekspresi **condition** senantiasa bernilai true (Uzayr, 2022).

Penggunaan perulangan **while**, **for**, dan **do-while** sebenarnya dapat saling menggantikan, tentunya dengan perlakuan tertentu. Berikut ini kita akan mencoba berlatih menyelesaiakan persoalan yang beberapa diantaranya sebelumnya juga telah diselesaikan menggunakan perulangan **for**.

1. Menampilkan nilai dari 0 sampai 9 menggunakan perulangan **while**.

   Tugas ini dapat kita selesaikan menggunakan kode program berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int i = 0;
    while(i<=9){
        cout << i <<" ";
        i++;
    }
    return 0;
}
```

   Output dari program diatas adalah:

```
0 1 2 3 4 5 6 7 8 9
```

   Disini kita menggunakan perulangan **while** untuk menyelesaikan tugas yang sebelumnya dikerjakan menggunakan perulangan **for**. Sebenarnya asalkan kita dapat melihat mana bagian **initializer**,

condition, dan **finalizer** maka kita juga dapat menggunakan perulangan **while** ini layaknya menggunakan perulangan **for**. Pada perulangan **while** ini kita dapat memandang bagian **initialize** int i = 0 ditulis sebelum perylangan while, sedangkan bagian **condition** i <= 9 sama yaitu pada bagian expresi **condition** yang diapit oleh tanda kurung ( ) di perulangan **while**. Terakhir untuk bagian **finalizer i++** ditulis dibagian bawah sebelum penutup kurung kurawal }.

2. Menampilkan nilai dari 0 sampai 10 menggunakan perulangan **while**.
   Kita hanya perlu melakukan sedikit perubahan pada bagian ekspresi **condition** pada jawaban tugas nomor 1 menjadi sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int i = 0;
    while(i<=10){
        cout << i <<" ";
        i++;
    }
    return 0;
}
```

   Output dari program diatas adalah:

```
0 1 2 3 4 5 6 7 8 9 10
```

3. Menampilkan nilai 0 2 4 6 8 10 12 14 16 18 20
   Untuk menyelesaikan tugas nomor 3 ini kita hanya perlu mengubah bagian **condition** i <=20 dan bagian **finalizer** i+=2 dari jawaban tugas nomor 2 menjadi sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int i = 0;
    while(i<=20){
        cout << i <<" ";
```

```
        i+=2;
    }
    return 0;
}
```

Output dari program di atas adalah sebagai berikut:

```
0 2 4 6 8 10 12 14 16 18 20
```

4. Menampilkan barisan 1 6 11 16 21 26 31 36 41 46 51

   Untuk menyelesaikan tugas ini dapat dilakukan dengan mengubah **initializer** i = 1, **condition** i<=51, dan **finalizer** i+=5 dari jawaban tugas nomor 3 menjadi sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int i = 1;
    while(i<=51){
        cout << i <<" ";
        i+=5;
    }
    return 0;

}
```

   Output dari program di atas adalah:

```
1 6 11 16 21 26 31 36 41 46 51
```

5. Menampilkan barisan 10 9 8 7 6 5 4 3 2 1

   Untuk menyelesaikan tugas ini kita akan menggunakan **initializer** int i = 10, **condition** i>0 serta menerapkan operasi *decrement* pada bagian **finalizer** menggunakan **i--** atau **i-=1** berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int i = 10;
    while(i>0){
        cout << i <<" ";
        i--;
    }
```

```
    return 0;
}
```

Output dari program ini adalah sebagai berikut:

```
10 9 8 7 6 5 4 3 2 1
```

6. Menampilkan barisan 21 18 15 12 9 6 3

   Barisan dimulai dengan menampilkan angka 21 kemudian dikurang-kurang 3 dan angka terakhir yang ditampilkan adalah angka 3. Berarti **initializer** i=21, **condition** i>=3 dan **finalizer** i-=3

```
#include <iostream>
using namespace std;
int main( ){
    int i = 21;
    while(i>=3){
        cout << i <<" ";
        i-=3;
    }
    return 0;
}
```

Output dari program di atas adalah sebagai berikut:

```
21 18 15 12 9 6 3
```

7. Menampilkan nilai 4 5 6 7 8 9 10 11 12 13 14 15

   Telah diinformasikan sebelumnya bahwa sebuah perulangan tidak harus selalu di mulai dari 0, 1, 3 atau 14 dan berakhir di nilai 10. Asalkan kita dapat mendefenisikan initializer, condition, dan finalizer yang dibutuhkan, maka kita dapat membuat perulangannya. Disinilah intuisi programer kembali dilatih. Untuk menyelesaikan tugas nomor 7 ini kita akan menggunakan initializer y=4, condition y<=15 dan finalizer y++. Maka source codenya akan menjadi seperti berikut ini:

```
#include <iostream>
using namespace std;
int main( ){
    int y = 4;
    while(y<=15){
```

```
        cout << y <<" ";
        y++;
    }
  return 0;
}
```

Output dari program ini adalah sebagai berikut:

```
4 5 6 7 8 9 10 11 12 13 14 15
```

8. Menampilkan elemen-elemen array menggunakan perulangan **while**.

   Misalkan kita memiliki sebuah array berisi lima elemen berikut 3, 20, 8, 15, 10 kemudian kita diminta untuk menampilkan elemen-lemen array tersebut menggunakan perulangan **while**, maka kita dapat melakukannnya menggunakan kode program berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int arr[5] = { 3, 20, 8, 15, 10 };
    int i = 0;
    while(i<5){
        cout << "arr[" << i << "] = " << arr[i] << endl;
        i++;
    }
  return 0;
}
```

   Output dari program di atas adalah:

```
arr[0] = 3
arr[1] = 20
arr[2] = 8
arr[3] = 15
arr[4] = 10
```

9. Menghitung hasil dari 1 + 2 + 3 + 4 + ... + n menggunakan perulangan **while**.

   Kita telang menyelesaikan tugas ini menggunakan perulangan **for** sebelumnya. Kali ini kita akan menyelesaikannya menggunakan perulangan **while** menggunakan kode program berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int n;
    cin >> n;
    int jum = 0;
    int i = 1;
    while(i<=n) {
        jum += i;
        i++;
    }
    cout << jum;
    return 0;
}
```

jika diberi input n = 10 akan menghasilkan hasil penjumlahan sama dengan 55 yang berasal dari 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55.

Output dari program di atas adalah sebagai berikut:

```
10
55
```

10. Menghitung nilai n! (dibaca n faktorial) menggunakan perulangan **while**.

    Nilai n! diperoleh dari hasil perkalian 1 sampai n atau dapat dituliskan n! = 1 x 2 x ... x n. kita telah menyesesaikan masalah n faktorial ini sebelumnya menggunakan perulangan **for** kali ini kita akan menyelesaikannya menggunakan perulangan **while** menggunakan kode program berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int n;
    cin >> n;
    int kal = 1;
    int i = 1;
    while(i<=n){
        kal *= i;
```

```
        i++;
    }
    cout << kal;
    return 0;
}
```

Output dari program di atas jika diberi input n = 10 adalah sebagai berikut:

```
10
3628800
```

11. Menghitung jumlah nilai elemen-elemen di dalam array menggunakan perulangan **while**.
    Misalkan kita memiliki sebuah array berisi lima elemen berikut 3, 20, 8, 15, 10 kemudian kita diminta untuk menghitung jumlah nilai elemen-lemen array tersebut menggunakan perulangan **while**. Operasi ini mirip dengan operasi menggunakan fungsi sum( ) di excel. Jika ditulis dalam formula matematika sederhana dapat dituliskan $sum = \sum_{i=0}^{4} arr[i]$. Berikut ini kode program yang dapat kita gunakan:

```
#include <iostream>
using namespace std;
int main( ){
    int arr[5] = { 3, 20, 8, 15, 10 };
    int sum = 0;
    int i = 0;
    while (i < 5) {
        sum += arr[i];
        i++;
    }
    cout << sum;
    return 0;
}
```

Output dari program di atas adalah:

```
56
```

12. Memberi label bilangan genap dan ganjil pada bilangan asli dari 1 sampai n menggunakan perulangan **while**.

Untuk menyelesaikan tugas ini menggunakan perulangan **while** dapat dilakukan dengan cara berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int i, n;
    cin >> n;
    i = 1;
    while (i <=n) {
        if(i%2==0){
            cout<< i <<" adalah bilangan GENAP\n";
        }else{
            cout<< i <<" adalah bilangan GANJIL\n";
        }
        i++;
    }
}
```

Output dari program di atas jika diberi input n = 15 adalah sebagai berikut:

```
15
1 adalah bilangan GANJIL
2 adalah bilangan GENAP
3 adalah bilangan GANJIL
4 adalah bilangan GENAP
5 adalah bilangan GANJIL
6 adalah bilangan GENAP
7 adalah bilangan GANJIL
8 adalah bilangan GENAP
9 adalah bilangan GANJIL
10 adalah bilangan GENAP
11 adalah bilangan GANJIL
12 adalah bilangan GENAP
13 adalah bilangan GANJIL
14 adalah bilangan GENAP
15 adalah bilangan GANJIL
```

13. Membuat program untuk memberikan label FIZZ dan BUZZ pada bilangan-bilangan dari 1 sampai n (nilai n dimasukkan melalui keyboard) dengan ketentuan jika bilangan tersebut adalah bilangan kelipatan 3 maka tambahkan label FIZZ, jika bilangan tersebut adalah bilangan kelipatan 5 maka tambahkan label BUZZ menggunakan perulangan **while**.
Sebelumnya kita telah menyelesaikan tugas ini menggunakan perulangan **for**. Kali ini kita akan menyelesaikannya menggunakan perulangan **while** berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int n;
    cin >> n;
    int i = 1;
    while(i <=n) {
        cout<<i<<" ";
        if(i%3==0){
            cout<<"FIZZ";
        }
        if(i%5==0){
            cout<<"BUZZ";
        }
        cout<<endl;
        i++;
    }
    return 0;
}
```

Jika diberi input n = 22, program diatas akan menghasilkan output berikut:

```
22
1
2
3 FIZZ
4
5 BUZZ
6 FIZZ
```

```
7
8
9 FIZZ
10 BUZZ
11
12 FIZZ
13
14
15 FIZZBUZZ
16
17
18 FIZZ
19
20 BUZZ
21 FIZZ
22
```

5.4 Perulangan Do-While

Perulangan **do-while** mirip dengan perulangan **while** hanya saja pada perulangan **do-while** pemeriksaan **condition** dilakukan dibagian akhir sehingga pada perulangan **do-while** akan tetap mengeksekusi operasi di dalam blok perulangan minimal satu kali meskipun syarat kondisi perulangannya tidak terpenuhi. Berikut ini deklarasi perulangan **do-while**:

```
do{ blok operasi }while(ekspresi) ;
```

Berikut ini contoh perulangan **do-while** yang tetap mengerjakan satu operasi meskipun syarat kondisinya tidak terpenuhi.

```
#include <iostream>
using namespace std;
int main( ){
    int i = 23;
    do{
        cout << i;
        i++;
```

```
    }while(i<10);
     return 0;
}
```
Output dari program di atas adalah sebagai berikut:
```
23
```

Di bagian awal program tersebut kita menetapkan int i = 23 atau dapat kita tuliskan **initializer** int i = 23. Sedangkan **condition** i < 10. Jadi sejak awal dengan nilai i = 23 tentu syarat kondisi i < 10 tidak terpenuhi (bernilai false) namun perulangan **do-while** tetap mengerjakan satu kali operasi untuk menampilkan nilai i yaitu 23.

Selanjutnya kita akan berlatih menggunakan perulangan **do-while** untuk menghitung nilai dari n!. berikut ini kode program yang dapat kita gunakan:

```
#include <iostream>
using namespace std;
int main( ){
    int n;
    cin >> n;
    int kal = 1;
    int i = 1;
    do{
        kal *= i;
        i++;
    }while(i<=n);
    cout << kal;
    return 0;
}
```
Output dari program di atas jika diberi input n = 9 adalah:
```
9
362880
```

5.5 Perulangan Range-Based For

Perulangan **range-based for** (*range-based for loop*) mulai diperkenalkan sejak versi C++ 11. Pada bahasa pemrograman lain perulangan jenis ini biasa juga disebut dengan perulangan **for-each**. Perulangan **range-based for** digunakan untuk melakukan

penelusuran (iterasi) terhadap elemen-elemen yang ada pada tipe data *container* (tipe data *collection*) seperti array, list, dan vector(Davis, 2014).

Perulangan **range-based for** dideklarasikan menggunakan simbol titik dua (:) dengan cara berikut:

```
for (tipe_data value : values) { blok operasi }
```

ekspresi di dalam **range-based for** terdiri dari tiga bagian yaitu:

1. **Tipe_data** menyatakan tipe data dari variabel **value**. Tipe data ini harus sesuai dengan tipe data elemen-elemen yang tersimpan di dalam *container*/*collection*.
2. **Value** adalah variabel yang akan mewakili elemen-elemen yang ada di *container* nilai dari variabel **value** ini akan berubah seiring dengan berjalannya iterasi.
3. **Values** adalah nama variabel bertipe *container*. *Container* dapat berupa array, vector, list, deque, dan lain-lain yang merupakan tipe yang memiliki method begin( ) dan end( ) yang mengembalikan iterator (Gregoire, 2021)(Grimes, 2017).

Berikut ini beberapa contoh penggunaan perulangan **range-based for** pada tipe *container*.

1. Menampilkan nilai elemen-elemen array int menggunakan perulangan **range-based for**.
   Misalkan kita memiliki array integer bernama data yang berisi lima elemen berikut 3, 20, 8, 15, 10 maka kita dapat menerapkan perulangan **range-based for** untuk menampilkan nilai elemen-lemennya dengan cara berikut ini:

```
#include <iostream>
using namespace std;
int main( ){
    int data[5] = { 3, 20, 8, 15, 10 };
    cout<<"[";
    for(int elemen : data){
        cout << elemen << " ";
    }
```

```
    cout<<"]";
    return 0;
}
```

Output dari program di atas adalah sebagai berikut:

```
[3 20 8 15 10 ]
```

Penting juga untuk diingat bahwa fitur perulangan **range-based for** ini baru tersedia di C++ 11 sehingga kita harus menjalankannya di compiler yang mendukung C++ 11 atau anda dapat menjalankannya pada *online compiler* yang mendukung C++ 11. Pada C++ 20 terdapat juga vitur tambahan lagi yaitu kita dapat membuat **initializer** di dalan tanda kurung dengan code program berikut:

```
#include <iostream>
using namespace std;
int main( ){
    cout<<"[";
    for(int data[5] = { 3, 20, 8, 15, 10 };int elemen : data){
        cout << elemen << " ";
    }
    cout<<"]";
    return 0;
}
```

2. Menampilkan nilai elemen-elemen array string menggunakan perulangan **range-based for**.

   Misalkan kita memiliki array string bernama arrHari yang terdiri dari 7 elemen:

```
string arrHari[7] = {"ahad","senin","selasa","rabu","kamis","jumat","sabtu"}
```

   maka kita dapat menggunakan perulangan **range-based for** untuk menampilkan nilai elemen-elemennya sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    string arrHari[7] = {"ahad","senin","selasa","rabu","kamis","jumat","sabtu"};
    cout<<"[";
```

```
    for(string hari : arrHari){
        cout << hari << " ";
    }
    cout<<"]";
    return 0;
}
```

Output dari program di atas adalah sebagai berikut:

```
[ahad senin selasa rabu kamis jumat sabtu ]
```

3. Menampilkan nilai elemen-elemen vector string menggunakan perulangan **range-based for**.
   Kita akan menerapkan perulangan **range-based for** pada tipe vector. Misalkan kita memiliki vectorHari = {"ahad","senin","selasa","rabu","kamis","jumat","sabtu"}. Maka kita dapat membuat kode program berikut untuk menampilkan elemen-elemen vectorHari:

```
#include <iostream>
#include <vector>
using namespace std;
int main( ){
    vector<string> vectorHari = {"ahad","senin","selasa","rabu","kamis","jumat","sabtu"};
    cout<<"[";
    for(string hari : vectorHari){
        cout << hari << " ";
    }
    cout<<"]";
    return 0;
}
```

Output dari program di atas adalah sebagai berikut:

```
[ahad senin selasa rabu kamis jumat sabtu ]
```

4. Menghitung jumlah elemen-elemen (*sum*) pada *vector* bertipe *double* menggunakan **range-based for**.
   Misalkan kita memiliki vector bertipe double berikut:
   vector<double> data = {4.5, 3.8, 7.2, 5.4, 8.5}; dan kita akan menghitung jumlah nilai elemen-elemen yang ada di dalamnya

atau yang kita kenal dengan operasi sum maka pertama kita akan membuat sebuah variabel untuk menyimpan hasil penjumlahan tersebut misalnya kita beri nama variabel **sum**. Nilai variabel sum ini mula-mula kita inisialisasi dengan nilai sama dengan 0. Kemudian seiring berjalannya perulangan **range-based for** maka nilai sum ini kita perbaharui menggunakan operasi *increment* **sum += value**. Untuk menyelesaikan tugas ini dapat kita gunakan kode program berikut:

```
#include <iostream>
#include <vector>
using namespace std;
int main( ){
    vector<double> data = {4.5, 3.8, 7.2, 5.4, 8.5};
    double sum = 0;
    for(double value : data){
        sum += value;
    }
    cout<<sum;
    return 0;
}
```

Output program di atas adalah:

```
29.4
```

5.6 Loop Control Statements

Control Statement adalah pernyataan untuk mengendalikan perulangan dengan cara mengubah aliran normalnya. *Control statement* ini memungkinkan program untuk keluar atau melompat dari perulangan. Pada C++ tersedia tiga macam *control statement* yaitu: **break**, **continue**, dan **goto** (Malhotra & Malhotra, 2023)(Uzayr, 2022).

1. **Break** digunakan untuk menghentikan operasi yang sedang dikerjakan di dalam perulangan. Pernyataan **break** ini biasanya digunakan melalui mekanisme *decision* misalnya **if** dan **switch-case**. Jadi jika kondisi pada *decision* terpenuhi maka perulangan akan melakukan **break** dan keluar dari perulangan. Pernyataan **break** ini dapat digunakan pada

semua mekanisme perulangan (**for**, **while**, **do-while**, **range-based for**).

2. **Continue** digunakan untuk melompati sebuah operasi di dalam perulangan tetapi akan dilanjutkan untuk mengerjakan operasi berikutnya jika **condition** masih terpenuhi. Jadi tidak serta-merta berhenti dan keluar dari perulangan tetapi melanjutkan operasi perulangan setelah melompati sebuah operasi selama **condition** masih terpenuhi.
3. **Goto** digunakan untuk melewatkan perulangan dan menuju ke pernyataan yang diberi label. *Control statement* **goto** ini tidak disarankan untuk digunakan dalam pemrograman, karena membutuhkan ketelitian dan kehati-hatian dalam penggunaannya, namun jika kita mampu teliti dan berhati-hati dalam menulis program penggunaan **goto** ini juga dapat dilakukan.

Selanjutnya kita akan berlatih menggunakan *loop control statement* untuk menyelesaikan masalah pemrograman.

1. Program yang menerima input bertipe double terus menerus sebanyak 10 kali, tetapi akan berhenti ketika diberi input bilangan negatif.
   Masalah ini dapat diselesaikan menggunakan perulangan **for** dan di dalam perulangan **for** tersebut kita menambahkan pernyataan break jika input lebih kecil dari 0 sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    double input;
    for(int a=1;a<=10;a++){
        cin>>input;
        cout<<"input-"<<a<<" = "<<input<<endl;
        if(input<0){
            cout << "input negatif ==> KELUAR"<<endl;
            break;
        }
    }
```

```
    return 0;
}
```

Program ini akan memaksa operasi untuk keluar dari perulangan **for** jika diberi input bilangan negatif meskipun proses perulangannya belum selesai. Berikut ini contoh output programnya:

```
3.57
input-1 = 3.57
829.152
input-2 = 829.152
3.14
input-3 = 3.14
-22.5
input-4 = -22.5
input negatif ==> KELUAR
```

2. Program yang menerima input bertipe double terus menerus, tetapi akan berhenti ketika diberi input bilangan negatif.

   Sedikit berbeda dengan program sebelumnya, kali ini program akan menerima input terus menerus tiada henti sampai diberi input negatif. Disini kita membutuhkan intuisi programer. Kita tidak mengetahui dengan pasti berapa kali perulangan akan dikerjakan maka sebaiknya kita memilih menggunakan perulangan **while** atau **do-while**. Berikut ini kode program yang dapat kita gunakan:

```
#include <iostream>
using namespace std;
int main( ){
    double input;
    int a = 1;
    while(true){
        cin>>input;
        cout<<"input-"<<a<<" = "<<input<<endl;
        if(input<0){
            cout << "input negatif ==> KELUAR"<<endl;
            break;
```

```
        }
        a++;
    }
    return 0;
}
```

Program di atas ketika diberi input bilangan *double* positif akan kembali meminta input, tetapi akan berhenti ketika diberi input bilangan negatif.

```
12.34
input-1 = 12.34
56.2
input-2 = 56.2
4567.321
input-3 = 4567.32
3.27
input-4 = 3.27
-22.7
input-5 = -22.7
input negatif ==> KELUAR
```

3. Program yang dapat menampilkan bilangan bulat dari 1 sampai n yang diinput dari *keyboard* tetapi mengabaikan semua bilangan kelipatan 3 (bilangan kelipatan 3 tidak ditampilkan).

   Tugas ini dapat kita selesaikan menggunakan pernyataan **continue** sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int n;
    cin >> n;
    for(int c=1;c<=n;c++){
        if(c%3==0){
            continue;
        }
        cout<<c<<endl;
    }
```

```
    return 0;
}
```

4. Penggunaan **goto** untuk melompat keluar dari perulangan.
   Meskipun penggunaan **goto** tidak disarankan, tetapi mungkin saja suatu saat akan berguna dan dibutuhkan, jadi tetap layak untuk diketahui dan dipelajari. Penggunaan **goto** ini tidak disarankan karena sulit untuk dimaintenance dan dapat mengakibatkan *infinite loop* jika tidak ditangani dengan baik (Naga, 2023)(Lei, 2021)(Uzayr, 2022). Pernyataan **goto** digunakan untuk melompat keluar dari perulangan menuju ke operasi yang telah diberi *label*. Jadi penggunaan **goto** ini berpasangan dengan **label**. *Label* dideklarasikan menggunakan simbol titik dua (:). Berikut ini contoh penggunaan goto pada program:

```
#include <iostream>
using namespace std;
int main( ){
    int i,n;
    double sum = 0,average = 0, value;
    cout<<"Input panjang data: ";
    cin >> n;
    double data[n];
    for(i=0;i<n;i++){
        cout<<"data["<<i<<"]: ";
        cin>>value;
        if(value>=0){
            data[i]=value;
            sum += value;
        }else{
            goto hitung_average;
        }
    }

    tampilkan_data:
    cout<<"[";
```

```
    for(int j=0;j<n;j++){
        if(j>0){
            cout<<", ";
        }
        cout<<data[j];
    }
    cout<<"]"<<endl;

    hitung_average:
    average = sum / (i - 1);
    cout << "Average = " << average<<endl;

    return 0;
}
```

Program di atas secara normal akan menerima input panjang data yaitu variabel n, kemudian membuat array data dengan panjang n, selanjutnya program akan menerima input sebanyak n elemen data. Pada program tersebut juga terdapat dua buah label program yaitu: **tampilkan_data** dan **hitung_average:**. Jika kita memasukkan input bilangan positif untuk variabel value maka nilai dari variabel value tersebut dimasukkan ke elemen data ke-i selanjutnya nilai **sum** diperbaharui dan perulangan akan berjalan normal. Tetapi, ketika kita memasukkan input *value* dengan bilangan negatif maka perulangan akan melakukan lompatan (*jump*) ke program yang diberi label **hitung_average**. Lompatan ini akan mengakibatkan program tidak mengerjakan tahaman menampilkan isi elemen-elemen data.

Berikut ini output ketika program berjalan normal:

```
Input panjang data: 5
data[0]: 2
data[1]: 6.21
data[2]: 3.14
data[3]: 8.75
data[4]: 9.21
[2, 6.21, 3.14, 8.75, 9.21]
Average = 7.3275
```

Dan berikut ini ketika terjadi lompatan yang disebabkan oleh eksekusi pernyataan **goto**:

```
Input panjang data: 5
data[0]: 2
data[1]: 6.21
data[2]: 3.14
data[3]: -8.75
Average = 5.675
```

5.7 Bagaimana Jika

Dalam proses belajar pemrograman kita tidak berhenti hanya ketika program telah selesai ditulis, tidak menemukan *error* dan program berhasil dijalankan. Dibutuhkan kemampuan untuk melakukan analisis dan memprediksi terjadinya *error* sebelum *error* tersebut terjadi. Kemampuan ini dapat diperoleh melalui proses latihan terus menerus. Pada bagian ini kita akan membahas beberapa hal yang mungkin tidak biasa tetapi hal tersebut dapat saja terjadi di dalam penggunaan perulangan. Kita akan melakukan sedikit analisa menggunakan pertanyaan bagaimana jika?.

1. Bagaimana Jika kita menuliskan operasi loop tanpa menggunakan pasangan kurung kurawal?
   Hal ini dapat saja dilakukan jika operasi yang dilakukan di dalam loop hanya terdiri dari satu baris operasi saja. Berikut ini contoh loop yang tidak menggunakan kurung kurawal { }:
   Contoh-1

```
#include <iostream>
using namespace std;
int main( ){
    for(int i=0;i<10;i++)
        cout<<"Pemrograman "<<i<<endl;

    return 0;
}
```

   Contoh-2

```
#include <iostream>
using namespace std;
int main( ){
```

```
    int i=0;
    while(i<10)
        cout<<"Pemrograman "<<i++<<endl;
    return 0;
}
```

Hal yang penting untuk diingat bahwa ini dapat dilakukan jika operasi di dalam perulangan hanya terdiri dari satu baris saja.

2. Bagaimana jika kita hendak membuat perulangan di dalam perulangan?

   Perulangan di dalam perulangan juga dapat dibuat dan hal tersebut memang dimungkinkan jika dibutuhkan. Perulangan di dalam perulangan ini biasa disebut dengan **Perulangan Bersarang** atau **Nested Loop**. Berbagai jenis perulangan berbeda juga dapat diletakkan di dalam perulangan bersarang. Misalnya kita membuat perulangan **while** di dalam perulangan **for** dan sebagainya (Tselikis, 2023). Berikut ini contoh penggunaan perulangan bersarang:

   Contoh-1

```
#include <iostream>
using namespace std;
int main( ){
    int i, j;
    for(i = 0; i < 2; i++) {
        cout << " Loop_Satu ";
        for(j = i+1; j < 2; j++){
            cout << " Loop_Dua ";
        }
    }
    return 0;
}
```

   Contoh-2

```
#include <iostream>
using namespace std;
int main( ){
    int i = 0;
    while(i<10){
```

```
        cout << "A";
        for(int j=0;j<=i;j++){
            cout<<"B";
            int k = 0;
            do{
                cout<<"C";
                k++;
            }while(k<j);
        }
        i++;
        cout<<endl;
    }
    return 0;
}
```

3. Bagaimana jika kita menuliskan perulangan for menggunakan for(; ;){ }
   Penulisan perulangan dengan cara tersebut akan menghasilkan perulangan tak hingga (**Infinite Loop**). Perulangan ini dapat berjalan, tetapi tidak dapat berhenti secara normal. Pada perulangan **for** di atas tidak memiliki nilai **initializer**, **condition** maupun **finalizer**. Perulangan ini akan berputar terus-menerus tiada henti sampai program di-**break** secara paksa. Berikut ini contoh program **infinite loop**:

```
#include <iostream>
using namespace std;
int main( ){
    for(;;){
        cout<<"perulangan"<<endl;
    }
    return 0;
}
```

4. Bagaimana jika menuliskan perulangan for menggunakan for(; ;){ }
5. Bagaimana jika kita benuliskan perulangan **for** dengan cara berikut ini: int a=20;for(;a>10;a--){ }

Hal ini dapat dilakukan dan tidak mengakibatkan error. Kita dapat mengosongkan bagian **initializer** di dalam for asalkan telah kita kerjakan sebelumnya.

```
#include <iostream>
using namespace std;
int main( ){
    int a = 20;
    for(;a>10;a--){
        cout<<"perulangan-"<<a<<endl;
    }
    return 0;
}
```

Hal yang hampir serupa juga dapat kita lakukan dengan cara berikut:

```
#include <iostream>
using namespace std;
int main( ){
    for(int a=20;a>10;){
        cout<<"perulangan-"<<a<<endl;
        a--;
    }
    return 0;
}
```

Kita juga dapat melakukannya dengan cara berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int a = 20;
    for(;a>10;){
        cout<<"perulangan-"<<a<<endl;
        a--;
    }
    return 0;
}
```

6. Bagaimana jika kita menuliskan perulangan **for** dengan cara berikut: for(;**constant**;){ }.

Contoh penulisan lengkap dari kode program di atas misalkan kita menggunakan bilangan konstan = 3 adalah sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    for(;3;){
        cout<<"perulangan"<<endl;
    }
    return 0;
}
```

Program perulangan di atas akan menjadi **infinite loop**.

7. Bagaimana jika kita menuliskan perulangan **while** dengan cara berikut: while(**constant**)
   Serupa dengan kasus nomor 6 sebelumnya hal ini akan mengakibatkan terjadinya **infinite loop**. Contoh kode program **infinite loop** ini adalah sebagai berikut:

```
#include <iostream>
using namespace std;
int main( ){
    while(3){
        cout<<"perulangan"<<endl;
    }
    return 0;
}
```

8. Bagaimana jika kita menuliskan perulangan **while** dengan cara berikut: while(true){ }
   Hal ini tidak akan mengakibatkan error sintaks program tetapi akan terjadi **infinite loop** serupa dengan contoh kasus 8.

9. Dapatkah perulangan **for** menggantikan perulangan **while**?
   Perulangan **for** dapat menggantikan perulangan **while** jika pada perulangan **for** kita tidak mengisi bagian **initializer** dan **finalizer-nya**. Berikut ini contoh perulangan **for** yang ekuivalen dengan perulangan **while**:

```
for(; a >= 3;) ekuivalen dengan while(a >= 3)
```

10. Bagaimana jika kita menggunakan operator selain --, ++, +=2, -=2, +=3, -=3 pada bagian **finalizer** di perulangan **for**?
    Kita dapat saja menggunakan operator aritmatika yang lain hanya saja perlu diingat bahwa *counting variable* (*loop variable*) harus bertipe *int* (bilangan bulat). Selain itu perlu juga untuk diperhatikan bahwa kondisi berhenti dapat tercapai agar tidak terjadi **infinite loop**. Berikut ini contoh programnya:
    Contoh-1

```
#include <iostream>
using namespace std;
int main( ){
    for(int i=1;i<20;i*=2){
        cout<<"perulangan"<<endl;
    }
    return 0;
}
```

    Contoh-2

```
#include <iostream>
using namespace std;
int main( ){
    for(int i=20;i>0;i/=2){
        cout<<"perulangan"<<endl;
    }
    return 0;
}
```

    Program contoh-1 dan contoh-2 ini dapat berjalan dengan benar tetapi program pada contoh-3 dan contoh-4 berikut mengalami **infinite loop**.
    Contoh-3

```
#include <iostream>
using namespace std;
int main( ){
    for(int i=0;i<20;i*=2){
        cout<<"perulangan"<<endl;
    }
    return 0;
}
```

Contoh-4

```
#include <iostream>
using namespace std;
int main( ){
    for(int i=1;i<20;i/=2){
        cout<<"perulangan"<<endl;
    }
    return 0;
}
```

11. Bagaimana jika kita akan membuat operasi penjumlahan berikut 1 + 2 + 3 + ... + n = hasil dengan hanya memberikan inpun nilai variabel n.
    Kita dapat menyelesaikannya menggunakan perulangan for berikut ini:

```
#include <iostream>
using namespace std;
int main( ){
    int n;
   cin >> n;
   int sum = 0;
   for(int i=1;i<=n;i++){
        if(i>1){cout<<" + ";}
        cout<<i;
        sum += i;
    }
    cout << " = " << sum << endl;
    return 0;
}
```

    Output dari program di atas jika diberi input n = 9 adalah sebagai berikut:

```
9
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45
```

12. Bagaimana jika kita hendak membuat pola menggunakan perulangan:

Program dapat menerima input sebuah variabel **n** yang menyatakan banyaknya baris pola. Berikut ini contoh pasangan input dan pola yang dihasilkan.

Untuk n = 4

```
4
#
# #
# # #
# # # #
```

Untuk n = 2

```
2
#
# #
```

Tugas ini dapat diselesaikan menggunakan perulangan bersarang (*nested loop*) berikut:

```
#include <iostream>
using namespace std;
int main( ){
    int n;
    cin >> n;
    for(int i=0;i<n;i++){
        for(int j=0;j<=i;j++){
            cout << "# ";
        }
        cout << endl;
    }
    return 0;
}
```

DAFTAR PUSTAKA

Davis, S. R. 2010. *Beginning Programming with C++ For Dummies*. Wiley Publishing, Inc.

Davis, S. R. 2014. *C++ For Dummies® , 7th Edition*. John Wiley & Sons, Inc.

Dmitrović, S. 2023. Modern C++ for Absolute Beginners. In *Modern C++ for Absolute Beginners*. Apress. https://doi.org/10.1007/978-1-4842-9274-7

Gregoire, M. 2021. Professional C++. In *Professional C++*. John Wiley & Sons, Inc. https://doi.org/10.1002/9781119421276

Grimes, R. 2017. *Beginning C++ Programming*. Packt Publishing.

Kirch-Prinz, U., & Prinz, P. 2002. *Programming in C++: A Complete Guide to C++*. Jones and Bartlett Publishers.

Lei, H. 2021. *C++ Programming For Beginners An Objected Oriented Language*. Hong Lei.

Malhotra, D., & Malhotra, N. 2023. *C++ Programming Fundamentals*. Mercury Learning and inforMation. www.merclearning.com

Mueller, J. P. 2021. *C++ All-In-One For Dummies, 4th Edition*. John Wiley & Sons, Inc. www.wiley.com

Naga, D. 2023. *Programming with C and C++*. Independently Published.

Point, T. 2014. *Learn C++ Programming Language*. Tutorials Point. www.tutorialspoin.com

Quickly, C. 2020. Learn C++ Quickly A Complete Beginner's Guide to Learning C++, Even If You're New to Programming. In *Code Quickly*. http://codequickly.org/

Tselikis, G. S. 2023. *Introduction to C++*. CRC Press. https://doi.org/10.1201/b16577-5

Uzayr, S. bin. 2022. *Mastering C++ Programming Language*. CRC Press.

BAB 6
FILE I/O

Oleh Muhammad Akbar Yasin

6.1 Pendahuluan

Pada Bab 3 Deklarasi I/O, telah dibahas mengenai I/O secara umum dalam bahasa pemrograman C++. Pada bab kali ini kita akan fokus pada *file* I/O.

File I/O atau dikenal pula sebagai *input/output* berkas, merupakan bagian penting dalam pemrograman karena memungkinkan program untuk mengkomunikasikan informasi dengan perangkat penyimpanan eksternal seperti *hard drive*.

Beberapa poin utama tentang pentingnya input/output file adalah sebagai berikut:

1. **Penyimpanan jangka panjang**: File I/O menawarkan cara untuk menyimpan data secara permanen, bahkan ketika program tidak digunakan. Hal ini diperlukan untuk melindungi informasi penting dan menghindari kehilangan data.
2. **Berbagi informasi**: File I/O memungkinkan pertukaran data antara berbagai program. Beberapa program dapat mengakses dan memodifikasi file yang sama, sehingga memungkinkan komunikasi dan kolaborasi.
3. **Pengaturan (konfigurasi)**: Banyak aplikasi yang memanfaatkan file untuk menyimpan pengaturan konfigurasi. Input/output file memungkinkan program untuk mengakses konfigurasi ini pada saat startup dan menyimpan modifikasi apapun pada file, sehingga pengguna dapat mempersonalisasi aplikasi.
4. ***Logging* (Pencatatan untuk penelusuran masalah)**: File I/O sering digunakan untuk merekam informasi tentang bagaimana suatu program berjalan. Hal ini dapat sangat berguna untuk memecahkan masalah dan memeriksa perilaku program selama periode waktu tertentu.

Singkatnya, file I/O adalah bagian dasar pemrograman yang memungkinkan data disimpan, dibagikan dan dipertukarkan antara program dan penyimpanan eksternal. Hal ini memungkinkan pengembangan aplikasi yang fleksibel dan lebih bermanfaat dengan memungkinkannya berkomunikasi dengan data di luar memori saat program berjalan.

6.2 File I/O Dasar

Berbicara mengenai I/O file pada C++, kita akan menggunakan *header* <fstream>. *Header* <fstream> adalah bagian dari *Standard Template Library* (STL) yang menyediakan fungsi-fungsi untuk melakukan operasi *input* dan *output* file. Istilah "fstream" berasal dari "*file stream*", yang menunjukkan fungsinya dalam mengelola berkas sebagai aliran data.

Header <fstream> memiliki tiga kelas yaitu ifstream, ofstream dan fstream. Kelas ifstream (*Input File Stream*) digunakan untuk mengambil data dari file. Kelas ini diturunkan dari kelas istream yang memungkinkannya bekerja dengan operasi input yang sebanding dengan cin.

Kelas ofstream (*Output File Stream*) digunakan untuk menulis data ke file. Kelas ini berasal dari kelas ostream, yang memungkinkannya untuk bekerja dengan operasi keluaran yang sama dengan cout.

Yang terakhir adalah kelas <fstream>, kelas ini enggabungkan fungsionalitas ifstream dan ofstream sehingga dapat digunakan untuk operasi membaca maupun menulis file.

```C/C++
#include <iostream>
#include <fstream>

int main() {
  // Menulis text ke dalam file
  std::ofstream outputFile("contoh.txt"); // membuka file untuk ditulis

  if (outputFile.is_open()) { // mengecek jika file berhasil dibuka
    outputFile << "Hello, File I/O!\n"; // Write a line to the file
    outputFile.close(); // menutup file
    std::cout << "Data berhasil dituliskan ke file.\n";
  } else {
    std::cerr << "Error: tidak dapat membuka file untuk penulisan.\n";
    return 1; // Return kode error 1
  }

  // Membaca isi file
  std::ifstream inputFile("contoh.txt"); // membuka file untuk dibaca

  if (inputFile.is_open()) { // mengecek jika file berhasil dibuka
    std::string content;
    std::getline(inputFile, content); // membaca satu baris dari file

    inputFile.close(); // menutup file

    std::cout << "Membaca isi file: " << content << std::endl;
  } else {
    std::cerr << "Error: tidak dapat membuka file untuk dibaca.\n";
    return 1; // Return kode error 1
  }

  return 0; // Return kode error 0 jika berhasil
}
```

Gambar 6.1. Kode File I/O Sederhana

Gambar 6.1 merupakan contoh kode sederhana bagaimana file I/O dilakukan dalam C++. Kode pada gambar 6.1 melakukan tindakan berikut:

1. Membuat file bernama "contoh.txt" untuk penulisan (ofstream).
2. Memverifikasi apakah file telah berhasil dibuka.
3. Menulis teks "Halo, File I/O!" ke file.
4. Menutup file yang dibuka.
5. Membuka file yang sama (contoh.txt) untuk dibaca (ifstream).

6. Memverifikasi apakah file telah berhasil dibuka.
7. Membaca sebuah baris dari file dan mencetaknya ke konsol.
8. Menutup berkas yang sedang dibuka.

Dalam contoh tersebut, header <fstream> disertakan untuk memanfaatkan ofstream untuk menulis dan ifstream untuk membaca. Kelas ofstream dan ifstream menawarkan metode yang mudah untuk bekerja dengan file I/O.

6.3 Membaca File

Untuk membaca sebuah file, kita perlu membuat sebuah *instance* dari kelas ifstream dan memanggil metode open(). Contoh kode sederhana operasi *read file* dapat dilihat pada gambar 62.

Pada contoh kode gambar 6.2, objek inputFile dari kelas ifstream dibuat dan kemudian dibuka dengan menggunakan nama file "contoh.txt" sebagai argumen dari konstruktor. Fungsi is_open() digunakan untuk memverifikasi apakah file berhasil dibuka. Proses pembacaan konten dari file dapat dilakukan dengan operator *stream* “>>” atau dengan fungsi std::getLine untuk membaca dari baris ke baris. Setelah pembacaan selesai, kita memanggil metode close() untuk menutup file.

```C/C++
#include <iostream>
#include <fstream>

int main() {
  // Membuka file untuk dibaca
  std::ifstream inputFile("contoh.txt");

  if (inputFile.is_open()) {
    // File berhasil dibuka, lakukan operasi read dari sini
    std::string content;
    // Bisa menggunakan operator stream >>
    inputFile >> content;

    std::string anotherContent;
    // atau bisa juga menggunakan std::getLine untuk mendapatkan baris per
baris
    while (std::getline(inputFile, anotherContent)) {
      std::cout << anotherContent<< "\n";
    }

    inputFile.close(); // Close file di akhir
  } else {
    std::cerr << "Error opening the file for reading.\n";
    return 1;
  }

  return 0;
}
```

Gambar 6.2. Kode File I/O *Read* Sederhana

6.4 Menulis ke File

Untuk mengakses sebuah file untuk ditulis, kita membuat sebuah *instance* dari kelas ofstream dan memanggil metode open(). Contoh kode dapat dilihat pada gambar 6.3.

Pada contoh kode gambar 6.3, objek outputFile dari kelas ofstream dibuat dan kemudian dibuka dengan menggunakan nama file "contoh.txt" (sebagai argument konstruktor). Fungsi is_open() digunakan untuk mengecek apakah file berhasil dibuka. Proses penulisan dapat dilakukan dengan operator stream “<<”. Setelah penulisan selesai, seperti biasa metode close() dipanggil untuk menutup file.

```C/C++
#include <iostream>
#include <fstream>

int main() {
  // Membuka file untuk penulisan
  std::ofstream outputFile("contoh.txt");

  if (outputFile.is_open()) {
    // Jika file dapat dibuka, lakukan operasi penulisan di bagian ini
    outputFile << "Saya menuliskan ini dari sebuah kode.\n";
    outputFile << "Ini baris lainnya\n";

    outputFile.close(); // Tutup file setelah menulis
  } else {
    std::cerr << "Terjadi error saat membuka file.\n";
    return 1;
  }

  return 0;
}
```

Gambar 6.3. Kode File I/O *Write* Sederhana

6.5 Mode-mode pada File

Mode berkas dalam C++ berkaitan dengan berbagai metode untuk membuka dan mengakses berkas. Ketika Anda mengakses sebuah file, Anda memilih sebuah mode yang mengindikasikan apakah Anda berencana untuk membaca, menulis, menambah, atau melakukan tindakan lain pada file tersebut.

Tabel 6.1. Mode File I/O

in	Membuka file untuk dibaca (mode default pada ifstream)
out	Membuka file untuk ditulis (mode default pada ofstream)
binary	Membuka file 126inary (non-text)
app	Membuka file untuk penulisan di bagian akhir konten yang sudah ada (*append*)
ate	Membuka file untuk penulisan atau pembacaan di bagian akhir konten yang sudah ada
trunc	Menghapus konten saat penulisan dilakukan

beg	Membuka file dari awal untuk pencarian, mode digunakan saat menggunakan fungsi seekg atau seekp
end	Membuka file dari akhir untuk pencarian, mode digunakan saat menggunakan fungsi seekg atau seekp
nocreate	Membuka file hanya jika file telah tersedia
noreplace	Membuka file hanya jika file belum tersedia

Sumber: C++ for Dummies (2004)

Pada contoh-contoh kode sebelumnya kita menggunakan mode bawaan (*default*) dari kelas ifstream yaitu mode in untuk membaca sedangkan mode default dari ofstream yakni out untuk menulis file sehingga secara tidak sadar sebenarnya mode file telah kita pilih.

Tabel 6.2. Mode default File I/O

Class	**Mode**
ifstream	ios::in
ofstream	ios:out
fstream	ios::in | ios::out

Sumber: C++ Primer Plus (2012)

Mode-mode file yang tersedia dapat dikombinasikan dengan menggunakan operator bitwise ("|"). Sebagai contoh, mengakses file untuk dibaca dan ditulis dalam mode binari, perhatikan contoh kode pada gambar 6.4, sebuah instance dari kelas fstream diwakili oleh variable fileStream untuk membuka file "contoh.bin" dengan 3 mode sekaligus yaitu ios::in, ios::out dan ios::binary yang artinya kita dapat membaca dan menulis ke file binari.

Sangatlah penting untuk memilih mode berkas yang sesuai dengan tindakan yang Anda inginkan dan menangani file yang dibuka dengan hati-hati dengan memeriksa masalah setelah mencoba membuka berkas.

```
C/C++
// menggunakan 3 mode sekaligus
cpp std::fstream fileStream("contoh.bin", std::ios::in | std::ios::out |
std::ios::binary)
```

Gambar 6.4. Mode File Binari

6.6 Posisi dan Pencarian pada File

Dalam C++, fungsi `seekg()` dan `seekp()` digunakan untuk memanipulasi posisi (pointer) berkas dalam *stream*. Fungsi-fungsi ini merupakan bagian dari header `<fstream>` dan dapat digunakan pada kelas `ifstream` maupun `ofstream`. Fungsi-fungsi ini memungkinkan Anda untuk mengatur posisi dimana operasi baca (`seekg()`) atau tulis (`seekp()`) yang berikutnya akan dilakukan.

```
C/C++
#include <iostream>
#include <fstream>

int main() {
  std::ifstream inputFile("contoh.txt", std::ios::in);

  if (inputFile.is_open()) {
    // Menempatkan pointer get pada byte ke-5 dari awal file
    inputFile.seekg(5, std::ios::beg);

    // Memulai pembacaan dari posisi yang ditetapkan
    std::string content;
    inputFile >> content;
    std::cout << "Membaca file: " << content << std::endl;

    inputFile.close();
  } else {
    std::cerr << "Error saat membuka file.\n";
    return 1;
  }

  return 0;
}
```

Gambar 6.5. Contoh kode seekg()

6.6.1 seekg()

Fungsi `seekg()` digunakan untuk mengatur posisi penunjuk (*pointer*) "get" yang digunakan untuk operasi pembacaan. Fungsi ini memiliki beberapa kelebihan tetapi penggunaan yang umum adalah menentukan *offset* dari titik referensi (biasanya awal file) dan arah dari titik referensi tersebut.

Perhatikan contoh kode pada gambar 6.5 `seekg(5, std::ios::beg)` memindahkan penunjuk get ke byte ke-5 dari awal berkas (`std::ios::beg` adalah titik referensinya).

6.6.2 seekp()

Fungsi `seekp()` digunakan untuk mengatur posisi penunjuk "put" yang digunakan untuk operasi penulisan. Mirip dengan `seekg()`, fungsi ini memungkinkan Anda untuk menentukan *offset* dan titik referensi.

```C/C++
#include <iostream>
#include <fstream>

int main() {
  std::ofstream outputFile("contoh.txt", std::ios::out | std::ios::app);

  if (outputFile.is_open()) {
    // Menempatkan pointer pada akhir file
    outputFile.seekp(0, std::ios::end);

    // Memulai proses penulisan dari posisi yang ditetapkan
    outputFile << "\nIni adalah data baru yang ditempatkan di akhir.";

    outputFile.close();
    std::cout << "Berhasil menambahkan data di akhir file.\n";
  } else {
    std::cerr << "Error saat membuka file.\n";
    return 1;
  }

  return 0;
}
```

Gambar 6.6. Contoh kode seekp()

Perhatikan contoh kode pada gambar 6.6, seekp(0, std::ios::end)` memindahkan penunjuk put ke bagian akhir berkas

(`std::ios::end` adalah titik referensinya). Hal ini memungkinkan Anda untuk menambahkan data baru pada akhir file.

6.6.3 Penggunaan Gabungan antara seekg() dan seekp()

Jika Anda bekerja dengan `fstream` (*stream* yang mendukung pembacaan dan penulisan), Anda dapat menggunakan `seekg()` dan `seekp()` untuk memindahkan *pointer* get dan put secara bersamaan.

```
C/C++
#include <iostream>
#include <fstream>

int main() {
  std::fstream file("contoh.txt", std::ios::in | std::ios::out);

  if (file.is_open()) {
    // Memindahkan pointer get and put pada byte ke-10 dari awal file
    file.seekg(10, std::ios::beg);
    file.seekp(10, std::ios::beg);

    //  Memulai operasi read dan write dari posisi yang ditetapkan
    std::string content;
    file >> content;

    outputFile << "\nIni adalah data baru yang ditambahkan.";

    file.close();
  } else {
    std::cerr << "Error saat membuka file.\n";
    return 1;
  }

  return 0;
}
```

Gambar 6.7. Contoh Kode Kombinasi seekg() dan seekp()

Perhatikan contoh kode pada gambar 6.7. baik `seekg()` maupun `seekp()` digunakan untuk mengatur posisi *pointer* get dan put ke byte ke-10 dari awal berkas. Hal ini memungkinkan Anda untuk melakukan operasi pembacaan dan penulisan dari posisi yang sama.

Perlu diperhatikan bahwa *offset* dapat bernilai positif atau negatif, dan titik referensi (`std::ios::beg`, `std::ios::cur`, atau `std::ios::end`) menentukan titik awal perhitungan offset.

Penggunaan yang tepat dari `seekg()` dan `seekp()` sangat penting untuk pemosisian dan manipulasi file yang tepat.

6.7 File I/O Berkas Biner

I/O berkas biner mengacu pada proses pembacaan dan penulisan ke file yang mempertahankan representasi biner yang tepat dari data. Tidak seperti I/O file teks yang berhubungan dengan karakter yang dapat dibaca manusia, I/O file biner digunakan untuk menangani data non-tekstual seperti gambar, audio, video, dan jenis informasi biner mentah lainnya.

6.7.1 Karakteristik File Biner

Berikut adalah karakteristik file biner:

1. **Representasi dari data mentah**, I/O file biner memungkinkan Anda bekerja dengan data mentah pada tingkat byte. Ini sangat berguna untuk menyimpan dan mengambil struktur data yang kompleks, format data khusus mesin, dan informasi apa pun yang tidak secara alami direpresentasikan sebagai teks
2. **Tanpa pengkodean karakter**, tidak seperti file teks, file biner tidak melibatkan pengkodean karakter seperti ASCII atau Unicode. Data disimpan dalam bentuk biner mentah, yang tidak dimaksudkan untuk dapat dibaca oleh manusia secara langsung.
3. **Penyimpanan yang ringkas**, format file biner sering kali lebih ringkas daripada format teks karena tidak memerlukan karakter tambahan untuk pemformatan, pembatas, atau baris baru.

```
C/C++
#include <iostream>
#include <fstream>

struct Data {
  int id;
  double value;
  char name[20];
};

int main() {
  std::ofstream binaryOutputFile("binary_data.dat", std::ios::out |
std::ios::binary);

  if (binaryOutputFile.is_open()) {
    Data dataWrite = {1, 3.14, "BinaryData"};
    binaryOutputFile.write(reinterpret_cast<char*>(&dataWrite),
sizeof(dataWrite));
    binaryOutputFile.close();
  }

  std::ifstream binaryInputFile("binary_data.dat", std::ios::in |
std::ios::binary);

  if (binaryInputFile.is_open()) {
    Data dataRead;
    binaryInputFile.read(reinterpret_cast<char*>(&dataRead),
sizeof(dataRead));
    // Menampilkan isi dari dataRead
    std::cout << "Read from file: " << dataRead.id << " | " << dataRead.value << "
| " << dataRead.name << "\n";

    binaryInputFile.close();
    std::cout << "Binary data read from file.\n";
  }

  return 0;
}
```

Gambar 6.8. *Read Write* File Binari

6.7.2 Cara Melakukan I/O pada File Binari

Dalam C++, header `<fstream>` yang sama digunakan untuk I/O file binari namun ketika membuka file untuk operasi binari, Anda perlu menyertakan flag mode `std::ios::binary`. Hal ini memastikan bahwa berkas tersebut diperlakukan sebagai berkas biner.

Contoh kode melakukan I/O pada file binari dapat dilihat pada gambar 6.9.

Berikut penjelasan dari contoh kode pada gambar 7.12:

1. Program mendefinisikan `struct Data` untuk merepresentasikan struktur data yang mencakup sebuah bilangan bulat (`id`), double (`nilai`), dan larik karakter (`nama`).
2. Membuka file biner dalam mode tulis lalu menyimpan *instance* dari `Data` ke dalam format biner ke file menggunakan fungsi `write()`.
3. Membuka kembali file biner yang sama dalam mode baca kemudian mengkonversikan data biner ke *instance* lain dari `Data` menggunakan fungsi `read()`.

6.8 Penanganan Kesalahan dalam File I/O

Penanganan kesalahan selama operasi file sangat penting untuk menghasilkan program yang aman dan dapat diandalkan. Operasi I/O file dapat gagal karena berbagai alasan seperti jalur file yang salah, izin akses file yang tidak terpenuhi atau masalah yang tidak terduga dengan sistem berkas.

Berikut ini adalah beberapa panduan untuk menangani kesalahan selama operasi file di C++:

1. Memeriksa status file dengan is_open(), selalu periksa apakah sebuah file telah berhasil dibuka sebelum melakukan operasi baca atau tulis. Baik `std::ifstream` maupun `std::ofstream` menyediakan metode `is_open()` untuk mengecek status file
2. Mengimplementasi penanganan *exception*, pertimbangkan untuk menggunakan *exception* untuk menangani kesalahan dengan lebih baik. Sebagai contoh, Anda dapat menggunakan blok `try-catch` untuk menangani error *exception* yang dapat terjadi sewaktu-waktu saat operasi file berlangsung. (Contoh kode ada pada gambar 6.9)
3. Memeriksa kondisi file dengan bad() atau fail(), setelah melakukan operasi file, periksa setiap kegagalan atau kondisi buruk menggunakan fungsi anggota `fail()` dan `bad()` pada *stream*. Jika operasi gagal, Anda dapat mengambil informasi kesalahan tambahan menggunakan `str()`.
4. Menutup file dengan close() saat operasi pada file telah selesai. Jika Anda telah selesa melakukan operasi pada file

atau menemukan kesalahan selama operasi pada file, pastikan untuk menutup file dengan benar. Hal ini penting untuk melepaskan sumber daya dan menghindari kebocoran sumber daya.

5. Memeriksa ketersediaan file. Sebelum melakukan operasi file, periksa apakah file tersebut ada, terutama jika program Anda bergantung pada keberadaan file tersebut.
6. Menangani izin akses file. Pastikan bahwa program Anda memiliki izin yang diperlukan untuk membaca atau menulis ke file yang ditentukan.

Perlu diingat bahwa penanganan kesalahan harus disesuaikan dengan kebutuhan spesifik program Anda. Sangatlah penting untuk memberikan pesan kesalahan yang informatif dan mengambil tindakan yang tepat berdasarkan error yang terjadi dan pertimbangkan untuk menerapkan strategi implementasi penanganan kesalahan yang tepat untuk meningkatkan keandalan operasi I/O file pada program Anda.

```C/C++
std::ifstream inputFile;
try {
  // Membuka file
  inputFile.open("example.txt");

  // Memeriksa status file apakah bisa dibuka atau tidak
  if (!inputFile.is_open()) {
    throw std::runtime_error("Error saat membuka file.");
  }

  // Operasi I/O file dilakukan dari baris ini

} catch (const std::exception& e) {
  // Handle exception
  std::cerr << "Exception: " << e.what() << "\n";

} finally {
  // Menutup file saat di akhir baik saat terjadi exception ataupun tidak
  if (inputFile.is_open()) {
    inputFile.close();
  }
}
```

Gambar 6.9. Contoh Implementasi Penanganan Kesalahan dengan try catch

DAFTAR PUSTAKA

Davies, S, R. 2004. *C++ for Dummies*: Wiley Publishing.
Prata, Stephen. 2012. *C++ Primer Plus*: Addison-Wesley.

BAB 7
ARRAY

Oleh Paray Theo Lonando

7.1 Pendahuluan

Array (biasa disebut array) adalah kumpulan data yang setiap elemennya memiliki nama dan tipe yang sama, dan setiap elemen diakses oleh indeks array yang berbeda. Semua data terperinci disebut elemen array. Setiap array dapat dimanipulasi menggunakan pemilih elemen yang disebut indeks. Oleh karena itu, indeks harus memiliki tipe terurut (ada penerus dan pendahulunya) seperti Integre, Boolean, atau Char. Oleh karena itu, array digunakan untuk merepresentasikan kumpulan data bertipe sama, disimpan secara berurutan sesuai dengan indeksnya. Contoh array: string, daftar nomor telepon, daftar belanja, tabel kontrol, vektor, matriks.

7.2 Tujuan Array

Tujuan penggunaan array adalah untuk bertindak sebagai wadah untuk mengumpulkan nilai-nilai dari tipe yang berbeda, atau untuk menyimpan tipe data yang berbeda seperti integer, floating point, karakter, dll untuk membentuk satu unit dan memfasilitasi pengelompokan.Buat skrip yang jelas dengan menyusun nilai dan skrip Anda.Tujuan Pembelajaran Kompetensi yang Diharapkan : a.

Memahami pengertian array itu sendiri B. Memahami cara penggunaan array pada program yang dibuat dengan algoritma dan struktur data. Memahami penerapan yang sesuai dari material array itu sendiri.

7.3 Deklarasi Array

Deklarasi Array Setiap pengenal yang digunakan harus diurutkan terlebih dahulu dalam kamus/deklarasi bersama dengan tipe data yang dimilikinya. Demikian pula, tipe data array dan tipe elemen yang dikandungnya harus diurutkan dalam kamus/deklarasi. Saat mengurutkan variabel reguler (tipe data dan string sederhana),

Anda cukup menentukan tipenya; saat mengurutkan array, Anda juga perlu menentukan jumlah elemen (misalnya, 1 hingga 100). Variabel array dideklarasikan dengan menentukan tipe dan nama variabel, diikuti dengan jumlah lokasi penyimpanan yang akan dibuat.

Contoh :
Dalam Algorima :
c : array[1..5] of integer
Dalam bahasa C++ :

int c[5];
C++ secara otomatis akan menyediakan lokasi memori sesuai dengan yang dideklarasikan, dimana nomor indeks selalu dimulai dari 0.

int c[5] = {-12, 0, 20, 85, 1551};

Nilai suatu variabel array dapat juga diinisialisasi secara langsung seperti yang terdapat di dalam tanda kurung kurawal pada saat deklarasi di atas.

int x[5] = [0];
Deklarasi variable array sekaligus mengisi setiap lokasi memorinya dengan nilai 0.

Tipe data yang akan ditampung array bisa sembarang tipe data misal Integer, Boolean, Char, Real, String bahkan yang terstruktur seperti Set, Record atau array lagi.

7.3.1 Array Dimensi 1

Bentuknya (Pseudocode) :
nama_var : array [rangeindeks] of tipe
Bentuknya (C++):
tipe nama_var[ukuran];
Dengan :
Tipe : menyatakan jenis elemen array (int, char dan lain-lain)
Rangeindeks: menyatakan indeks awal sampai demgan indeks akhir dari var array yang dibuat

Ukuran : menyatakan jumlah maksimal elemen array

Nilai indeks awal dan indeks akhir yang dideklarasikan di algoritma tidak diperlukan pada deklarasi bahasa C++. Pada bahasa C++ hanya banyak elemennya saja (ukuran). Jadi ukuran = indeksakhir – indeksawal + 1.

Contoh :

Algoritma :

```
Algoritma Pangkat Dua
Deklarasi
square : array [1..100] of integer i,k : integer
Deskripsi
for i  1 to 10 do k  i + 1 square[i]  k * k
output(”Pangkat dari ”,k,” adalah ”, square[i])
endfor
```

Program C++ :

```
#include<iostream.h>
main()
{
int square[100];
int i,k;
for(i=0; i<10; i++)
{
k = i + 1;
square[i] = k * k;
cout<<”\nPangkat dari ”<<k<<” adalah ”<< square[i];
}
}
```

Output :

```
Pangkat dari 1 adalah 1
Pangkat dari 2 adalah 4
Pangkat dari 3 adalah 9
Pangkat dari 4 adalah 16
Pangkat dari 5 adalah 25
Pangkat dari 6 adalah 36
Pangkat dari 7 adalah 49
```

Pangkat dari 8 adalah 64
Pangkat dari 9 adalah 81
Pangkat dari 10 adalah 100

7.3.2 Array Dimensi 2

Array dua dimensi adalah array dengan m baris dan n kolom. Formatnya bisa matriks atau tabel. Pada prinsipnya matriks mirip dengan array. Matriks adalah kumpulan informasi serupa, di mana setiap elemen ditentukan oleh dua indeks (biasanya dikaitkan dengan baris dan kolom). Indeks matriks harus memiliki batasan dan properti yang sama dengan indeks array satu dimensi. Sering dikatakan matriks adalah array dua dimensi, namun perlu diketahui bahwa pengertian dua dimensi (baris dan kolom) ada pada pemikiran kita (ekspresi logikanya). Susunan elemen matriks dalam memori komputer selalu berupa rangkaian "sel linier". Pemahaman dua dimensi ini hanya membantu pemrogram merancang programnya. Secara umum, dimensi dan jumlah elemen array/matriks tidak dibatasi oleh compiler, tetapi oleh keterbatasan RAM yang disediakan. Karena matriks menggunakan memori internal. Struktur data ini mudah digunakan, namun menggunakan lebih banyak memori.
Contoh matriks yang sering kita lihat sehari-hari adalah tabel.
Deklarasi array (Pseudocode) :
nama_var : array [rangeindeks1,rangeindeiks2] of tipe

Deklarasi array (C++):
tipe nama_var[ukuran1][ukuran2];

Contoh :
Sebuah matrik A berukuran 2x3 dapat dideklarasikan sebagai berikut :
Algoritma :
a : array [1..2,1..3] of integer
$a_{1,1}$ □ 11
$a_{1,2}$ □ 7
$a_{1,3}$ □ 4
$a_{2,1}$ □ 12

$a_{2,2} = 3$
$a_{2,3} = 9$

C++ :
int a[2][3] = {{11, 7, 4},{12, 3, 9}};
yang akan menempati lokasi memori dengan susunan berikut :

	0	1	2
0	11	7	4
1	12	3	9

Dan definisi variabel untuk setiap elemen tersebut adalah :

	0	1	2
0	a[0][0]	a[0][1]	a[0][2]
1	a[1][0]	a[1][1]	a[1][2]

Contoh Program 1 :

```
#include<iostream.h>
main()
{
int A[3][3]={{1,2,3},{4,5,6},{7,8,9}};
int i,j;
for(i=0; i<3; i++)
{
for(j=0; j<3; j++)
cout<<A[i][j]<<" ";
cout<<endl;
}
}
```

Output :
1 2 3
4 5 6

7 8 9

Contoh Program 2 :

```
#include<stdio.h>
void printArray(int [][3]);
main()
{
int matrik1 [2][3] = { {1, 2, 3}, {4, 5, 6}};
int matrik2 [2][3] = { {1, 2, 3}, {4, 5}};
int matrik3 [2][3] = { {1, 2}, {4} };
printArray(matrik1);
printArray(matrik2); p
rintArray(matrik3); return 0;
}
void printArray(int a[][3])
{  int i, j;
for(i=0; i<=1; i++)
{ for(j=0; j<=2; j++)
printf("%d ", a[i][j]);
printf("\n");
}
}
```

Output :

```
1 2 3
4 5 6
1 2 3
4 5 0
1 2 0
4 0 0
```

Contoh Program 3 :

```
//program penjumlahan matrik
#include<iostream.h>
main()
{
int A[3][2]={{1,2},{3,4},{5,6}};
```

```
int B[3][2]={{7,8},{9,10},{11,12}};
int C[3][2];
int i,j;
cout<<"Matrik A = \n";
for(i=0; i<3 ; i++)
{

for(j=0; j<2 ; j++)
cout<<A[i][j]<<" ";
cout<<endl;
}
cout<<"Matrik B = \n";
for(i=0; i<3 ; i++)
{
for(j=0; j<2 ; j++)
cout<<B[i][j]<<" ";
cout<<endl;
}
//penjumlahan matrix for(i=0; i<3 ; i++)
for(j=0; j<2 ; j++)
C[i][j]=A[i][j] + B[i][j];
cout<<"Matrik C = \n";
for(i=0; i<3 ; i++)
{
for(j=0; j<2 ; j++)
cout<<C[i][j]<<" ";
cout<<endl;
} }
```

7.3.3 Array Dimensi Banyak

Bentuknya dalam bahasa C++ :
tipe nama_var[ukuran1[ukuran2]. . .[ukuranN];

Contoh :
int data_huruf[2][4][6];

DAFTAR PUSTAKA

Edy Budiman. 2015. *Belajar Dasar Algoritma dan Pemrograman.* Samarinda : ., 2015

BAB 8
PENGURUTAN (SORTING)

Oleh Wirawan Istiono

8.1 Apa yang dimaksud dengan Sorting dalam Bahasa Pemrograman?

Algoritma sorting adalah prosedur sistematis yang digunakan untuk mengatur ulang sejumlah besar elemen ke dalam urutan tertentu, seperti alfabet, urutan nilai menurun, atau urutan jarak menaik. Algoritma sorting menerima data masukan dalam bentuk daftar, menjalankan operasi tertentu pada daftar tersebut, dan menghasilkan array terurut sebagai keluaran. Algoritma sorting memiliki banyak penerapan, seperti memfasilitasi penataan barang berdasarkan harga di situs web ritel dan menetapkan urutan situs pada halaman hasil mesin pencari atau search engine results page (SERP) (Wigmore, 2023).

Algoritma sorting adalah prosedur komputasi yang mengatur isi daftar dalam urutan tertentu. Dua urutan yang paling umum digunakan adalah urutan numerik dan urutan leksikografis, yang dapat berupa menaik atau menurun. Mengoptimalkan efisiensi algoritma lain, seperti algoritma pencarian dan penggabungan, bergantung pada sorting data masuk yang efisien ke dalam daftar yang diurutkan. Sorting seringkali bermanfaat untuk membakukan data dan menghasilkan keluaran yang mudah dipahami oleh manusia (Wikipedia, 2021).

Beberapa contoh Algoritma sorting antara lain (Wigmore, 2023):

1. *Bubble sort algorithm* menelusuri daftar secara berulang, membandingkan item di dekatnya, dan menukar lokasinya jika urutannya tidak benar. Algoritma mengulangi daftar dengan cara ini hingga seluruh daftar telah diurutkan.
2. *Insertion sort algorithm* dimulai dengan menyusun dua item pertama dalam urutan menaik. Kemudian dilanjutkan dengan membandingkan item ketiga dengan item kedua, menukar lokasinya jika perlu. Proses ini diulangi dengan item pertama.

Benda-benda yang berurutan yang menjalani prosedur yang sama sering kali tidak memerlukan perpindahan yang signifikan dalam benda-benda yang disusun.

3. *Shell sort algorithm* melakukan perbandingan dan mengurutkan elemen pada interval tertentu, secara bertahap mengurangi ukuran interval pada setiap iterasi daftar. Proses terakhir melibatkan Algoritma sorting bubble, namun efisiensinya ditingkatkan karena fakta bahwa item sudah berada di dekat tempat yang dituju.
4. Metode quicksort menggunakan pemilihan item secara acak dari daftar, diikuti dengan perbandingan semua item lain yang ada di dalamnya. Kemudian mengatur item menjadi dua kelompok: item yang harus diposisikan sebelum item yang dipilih, dan item yang harus diposisikan setelahnya. Oleh karena itu, tidak perlu lagi membandingkan item mana pun dengan item di kelompok lain. Prosedurnya melibatkan pemilihan objek secara acak dari masing-masing dua kategori dan mengulangi proses ini. Pada akhirnya, sorting akhir dilakukan dengan metode lain, seperti algoritma penyisipan.

Secara formal, setiap Algoritma sorting harus mematuhi dua batasan untuk menghasilkan keluaran yang diinginkan (Wikipedia, 2021):

1. Outputnya disusun secara monotonik, artinya setiap bagian tidak lebih kecil atau lebih besar dari elemen sebelumnya, sebagaimana ditentukan oleh urutan yang diperlukan.
2. Hasilnya adalah permutasi, yaitu penataan ulang masukan dengan tetap mempertahankan semua elemen aslinya.

Untuk mencapai efisiensi maksimum, disarankan untuk menyimpan data masukan dalam struktur data yang mendukung akses acak, dibandingkan dengan struktur data yang hanya memungkinkan akses berurutan.

8.2 Sejarah dan konsep

Sejak dimulainya komputasi, masalah sorting telah menarik perhatian yang signifikan dalam penelitian, mungkin karena rumitnya penanganan masalah tersebut secara efisien meskipun deskripsinya mudah dan terkenal. Salah satu pengembang Algoritma sorting awal pada tahun 1951 adalah Betty Holberton, yang berkontribusi pada pengembangan ENIAC dan UNIVAC (Sheppard, 2023). Analisis bubble sort sudah ada sejak tahun 1956 (Debra, 1996). Algoritma optimal asimtotik telah ditemukan sejak pertengahan abad ke-20. Meskipun demikian, algoritma baru terus dikembangkan, seperti Timsort yang diadopsi secara luas yang berasal dari tahun 2002, dan Library Sort yang pertama kali diterbitkan pada tahun 2006 (Wikipedia, 2021).

Algoritma sorting perbandingan memerlukan minimal perbandingan Ω(n log n), dengan urutan masukan tertentu memerlukan kelipatan n log n perbandingan, di mana n adalah jumlah elemen dalam larik yang akan diurutkan. Algoritma berbasis non-perbandingan, seperti pengurutan penghitungan, dapat menunjukkan kinerja yang unggul (Awasthi, 2021).

Algoritma sorting biasanya diajarkan dalam kursus pengantar ilmu komputer. Mereka berfungsi sebagai pengantar yang berguna untuk berbagai konsep algoritma dasar, termasuk notasi O besar, algoritma bagi-dan-taklukkan, struktur data seperti heap dan pohon biner, algoritma acak, analisis kasus terbaik, terburuk, dan rata-rata, tradeoff ruang-waktu, dan batas atas dan bawah.

Mengurutkan array kecil secara efisien, baik dengan meminimalkan jumlah perbandingan dan pertukaran atau dengan mempertimbangkan karakteristik spesifik mesin, masih merupakan tantangan penelitian yang berkelanjutan. Saat ini, solusi hanya tersedia untuk array dengan anggota kurang dari 20. Masalah mencapai sorting optimal pada mesin paralel, sebagaimana didefinisikan oleh berbagai kriteria, saat ini sedang diteliti (Naeem Akhter, Muhammad Idrees dan Furqan-ur-Rehman, 2017).

8.3 Klasifikasi

Algoritma sorting dapat dikategorikan berdasarkan (Wigmore, 2023):

1. Kompleksitas komputasi mengacu pada studi tentang sumber daya yang diperlukan untuk memecahkan masalah komputasi, seperti waktu dan ruang.
2. Performa optimal, suboptimal, dan tipikal berdasarkan ukuran daftar. Kompleksitas waktu dari Algoritma sorting serial konvensional adalah O(n log n) untuk perilaku baik, O(log2 n) untuk pengurutan paralel, dan O(n2) untuk perilaku buruk. Kompleksitas waktu optimal untuk pengurutan serial adalah O(n), namun hal ini tidak dapat dicapai dalam skenario biasa. Kompleksitas waktu pengurutan paralel optimal adalah O(log n).
3. Algoritma yang melakukan swap tanpa memerlukan memori tambahan.
4. Pemanfaatan sumber daya komputer, khususnya memori. Secara khusus, Algoritma sorting tertentu dianggap "di tempat". Secara formal, Algoritma sorting di tempat hanya memerlukan jumlah memori yang konstan melebihi item yang diurutkan. Kadang-kadang, memori O(log n) tambahan juga dianggap "di tempat".
5. Rekursi: Algoritma tertentu dapat diklasifikasikan sebagai rekursif atau non-rekursif, namun ada juga yang dapat menjadi keduanya, seperti pengurutan gabungan.
6. Stabilitas: Algoritma sorting yang stabil memastikan bahwa catatan dengan kunci yang sama (yaitu nilai) mempertahankan urutan relatifnya.
7. Terlepas dari apakah itu jenis perbandingan. Algoritma sorting perbandingan mengevaluasi data hanya dengan membandingkan dua komponen menggunakan operator perbandingan.
8. Teknik umum antara lain penyisipan, pertukaran, seleksi, penggabungan, dan lain-lain. Algoritma sorting pertukaran mencakup pengurutan bubble dan pengurutan cepat. Algoritma sorting seleksi mencakup pengurutan siklus dan pengurutan heap.

9. Penentuan apakah algoritma dijalankan secara berurutan atau paralel. Wacana selanjutnya sebagian besar berfokus pada algoritma serial dan mengasumsikan mode operasi berurutan.
10. Kemampuan beradaptasi mengacu pada dampak urutan masukan yang telah diatur sebelumnya terhadap durasi operasi. Algoritma yang mempertimbangkan faktor ini dianggap adaptif.
11. Berbasis internet: Algoritma online, seperti Penyisipan Penyisipan, mampu mengurutkan aliran masukan yang berkelanjutan.

8.4 Jenis Sorting dalam Struktur Data

1. Sorting berbasis perbandingan melibatkan penggunaan pembanding untuk membandingkan komponen atau item dalam sampel data. Pembanding ini menetapkan susunan item. Beberapa contoh Algoritma sorting adalah Bubble Sort dan Merge Sort.
2. Algoritma sorting berbasis penghitungan tidak menyertakan perbandingan antar elemen. Sebaliknya, mereka mengandalkan asumsi yang diperhitungkan selama eksekusi. Dua contoh Algoritma sorting adalah Counting Sort dan Radix Sort.
3. Perbandingan antara Sorting Di Tempat dan Tidak Di Tempat: Dalam struktur data, teknik pengurutan di tempat mengubah susunan elemen dalam larik secara langsung, tanpa membuat larik baru. 'Di tempat' mengacu pada proses sorting di mana tidak diperlukan ruang tambahan selain ruang masukan, kecuali ruang yang digunakan untuk variabel atau iterator. Selain itu, ruang yang digunakan oleh tumpukan dalam algoritma rekursif tidak diperhitungkan. Sebaliknya, pendekatan sorting *Not-in-Place* menggunakan struktur data tambahan untuk mengurutkan array asli. Beberapa contoh teknik pengurutan di tempat antara lain Bubble Sort, Selection Sort, dan Quick Sort. Contoh Algoritma sorting Not in Place adalah Merge Sort.

8.5 Sorting Stabil dan Tidak Stabil

Algoritma sorting stabil menyusun elemen identik dalam urutan yang sama seperti yang muncul pada masukan asli. Misalnya, dalam demonstrasi sorting kartu di sebelah kanan, kartu-kartu

disusun menurut peringkatnya, tetapi mengabaikan jenisnya. Hal ini memungkinkan potensi banyak iterasi berbeda yang terorganisir secara akurat dari daftar awal. Algoritma sorting stabil mematuhi aturan berikut: ketika dua hal dianggap sama, urutan aslinya dipertahankan. Dengan kata lain, jika satu item sebelum item lainnya di input, maka item tersebut juga akan mendahuluinya di output (Awasthi, 2021).

Menjaga stabilitas sangat penting untuk menjaga ketertiban di berbagai jenis data yang dilakukan pada pengumpulan data yang sama. Misalnya, pertimbangkan skenario dimana data siswa, yang mencakup nama dan bagian kelas, disusun secara dinamis. Sorting dilakukan terutama berdasarkan nama, diikuti oleh bagian kelas. Menggunakan Algoritma sorting yang stabil memastikan bahwa urutan nama tidak akan terpengaruh saat mengurutkan berdasarkan bagian kelas. Namun, jika Algoritma sorting yang tidak stabil digunakan, pengurutan berdasarkan bagian dapat mengatur ulang urutan nama, sehingga menghasilkan daftar siswa yang tidak berdasarkan abjad (Sheppard, 2023).

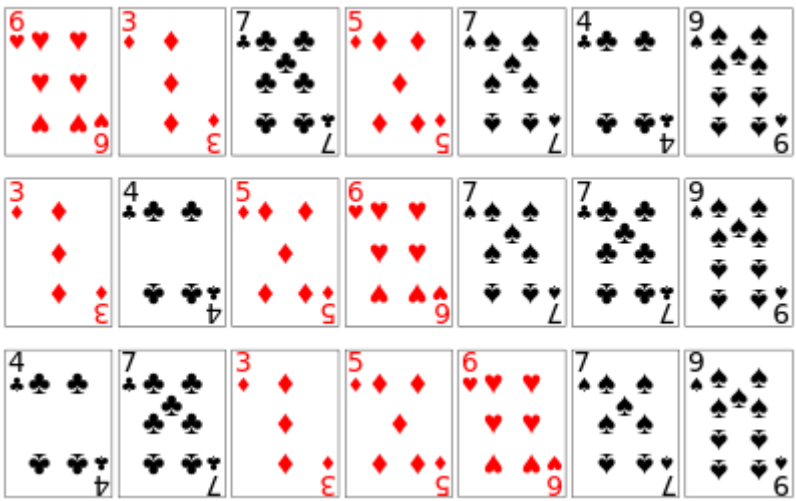
Secara formal, data yang diurutkan dapat direpresentasikan sebagai catatan atau tupel nilai, dan bagian tertentu dari data yang digunakan untuk sorting disebut sebagai kunci. Dalam contoh kartu, kartu dilambangkan sebagai catatan yang terdiri dari pangkat dan jenis, dengan pangkat sebagai kuncinya. Suatu metode pengurutan dianggap stabil jika, dalam kasus ketika dua catatan R dan S memiliki kunci yang sama dan R sebelum S dalam daftar asli, R akan selalu mendahului S dalam daftar yang diurutkan (Wikipedia, 2021).

Ketika elemen yang identik tidak dapat dibedakan, misalnya dengan bilangan bulat atau data apa pun yang seluruh elemennya berfungsi sebagai kuncinya, konsep stabilitas menjadi tidak relevan. Stabilitas tidak menjadi masalah ketika semua kunci berbeda.

Algoritma sorting yang tidak stabil memiliki kemampuan untuk dimodifikasi secara sengaja agar menjadi stabil. Pendekatan untuk mencapai hal ini adalah dengan memperpanjang proses perbandingan kunci secara artifisial. Dengan cara ini, ketika membandingkan dua objek dengan kunci yang identik, urutan entri dalam daftar masukan asli digunakan sebagai faktor penentu. Namun

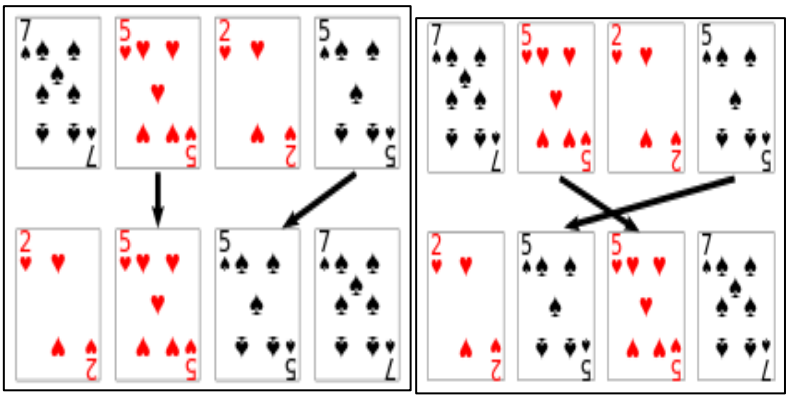
mengingat urutan ini mungkin memerlukan waktu dan kapasitas fisik ekstra (Awasthi, 2021).

Penerapan Algoritma sorting yang stabil melibatkan pengurutan daftar berdasarkan kunci primer dan sekunder. Sebagai contoh, mari kita pertimbangkan skenario di mana kita ingin menyusun satu set kartu remi dalam urutan tertentu: pertama berdasarkan jenis kartu dalam urutan klub (♣), berlian (♦), hati (♥), sekop (♠), dan kemudian di dalam setiap jenis, kartu disusun berdasarkan peringkatnya. Untuk mencapai hal ini, mulailah dengan menyusun kartu-kartu dalam urutan peringkat menggunakan metode sorting apa pun. Selanjutnya, lakukan pengurutan stabil berdasarkan jenis kartu:



Gambar 8.1. Contoh sorting kartu stabil
(Sumber: Wikipedia, 2021)

Dalam setiap setelan, jenis stabil mempertahankan urutan peringkat yang ada. Konsep ini dapat diperluas untuk mengakomodasi sejumlah kunci dan digunakan oleh jenis radix. Hasil yang setara dapat dicapai dengan metode sorting tidak stabil dengan menggunakan perbandingan kunci leksikografis. Perbandingan ini misalnya mengutamakan perbandingan jenis pakaian terlebih dahulu, baru kemudian membandingkan peringkat jika jenis pakaiannya identik (Sheppard, 2023).



Gambar 8.2. Contoh sortir kartu Stabil (kiri) dan Tidak Stabil (kanan). (Sumber: Wikipedia, 2021)

Gambar 8.2 menunjukkan ilustrasi Algoritma sorting stabil yang diterapkan pada setumpuk kartu remi. Saat kartu diurutkan menggunakan Algoritma sorting stabil berdasarkan peringkatnya, kedua kartu 5 harus mempertahankan urutan relatif aslinya dalam keluaran yang diurutkan. Jika Algoritma sorting tidak stabil digunakan, urutan angka 5 pada keluaran yang diurutkan mungkin terbalik (Awasthi, 2021).

1. Metode pengurutan yang stabil memastikan bahwa ketika dua item dengan kunci yang sama ditemukan, urutan relatifnya dalam keluaran yang diurutkan sama dengan urutannya dalam masukan yang tidak diurutkan.
2. Suatu metode pengurutan dianggap tidak stabil jika tidak menjaga urutan relatif objek dengan kunci yang sama sebelum dan sesudah pengurutan.

Tabel 8.1. Klasifikasi jenis sorting stabil dan tidak stabil

Algoritma Sorting	In-place	Stabil
Bubble sort	Ya	Ya
Selection sort	Ya	Tidak
Insertion sort	Ya	Ya
Quick sort	Ya	Tidak
Merge sort	Tidak	Ya
Heap sort	Ya	Tidak

(Sumber: Awasthi, 2021)

8.6 Algoritma Sorting Adaptif dan Non-Adaptif

Suatu algoritma dianggap "adaptif" ketika kompleksitas waktunya dipengaruhi oleh urutan elemen dalam array input yang perlu diurutkan (Sheppard, 2023).

Pengurutan penyisipan adalah Algoritma sorting adaptif. Jika input sudah diurutkan, kompleksitas waktunya diketahui O(n). Oleh karena itu, jika array masukan hampir terurut, disarankan untuk menggunakan pengurutan penyisipan. Namun, penting untuk dicatat bahwa ini bukan satu-satunya pertimbangan ketika membandingkan penyisipan semacam dengan Algoritma sorting lainnya (Debra, 1996).

Pengurutan Penggabungan diklasifikasikan sebagai Algoritma sorting "Non-Adaptif" karena tidak mempertimbangkan urutan awal elemen dalam array. Algoritma akan secara konsisten memiliki kompleksitas waktu O(nlogn).

Tabel 8.2. Klasifikasi Algoritma sorting adaptif atau non-adaptif

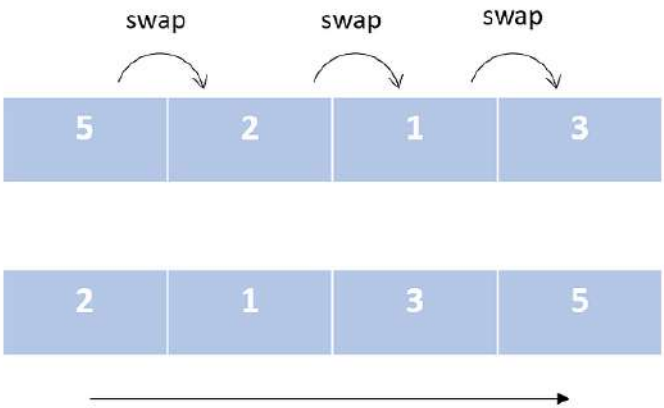
Algoritma sorting adaptif	Algoritma sorting non-adaptif
Bubble Sort	Selection Sort
Insertion Sort	Merge Sort
Quick Sort	Heap Sort

(Sumber: Awasthi, 2021)

8.7 Jenis Algoritma sorting

1. Bubble sort

   Bubble sort merupakan algoritma sorting paling dasar. Algoritma melakukan iterasi melalui daftar dan membandingkan elemen secara berpasangan pada setiap iterasi. Ini menukar elemen sehingga elemen yang lebih besar ditempatkan di akhir daftar (Mishira Deepak dan Garg, 2008).



Pseudocode dari bubble sort:

```
procedure bubbleSort(A: list)
  n = length(A)
  for i from 0 to n-1
    for j from 0 to n-i-1
      if A[j] > A[j+1]
        swap(A[j], A[j+1])
```

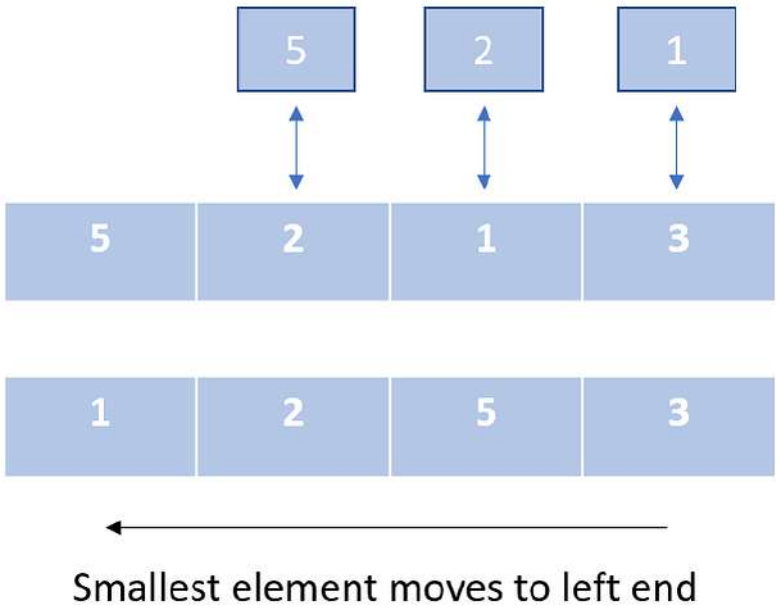
Penting untuk dicatat bahwa Bubble Sort bukanlah Algoritma sorting yang paling efisien, terutama untuk kumpulan data yang besar. Namun, ini sering digunakan untuk tujuan pengajaran karena kesederhanaannya (Awasthi, 2021).

Skenario aplikasi

a. Ini berfungsi sebagai pengenalan konsep Algoritma sorting untuk siswa yang mempelajari Ilmu Komputer.
b. Sorting bubble adalah teknik yang banyak digunakan dalam grafik komputer untuk mengidentifikasi kesalahan kecil, seperti pertukaran dua elemen, dalam array yang hampir terurut.

2. Selection sort

Selection sort adalah metode pengurutan yang mempartisi array tertentu menjadi dua bagian: segmen kiri yang diurutkan dan bagian kanan yang tidak diurutkan.



Pseudocode selection sort:

```
procedure selectionSort(A: list)
   n = length(A)
   for i from 0 to n-1
      minIndex = i

      for j from i+1 to n-1
         if A[j] < A[minIndex]
            minIndex = j
      swap(A[i], A[minIndex])
```

Selection Sort, mirip dengan Bubble Sort, bukanlah pilihan yang paling optimal untuk menangani kumpulan data yang sangat besar, namun mudah untuk dipahami dan dijalankan (Awasthi, 2021).

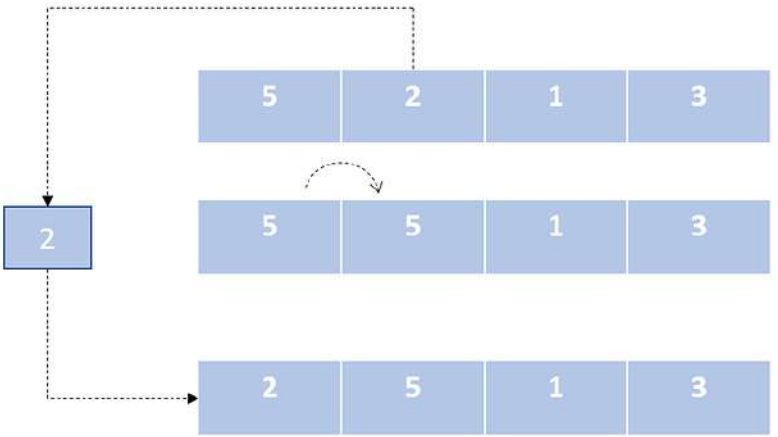
Skenario aplikasi

a. Ini digunakan ketika besarnya daftar kecil. Kompleksitas waktu pengurutan pilihan adalah $O(N^2)$, menjadikannya tidak efisien untuk daftar besar.

b. Selain itu, ini digunakan dalam situasi di mana terdapat kendala pada kapasitas memori, karena meminimalkan jumlah swap yang diperlukan selama proses sorting.

3. Insertion sort

Insertion sort adalah metode pengurutan yang mempartisi array yang disediakan menjadi bagian yang sudah terurut dan bagian yang belum terurut. Selanjutnya, kita menelusuri bagian yang tidak disortir secara berurutan dan menempatkan setiap bagian dari bagian ini ke posisi yang sesuai dalam bagian yang diurutkan (Awasthi, 2021).



Pseudocode dari Insertion sort:

```
procedure insertionSort(A: list)
   n = length(A)
   for i from 1 to n-1
      key = A[i]
      j = i - 1

      while j >= 0 and A[j] > key
         A[j + 1] = A[j]
         j = j - 1

      A[j + 1] = key
```

Penyisipan Sorting sangat efektif untuk kumpulan data kecil dan sering digunakan dalam skenario dunia nyata untuk kumpulan data yang diurutkan sebagian atau sebagai komponen dasar dari Algoritma sorting yang lebih canggih (Naeem Akhter, Muhammad Idrees, dan Furqan-ur-Rehman, 2017).

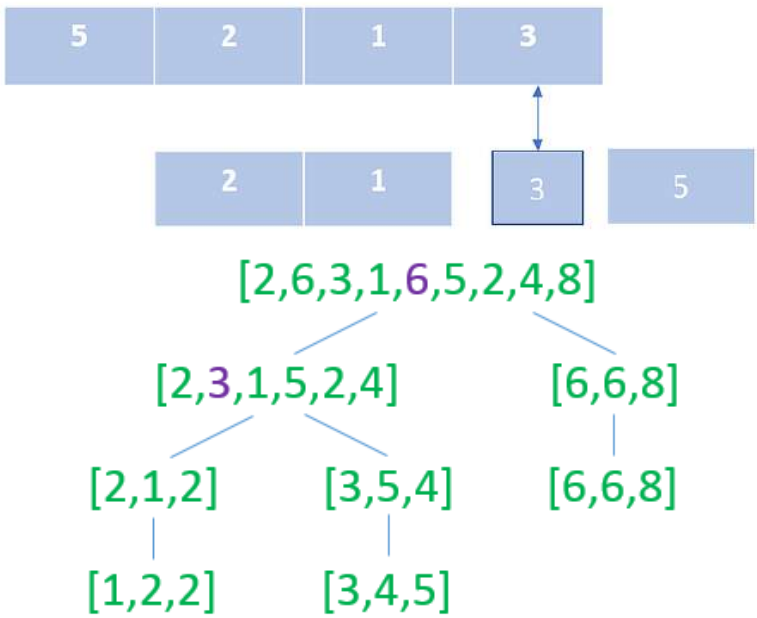
Skenario aplikasi

a. Algoritma ini menunjukkan stabilitas dan menunjukkan efisiensi tinggi ketika daftar hampir diurutkan.
b. Ini menunjukkan efisiensi tinggi dalam mengurutkan daftar yang sangat pendek, seperti daftar yang berisi 40 elemen.

4. Quicksort

Algoritma quicksort adalah metode yang mengikuti pendekatan membagi dan menaklukkan. Itu berasal dari pembagian. Ini juga disebut sebagai sorting pertukaran partisi.

Konsep dasar yang mendasari quicksort adalah memilih elemen pivot dari array tertentu dan kemudian membagi array menjadi beberapa partisi berdasarkan elemen pivot tersebut. Setelah itu, ia memanggil dirinya sendiri secara rekursif dan kemudian membagi dua subarray.



Pseudocode Quicksort:

```
procedure quickSort(A: list, low: integer, high: integer)
  if low < high
    pivotIndex = partition(A, low, high)
     quickSort(A, low, pivotIndex - 1)
    quickSort(A, pivotIndex + 1, high)

procedure partition(A: list, low: integer, high: integer): integer
   pivot = A[high]
    i = low - 1
  for j from low to high - 1
    if A[j] <= pivot
      i = i + 1
      swap(A[i], A[j])

  swap(A[i + 1], A[high])
   return i + 1
```

Algoritma Quick Sort banyak digunakan untuk menyortir data karena efisiensinya yang tinggi, menjadikannya pilihan umum sebagai Algoritma sorting default di berbagai perpustakaan komputer (Mishira Deepak dan Garg, 2008).

Algoritma Quick Sort terdiri dari langkah logis berikut:

a. Pilihan pivot: Memilih elemen sebagai pivot (dalam hal ini, kami memilih elemen terakhir sebagai pivot)
b. Partisi: Array dibagi sedemikian rupa sehingga semua elemen yang lebih kecil dari pivot ditempatkan di subarray kiri, sedangkan semua elemen yang lebih besar dari pivot disimpan di subarray kanan.
c. Pemanggilan Quicksort menggunakan rekursi: Fungsi Quicksort dipanggil secara rekursif untuk dua subarray yang dihasilkan sebelumnya, dan prosesnya diulangi.

Skenario aplikasi

a. Quicksort sangat efisien untuk kumpulan data yang dapat ditampung dalam RAM yang tersedia.

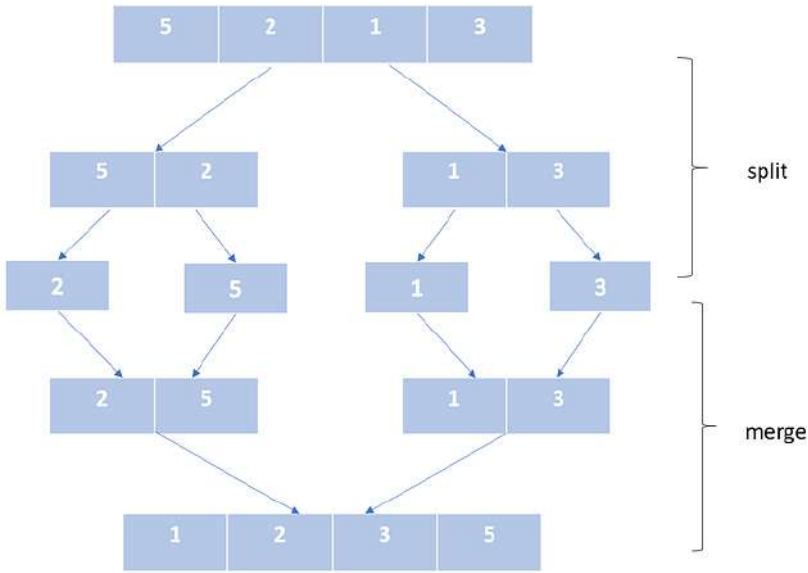
b. Sorting Cepat cocok untuk array karena merupakan metode pengurutan di tempat, artinya tidak memerlukan penyimpanan tambahan.

Skenario apa yang paling tidak menguntungkan untuk Sorting cepat?

a. Keadaan ini terjadi ketika daftar atau array elemen sudah diurutkan. Oleh karena itu, pivotnya adalah: pivot : array(rendah+tinggi)/2.
b. pivot: acak (rendah, tinggi).
c. Berdasarkan skenario, kompleksitas waktu quicksort harus O(n2) dan kompleksitas ruang harus O(n) karena tumpukan rekursif.

5. Merge sort

Ini adalah algoritma yang mengikuti pendekatan membagi dan menaklukkan. Algoritma ini menggunakan proses membagi daftar menjadi dua secara berulang hingga hanya berisi satu anggota.



Pseudocode dari Merge sort:

```
procedure mergeSort(A: list)
  if length(A) > 1
    mid = length(A) / 2
    left_half = A[0:mid-1]
    right_half = A[mid:length(A)-1]

    mergeSort(left_half)
    mergeSort(right_half)

    merge(A, left_half, right_half)

procedure merge(A: list, left: list, right: list)
  i = j = k = 0

  while i < length(left) and j < length(right)
    if left[i] <= right[j]
      A[k] = left[i]
      i = i + 1
    else
      A[k] = right[j]
      j = j + 1
    k = k + 1

  while i < length(left)
    A[k] = left[i]
    i = i + 1
    k = k + 1

  while j < length(right)
    A[k] = right[j]
    j = j + 1
    k = k + 1
```

Merge Sort terkenal dengan stabilitas dan kinerjanya yang andal. Pendekatan ini memastikan kompleksitas waktu sebesar O(n log n) tetapi memerlukan ruang ekstra untuk operasi penggabungan,

menjadikannya kurang efisien dalam memori dibandingkan dengan beberapa Algoritma sorting alternatif.

Skenario aplikasi

1. Performa pengurutan gabungan sangat efisien bila diterapkan pada daftar tertaut.
2. Hal ini umumnya digunakan untuk sorting eksternal dalam situasi ketika biaya akses acak jauh lebih tinggi dibandingkan dengan akses sekuensial.

8.8 Apa Algoritma Sorting Terbaik?

Pemilihan Algoritma sorting yang optimal bergantung pada tuntutan dan atribut tertentu dari data yang diurutkan. Algoritma sorting menunjukkan kompleksitas waktu dan ruang yang bervariasi, dan efisiensinya bergantung pada parameter seperti ukuran kumpulan data, penyebaran data, dan ketersediaan memori (Naeem Akhter, Muhammad Idrees, dan Furqan-ur-Rehman, 2017).

Di bawah ini beberapa Algoritma sorting beserta karakteristiknya masing-masing (Mishira Deepak dan Garg, 2008):

1. Quick Sort:
   a. Kompleksitas Waktu Kasus Rata-rata: O(n log n)
   b. Kompleksitas Waktu Kasus Terburuk: O(n^2) (jarang, tetapi bisa terjadi)
   c. Kompleksitas Ruang: O(log n)
   d. Algoritma sorting di tempat (membutuhkan memori tambahan minimal)
   e. Efisien untuk kumpulan data besar dan umumnya lebih cepat dalam praktiknya dibandingkan algoritma O(n log n) lainnya.
2. Merge sort:
   a. Kompleksitas Waktu Kasus Rata-rata: O(n log n)
   b. Kompleksitas Waktu Kasus Terburuk: O(n log n)
   c. Kompleksitas Ruang: O(n) (membutuhkan ruang tambahan untuk penggabungan)
   d. Algoritma sorting yang stabil (mempertahankan urutan elemen yang sama)
   e. Cocok untuk kumpulan data besar dan daftar tertaut.
3. Heap sort:
   a. Kompleksitas Waktu Kasus Rata-rata: O(n log n)

   b. Kompleksitas Waktu Kasus Terburuk: O(n log n)
   c. Kompleksitas Ruang: O(1) (sorting di tempat)
   d. Algoritma sorting di tempat tanpa persyaratan memori tambahan.
   e. Tidak ramah cache seperti quicksort.
4. Insertion sort:
   a. Kompleksitas Waktu Kasus Rata-rata: O(n^2)
   b. Kompleksitas Waktu Kasus Terburuk: O(n^2)
   c. Kompleksitas Ruang: O(1) (sorting di tempat)
   d. Sederhana dan efisien untuk kumpulan data kecil atau kumpulan data yang diurutkan sebagian.
5. Bubble sort:
   a. Kompleksitas Waktu Kasus Rata-rata: O(n^2)
   b. Kompleksitas Waktu Kasus Terburuk: O(n^2)
   c. Kompleksitas Ruang: O(1) (sorting di tempat)
   d. Sederhana dan mudah dipahami tetapi kurang efisien dibandingkan algoritma lainnya.

Algoritma optimal bergantung pada atribut berbeda dari data Anda dan spesifikasi aplikasi Anda. Quicksort atau mergesort umumnya disukai untuk menyortir kumpulan data besar dan untuk sorting tujuan umum. Namun demikian, dalam skenario tertentu (misalnya ketika menangani data yang hampir terurut), jenis penyisipan atau jenis bubble mungkin menunjukkan efisiensi yang lebih baik. Saat memilih Algoritma sorting, penting untuk selalu mempertimbangkan konteks yang tepat dan kualitas unik data Anda.

DAFTAR PUSTAKA

Awasthi, A. K. 2021. *Top 5 Sorting Algorithms, built-in function in programming languages which sorting internally used? When to use which sorting technique?*, *atulkawasthi.medium.com*. Available at: https://atulkawasthi.medium.com/top-5-sorting-algorithms-built-in-function-in-programming-languages-which-sorting-internally-used-f1c1e3027ef4 (Accessed: 5 December 2023).

Debra, D. 1996/ 'Approved for the Department : Approved for the University Committee on Graduate Studies ':, *Society*, (March).

Mishira Deepak and Garg, D. 2008. 'Selection of Best Sorting Algorithm', *International Journal of Intelligent Information Processing*, 2(December), pp. 363–368. Available at: http://www.gdeepak.com/pubs/Selection of best sorting algorithm.pdf.

Naeem Akhter, Muhammad Idrees and Furqan-ur-Rehman. 2017. 'Sorting Algorithms – A Comparative Study', *International Journal of Computer Science and Information Security*, 14(12), pp. 930–936.

Sheppard, A. 2023. *Meet the 'Refrigerator Ladies' Who Programmed the ENIAC*, *Mental Floss*. Available at: https://www.mentalfloss.com/article/53160/meet-refrigerator-ladies-who-programmed-eniac (Accessed: 12 December 2023).

Wigmore, I. 2023. *Definition sorting algorithm*, *techtarget.com*. Available at: https://www.techtarget.com/whatis/definition/sorting-algorithm (Accessed: 12 December 2023).

Wikipedia. 2021. *Sorting algorithm*, *wikipedia.org*. Available at: https://en.wikipedia.org/wiki/Sorting_algorithm.

BAB 9
STRUCT

Oleh Wulan Sri Lestari

9.1 Pendahuluan

Dalam pemrograman C++, programmer seringkali menghadapi kebutuhan untuk mengelompokkan beberapa data yang saling berhubungan ke dalam satu kesatuan. Hal tersebut menjadi alasan munculnya konsep struct. Struct, merupakan singkatan dari "structure," yang memungkinkan para programmer menciptakan tipe data khusus untuk menggabungkan variabel dengan tipe data yang berbeda ke dalam satu variabel tertentu. Dengan memanfaatkan struct, kita dapat dengan mudah mengorganisasi dan merepresentasikan informasi terkait dalam program.

Dalam pengembangan perangkat lunak, memiliki struktur data yang baik seperti struct dapat menjadi elemen kunci untuk membangun program yang efisien dan mudah dipahami. Melalui struct, pengorganisasian dan penyimpanan data terstruktur di memori menjadi lebih efisien, memudahkan manipulasi dan pengelolaan data. Kelebihan lainnya adalah kemampuan struct memungkinkan para programmer untuk menggabungkan data dan fungsionalitas terkait dalam satu kesatuan, menghasilkan kode yang lebih bersih dan terstruktur secara optimal.

Bab ini akan membahas tentang definisi struct, deklarasi variabel struct, mengakses anggota struct, penerapan struct sebagai array, parameter fungsi, dan pointer serta nested struct. Selain itu, bab ini akan memberikan contoh penggunaan struct yang dapat dipelajari untuk memahami konsep struct dengan baik.

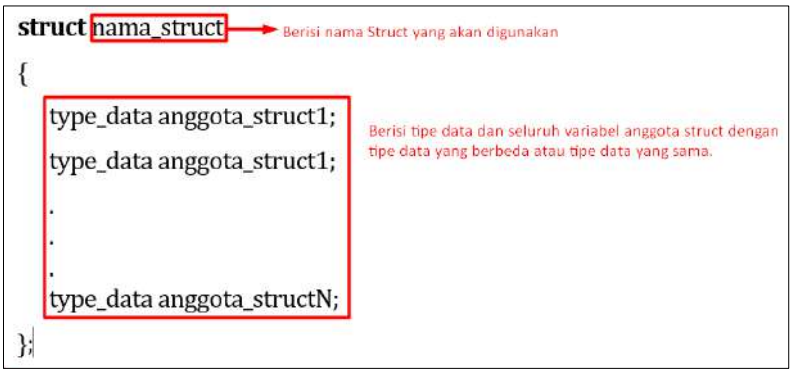
9.2 Definisi Struct

Struct merupakan tipe data yang ditentukan oleh pengguna untuk mengelompokkan data/variabel dengan tipe data yang berbeda. Sebagai contoh, sebuah struct dapat menyimpan informasi

tentang sebuah buku, seperti judul, penulis, penerbit, tahun terbit, harga buku dan data lainnya yang relevan.

9.2.1 Sintaks dan Deklarasi Struct

Untuk mendeklarasikan sebuah struct, kita dapat menggunakan keyword struct. Gambar 9.1 merupakan sintaks umum penggunaan struct.



Gambar 9.1. Sintaks Struct

Variabel struct dapat dideklarasikan dengan dua cara, yaitu deklarasi langsung pada saat pembuatan struct atau deklarasi seperti tipe data biasa di dalam fungsi **'main'** dengan menambahkan keyword struct. Sebagai contoh sederhana, kita akan mendeklarasikan sebuah struct dengan nama **'Buku'** untuk merepresentasikan informasi tentang sebuah buku. Perhatikan contoh deklarasi struct pada Gambar 9.2 berikut.

```
#2 Deklarasi pada fungsi main
struct Buku{
  std::string judul;
  std::string penulis;
  std::string penerbit;
  int tahunTerbit;
  float harga;
};

int main()
{
  struct Buku Buku1, Buku2, Buku3;
  return 0;
}
```

```
#1 Deklarasi langsung pada struct
struct Buku{
  std::string judul;
  std::string penulis;
  std::string penerbit;
  int tahunTerbit;
  float harga;
}Buku1, Buku2, Buku3;

int main()
{
  return 0;
}
```

Gambar 9.2. Deklarasi Struct

Huruf pertama dari nama struct pada umumnya ditulis dengan huruf kapital dan setiap anggota struct dideklarasikan sesuai dengan tipe data dan namanya masing-masing.

9.2.2 Inisialisasi dan Mengakses Struct

Metode paling umum untuk menginisialisasi anggota struct adalah dengan menggunakan operator **dot (.)** atau biasa disebut dengan *member access operator.* Perhatikan contoh berikut untuk menginisialisasi dan mengakses anggota struct '**Buku'**.

```
#include <iostream>
struct Buku{
    std::string judul;
    std::string penulis;
    std::string penerbit;
    int tahunTerbit;
    float harga;
};

int main() {
    struct Buku Buku1;
    // Inisialisasi nilai anggota struct
    Buku1.judul = "Pemrograman C++ for Beginners";
    Buku1.penulis = "Wulan Sri Lestari";
    Buku1.penerbit = "Get Press Indonesia";
    Buku1.tahunTerbit = 2024;
    Buku1.harga = 75000;

    // Menampilkan informasi buku
    std::cout << "Informasi Buku 1\n";
    std::cout << "Judul: " << Buku1.judul << "\n";
    std::cout << "Pengarang: " << Buku1.penulis << "\n";
    std::cout << "Pengarang: " << Buku1.penerbit << "\n";
    std::cout << "Tahun Terbit: " << Buku1.tahunTerbit << "\n";
    std::cout << "Harga: $" << Buku1.harga << "\n";

    return 0;
}
```

Pada contoh di atas, inisialisasi struct '**Buku'** dilakukan dengan memanggil setiap anggota struct. Selain dengan cara di atas, anggota struct juga dapat diinisialisasi menggunakan kurung kurawal **{ }** dan setiap variabel secara berurutan dipisah dengan tanda koma sebagai berikut.

```cpp
#include <iostream>
struct Buku{
    std::string judul;
    std::string penulis;
    std::string penerbit;
    int tahunTerbit;
    float harga;
};

int main() {
    struct Buku Buku1 = {"Pemrograman C++ for Beginners",
                         "Wulan Sri Lestari", "Get Press Indonesia",
                         2024, 75000};

    // Menampilkan informasi buku
    std::cout << "Informasi Buku 1\n";
    std::cout << "Judul: " << Buku1.judul << "\n";
    std::cout << "Pengarang: " << Buku1.penulis << "\n";
    std::cout << "Pengarang: " << Buku1.penerbit << "\n";
    std::cout << "Tahun Terbit: " << Buku1.tahunTerbit << "\n";
    std::cout << "Harga: $" << Buku1.harga << "\n";

    return 0;
}
```

Pada program di atas, maka **'Buku1.judul'** akan berisi nilai "Pemrograman C++ for Beginners", **'Buku1.penulis'** berisi "Wulan Sri Lestari", **'Buku1.penerbit'** berisi "Get Press Indonesia", **'Buku1.tahunTerbit'** berisi nilai 2024, dan **'Buku1.harga'** berisi nilai 75000 sesuai dengan urutan nilai yang dimasukkan di dalam kurung kurawal.

Note:

1. Anggota dalam struct tidak dapat diinisialisasi saat mendeklarasikan anggotanya. Perhatikan contoh inisialisasi saat deklarasi berikut.

```
struct Buku{
  std::string judul = "Pemrograman C++";
  std::string penulis = "Wulan Sri Lestari";
  std::string penerbit= "Get Press Indonesia";
  int tahunTerbit = 2024;
  float harga = 75000;
};
```

Tidak dapat dilakukan inisialisasi saat deklarasi

```
int main()
{
  return 0;
}
```

2. Ketika tipe data anggota dalam struct dideklarasikan, tidak akan ada memori yang dialokasikan untuk menyimpan nilai. Memori dialokasikan hanya ketika variabel dibuat.

9.3 Struct sebagai Array

Penggunaan struct sebagai array memungkinkan kita untuk menyimpan dan mengelola kumpulan data terstruktur secara efisien. Struct dapat dideklarasikan sebagai array dengan cara menggabungkan keduanya. Sebagai contoh, jika kita memiliki struct '**Buku'** dan ingin membuat array buku, kita dapat mendeklarasikan array sebagai berikut:

```
#include <iostream>
struct Buku{
    std::string judul;
    std::string penulis;
    std::string penerbit;
    int tahunTerbit;
    float harga;
};

int main() {
    //Deklarasi struct sebagai Array
    struct Buku buku [10];

    return 0;
}
```

Pada contoh di atas, kita membuat struct **'Buku'** dengan array buku yang dapat menampung informasi sebanyak 10 buku.

9.3.1 Inisialisasi dan Akses Data Sebagai Array

Setelah deklarasi, kita dapat menginisialisasi dan mengakses data dalam array struct seperti mengelola array pada umumnya. Contoh, kita akan menginisialisasi dan mengakses struct '**buku'** indeks ke-0 seperti berikut.

```
#include <iostream>
struct Buku{
    std::string judul;
    std::string penulis;
    std::string penerbit;
    int tahunTerbit;
    float harga;
};

int main() {
    //Deklarasi struct sebagai Array
    struct Buku buku [10];

    //Inisialisasi informasi buku indeks-0
    buku[0].judul = "Pemrograman C++";
    buku[0].penulis = "Wulan Sri Lestari";
    buku[0].penerbit = "Get Press Indonesia";
    buku[0].tahunTerbit = 2024;
    buku[0].harga = 75000;

    // Menampilkan informasi buku indeks-0
    std::cout << "Informasi Buku 1\n";
    std::cout << "Judul: " << buku[0].judul << "\n";
    std::cout << "Pengarang: " << buku[0].penulis << "\n";
    std::cout << "Pengarang: " << buku[0].penerbit << "\n";
    std::cout << "Tahun Terbit: " << buku[0].tahunTerbit << "\n";
    std::cout << "Harga: $" << buku[0].harga << "\n";
    return 0;
}
```

Pad program di atas, kita telah menginisialisasi buku indeks-0 dengan judul "Pemrograman C++", penulis "Wulan Sri Lestari", penerbit "Get Press Indonesia", tahunTerbit 2024, dan harga 75000.

Untuk menampilkan informasi buku, maka wajib menggunakan indeks dan diikuti dengan operator dot dan anggota struct.

9.3.2 Inisialisasi dan Akses Data Menggunakan Perulangan

Inisialisasi struct sebagai array juga dapat dilakukan dengan menggunakan perulangan. Perhatikan program berikut ini untuk mengisi nilai struct sebagai array dengan perulangan.

```
#include <iostream>
struct Buku{
    std::string judul;
    std::string penulis;
    std::string penerbit;
    int tahunTerbit;
    float harga;
};

int main() {
    int JUMLAH_BUKU = 10; // Tentukan jumlah buku
    Buku daftarBuku[JUMLAH_BUKU]; // Deklarasi array buku
    // Input informasi buku menggunakan perulangan
    for (int i = 0; i < JUMLAH_BUKU; ++i) {
        std::cout << "Masukkan informasi buku ke-" << i + 1 << ":\n";
        std::cout << "Judul: ";
        std::getline(std::cin, daftarBuku[i].judul);
        std::cout << "Penulis: ";
        std::getline(std::cin, daftarBuku[i].penulis);
        std::cout << "Penerbit: ";
        std::getline(std::cin, daftarBuku[i].penerbit);
        std::cout << "Tahun Terbit: ";
        std::cin >> daftarBuku[i].tahunTerbit;
        std::cout << "Harga: $";
        std::cin >> daftarBuku[i].harga;
        // Membersihkan buffer setelah input
        std::cin.ignore();
        std::cout << "\n";
    }
    // Menampilkan informasi buku yang telah diinput
    std::cout << "Informasi Buku:\n";
    for (int i = 0; i < JUMLAH_BUKU; ++i) {
        std::cout << "Buku ke-" << i + 1 << ":\n";
        std::cout << "Judul: " << daftarBuku[i].judul << "\n";
        std::cout << "Penulis: " << daftarBuku[i].penulis << "\n";
        std::cout << "Penerbit: " << daftarBuku[i].penerbit << "\n";
        std::cout << "Tahun Terbit: " << daftarBuku[i].tahunTerbit << "\n";
        std::cout << "Harga: $" << daftarBuku[i].harga << "\n\n";
    }

    return 0;
}
```

Program di atas, variabel JUMLAH_BUKU digunakan sebagai variabel yang akan menentukan jumlah buku yang akan dimasukkan ke dalam array struct '**daftarBuku**' dan digunakan dalam perulangan saat memasukkan dan mencetak informasi buku. Perhatikan dalam memasukkan dan menampilkan informasi ke dalam struct '**daftarBuku'** wajib menggunakan indeks.

9.4 Struct sebagai Parameter Fungsi

Struct dalam pemrograman C++ dapat digunakan sebagai parameter fungsi untuk mempermudah dan mengorganisir pengiriman data yang terstruktur. Hal ini berguna ketika kita ingin mengoperasikan atau memanipulasi data dari tipe struct tertentu dalam fungsi tertentu.

9.4.1 Inisialisasi dan Akses Data

Pertama-tama, kita perlu mendefinisikan struct yang akan digunakan. Dalam contoh ini, kita akan menggunakan struct **'Buku'** yang memiliki beberapa anggota seperti judul, penulis, penerbit, tahun terbit, dan harga. Struct dapat digunakan sebagai parameter fungsi dengan menyertakan tipe struct tersebut di dalam daftar parameter fungsi. Contoh penggunaan struct sebagai parameter fungsi untuk menampilkan informasi buku sebagai berikut.

```cpp
#include <iostream>
// Mendefinisikan struct Buku
struct Buku {
    std::string judul;
    std::string penulis;
    std::string penerbit;
    int tahunTerbit;
    float harga;
};
// Fungsi untuk menampilkan informasi buku dengan parameter struct Buku
void tampilkanInformasiBuku(Buku buku) {
    std::cout << "Informasi Buku\n";
    std::cout << "Judul: " << buku.judul << "\n";
    std::cout << "Penulis: " << buku.penulis << "\n";
    std::cout << "Penerbit: " << buku.penerbit << "\n";
    std::cout << "Tahun Terbit: " << buku.tahunTerbit << "\n";
    std::cout << "Harga: $" << buku.harga << "\n";
}

int main() {
    // Membuat variabel struct Buku
    Buku buku = {"Pemrograman C++", "Wulan Sri Lestari",
                 "Get Press Indonesia", 2024, 75000};
    // Memanggil fungsi dengan struct sebagai parameter
    tampilkanInformasiBuku(buku);

    return 0;
}
```

Ketika program menjalankan fungsi '**tampilkan Informasi Buku**' maka akan ditampilkan informasi buku sesuai dengan informasi yang telah diisi pada struct buku.

9.5 Struct sebagai Pointer

Struct dalam pemrograman C++ dapat digunakan sebagai parameter pointer untuk mengoperasikan dan memanipulasi data secara efisien. Penggunaan pointer memungkinkan untuk mengakses dan mengubah nilai dari variabel struct secara langsung. Mendefinisikan pointer ke struct sama seperti mendefinisikan pointer ke variabel biasa. Perhatikan contoh berikut.

*int *variabelPointer; // Definisi pointer pada variabel biasa*
*struct Buku *structBukuPointer; // Definisi pointer pada struct*

structBukuPointer* berfungsi untuk menyimpan alamat dari variabel struct dan untuk menampilkan alamat variabel struct, kita dapat menggunakan operator **'&' sebelum nama struct. Contoh.

alamatStructBuku = & *structBukuPointer;*

variabel *alamatStructBuku* berisi alamat memory dari variabel struct pointer '*structBukuPointer'.*

9.5.1 Inisialisasi dan Akses Data Struct Pointer

Pertama-tama, kita perlu mendefinisikan struct yang akan digunakan. Dalam contoh ini, kita akan menggunakan struct **'Buku'**. Struct dapat digunakan sebagai parameter pointer dengan mendeklarasikan parameter fungsi sebagai pointer ke tipe struct. Contoh penggunaan struct sebagai parameter pointer untuk mengubah informasi buku sebagai contoh:

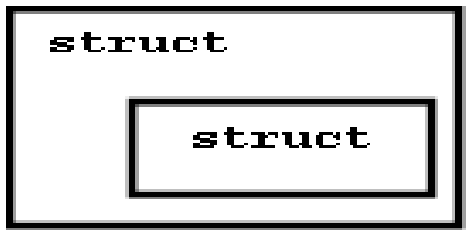
```
#include <iostream>
// Mendefinisikan struct Buku
struct Buku {
    std::string judul;
    std::string penulis;
    std::string penerbit;
    int tahunTerbit;
    float harga;
};

// Fungsi untuk mengubah informasi buku menggunakan pointer
void ubahInformasiBuku(Buku* buku) {
    buku->judul = "Pemrograman C++ Revisited";
    buku->penulis = "Wulan Sri Lestari";
    buku->penerbit = "Get Press Indonesia";
    buku->tahunTerbit = 2024;
    buku->harga = 90000;
}
```

```
int main() {
    // Membuat variabel struct Buku
    Buku contohBuku = {"Pemrograman C++", "Lestari", "GetPress", 2024, 75000};

    // Menampilkan informasi buku sebelum perubahan
    std::cout << "Informasi Buku Sebelum Perubahan\n";
    std::cout << "Judul: " << contohBuku.judul << "\n";
    std::cout << "Penulis: " << contohBuku.penulis << "\n";
    std::cout << "Penerbit: " << contohBuku.penerbit << "\n";
    std::cout << "Tahun Terbit: " << contohBuku.tahunTerbit << "\n";
    std::cout << "Harga: $" << contohBuku.harga << "\n\n";

    // Memanggil fungsi dengan struct sebagai parameter pointer
    ubahInformasiBuku(&contohBuku);

    // Menampilkan informasi buku setelah perubahan
    std::cout << "Informasi Buku Setelah Perubahan\n";
    std::cout << "Judul: " << contohBuku.judul << "\n";
    std::cout << "Penulis: " << contohBuku.penulis << "\n";
    std::cout << "Penerbit: " << contohBuku.penerbit << "\n";
    std::cout << "Tahun Terbit: " << contohBuku.tahunTerbit << "\n";
    std::cout << "Harga: $" << contohBuku.harga << "\n";

    return 0;
}
```

Pada fungsi '**ubahInformasiBuku'** struct pointer digunakan sebagai parameter fungsi. Untuk mengubah setiap variabel struct pointer pada fungsi maka wajib menggunakan operator "**->**" untuk mengakses nilai dari anggota struct yang menggunakan pointer.

9.6 Nested Struct

Nested struct adalah konsep dalam pemrograman C++ di mana sebuah struct didefinisikan di dalam struct lainnya. Dengan kata lain, kita dapat menempatkan satu struct ke dalam anggota lain dari struct yang lebih besar.



Hal ini memungkinkan pengelompokan data yang terkait secara hierarkis dan terorganisir. Contoh sederhana nested struct dapat ditemukan dalam situasi di mana sebuah objek memiliki bagian-bagian yang lebih terinci. Misalnya, kita memiliki struct "Buku" yang memiliki informasi tentang **tanggalTerbit** yang menyimpan informasi tentang tanggal terbit buku. Perhatian contoh berikut.

```
#include <iostream>

// Mendefinisikan nested struct TanggalTerbit
struct TanggalTerbit {
    int hari;
    int bulan;
    int tahun;
};
// Mendefinisikan struct Buku dengan nested struct
struct Buku {
    std::string judul;
    std::string penulis;
    std::string penerbit;
    TanggalTerbit tanggalTerbit; // Nested struct
    float harga;
};
```

Nested struct dapat diakses melalui variabel struct yang bersangkutan. Dalam contoh ini, kita dapat mengakses anggota tanggalTerbit dari struct Buku seperti berikut:

```
int main(){
    // Membuat variabel struct Buku
    Buku buku = {"Pemrograman C++", "Wulan Sri Lestari",
                 "GetPress Indonesia", {12, 2, 2024}, 89000};

    // Mengakses dan menampilkan informasi tanggal terbit
    std::cout << "Tanggal Terbit: " << buku.tanggalTerbit.hari << "/"
              << buku.tanggalTerbit.bulan << "/"
              << buku.tanggalTerbit.tahun << "\n";

    return 0;
}
```

Pada contoh di atas, untuk mengakses variabel hari, bulan, dan tahun maka kita harus memanggil struct '**buku'** terlebih dahulu kemudian diikuti dengan nested struct '**tanggalTerbit'** sehingga seluruh variabel dalam nested struct dapat ditampilkan.

Latihan

1. Buatlah struct **'Barang'** yang berisi informasi tentang kode barang, nama, harga, dan jumlah stok. Tulis program yang memungkinkan pengguna untuk mengelola data barang, seperti menambah stok, mengupdate harga, dan menampilkan daftar barang.
2. Buatlah nested struct tentang '**Barang'** dan '**DetailBarang**' kemudian manfaat array dan pointer untuk mengelolah nested struct.

DAFTAR PUSTAKA

Malhotra, D., & Malhotra, N. 2023. C*++ Programming Fundamentals.* Dulles: Mercury Learning and Information LLC.

Tselikis, George S. 2023. *Introduction to C++*. London New York: CRC Press.

Uzayr, Sufyan. 2022. *Mastering C++ Programming Language.* London New York: CRC Press.

Stroustrup, Bjarne. 2014. *Programming: Principles and Practice Using C++ (2nd Edition).* United States of America: Pearson Education, Inc.

Josuttis, Nicolai M. 2012. *The C++ Standard Library : a Tutorial and Reference (2nd Edition)*. United States of America: Pearson Education, Inc.

cplusplus.com. Structures (C++). Tersedia di: https://www.cplusplus.com/doc. [Diakses pada 23 Januari 2024].

BAB 10
FUNGSI REKURSIF

Oleh Hendra Nusa Putra

10.1 Pengenalan Fungsi Rekursif dalam Bahasa Pemrograman C++

Fungsi rekursif adalah konsep penting dalam bahasa pemrograman C++. Fungsi jenis ini merujuk pada kemampuan sebuah fungsi untuk memanggil dirinya sendiri. Konsep ini mungkin terdengar agak rumit bagi pemula dalam pemrograman, tetapi dengan pemahaman yang tepat, dapat membantu Anda menyelesaikan masalah yang lebih kompleks dengan lebih mudah.(Kumparan 2022)

10.1.1 Apa itu Fungsi Rekursif dan Bagaimana Cara Kerjanya?

Fungsi rekursif adalah fungsi yang memanggil dirinya sendiri. Ketika fungsi tersebut dipanggil, program akan masuk ke dalam blok kode fungsi itu sendiri secara berulang sampai suatu kondisi terpenuhi. Pada saat kondisi tersebut terpenuhi, program keluar dari fungsi dan melanjutkan eksekusi ke bagian berikutnya. Misalnya, kita memiliki fungsi rekursif sederhana yang menghitung faktorial angka:

```
int faktorial(int n) {
 if (n == 0 || n == 1) {
  return 1;
 } else {
  return n * faktorial(n - 1);
 }
}
```

Fungsi ini akan memanggil dirinya sendiri dengan mengurangi parameter n hingga mencapai kondisi basis, yaitu ketika n adalah 0 atau 1. Saat kondisi ini terpenuhi, fungsi akan kembali ke pemanggilnya dan mengalikan hasilnya dengan n, dan

seterusnya hingga mencapai angka asli yang ingin dihitung faktorialnya.

10.1.2 Manfaat dan Kelebihan Fungsi Rekursif dalam Bahasa Pemrograman C++

Penggunaan fungsi rekursif dalam bahasa pemrograman C++ memiliki beberapa manfaat dan kelebihan. Berikut ini adalah beberapa di antaranya:

1. **Simpel dan Elegan**: Dalam beberapa kasus, pemecahan masalah dengan menggunakan fungsi rekursif dapat lebih sederhana dan elegan daripada pendekatan lainnya. Konsep rekursif memungkinkan untuk mengekspresikan gagasan secara terstruktur dan terorganisir dengan kode yang lebih bersih dan rapi.
2. **Kemampuan Memahami Masalah yang Kompleks**: Fungsi rekursif sangat berguna dalam memecahkan masalah yang kompleks. Dalam beberapa situasi, masalah kompleks dapat dibagi menjadi submasalah yang lebih kecil dan lebih mudah dipecahkan melalui pemanggilan rekursif. Ini memungkinkan pemecahan masalah yang lebih terstruktur dan memudahkan pemahaman kita terhadap masalah yang lebih besar.
3. **Fleksibilitas**: Fungsi rekursif memberikan fleksibilitas dalam pemrograman karena memungkinkan kita untuk melakukan pemanggilan yang berulang tanpa batasan jumlah pemanggilan. Hal ini memungkinkan penggunaan fungsi rekursif dalam berbagai aplikasi dan pemecahan masalah.

10.1.3 Dampak Fungsi Rekursif terhadap Efisiensi Kode

Meskipun fungsi rekursif mempunyai manfaat dan kelebihan, namun kita perlu mempertimbangkan dampaknya terhadap efisiensi kode. Pemanggilan fungsi rekursif dapat mempengaruhi kinerja dan penggunaan memori program kita.

Dalam beberapa kasus, menggantikan pemanggilan fungsi rekursif dengan pendekatan iteratif (pemanggilan menggunakan loop) dapat meningkatkan efisiensi kode. Karena setiap pemanggilan

fungsi rekursif akan menambah beban pada stack, dengan masalah yang lebih besar, dapat menyebabkan *stack overflow*.

Oleh karena itu, penting untuk memahami kapan menggunakan fungsi rekursif dan kapan menggunakan pendekatan iteratif dalam pemrograman. Hal ini dapat membantu memastikan efisiensi kode kita dan mencegah terjadinya *stack overflow* yang mengakibatkan program crash. Sekali lagi, fleksibilitas dalam pemilihan strategi pemrograman yang tepat adalah kunci untuk mengoptimalkan kode kita.

Dengan memahami konsep dan cara kerja fungsi rekursif, kita dapat meningkatkan kemampuan dalam pemrograman C++. Mengingat manfaatnya dapat memecahkan masalah secara elegan dan fleksibel, serta mempertimbangkan dampak pada efisiensi kode, kita dapat menjadi pemrogram yang lebih terampil dan efisien dalam menggunakan fungsi rekursif.

10.2 Langkah-langkah Menerapkan Fungsi Rekursif

10.2.1 Pengaturan Awal dan Membuat Fungsi Rekursif Dasar

Untuk memulai, Anda perlu memahami struktur dasar fungsi rekursif dalam bahasa pemrograman C++. Struktur dasar ini terdiri dari dua elemen utama: basis dan rekursi. Basis adalah kondisi saat fungsi berhenti memanggil dirinya sendiri, sedangkan rekursif adalah pemanggilan kembali fungsi pada dirinya sendiri.

Contoh Implementasi Fungsi Rekursif Sederhana

Mari kita lihat contoh implementasi fungsi rekursif sederhana dalam bahasa pemrograman C++. Misalkan kita ingin menghitung faktorial suatu bilangan menggunakan fungsi rekursif.

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

int faktorial(int n) {
 if (n == 0 || n == 1) {
  return 1;
 } else {
  return n * faktorial(n - 1);
 }
```

```
}

int main() {
  int bilangan;
  cout << "Masukkan bilangan: ";
  cin >> bilangan;
  cout << "Faktorial dari " << bilangan << " adalah " << faktorial(bilangan) << endl;
  return 0;
}
```

Dalam contoh di atas, kita membuat fungsi faktorial yang menerima satu parameter n yang merupakan bilangan yang ingin kita hitung faktorialnya. Pada bagian awal fungsi, kita mengecek apakah bilangan n adalah 0 atau 1. Jika ya, maka hasil faktorialnya adalah 1. Jika tidak, maka fungsi akan memanggil dirinya sendiri dengan parameter n - 1 dan mengalikan n dengan hasil rekursif tersebut. Proses ini berulang sampai basis tercapai.

Kelebihan dan Keterbatasan Penggunaan Fungsi Rekursif

Penggunaan fungsi rekursif memiliki beberapa kelebihan dan keterbatasan yang perlu dipertimbangkan.

Kelebihan Penggunaan Fungsi Rekursif:

1. *Elegan dan mudah dibaca*: Implementasi menggunakan fungsi rekursif sering kali lebih mudah dipahami dibandingkan dengan implementasi menggunakan perulangan.
2. *Penyelesaian masalah kompleks*: Fungsi rekursif dapat digunakan untuk menyelesaikan masalah kompleks dalam pemrograman seperti permutasi, kombinasi, dan pencarian dalam struktur data rekursif.

Keterbatasan Penggunaan Fungsi Rekursif:

1. *Memakan memori*: Pemanggilan rekursif yang dalam dan tidak efisien dapat memakan banyak memori sehingga dapat menyebabkan *stack overflow*.

2. *Kecepatan eksekusi*: Implementasi menggunakan rekursi sering kali lebih lambat dibandingkan dengan implementasi menggunakan perulangan.

Meskipun memiliki keterbatasan, penggunaan fungsi rekursif tetap menjadi teknik penting dalam pemrograman. Dengan pemahaman yang baik tentang struktur dasar dan langkah-langkah menerapkannya, pemula dalam bahasa pemrograman C++ dapat mengoptimalkan penggunaan fungsi rekursif dalam menyelesaikan masalah-masalah yang ditemui.

Dengan demikian, Anda telah mengenal lebih dalam tentang fungsi rekursif dalam bahasa pemrograman C++. Memahami konsep dasar dan langkah-langkah menerapkannya akan memberikan Anda fondasi yang baik untuk mengoptimalkan implementasi dan menyelesaikan masalah dengan efisien menggunakan teknik ini.

10.2.2 Menerapkan Fungsi Rekursif dalam Kasus-Kasus Khusus

1. **Perhitungan Faktorial**
   Faktorial dari suatu bilangan nn (ditulis sebagai n!n!) adalah produk dari semua bilangan bulat positif sampai dengan nn. Fungsi rekursif untuk menghitung faktorial sangat sederhana dan merupakan salah satu contoh klasik penggunaan rekursif.
2. **Menyelesaikan Masalah Fibonacci**
   Deret Fibonacci adalah seri angka di mana setiap angka adalah penjumlahan dua angka sebelumnya. Fungsi rekursif dapat digunakan untuk menghitung angka pada posisi tertentu dalam deret Fibonacci.
3. **Pencarian dan Pengurutan**
   Algoritma seperti Binary Search dan Quick Sort memanfaatkan rekursi untuk memecah masalah menjadi sub-masalah yang lebih kecil, yang kemudian diselesaikan secara rekursif.
4. **Menyelesaikan Tower of Hanoi**
   Tower of Hanoi adalah teka-teki klasik yang solusinya dapat dengan elegan diwakili menggunakan fungsi rekursif. Ini

melibatkan pemindahan cakram dari satu tiang ke tiang lain dengan aturan tertentu.

5. **Menjelajahi Struktur Data Pohon**
   Dalam struktur data seperti pohon biner, fungsi rekursif sering digunakan untuk menjelajahi (traversal) pohon, seperti dalam pre-order, in-order, dan post-order traversal.
6. **Algoritma Backtracking**
   Digunakan dalam berbagai masalah seperti N-Queens, Labirin, dan Subset Sum. Fungsi rekursif memungkinkan kembali ke langkah sebelumnya (backtrack) dan mencoba alternatif lain.
7. **Permasalahan Pembagian dan Penguasaan (Divide and Conquer)**
   Konsep ini digunakan dalam berbagai algoritma, di mana masalah dibagi menjadi sub-masalah yang lebih kecil, diselesaikan secara rekursif, dan solusi sub-masalah digabungkan.
8. **Pemrosesan String dan Permutasi**
   Dalam kasus seperti permutasi string, fungsi rekursif dapat digunakan untuk menghasilkan semua kemungkinan pengaturan karakter.
9. **Dynamic Programming**
   Untuk masalah-masalah tertentu, pendekatan rekursif dengan penyimpanan hasil sub-problem (memoization) dapat sangat meningkatkan efisiensi.
10. **Algoritma Graf**
    Penjelajahan graf, seperti Depth-First Search (DFS), seringkali diimplementasikan dengan rekursif.

Contoh Kode

Misalnya, mari kita lihat contoh kode sederhana untuk fungsi rekursif yang menghitung faktorial :

```
def factorial(n):
  if n == 0 or n == 1:
    return 1
  else:
    return n * factorial(n-1)
```

Dan untuk Fibonacci :

```
def fibonacci(n):
  if n <= 1:
    return n
  else:
    return fibonacci(n-1) + fibonacci(n-2)
```

Penting untuk dicatat bahwa dalam fungsi rekursif, kasus basis (kondisi berhenti) sangat penting untuk mencegah rekursi tak terbatas. Juga, terkadang pendekatan rekursif bisa kurang efisien daripada iteratif, terutama dalam hal penggunaan memori dan waktu eksekusi, tergantung pada kasusnya.

10.2.3 Teknik Debugging untuk Meningkatkan Kualitas Kode Rekursif

Debugging kode rekursif dapat menjadi tantangan karena sifat rekursi itu sendiri, yang seringkali membuat alur eksekusi lebih sulit untuk diikuti. Berikut adalah beberapa teknik dan strategi yang dapat digunakan:

1. **Pemahaman Mendalam tentang Kasus Basis dan Kasus Rekursif**
   Kasus Basis: Pastikan kasus basis (kondisi pemberhentian) Anda benar. Kesalahan dalam kasus basis bisa menyebabkan rekursi tak terbatas atau kondisi berhenti prematur.
   Kasus Rekursif: Pahami dengan jelas bagaimana setiap panggilan rekursif mengubah keadaan masalah. Setiap panggilan harus mendekatkan ke solusi.
2. **Visualisasi Panggilan Fungsi**
   Gunakan diagram atau pohon panggilan rekursif untuk memvisualisasikan bagaimana fungsi dipanggil dan bagaimana kasus basis dicapai.
   Alat seperti Python Tutor dapat membantu dalam visualisasi ini.

3. **Penyertaan Pernyataan Print untuk Pelacakan**
   Sertakan pernyataan cetak dalam fungsi untuk melacak nilai-nilai variabel dan alur panggilan. Ini membantu dalam memahami urutan dan konteks eksekusi fungsi rekursif.
4. **Pembatasan Rekursi**
   Tetapkan batas atas untuk rekursif untuk menghindari rekursif tak terbatas. Python, misalnya, memiliki batas rekursif default yang mencegah overflow stack.
5. **Pemecahan Masalah Secara Iteratif**
   Kadang-kadang, menulis solusi yang sama secara iteratif dapat membantu memahami logika yang harus diikuti oleh versi rekursif.
6. **Memanfaatkan Debugger**
   Gunakan debugger untuk melangkah melalui setiap panggilan rekursif. Ini memungkinkan inspeksi kondisi saat ini dari stack panggilan, variabel, dan alur program.
7. **Uji Kasus Tepi dan Kasus Kecil**
   Mulai dengan kasus yang sangat sederhana untuk memastikan bahwa fungsi rekursif berfungsi seperti yang diharapkan. Kemudian, uji kasus tepi (seperti nilai negatif atau nol, tergantung pada masalahnya) untuk memastikan robustness.
8. **Memperhatikan Stack Overflow**
   Kesalahan umum dalam fungsi rekursif adalah stack overflow. Jika fungsi rekursif Anda memicu kesalahan ini, Anda mungkin perlu mengoptimasi rekursi Anda atau mengubah pendekatan.
9. **Refactoring dan Pengoptimalan**
   Sesuaikan dan optimalkan kode rekursif Anda. Pertimbangkan teknik seperti memoization untuk menghindari perhitungan berulang dalam kasus rekursif yang berat.
10. **Review Kode dan Peer Feedback**
    Dapatkan feedback dari orang lain. Review peer dapat mengungkapkan kesalahan atau area untuk perbaikan yang mungkin Anda lewatkan.

Contoh Debugging
Misalkan Anda memiliki fungsi rekursif sederhana dan Anda ingin men-debugnya. Mari kita gunakan contoh faktorial lagi:

```
def factorial(n):
  assert n >= 0, "Faktorial dari bilangan negatif tidak didefinisikan"
  if n == 0 or n == 1:
    return 1
  else:
    return n * factorial(n-1)
```

Dalam proses debugging, Anda bisa menambahkan print statement untuk melacak nilai n pada setiap panggilan:

```
def factorial(n):
  print(f"Memanggil faktorial({n})")
  assert n >= 0, "Faktorial dari bilangan negatif tidak didefinisikan"
  if n == 0 or n == 1:
    return 1
  else:
    return n * factorial(n-1)
```

Dalam proses debugging, Anda bisa menambahkan print statement untuk melacak nilai n pada setiap panggilan:

```
def factorial(n):
  print(f"Memanggil faktorial({n})")
  assert n >= 0, "Faktorial dari bilangan negatif tidak didefinisikan"
  if n == 0 or n == 1:
    return 1
  else:
    return n * factorial(n-1)
```

10.3 Prinsip-Prinsip Desain Fungsi Rekursif yang Efisien

Dalam membahas prinsip-prinsip desain fungsi rekursif yang efisien, ada beberapa aspek penting yang perlu diperhatikan. Berikut adalah penjelasan terkait dengan prinsip-prinsip tersebut:

1. **Basis Rekursi (Base Case)**
   Definisi yang Jelas : Basis rekursif harus jelas dan menangani kasus-kasus terkecil atau paling sederhana dari masalah.
   Pencegahan Loop Tak Terbatas : Basis rekursi mencegah rekursif berlanjut secara tak terbatas dan memastikan rekursif berakhir.
2. **Langkah Rekursi (Recursive Step)**
   Mendekati Basis : Setiap pemanggilan rekursif harus mendekatkan masalah ke basis rekursif.
   Pemecahan Masalah menjadi Sub-masalah yang Lebih Kecil : Fungsi harus memecah masalah menjadi sub-masalah yang lebih kecil dan menangani sub-masalah tersebut secara rekursif.
3. **Efisiensi Komputasi**
   Menghindari Pengulangan Perhitungan : Gunakan teknik seperti memoization untuk menyimpan hasil perhitungan yang sudah dilakukan.
   Kompleksitas Waktu dan Ruang : Pertimbangkan kompleksitas waktu dan ruang. Rekursif yang tidak efisien dapat menyebabkan overhead memori yang besar atau waktu eksekusi yang lambat.

   Contoh Fibonacci dengan Memoization dalam Python:

```
memo = {}

def fibonacci(n):
  if n in memo:
    return memo[n]
  if n <= 2:
    return 1
  else:
```

```
    result=fibonacci(n - 1)+fibonacci(n-2)
    memo[n] = result
    return result
```

4. **Keterbacaan dan Kejelasan**
   Kode yang Mudah Dipahami : Tulis fungsi rekursif dengan cara yang mudah dipahami dan diikuti.
   Komentar dan Dokumentasi : Berikan komentar dan dokumentasi yang cukup untuk menjelaskan bagaimana rekursi bekerja.
5. **Kasus Uji (Test Cases)**
   Melakukan Pengujian yang Menyeluruh : Uji fungsi rekursif dengan berbagai kasus uji, termasuk kasus-kasus ekstrem.
   Validasi Hasil : Pastikan bahwa hasil rekursi valid dan sesuai dengan harapan.
6. **Penggunaan Sumber Daya**
   Stack Overflow : Waspadai potensi stack overflow pada rekursif yang sangat dalam.
   Optimasi Tail Recursion : Jika memungkinkan, gunakan optimasi tail recursion untuk mengurangi beban pada stack pemanggilan.

   Berikut contohnya dalam bahasa pemrograman Python:

```
def factorial(n, result=1):
  if n == 0:
    return result
  else:
    return factorial(n - 1, result * n)
```

7. **Pemilihan Kasus untuk Rekursif**
   Sesuai untuk Masalah : Pastikan bahwa rekursi adalah pendekatan yang tepat untuk masalah yang dihadapi.
   Alternatif Iteratif : Pertimbangkan apakah solusi iteratif mungkin lebih efisien dari segi waktu dan ruang.

10.3.1 Meningkatkan Keterbacaan Kode Rekursif

1. Nama Fungsi dan Variabel yang Deskriptif
   Penting untuk menggunakan nama fungsi dan variabel yang deskriptif agar kode rekursif Anda mudah dipahami oleh orang lain (atau bahkan oleh Anda sendiri di masa depan). Nama yang jelas akan membantu dalam memahami tujuan dan fungsionalitas setiap bagian kode. Misalnya :

```
# Versi tidak jelas
def f(n):
    if n <= 0:
        return 0
    else:
        return n + f(n - 1)
```

2. Komentari Kode Anda
   Tambahkan komentar yang jelas untuk menjelaskan langkah-langkah utama dalam fungsi rekursif Anda. Ini akan membantu pembaca untuk memahami alur logika yang Anda implementasikan. Misalnya :

```
def fibonacci(n):
    # Basis case
    if n <= 1:
        return n
    # Hitung Fibonacci untuk n-1 dan n-2
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
```

3. Gunakan Indentasi yang Konsisten
   Pastikan untuk mengatur indentasi kode Anda dengan konsisten. Ini akan membuat kode rekursif Anda lebih mudah dibaca dan dipahami. Gunakan jumlah spasi atau tab yang sama di seluruh kode. Contohnya:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)
```

4. Pertimbangkan Penggunaan Helper Function
   Jika Anda merasa kode rekursif Anda menjadi terlalu rumit, pertimbangkan untuk menggunakan fungsi bantuan (helper function) yang dapat memecah masalah menjadi bagian yang lebih kecil dan lebih mudah dipahami. Ini juga dapat meningkatkan keterbacaan kode. Misalnya :

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr) // 2
    left_half = arr[:mid]
    right_half = arr[mid:]

    left_half = merge_sort(left_half)
    right_half = merge_sort(right_half)

    return merge(left_half, right_half)
#Fungsi helper untuk menggabungkan dua list terurut
```

5. Tes dan Dokumentasikan Kode
   Terakhir, pastikan Anda menguji kode rekursif Anda dengan baik dan mendokumentasikannya. Dokumentasi akan membantu orang lain untuk memahami cara menggunakan fungsi Anda dengan benar. Juga, pastikan untuk melakukan uji coba dengan berbagai kasus uji, termasuk kasus dasar dan kasus ekstrem.

10.3.2 Menerapkan Rekursif pada Struktur Data yang Lebih Kompleks

Rekursif tidak hanya dapat diterapkan pada fungsi sederhana, tetapi juga pada struktur data yang lebih kompleks seperti pohon (tree), graf (graph), atau bahkan struktur data yang Anda definisikan sendiri. Berikut adalah beberapa poin penting dalam menerapkan rekursif pada struktur data yang lebih kompleks:

1. **Pemahaman Struktur Data**
   Sebelum menerapkan rekursi pada struktur data kompleks, Anda harus memahami dengan baik bagaimana struktur data tersebut bekerja. Misalnya, jika Anda akan menerapkan rekursi pada pohon biner, Anda perlu tahu tentang node, anak-anak (children), dan bagaimana cabang-cabang (branches) berhubungan satu sama lain.
2. **Mendefinisikan Kasus Basis**
   Setiap fungsi rekursif perlu memiliki kasus basis yang mendefinisikan kondisi di mana rekursi berhenti. Dalam konteks struktur data kompleks, kasus basis sering kali terkait dengan node atau elemen terakhir dalam struktur data. Misalnya, jika Anda bekerja dengan pohon biner, kasus basis dapat didefinisikan ketika Anda mencapai node daun (leaf node).
3. **Pemanggilan Rekursif**
   Pada setiap langkah dalam rekursi, Anda harus memanggil fungsi rekursif untuk mengatasi sub-masalah yang lebih kecil. Dalam konteks struktur data kompleks, ini mungkin berarti bergerak dari satu node ke node lain dalam pohon atau mengunjungi simpul (vertex) yang terhubung dalam graf.
4. **Penanganan Data**
   Selama proses rekursi, Anda harus memikirkan bagaimana data diubah atau diakses pada setiap langkah. Ini bisa melibatkan penambahan atau pengurangan data, pencarian, atau manipulasi data lainnya sesuai dengan struktur data yang Anda kerjakan.
5. **Memperhatikan Efisiensi**
   Pada struktur data yang lebih kompleks, penting untuk mempertimbangkan efisiensi. Penggunaan memori dan waktu komputasi harus diperhatikan agar rekursif tidak menjadi terlalu lambat atau memakan banyak sumber daya. Anda bisa menerapkan prinsip-prinsip pengoptimalan yang telah dibahas sebelumnya.

Contoh: Rekursif pada Pohon Biner

Misalnya, jika Anda ingin menerapkan rekursif pada pohon biner untuk melakukan pencarian dalam pohon, Anda dapat mengikuti pola seperti ini:

```
class TreeNode:
  def __init__(self, value):
    self.value = value
    self.left = None
    self.right = None

def search_in_binary_tree(node, target):
  if node is None:
    return False
  if node.value == target:
    return True
  left_result = search_in_binary_tree(node.left, target)
  right_result = search_in_binary_tree(node.right, target)
  return left_result or right_result
```

10.4 Kesimpulan

Berikut adalah beberapa kesimpulan penting tentang Fungsi Rekursif:

1. **Pengenalan Fungsi Rekursif**: Fungsi rekursif adalah konsep penting dalam bahasa pemrograman, memungkinkan fungsi untuk memanggil dirinya sendiri. Dengan pemahaman yang tepat, ini dapat membantu menyelesaikan masalah yang lebih kompleks.
2. **Apa itu Fungsi Rekursif:** Fungsi rekursif adalah fungsi yang memanggil dirinya sendiri, dan ini terjadi sampai kondisi basis tercapai. Contoh faktorial adalah ilustrasi yang baik.
3. **Manfaat dan Kelebihan**: Penggunaan fungsi rekursif memiliki manfaat seperti kesederhanaan, kemampuan pemahaman masalah yang kompleks, dan fleksibilitas dalam pemrograman.
4. **Dampak terhadap Efisiensi Kode**: Fungsi rekursif bisa mempengaruhi kinerja dan penggunaan memori. Dalam beberapa kasus, rekursif dapat diganti dengan pendekatan iteratif untuk meningkatkan efisiensi kode.
5. **Langkah-langkah Menerapkan Fungsi Rekursif:** Langkah-langkah awal dalam menerapkan fungsi rekursif termasuk

memahami basis rekursi dan langkah rekursi. Ini adalah dasar dari desain rekursi yang efisien.

6. **Kelebihan dan Keterbatasan Penggunaan Fungsi Rekursif:** Fungsi rekursif dapat digunakan dalam berbagai kasus, tetapi perlu memperhatikan keterbatasannya terkait memori dan waktu eksekusi.
7. **Teknik Debugging**: Debugging kode rekursif memerlukan pemahaman mendalam tentang kasus basis dan kasus rekursif, serta penggunaan alat bantu seperti cetak dan visualisasi.
8. **Prinsip-Prinsip Desain Fungsi Rekursif yang Efisien:** Desain fungsi rekursif yang efisien melibatkan pemahaman tentang basis rekursi, langkah rekursi, efisiensi komputasi, keterbacaan, kasus uji, penggunaan sumber daya, pemilihan kasus rekursif, dan pengoptimalan tail recursion.
9. **Meningkatkan Keterbacaan Kode Rekursif:** Keterbacaan kode rekursif dapat ditingkatkan dengan menggunakan nama fungsi dan variabel yang deskriptif, komentar, indentasi yang konsisten, penggunaan fungsi bantu, dan pengujian serta dokumentasi kode.
10. **Menerapkan Rekursi pada Struktur Data yang Lebih Kompleks:** Rekursi dapat diterapkan pada struktur data kompleks seperti pohon, graf, atau struktur data khusus lainnya dengan memahami struktur data, mendefinisikan kasus basis, pemanggilan rekursif, penanganan data, dan memperhatikan efisiensi.

DAFTAR PUSTAKA

George, Otu., M., S., Oyebanji., F, Okonkwo., R., U., Ugbe., Andrew, Chukwuemeka, Okafor., Shehu, Usman. 2023. Evaluation, computation and coding of iterative function using recursive approach. Dutse Journal of Pure and Applied Sciences, 9(1b), 209-218. Available from: 10.4314/dujopas.v9i1b.20

Usmonov, Shavkatjon, Shukurovich. 2023. Improving Popular Textbook Recursive Algorithms with Tail Recursion. Available from: 10.1109/icrai57502.2023.10089594

Richard, Mikael, Slevinsky., Hassan, Safouhi. 2022. A recursive algorithm for an efficient and accurate computation of incomplete Bessel functions. Numerical Algorithms, 92(1), 973-983. Available from: 10.1007/s11075-022-01438-0

Kumparan. 2022. "Fungsi Rekursif Dan Contoh Nyatanya." *kumparan*. https://kumparan.com/how-to-tekno/fungsi-rekursif-dan-contoh-nyatanya-1xqEEPPHXFg.

Rodrigo, C., O., Rocha., Luís, F., W., Góes., Fernando, Magno, Quintão, Pereira. 2018. Automatic parallelization of recursive functions with rewriting rules. Science of Computer Programming, 173, 128-152. Available from: 10.1016/J.SCICO.2018.01.004

Manuel, Rubio-Sánchez. 2017. Introduction to Recursive Programming.

Alexandros, Agapitos., Michael, O'Neill., Ahmed, Kattan., Simon, M., Lucas. 2017. Recursion in tree-based genetic programming. Genetic Programming and Evolvable Machines, 18(2), 149-183. Available from: 10.1007/S10710-016-9277-5

Rashkovits, Rami., Ilana, Lavy. 2017. Interactive Debug and Exploration of Recursive Functions. International Journal of Education, 02

Jinping, Song. 2013. Discussion on Writing of Recursive Algorithm. International Journal of Hybrid Information Technology, 6(6), 127-134. Available from: 10.14257/IJHIT.2013.6.6.11

BAB 11
STRUCTURAL QUERY LANGUAGE (SQL) 2

Oleh Ade Wisandra

11.1 Sejarah Awal Mula Sql

SQL, atau *Structured Query Language*, adalah sebuah bahasa pemrograman yang digunakan untuk mengelola dan mengakses sistem manajemen basis data relasional (RDBMS). SQL dikembangkan pada tahun 1970-an oleh IBM, dan sejak itu telah menjadi standar de facto dalam pengelolaan data di berbagai platform. Dengan SQL, pengguna dapat membuat, mengubah, dan menghapus data dalam basis data, serta melakukan berbagai operasi lainnya seperti penggabungan data, pengurutan, dan pengelompokan.

Definisi *Structured Query Language* (SQL) adalah sekumpulan perintah khusus yang digunakan untuk mengakses data dalam database relasional. Untuk mencari data SQL pada query operator LIKE yang telah ditentukan dapat dilakukan melalui pencocokan string pada data yang ada untuk mendapatkan hasil. Tobing, F. A. T(2022).

Salah satu aspek utama dari SQL adalah kemampuannya untuk melakukan Queri, yang memungkinkan pengguna untuk mengambil informasi dari basis data dengan menggunakan perintah-perintah yang sederhana dan mudah dimengerti. Dengan sintaksis yang jelas dan terstruktur, SQL menjadi alat yang sangat efektif untuk bekerja dengan basis data relasional. Selain itu, SQL juga mendukung transaksi, yang memastikan integritas data dan konsistensi dalam lingkungan multi-pengguna.

Basis data relasional yang menggunakan SQL sebagai antarmuka utamanya mencakup sistem seperti MySQL, PostgreSQL, Microsoft SQL Server, dan Oracle Database. SQL telah berkembang dari versi awalnya menjadi standar industri yang luas digunakan, dan

peningkatan terus-menerus telah diintegrasikan untuk mendukung fungsionalitas baru dan meningkatkan kinerja. Dengan SQL, para pengembang dan administrator basis data dapat dengan efisien mengelola dan memanipulasi data, menjadikannya elemen kunci dalam ekosistem teknologi informasi modern. Yanto, R. (2016).

Sebagai seorang ahli basis data terkenal, Donald D. Chamberlin, yang juga salah satu pencipta SQL, pernah menggambarkan peran SQL dalam mengelola data: "SQL memberikan cara untuk menyusun pertanyaan dan pernyataan pengelolaan data dengan cara yang intuitif dan deklaratif. Ini memungkinkan pengguna untuk fokus pada apa yang ingin mereka capai, bukan pada cara mendapatkan itu."

Dengan sejarah panjang dan keberhasilannya dalam dunia teknologi informasi, SQL tetap menjadi fondasi utama dalam pengelolaan data relasional, memainkan peran krusial dalam berbagai aplikasi dan sistem yang kita gunakan sehari-hari.

11.2 Para Penemu SQL (*Structured Query Language*)

Tiga tokoh penting yang berkontribusi dalam penemuan SQL

1. **Edgar F. Codd**

   Edgar F. Codd adalah seorang ilmuwan komputer asal Inggris yang dianggap sebagai bapak SQL. Pada tahun 1970, Codd menerbitkan makalah yang berjudul "A Relational Model of Data for Large Shared Data Banks" di jurnal IBM Research. Makalah tersebut mengusulkan model hubungan dalam pengelolaan data. Model ini melibatkan penggunaan bahasa query yang kemudian menjadi cikal bakal SQL.(2)

   Codd juga mengusulkan 12 aturan atau prinsip yang harus dipenuhi oleh sistem manajemen basis data (SMBD) untuk disebut sebagai sistem basis data relasional yang sejati. Kontribusi Codd dalam pengembangan SQL sangatlah signifikan, karena ia menetapkan fondasi dan prinsip-prinsip dasar yang masih digunakan dalam SQL hingga saat ini.

2. **Raymond F. Boyce**

Raymond F. Boyce adalah seorang ilmuwan komputer yang bekerja sama dengan Edgar F. Codd. Pada tahun 1970-an, bersama dengan Codd, Boyce mengembangkan bahasa SQL pertama yang disebut SEQUEL (Structured English Query Language). Bahasa ini kemudian disederhanakan menjadi SQL.

Boyce juga dikenal karena menemukan dan mengembangkan teori fungsional dependensi, yang digunakan dalam mendefinisikan kunci primer (*primary key*) pada tabel yang relasional. Kontribusi Boyce dalam SQL tidak dapat diabaikan, karena ia berperan penting dalam pengembangan dan pengklasifikasian bahasa ini.

3. **Josephine M. Widom**

   Josephine M. Widom adalah seorang profesor ilmu komputer di Universitas Stanford dan salah satu tokoh penting dalam bidang pangkalan data. Widom berfokus pada pengembangan perangkat lunak untuk memperbaiki kinerja sistem pangkalan data dan mengoptimalkan pertanyaan SQL. Selain menjadi seorang pengajar dan peneliti, Widom juga terlibat dalam pengembangan produk komersial yang berkaitan dengan pangkalan data, seperti sistem manajemen basis data yang sangat dipengaruhi oleh SQL.

   Melalui penelitiannya, Widom berhasil memperbaiki efisiensi SQL dan mengurangi beban yang dikenakan pada sistem pangkalan data. Kontribusinya dalam mengoptimalkan kinerja SQL sangat berharga bagi pengguna SQL di seluruh dunia.

Pada umumnya, SQL digunakan bersamaan dengan bahasa pemrograman lain, seperti Python, Java, atau PHP, untuk mengembangkan aplikasi lengkap. Bahasa pemrograman tersebut digunakan untuk mengintegrasikan logika bisnis, antarmuka pengguna, dan fungsionalitas tambahan dengan operasi basis data yang dijalankan dengan SQL.

11.3 SQL dan Fungsinya

Structured Query Language mempunyai berbagai fungsi yang penting dalam pengelolaan basis data. Salah satu fungsi utama SQL adalah untuk melakukan operasi CRUD pada data yang tersimpan dalam basis data. CRUD yaitu *Create* (membuat), *Read* (membaca), *Update* (memperbarui), dan *Delete* (menghapus). Kusumo, A. S. (2021).

11.4 Konsep dan Perintah Dasar Sql

SQL, atau *Structured Query Language*, adalah bahasa pemrograman khusus yang digunakan untuk mengelola dan mengakses database relasional. Nugroho, B. (2008).

1. **Dasar-dasar Database Relasional:**
    a. Memahami konsep dasar database relasional.
    b. Mengetahui perbedaan antara tabel, baris (record), dan kolom (field).
    c. Mempelajari cara mengorganisir data dalam basis data.
2. **Perintah Dasar SQL:**
    a. SELECT: Mengambil data dari satu atau lebih tabel.
    b. INSERT: Menambahkan data baru ke dalam tabel.
    c. UPDATE: Memperbarui data yang sudah ada dalam tabel.
    d. DELETE: Menghapus data dari tabel.
3. **Kondisi dan Klausa WHERE:**
    a. Memahami cara menggunakan klausa WHERE untuk menyaring data berdasarkan kondisi tertentu.
    b. Mempelajari operator perbandingan (misalnya, "=", "<>", "<", ">", "<=", ">=").
4. **Urutan dan Pengelompokan Data:**
    a. Menggunakan perintah ORDER BY untuk mengurutkan hasil query.
    b. Menggunakan klausa GROUP BY untuk mengelompokkan data.
5. **Fungsi Agregat:**
    a. Memahami fungsi agregat seperti SUM, AVG, COUNT, MIN, dan MAX.
    b. Menggunakan fungsi-fungsi ini dalam query untuk menghasilkan ringkasan data.

6. **Gabungan Tabel (JOIN):**
    a. Memahami jenis-jenis join seperti INNER JOIN, LEFT JOIN, RIGHT JOIN, dan FULL JOIN.
    b. Menggunakan join untuk menggabungkan data dari dua atau lebih tabel.
7. **Subquery:**
    a. Memahami konsep subquery dan cara menggunakannya dalam query.
    b. Menggunakan subquery untuk mendapatkan informasi yang lebih spesifik.
8. **Indeks dan Kinerja:**
    a. Mengetahui peran indeks dalam mengoptimalkan kinerja query.
    b. Mempelajari cara membuat dan menggunakan indeks dengan bijak.
9. **Transaksi:**
    a. Memahami konsep transaksi dan pengendalian transaksi.
    b. Mempelajari perintah COMMIT dan ROLLBACK untuk mengelola transaksi.
10. **Keamanan:**
    a. Memahami cara melindungi database dari serangan seperti SQL injection.
    b. Mengetahui cara memberikan hak akses dengan bijaksana.
11. **Views dan Stored Procedures:**
    a. Memahami penggunaan Views untuk menyederhanakan kompleksitas query.
    b. Mengetahui cara membuat dan menggunakan Stored Procedures untuk mengelola logika bisnis di database.
12. **Manajemen Database:**
    a. Mempelajari perintah-perintah untuk membuat, mengubah, dan menghapus basis data dan tabel.
    b. Mengetahui cara mengelola indeks, tata letak tabel, dan parameter database lainnya.

11.5 Perintah Dasar SQL / Structured Query Language

1. **Perintah** SELECT.
2. **Perintah** SELECT DISTINCT.
3. **Perintah** WHERE.
4. **Perintah** (operator) AND, OR dan NOT.
5. **Perintah** ORDER BY.
6. **Perintah** INSERT INTO.
7. **Perintah** UPDATE.
8. **Perintah** DELETE

11.6 Penjelasan Fungsi Dan Contoh Penerapannya SQL

1. **Membuat Tabel Baru**

```
CREATE TABLE mahasiswa (
  id INT,
  nama VARCHAR(50),
  alamat VARCHAR(100)
);
```

Di sinilah kita menggunakan perintah CREATE TABLE untuk membuat tabel baru dalam database. Pada contoh ini, kita membuat tabel dengan nama "mahasiswa" yang memiliki tiga kolom, yaitu id, nama, dan alamat. Tipe data kolom dijelaskan di samping nama kolom. Misalnya, kolom id menggunakan tipe data INT (integer), kolom nama menggunakan tipe data VARCHAR dengan panjang maksimal 50 karakter, dan kolom alamat menggunakan tipe data VARCHAR dengan panjang maksimal 100 karakter.

2. **Menambahkan Data ke Tabel**

```
INSERT INTO mahasiswa (id, nama, alamat)
VALUES (1, 'John Doe', 'Jalan Sudirman No. 123');
```

Perintah INSERT INTO digunakan untuk menambahkan data ke dalam tabel. Pada contoh ini, kita menambahkan satu baris data ke dalam tabel "mahasiswa" dengan nilai id, nama, dan alamat yang sesuai.

3. **Mengambil Data dari Tabel**
   SELECT * FROM mahasiswa;
   Perintah SELECT digunakan untuk mengambil data dari tabel. Pada contoh ini, kita menggunakan tanda asterisk (*) untuk mengambil semua kolom pada tabel "mahasiswa". Hasil dari perintah ini akan mengembalikan semua baris data yang ada dalam tabel.
4. **Mengupdate Data Tabel**
   UPDATE mahasiswa
   SET alamat = 'Jalan Thamrin No. 456'
   WHERE id = 1;
   Perintah UPDATE digunakan untuk memperbarui data dalam tabel. Pada contoh ini, kita memperbarui nilai kolom "alamat" pada baris data dengan id 1 menjadi "Jalan Thamrin No. 456". Perintah WHERE digunakan untuk menentukan kondisi di mana perubahan harus dilakukan.
5. **Menghapus Data dari Tabel**
   DELETE FROM mahasiswa
   WHERE id = 1;
   Perintah DELETE FROM digunakan untuk menghapus data dari tabel. Pada contoh ini, kita menghapus baris data yang memiliki id 1 dari tabel "mahasiswa". Seperti halnya pada perintah UPDATE, perintah WHERE digunakan untuk menentukan kondisi di mana penghapusan harus dilakukan.

Penjelasan Koding

Nama_field adalah nama kolom (*field*) yang akan dibuat. Beberapa sistem manajemen basis data mengizinkan penggunaan spasi dan karakter nonhuruf pada nama kolom.

Tipe_data tergantung implementasi sistem manajemen basis data. Misalnya, pada MySQL, tipe data dapat berupa `ARCHAR`, `TEXT`, `BLOB`, `ENUM`, dan sebagainya. *Constraints* adalah batasan-batasan yang diberikan untuk tiap kolom. Ini juga tergantung implementasi sistem manajemen basis data, misalnya `NOT NULL`, `UNIQUE`, dan sebagainya. Ini dapat digunakan untuk mendefinisikan kunci primer (*primary key*) dan kunci asing (*foreign key*).

Satu tabel boleh tidak memiliki kunci primer sama sekali, namun sangat disarankan mendefinisikan paling tidak satu kolom sebagai kunci primer.
Contoh:

```
CREATE TABLE user
(
username VARCHAR(30) CONSTRAINT PRIMARY KEY,
passwd VARCHAR(20) NOT NULL,
tanggal_lahir DATETIME
);
```

akan membuat tabel user seperti berikut:

username	passwd	tanggal_lahir

Jenis perintah SQL yang dimaksud adalah sebagai berikut:

1. **DML (*Data Manipulation Language*)**
2. **DDL (*Data Definition Language*)**
3. **DCL (*Data Control Language*)**

1. ***Data Manipulation Language* (DML)**
   DML adalah jenis SQL yang digunakan untuk mengubah atau memanipulasi data dalam sebuah database. Dalam DML, terdapat perintah-perintah seperti INSERT, UPDATE, DELETE, dan SELECT.
   Misalnya, jika Anda ingin menambahkan data baru ke dalam sebuah tabel, Anda dapat menggunakan perintah INSERT. Perintah ini akan memasukkan data baru ke dalam tabel yang ditentukan. Contoh penggunaannya adalah sebagai berikut:
   INSERT INTO nama_tabel (kolom1, kolom2, kolom3)
   VALUES (nilai1, nilai2, nilai3);
   Dengan menggunakan perintah INSERT ini, Anda dapat dengan mudah menambahkan data baru ke dalam tabel yang ada di dalam database.

2. **Data Definition Language (DDL)**
   DDL adalah jenis SQL yang digunakan untuk mendefinisikan struktur atau skema database. Dalam DDL, terdapat perintah-perintah seperti CREATE, ALTER, dan DROP.
   Misalnya, jika Anda ingin membuat tabel baru dalam database, Anda dapat menggunakan perintah CREATE TABLE. Perintah ini akan membuat tabel baru sesuai dengan struktur yang Anda tentukan. Contoh penggunaannya adalah sebagai berikut:
   CREATE TABLE nama_tabel
   (kolom1 tipe_data1,
   kolom2 tipe_data2,
   kolom3 tipe_data3);
   Dengan menggunakan perintah CREATE TABLE ini, Anda dapat membuat tabel baru dengan kolom-kolom yang sesuai dengan kebutuhan Anda.

3. **Data Control Language (DCL)**
   DCL adalah jenis SQL yangunakan untuk mengontrol akses dan izin pengguna terhadap database Dalam DCL, terdapat perintah-perintah seperti GRANT dan REVOKE.

11.7 Perintah Data Control Sql Dan Contoh Coding

Perintah SQL ini digunakan khususnya untuk mengatur hak apa saja yang dimiliki oleh pengguna. Baik itu hak terhadap sebuah database ataupun pada tabel maupun field yang ada. Contoh Coding sebagai Berikut: Troels Arvin(2007).

1. **SELECT Statement:**
   Mengambil data dari satu atau lebih tabel.
   -- Mengambil semua data dari tabel "pegawai"
   SELECT * FROM pegawai;

   -- Mengambil data tertentu dari tabel "pelanggan"
   SELECT nama, alamat FROM pelanggan WHERE kota = 'Jakarta';
2. **INSERT Statement:**
   Menambahkan data baru ke dalam tabel.
   -- Menambahkan pegawai baru

INSERT INTO pegawai (nama, posisi, gaji) VALUES ('John Doe', 'Manajer', 50000);

3. **UPDATE Statement:**
   Memperbarui data yang sudah ada dalam tabel.
   -- Memperbarui gaji pegawai dengan ID 101
   UPDATE pegawai SET gaji = 55000 WHERE id = 101;
4. **DELETE Statement:**
   Menghapus data dari tabel.
   -- Menghapus pegawai dengan ID 102
   DELETE FROM pegawai WHERE id = 102;
5. **JOIN:**
   Menggabungkan data dari dua atau lebih tabel.
   -- Menggabungkan data dari tabel "pelanggan" dan "pesanan"
   SELECT pelanggan.nama, pesanan.nomor_pesanan
   FROM pelanggan
   JOIN pesanan ON pelanggan.id = pesanan.id_pelanggan;
6. **Fungsi Agregat:**
   Menggunakan fungsi-fungsi seperti COUNT, SUM, AVG, MIN, dan MAX.
   -- Menghitung jumlah pesanan untuk setiap pelanggan
   SELECT pelanggan.id, pelanggan.nama, COUNT(pesanan.nomor_pesanan) AS jumlah_pesanan
   FROM pelanggan
   LEFT JOIN pesanan ON pelanggan.id = pesanan.id_pelanggan
   GROUP BY pelanggan.id, pelanggan.nama;
7. **Subquery:**
   Menggunakan subquery dalam query utama.
   -- Menampilkan pegawai dengan gaji di atas rata-rata
   SELECT * FROM pegawai
   WHERE gaji > (SELECT AVG(gaji) FROM pegawai);
8. **Views:**
   Menggunakan view untuk menyederhanakan query.
   -- Membuat view untuk menampilkan data pelanggan di Jakarta
   CREATE VIEW pelanggan_jakarta AS
   SELECT * FROM pelanggan WHERE kota = 'Jakarta';
   -- Mengambil data dari view

SELECT * FROM pelanggan_jakarta;

9. **Stored Procedures:**
   Menggunakan stored procedures untuk menjalankan logika bisnis.
   -- Membuat stored procedure untuk menambah pelanggan
   DELIMITER //
   CREATE PROCEDURE tambah_pelanggan (IN p_nama VARCHAR(255), IN p_alamat VARCHAR(255))
   BEGIN
   INSERT INTO pelanggan (nama, alamat) VALUES (p_nama, p_alamat);
   END //
   DELIMITER ;
   -- Menjalankan stored procedure
   CALL tambah_pelanggan('Jane Doe', 'Jl. Sudirman');
10. **Indeks:**
    Menggunakan indeks untuk mengoptimalkan kinerja query.
    -- Membuat indeks pada kolom "nama" dalam tabel "pelanggan"
    CREATE INDEX idx_nama_pelanggan ON pelanggan(nama);
11. **Izin Akses Kepada Pengguna Tabel:**
    GRANT jenis_izin ON nama_tabel TO nama_pengguna;

Kegunaan dan manfaat bahasa SQL (*structured query language)* dalam pengelolaan basis data. berikut adalah beberapa di antaranya:

1. **Manipulasi Data:**
   a. Query Data: SQL digunakan untuk mengekstrak data dari basis data menggunakan perintah SELECT.
   b. Insert Data: SQL memungkinkan Anda menambahkan data baru ke dalam tabel.
   c. Update Data: SQL memungkinkan Anda memperbarui data yang sudah ada dalam tabel.
   d. Delete Data: SQL dapat digunakan untuk menghapus data dari tabel.

2. **Pengelolaan Basis Data:**
   a. Membuat Tabel: Dengan SQL, Anda dapat membuat struktur tabel baru, menentukan tipe data, dan relasi antar tabel.
   b. Mengubah Tabel: SQL memungkinkan pengubahan struktur tabel, seperti menambahkan kolom atau mengubah tipe data.
   c. Menghapus Tabel: SQL memungkinkan Anda menghapus tabel dan struktur terkait lainnya dari basis data.
3. **Pengendalian Akses:**
   a. Pemberian Hak Akses: SQL memungkinkan administrator basis data memberikan hak akses yang terbatas kepada pengguna atau aplikasi tertentu.
   b. Pengaturan Keamanan: SQL dapat digunakan untuk mengimplementasikan mekanisme keamanan seperti enkripsi data dan otentikasi pengguna.
4. **Pertanyaan Kompleks:**
   SQL memungkinkan pengguna untuk membuat pertanyaan kompleks menggunakan klausa JOIN, GROUP BY, dan HAVING, sehingga dapat mengekstrak informasi yang sangat spesifik dari basis data.
5. **Pemeliharaan Integritas Data:**
   SQL mendukung pembuatan aturan dan constraint untuk memastikan integritas data, seperti constraint kunci utama (*primary key*) dan kunci asing (*foreign key*).
6. **Optimasi Kueri:**
   SQL memungkinkan pengguna untuk menulis kueri yang dioptimalkan untuk meningkatkan kinerja basis data, termasuk penggunaan indeks dan peningkatan struktur kueri.
7. **Replikasi dan Backup:**
   SQL menyediakan perintah untuk membuat salinan (replikasi) basis data atau membuat cadangan (backup) data, sehingga memastikan keamanan data.
8. **Pemrograman Terkait Database:**
   SQL dapat digunakan dalam pengembangan aplikasi yang terhubung ke basis data, memungkinkan penggunaan perintah SQL langsung dalam kode program.

11.8 Sisi Lain SQL

Perbedaan SQL server dan MySQL

Kedua teknologi bekerja sama sehingga Anda dapat menyimpan dan memproses data dalam tabel terstruktur dengan baris dan kolom. MySQL adalah teknologi dasar yang menyimpan data, sedangkan SQL adalah bahasa yang Anda gunakan untuk membaca, mengubah, dan menghapus data tersebut.

SQL dan NoSQL

Bahasa. NoSQL menggunakan bahasa yang bervariasi tergantung pada jenis NoSQL yang digunakan. Sedangkan bahasa query SQL menggunakan bahasa query SQL standar untuk memanipulasi data. Sharma, V., & Dave, M. (2012)

11.9 Kesimpulan

Secara keseluruhan, SQL (*Structured Query Language*) adalah 209ontro pemrograman yang digunakan untuk mengelola dan mengakses 209ontro manajemen basis data relasional (RDBMS). Dengan sintaksis yang jelas dan terstruktur, SQL memungkinkan pengguna untuk melakukan berbagai operasi, termasuk pengambilan, penyisipan, pembaruan, dan penghapusan data dalam basis data. Beberapa perintah utama SQL mencakup SELECT untuk mengekstrak data, INSERT INTO untuk menyisipkan data baru, UPDATE untuk memperbarui data, DELETE untuk menghapus data, CREATE TABLE untuk membuat 209ontr baru, dan lainnya.

Selain itu, SQL memiliki kemampuan untuk mengelola struktur basis data melalui perintah seperti CREATE TABLE, ALTER TABLE, dan DROP TABLE. Perintah-perintah ini memungkinkan pengguna untuk mendefinisikan dan mengubah struktur 209ontr sesuai kebutuhan. SQL juga mendukung operasi JOIN, yang memungkinkan pengguna untuk menggabungkan data dari 209ontr yang berbeda berdasarkan kolom terkait.

Dalam konteks keamanan, SQL memberikan 209ontrol akses melalui perintah GRANT dan REVOKE, memungkinkan administrator untuk mengatur hak akses pengguna atau peran terhadap basis data atau objek tertentu. Dengan demikian, SQL tidak

hanya berfungsi sebagai alat untuk mengelola data, tetapi juga sebagai sarana untuk menjaga keamanan dan integritas data dalam lingkungan multi-pengguna.

Secara umum, SQL telah menjadi standar de facto dalam pengelolaan basis data relasional, dan pemahaman tentang perintah-perintah SQL menjadi keterampilan yang penting bagi para pengembang dan administrator basis data dalam mendukung pengembangan dan operasional aplikasi modern.

DAFTAR PUSTAKA:

"ISO/IEC 9075-2:2023". ISO.org (dalam bahasa Inggris). Juni 2023. Diakses tanggal 31 Agustus 2023.

Paul, Ryan. "A guided tour of the Microsoft Command Shell". Ars Technica. Diakses tanggal 10 April 2011.

"ISO/IEC 9075-1:2008: Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)"

Beaulieu, Alan (April 2009). Mary E Treseler, ed. Learning SQL (edisi ke-2nd). Sebastapol, CA, USA: O'Reilly. ISBN 978-0-596-52083-0.

Chamberlin, Donald D. (2001-10-03). "Oral history interview with Donald D. Chamberlin". Diakses tanggal 2020-01-14. We changed the original name "SEQUEL" to SQL because we got a letter from somebody's lawyer that said the name "SEQUEL" belonged to them. We shortened it to SQL, for Structured Query Language, and the product was known as SQL/DS.

Kusumo, A. S. 2021. *Pemrograman SQL Server 2019*. Elex Media Komputindo.

Idwar, I., Murniati, M., Octavera, S., & Sovita, I. 2023. TEKNOLOGI DATABASE AKUNTANSI PERUSAHAAN JASA FLATFORM MICROSOFT ACCESS oleh DOSEN dan MAHASISWA di UNIDHA. *Prosiding Konferensi Nasional Pengabdian Kepada Masyarakat dan Corporate Social Responsibility (PKM-CSR)*, *6*, 1-12.

Tobing, F. A. T., Chandra, A., & Nainggolan, R. 2022. Penerapan Algoritma Knuth Morris Pratt (KMP) pada Pencarian Data di SQL LIKE Operators. *JURNAL WIDYA*, *3*(1), 50-58.

Troels Arvin, 2007-05-26, http://troels.arvin.dk/db/rdbms/ Comparison of different SQL implementations

Monica, A. 2023. Pengukuran Efektivitas Serangan SQL Injection Pada Website Dengan Menggunakan Tools JSQL, Havij, Dan The Mole. *Jurnal Informatika dan Teknologi Komputer (J-ICOM)*, *4*(2), 65-71.

Nugroho, B. 2008. *Panduan lengkap Menguasai perintah SQL*. Mediakita.

Sharma, V., & Dave, M. 2012. Sql and nosql databases. *International Journal of Advanced Research in Computer Science and Software Engineering*, *2*(8).

Yanto, R. 2016. *Manajemen Basis Data Menggunakan MySQL*. Deepublish.

BIODATA PENULIS

Dr. Irmawati, S.Kom., MMSI.
Dosen Program Studi Sistem Informasi
Fakultas Teknik dan Informatika
Universitas Multimedia Nusantara

Penulis adalah dosen tetap pada Program Studi Sistem Informasi Fakultas Teknik dan Informatika Universitas Multimedia Nusantara. Menyelesaikan pendidikan S1 dan S2 pada Program Studi Sistem Informasi Universitas Gunadarma dan S3 pada departemen Teknik Elektro Universitas Indonesia. Penulis menekuni bidang keilmuan meliputi *Artificial Intelligence*, *Image Processing*, *Machine Learning*, dan *Deep Learning*.

BIODATA PENULIS

Dani Anggoro, S.Kom., M.Kom.
Dosen Program Studi Ilmu Komputer
Fakultas Ilmu Komputer, Universitas Muhammadiyah Metro

Penulis lahir di Metro (Lampung). Penulis adalah dosen tetap pada Program Studi Ilmu Komputer Universitas Muhammadiyah Metro (UM Metro). Menyelesaikan pendidikan S1 pada Program Studi Sistem Informasi Universitas Budi Luhur Tahun 2011 dan melanjutkan S2 Ilmu Komputer di universitas yang sama di tahun 2014. Selain dosen, penulis adalah programmer di Direktorat Teknologi Informasi Universitas Budi Luhur dari tahun 2007 sampai 2021. Konsentrasi penelitian di bidang Basis Data, Data Warehouse, Big Data, Sistem Informasi dan Kecerdasan Buatan (AI).

BIODATA PENULIS

Zumhur Alamin, S.Kom., M.Kom.
Dosen Program Studi Ilmu Komputer
Fakultas Teknik dan Komputer Universitas Muhammadiyah Bima

Penulis lahir di Sumbawa Besar tanggal 24 Februari 1987. Minat penulis terhadap ilmu komputer dimulai pada tahun 2005, ketika penulis memulai perjalanan pendidikan di Universitas Muhammadiyah Malang, mengambil jurusan Teknik Informatika, dan berhasil meraih gelar sarjana pada tahun 2009. Pada awal tahun 2014, penulis melanjutkan studi pada program magister Ilmu Komputer di Universitas Budi Luhur Jakarta dan berhasil menyelesaikan studi pada tahun 2015.

Sebagai anak kedua dari enam bersaudara dari pasangan Ir. Zainuddin H. Hamzah dan Nurjanah, S. Sos, S. Pd., penulis telah menunjukkan komitmennya dalam dunia akademis. Kini, sebagai seorang dosen di Program Studi Ilmu Komputer di Universitas Muhammadiyah Bima sejak tahun 2022, penulis mengampu berbagai mata kuliah, termasuk Algoritma dan Pemrograman, serta memiliki ketertarikan di bidang Software Engineering dan Data Science. Untuk mewujudkan karir sebagai dosen profesional, penulis aktif sebagai peneliti di bidang kepakarannya tersebut. Selain sebagai peneliti, penulis juga aktif dalam menulis buku dengan harapan dapat memberikan kontribusi positif bagi bangsa dan negara.

BIODATA PENULIS

Rudiansyah, S.Kom., M.Kom
Dosen Program Studi Sistem Informasi
Fakultas Ilmu Komputer Universitas Sjakhyakirti Palembang

Penulis lahir di Palembang tanggal 20 November 1979. Penulis adalah dosen tetap pada Program Studi Sistem Informasi Fakultas Ilmu Komputer, Universitas Sjakhyakirti Palembang. Menyelesaikan pendidikan S1 pada Jurusan Sistem Informasi dan melanjutkan S2 pada Jurusan Teknik Informatika. Penulis menekuni bidang Menulis.

BIODATA PENULIS

Sugiarto Cokrowibowo
Dosen Program Studi Informatika
Fakultas Teknik Universitas Sulawesi Barat

Penulis lahir di Palopo tanggal 24 Mei 1986. Penulis adalah dosen tetap pada Program Studi Informatika Fakultas Teknik, Universitas Sulawesi Barat. Menyelesaikan pendidikan S1 pada Program Studi Matematika Universitas Hasanuddin dan melanjutkan S2 pada Program Studi Teknik Elektro Universitas Hasanuddin. Bidang kajian yang diminati oleh penulis, antara lain artificial intelligence; soft computing; internet of things; programming; algoritma dan struktur data. Penulis dapat dihubungi melalui surel: sugiarto.cokrowibowo.id@gmail.com penulis juga aktif membuat konten video pembelajaran algoritma dan pemrograman di channel Youtube FOSALGO https://www.youtube.com/@FOSALGO

BIODATA PENULIS

Muhammad Akbar Yasin, S.T., M.T.
Dosen Program Studi Sistem dan Teknologi Informasi
Fakultas Teknik Universitas Muhammadiyah Kendari

Penulis berasal lahir di Kendari, Sulawesi Tenggara. Penulis adalah dosen tetap pada Program Studi Pendidikan Sistem dan Teknologi Informasi Fakultas Teknik, Universitas Muhammadiyah Kendari. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika, Universitas Halu Oleo dan melanjutkan S2 pada Jurusan Teknik Elektro opsi Manajemen dan Rekayasa Keamanan Informasi, Institut Teknologi Bandung.

BIODATA PENULIS

Paray Theo Lonando, S.Kom., M.Kom
Dosen Program Studi Sistem Informasi
Fakultas Sains dan Teknologi Institut Ilmu Kesehatan dan Tenologi Muhammadiyah Palembang

Penulis lahir di Palembang tanggal 29 Juli 1992. Penulis adalah dosen tetap pada Program Studi Sistem Informasi Fakultas Sains dan Teknologi Institut Ilmu Kesehatan dan Tenologi Muhammadiyah Palembang. Menyelesaikan pendidikan S1 pada Jurusan Sistem Informasi dan melanjutkan S2 pada Jurusan Teknik Informatika. Penulis menekuni bidang Menulis.

BIODATA PENULIS

Wirawan Istiono, S.Kom., M.Kom.
Dosen Informatika
Fakultas Teknik dan Informatika, Universitas Multimedia Nusantara

Penulis lahir di Jakarta tanggal 13 April 1983. Penulis adalah dosen pada Program Studi Informatika Fakultas Teknik dan Informatika, Universitas Multimedia Nusantara. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika dan melanjutkan S2 pada bidang jurusan yang Teknik Informatika. Pada saat ini, penulis menekuni bidang Penelitian terkait komputer sains, sistem komputer dan teknologi informasi

BIODATA PENULIS

Wulan Sri Lestari, S.Kom., M.Kom.
Dosen Program Studi Teknologi Informasi
Fakultas Informatika Universitas Mikroskil

Penulis lahir di Tanjung Merawa tanggal 31 Maret 1993. Penulis adalah dosen tetap pada Program Studi Teknologi Informasi Fakultas Informatika, Universitas Mikroskil. Menyelesaikan pendidikan S1 pada Jurusan Teknik Informatika dan melanjutkan S2 pada Jurusan Teknologi Informasi. Selain menulis buku dibidang Teknologi Informasi, penulis telah melakukan beberapa penelitian di bidang Keamanan Data, Deep Learning dan Artificial Inteligence.

BIODATA PENULIS

Hendra Nusa Putra, S.Kom., M.Kom.
Dosen Program Studi D3 Rekam Medis & Informasi Kesehatan
STIKES Dharma Landbouw Padang

Penulis lahir di Padang tanggal 07 Desember 1984. Saat ini penulis sedang menempuh pendidikan Program Doktoral (S3) Sistem Informasi UPI " YPTK" Padang. Aktivitas penulis saat ini berprofesi sebagai Dosen di STIKES Dharma Landbow Padang, dengan mengajar di mata kuliah Kodefiksi IT, Jaringan Komputer dan Perangkat Lunak Yankes serta Algoritma Pemograman. Penulis juga aktif menjadi pembicara di beberapa pertemuan yang membahas seputar teknologi informasi, seperti "Dampak Negatif Kemajuan Teknologi" (2021), "Pembelajaran Daring bagi Mahasiswa Baru" (2020), "Blending Learning Solusi Ngampus di Masa New Normal" (2020), "Teknik Searching Efektif di Internet" (2020) dan "Pemanfaatan Learning Management System (LMS) dengan EDMOOD" (2020).

BIODATA PENULIS

Ade Wisandra, M.Kom
Stikes dharma Lanbouw Padang

Lahir di Pangian Lintau Tanah Datar, 15 Mai 1989. Telah menikah dengan satu istri dan dua putra dan satu putri. Menggeluti hal-hal yang bersifat religiusitas dan komputer. Aktif dalam berbagai kegiatan pelatihan dan teknis di bidang komputer di beberapa intansi Swasta dan sosial.

Riwayat pekerjaan/profesi (10 tahun terakhir):

1. 2018–kini: Dosen Tetap Rekam Medis STKES dharma Landbouw Padang
2. 2018 : Dosen LB Komputer di STKIP PGRI Padang
3. 2010–2018: Pengolah bahan pustaka Stkip PGRI Padang (sekarang UPGRISBA)

Riwayat Pendidikan Tinggi dan Tahun Belajar:

1. S-2: Ilmu Komputer (2014—2016)
2. S-1: Sistem Informasi Komputer (2012—2014)

Judul Penelitian dan Buku Tahun Terbit (10 tahun terakhir):

1. Penerapan algoritma k-means untuk klasterisasi data obat pasien rawat jalan berdasarkan 3 penyakit terbanyak di rumah sakit m. Natsir solok (2022)

2. Perancangan Sitem Informasi Pendaftaran Online Pasien Rawat I Pada Puskesmas Menggunakan Metode Prototype (2023)
3. Pembelajaran dan Pemanfaatan Blog Bagi Puskesmas Puskesmas di Lingkungan Kota Padang (2023)

Alamat email : adeperpus@gmail.com