

LAPORAN PRAKTIKUM

Programming Algorithm

HALAMAN SAMPUL



Disusun Oleh :

Lutvan Muzammil / 2502310085

JURUSAN INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS BHAUDIN MUDHARY MADURA

SUMENEP

2025



LAPORAN PRAKTIKUM

Programming Algorithm

HALAMAN PENGESAHAN

Diajukan guna melengkapi tugas akhir dan memenuhi salah satu syarat untuk menyelesaikan Mata Kuliah *Programming Algorithm*

Disusun Oleh :

Lutvan Muzammil / 2502310085

JURUSAN INFORMATIKA

FAKULTAS SAINS DAN TEKNOLOGI

UNIVERSITAS BHAUDIN MUDHARY MADURA

SUMENEP

2025

LEMBAR ASISTENSI

DAFTAR ISI

HALAMAN SAMPUL.....	i
HALAMAN PENGESAHAN.....	ii
LEMBAR ASISTENSI	iii
DAFTAR ISI.....	iv
DAFTAR GAMBAR	vi
DAFTAR TABEL	viii
MODUL I DASAR ALGORITMA C	1
1.1 Tujuan Praktikum	1
1.2 Landasan Teori.....	1
1.3 Alat-alat dan Komponen.....	3
1.4 Prosedur Percobaan	3
1.5 Data Hasil Percobaan.....	4
1.6 Analisis Data dan Pembahasan.....	9
1.7 Kesimpulan.....	11
MODUL 2 PENGAMBILAN KEPUTUSAN (<i>IF, IF-ELSE, SWITCH</i>)	13
2.1 Tujuan Praktikum	13
2.2 Landasan Teori.....	13
2.3 Alat-alat dan Komponen.....	17
2.4 Prosedur Percobaan	18
2.5 Data Hasil Percobaan.....	19
2.6 Analisis Data dan Pembahasan.....	22
2.7 Kesimpulan.....	23
MODUL III PERULANGAN (<i>FOR, WHILE, DO-WHILE</i>)	25
3.1 Tujuan Praktikum	25
3.2 Landasan Teori.....	25
3.3 Alat-alat dan Komponen.....	31
3.4 Prosedur Percobaan	31

3.5 Data Hasil Percobaan.....	32
3.6 Analisa Hasil Percobaan	36
3.7 Kesimpulan	36
MODUL IV ARRAY	40
4.1 Tujuan Praktikum	40
4.2 Landasan Teori.....	40
4.3 Alat-alat dan Komponen.....	43
4.4 Prosedur Percobaan	43
4.5 Data Hasil Percobaan.....	44
4.6 Analisa Hasil Percobaan	48
4.7 Kesimpulan	51
DAFTAR PUSTAKA.....	52

DAFTAR GAMBAR

Gambar 1.1 <i>Program Kegiatan 1</i>	3
Gambar 1.2 <i>Program Kegiatan 2</i>	3
Gambar 1.3 <i>Program Keiatan 3</i>	4
Gambar 1.4 <i>Program Kegiatan 4</i>	4
Gambar 1.5 <i>Program Kegiatan 5</i>	4
Gambar 1.6 <i>Output Program Kegiatan 1</i>	5
Gambar 1.7 <i>Hasil Output Kegiatan 2</i>	6
Gambar 1.8 <i>Hasil Output Kegiatan 4</i>	8
Gambar 2.1 <i>Pernyataan If</i>	14
Gambar 2.2 <i>Diagram Alir If</i>	15
Gambar 2.3 <i>Pernyataan If-Else</i>	15
Gambar 2.4 <i>Diagram Alir If-Else</i>	15
Gambar 2.5 <i>Pernyataan Nested If</i>	16
Gambar 2.6 <i>Pernyataan Switch-Case</i>	17
Gambar 2.7 <i>Program Kegiatan 1</i>	18
Gambar 2.8 <i>Program Kegiatan 2</i>	18
Gambar 2.9 <i>Output Program 1</i>	19
Gambar 2.10 <i>Hasil Output Kegiatan 2</i>	20
Gambar 2.11 <i>Output Program 3</i>	21
Gambar 3.1 <i>Pernyataan For</i>	25
Gambar 3.2 <i>Perulangan For</i>	26
Gambar 3.3 <i>Pernyataan While</i>	26
Gambar 3.4 <i>Penggunaan While</i>	26
Gambar 3.5 <i>Pernyataan Do-While</i>	27
Gambar 3.6 <i>Penggunaan Do-While</i>	27
Gambar 3.7 <i>Penggunaan Tabel</i>	28
Gambar 3.8 <i>Penggunaan Tabel For</i>	28
Gambar 3.9 <i>Bagian Terdalam Nested Loop</i>	29

Gambar 3.10 Mencetak Baris	29
Gambar 3.11 Pernyataan Mencetak Untuk Semua baris	29
Gambar 3.12 Pernyataan Break	29
Gambar 3.13 Pernyataan Break	30
Gambar 3.14 Pernyataan Continue	30
Gambar 3.15 Deklarasi Untuk Fungsi Exit()	31
Gambar 3.16 Program Menggunakan Exit()	31
Gambar 3.17 Kegiatan 3.....	31
Gambar 3.18 Kegiatan 3.....	32
Gambar 3.19 Hasil Output Kegiatan 1	33
Gambar 3.20 Hasil Output Kegiatan 2	34
Gambar 3.21 Hasil Output Kegiatan 3	35
Gambar 4.1 Bentuk Deklarasi Array	41
Gambar 4.2 Contoh Deklarasi Dan Inisialisasi Array	41
Gambar 4.3 Deklarasi Array Multimedia	41
Gambar 4.4 Deklarasi & Inisialisasi Array Multimedia	41
Gambar 4.5 Program Kegiatan 1	43
Gambar 4.6 Program Kegiatan 2	44
Gambar 4.7 Program Kegiatan 3	44
Gambar 4.8 Hasil Output Kegiatan 1	45
Gambar 4.9 Hasil Output Kegiatan 2	46
Gambar 4.10 Hasil Output Kegiatan 3	48

DAFTAR TABEL

Tabel 1.1 <i>Listing Program Kegiatan 1</i>	5
Tabel 1.2 <i>Listing Program Kegiatan 2</i>	6
Tabel 1.3 <i>Listing Program Kegiatan 3</i>	7
Tabel 1.4 <i>Listing Program Kegiatan 4</i>	7
Tabel 1.5 <i>Listing Program Kegiatan 5</i>	8
Tabel 2.1 <i>Operator Relasi</i>	13
Tabel 2.2 <i>Operator Logika</i>	14
Tabel 2.3 <i>Listing Program Kegiatan 1</i>	19
Tabel 2.4 <i>Listing Program Kegiatan 2</i>	20
Tabel 2.5 <i>Listing Program 3</i>	21
Tabel 3.1 <i>Nested Loop</i>	28
Tabel 3.2 <i>Listing Program Kegiatan 1</i>	33
Tabel 3.3 <i>Listing Program Kegiatan 2</i>	34
Tabel 3.4 <i>Listing Program Kegiatan 3</i>	35
Tabel 4.1 <i>Elemen Array</i>	40
Tabel 4.2 <i>Ilustrasi Array 2D</i>	42
Tabel 4.3 <i>Listing Program Kegiatan 1</i>	45
Tabel 4.4 <i>Listing Program Kegiatan 2</i>	46
Tabel 4.5 <i>Listing Program Kegiatan 3</i>	47

MODUL I

DASAR ALGORITMA C

1.1 Tujuan Praktikum

- A. Menjelaskan tentang beberapa tipe data dasar.
- B. Menjelaskan tentang variabel.
- C. Menjelaskan tentang konstanta.
- D. Menjelaskan tentang berbagai jenis operator dan pemakaiannya.
- E. Menjelaskan tentang instruksi I/O.

1.2 Landasan Teori

1.2.1 Tipe Data Dasar

Data berdasarkan jenisnya dapat dibagi menjadi lima kelompok, yang dinamakan sebagai tipe data dasar. Kelima tipe data dasar adalah:

- A. Bilangan bulat (*integer*).
- B. Bilangan real presisi-tunggal.
- C. Bilangan real presisi-ganda.
- D. Karakter.
- E. Tak-bertipe (*void*).

Kata-kunci yang berkaitan dengan tipe data dasar secara berurutan di antaranya adalah *int* (*short int*, *long int*, *signed int* dan *unsigned int*), *float*, *double*, dan *char*.

1.2.2 Variabel

Aturan penulisan pengenalan untuk sebuah variabel, konstanta atau fungsi yang didefinisikan oleh pemrogram adalah sebagai berikut :

- A. Pengenal harus diawali dengan huruf (A..Z, a..z) atau karakter garisbawah (_).
- B. Selanjutnya dapat berupa huruf, digit (0..9) atau karakter garis bawah atau tanda dollar (\$).
- C. Panjang pengenalan boleh lebih dari 31 karakter, tetapi hanya 31 karakter

pertama yang akan dianggap berarti.

D. Pengenal tidak boleh menggunakan nama yang tergolong sebagai kata-kata cadangan (*reserved words*) seperti *int*, *if*, *while* dan sebagainya.

1.2.3 Konstanta

Konstanta menyatakan nilai yang tetap. Berbeda dengan variabel, suatu konstanta tidak dideklarasikan. Namun seperti halnya variabel, konstanta juga memiliki tipe. Penulisan konstanta mempunyai aturan tersendiri, sesuai dengan tipe masing-masing.

A. Konstanta karakter misalnya ditulis dengan diawali dan diakhiri dengan tanda petik tunggal, contohnya : 'A' dan '@'.

B. Konstanta integer ditulis dengan tanda mengandung pemisah ribuan dan tak mengandung bagian pecahan, contohnya : -1 dan 32767

C. Konstanta real (*float* dan *double*) bisa mengandung pecahan (dengan tanda berupa titik) dan nilainya bisa ditulis dalam bentuk eksponensial (menggunakan tanda e), contohnya : 27.5f (untuk tipe *float*) atau 27.5 (untuk tipe *double*) dan 2.1e+5 (maksudnya $2,1 \times 10^5$).

D. Konstanta string merupakan deretan karakter yang diawali dan diakhiri dengan tanda petik-ganda ("), contohnya : "Pemrograman Dasar C".

1.2.4 Operator

Operator merupakan simbol atau karakter yang biasa dilibatkan dalam program untuk melakukan sesuatu operasi atau manipulasi, seperti menjumlahkan dua buah nilai, memberikan nilai ke suatu variabel, membandingkan kesamaan dua buah nilai. Sebagian operator C tergolong sebagai operator *binary*, yaitu operator yang dikenakan terhadap dua buah nilai (*operand*).

Contoh :

$a + b$

Simbol + merupakan operator untuk melakukan operasi penjumlahan dari kedua *operand*-nya (yaitu a dan b). Karena operator penjumlahan melibatkan dua operator ini tergolong sebagai operator *binary*.

-c

Simbol - (minus) juga merupakan operator. Simbol ini termasuk sebagai operator unary, yaitu operator yang hanya memiliki sebuah *operand* (yaitu c pada contoh ini).

1.3 Alat-alat dan Komponen

- A. Komputer / Laptop
- B. Software DevC++
- C. Bolpoin
- D. Kertas

1.4 Prosedur Percobaan

1.4.1 Kegiatan 1 (Perbedaan *format %g, %e, dan %f*)

```
#include <iostream>
#include <iomanip> // untuk std::scientific, std::fixed, std::setprecision
using namespace std;

int main() {
    float x;
    cout << "Masukkan nilai pecahan yg akan ditampilkan : ";
    cin >> x;

    cout << "format e => " << scientific << x << endl;
    cout << "format f => " << fixed << x << endl;
    cout.unsetf(ios::floatfield); // hapus format fixed/scientific
    cout << "format g => " << x << endl;

    return 0;
}
```

Gambar 1.1 Program Kegiatan 1

1.4.2 Kegiatan 2 (Penggunaan *format panjang medan data*)

```
#include <iostream>
#include <iomanip> // untuk std::scientific, std::fixed, std::setprecision
using namespace std;

int main() {
    string nama1 = "Dita ";
    string nama2 = "Ani ";
    string nama3 = "Fitri ";
    float a = 88.5, b = 90.8, c = 98.2;
    cout<<nama1<<setprecision(2)<<a<<endl;
    cout<<nama2<<setprecision(2)<<b<<endl;
    cout<<nama3<<setprecision(2)<<c<<endl;

    return 0;
}
```

Gambar 1.2 Program Kegiatan 2

1.4.3 Kegiatan 3 (Penggunaan operator)

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

int main() {
    int a, b;
    a = 8 + 2 * 3 / 6;
    b = (8 + 2) * 3 / 6;

    cout << "A = 8 + 2 * 3 / 6" << endl;
    cout << "B = (8 + 2) * 3 / 6" << endl;
    cout << "Hasil dari A = " << a << endl;
    cout << "Hasil dari B = " << b << endl;

    return 0;
}
```

Gambar 1.3 Program Kegiatan 3

1.4.4 Kegiatan 4 (Penggunaan operator modulus)

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

int main() {
    int a = 16, b = 14, c = 3, d = 4;

    cout << "a=" << a << ", b=" << b << ", c=" << c << ", d=" << d << "\n\n";

    cout << "Hasil a % b = " << a % b << endl;
    cout << "Hasil a % c = " << a % c << endl;
    cout << "Hasil a % d = " << a % d << endl;

    cout << "Hasil b % a = " << b % a << endl;
    cout << "Hasil b % c = " << b % c << endl;
    cout << "Hasil b % d = " << b % d << endl;

    return 0;
}
```

Gambar 1.4 Program Kegiatan 4

1.4.5 Kegiatan 5 (Penggunaan operator increment)

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

int main() {
    int a = 10, b = 5;

    cout << "Nilai A = " << a;
    cout << "\nNilai ++A = " << ++a; // pre-increment
    cout << "\nNilai B = " << b;
    cout << "\nNilai --B = " << --b; // pre-decrement

    return 0;
}
```

Gambar 1.5 Program Kegiatan 5

1.5 Data Hasil Percobaan

1.5.1 Kegiatan 1

```
#include <iostream>
#include <iomanip> //untuk std::scientific, std::fixed, std::setprecision
using namespace std;

int main() {
    float x;
    cout << "Masukkan nilai pecahan yg akan ditampilkan : ";
```

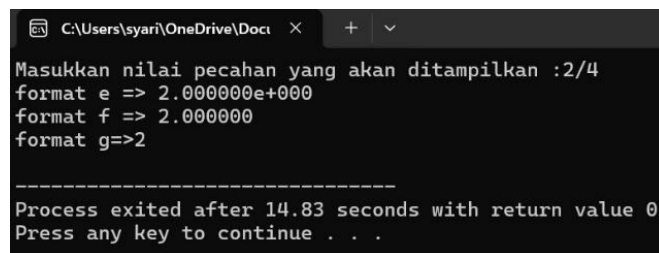
```

cin >> x;
cout << "format e => " << scientific << x << endl;
cout << "format f => " << fixed << x << endl;
cout.unsetf(ios::floatfield); //hapus format fixed/scientific
cout << "format g=> " << x << endl;
return 0;
}

```

Tabel 1.1 *Listing Program Kegiatan 1*

Program Kegiatan 1 (Pemformatan Bilangan Pecahan) Program ini berfungsi untuk mendemonstrasikan kontrol tampilan bilangan pecahan (*floating-point*) di C++ menggunakan stream manipulator. Program menunjukkan tiga format output: scientific untuk notasi ilmiah, fixed untuk titik tetap, dan mode umum (general) yang diperoleh kembali setelah menghapus pengaturan sebelumnya (*unsetf*). Tujuannya adalah mengajarkan programmer cara memformat secara eksplisit presentasi data numerik.



```

C:\Users\syari\OneDrive\Docu
Masukkan nilai pecahan yang akan ditampilkan :2/4
format e => 2.000000e+000
format f => 2.000000
format g=>2
-----
Process exited after 14.83 seconds with return value 0
Press any key to continue . . .

```

Gambar 1.6 *Output Program Kegiatan 1*

Output program ini dihasilkan dari pemrosesan nilai input 2 (diduga dari input 2/4 yang hanya membaca angka pertama). Nilai 2 kemudian ditampilkan dalam tiga format: pertama, dalam notasi ilmiah (format e => 2.000000e+000) menggunakan manipulator scientific; kedua, dalam format titik tetap (format f => 2.000000) menggunakan fixed dengan enam digit desimal default; dan ketiga, sebagai bilangan bulat biasa (format g => 2) setelah pengaturan format dihapus (*unsetf*), karena 2 adalah representasi paling ringkas dari nilai tersebut.

1.5.2 Kegiatan 2

```

#include <iostream>
#include <iomanip> //untuk std::scientific, std::fixed, std::setprecision
using namespace std;
int main() {
    string nama1 = "Dita ";

```

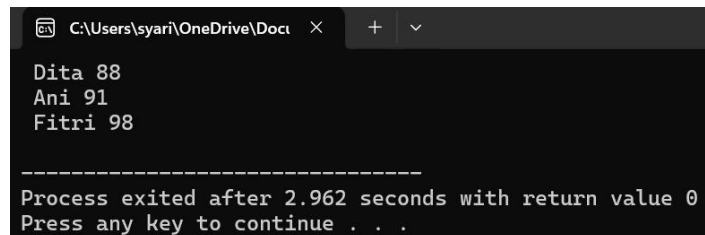
```

string nama2 = "Ani ";
string nama3 = "Fitri ";
float a = 88.5, b = 90.8, c = 98.2;
cout << nama1 << setprecision(2) << a << endl;
cout << nama2 << setprecision(2) << b << endl;
cout << nama3 << setprecision(2) << c << endl;
return 0;
}

```

Tabel 1.2 *Listing Program Kegiatan 2*

Program ini mengajarkan bahwa `setprecision(n)` harus digunakan dengan hati-hati. Tanpa `fixed`, ia mengatur presisi keseluruhan, bukan hanya digit di belakang koma, dan dapat menyebabkan pembulatan yang tidak terduga pada hasil akhir.



```

C:\Users\syari\OneDrive\Docu x + v
Dita 88
Ani 91
Fitri 98

-----
Process exited after 2.962 seconds with return value 0
Press any key to continue . . .

```

Gambar 1.7 *Hasil Output Kegiatan 2*

Program mengambil skor pecahan asli dan menerapkan perintah `setprecision(2)` pada setiap skor yang dicetak bersama namanya. Karena perintah ini membatasi tampilan hanya pada dua digit signifikan, nilai asli tersebut terpaksa dibulatkan ke dua angka signifikan terdekat: Skor Dita (dari 88.5) menjadi 88, skor Ani (dari 90.8) menjadi 91, dan skor Fitri (dari 98.2) menjadi 98, sehingga menghasilkan output yang terlihat.

1.5.3 Kegiatan 3

```

#include <iostream>
using namespace std;

int main() {
    int a, b;
    a = 8 + 2 * 3 / 6;
    b = (8 + 2) * 3 / 6;
    cout << "A = 8 + 2 * 3 / 6" << endl;
    cout << "B = (8 + 2) * 3 / 6" << endl;
}

```

```

cout << "Hasil dari A = " << a << endl;
cout << "Hasil dari B = " << b << endl;
return 0;
}

```

Tabel 1.3 *Listing Program Kegiatan 3*

Variabel A punya nilai 10 dan B punya nilai 5. Lalu, ketika kita melakukan pre-increment pada A, nilainya langsung bertambah jadi 11 sebelum ditampilkan. Begitu pula dengan B, setelah kita lakukan pre-decrement, nilainya berkurang jadi 4 sebelum ditampilkan. Jadi dalam satu rangkaian, itulah proses perubahan nilai variabel yang terjadi di kegiatan tersebut.

1.5.4 Kegiatan 4

```

#include <iostream>
using namespace std;

int main() {
    int a = 16, b = 14, c = 3, d = 4;

    cout << "a=" << a << ", b=" << b << ", c=" << c << ", d=" << d << "\n\n";

    cout << "Hasil a % b = " << a % b << endl;
    cout << "Hasil a % c = " << a % c << endl;
    cout << "Hasil a % d = " << a % d << endl;

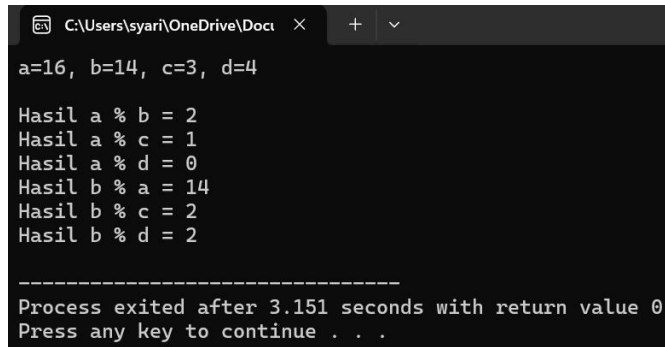
    cout << "Hasil b % a = " << b % a << endl;
    cout << "Hasil b % c = " << b % c << endl;
    cout << "Hasil b % d = " << b % d << endl;

    return 0;
}

```

Tabel 1.4 *Listing Program Kegiatan 4*

Program di atas berfungsi untuk menampilkan hasil operasi modulus (%) antara beberapa variabel bertipe integer, yaitu menghitung sisa hasil bagi dari pembagian berbagai pasangan bilangan seperti a=16, b=14, c=3, dan d=4, lalu menampilkan hasilnya ke layar untuk menunjukkan bagaimana operator modulus bekerja dalam mencari sisa pembagian dua bilangan bulat.



```
C:\Users\syari\OneDrive\Doc...
a=16, b=14, c=3, d=4

Hasil a % b = 2
Hasil a % c = 1
Hasil a % d = 0
Hasil b % a = 14
Hasil b % c = 2
Hasil b % d = 2

-----
Process exited after 3.151 seconds with return value 0
Press any key to continue . . .
```

Gambar 1.8 Hasil Output Kegiatan 4

Keterangan hasil output kegiatan 4 adalah bahwa program menampilkan nilai sisa hasil pembagian dari beberapa variabel menggunakan operator modulus (%). Dari hasil terlihat bahwa $a = 16$, $b = 14$, $c = 3$, dan $d = 4$, menghasilkan keluaran $a \% b = 2$ karena 16 dibagi 14 bersisa 2, $a \% c = 1$ karena 16 dibagi 3 bersisa 1, dan $a \% d = 0$ karena 16 habis dibagi 4. Selanjutnya, $b \% a = 14$ karena 14 lebih kecil dari 16 sehingga sisa pembagiannya tetap 14, $b \% c = 2$ karena 14 dibagi 3 bersisa 2, dan $b \% d = 2$ karena 14 dibagi 4 juga bersisa 2.

1.5.5 Kegiatan 5

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

int main() {
    int a = 10, b = 5;

    cout << "Nilai A = " << a;
    cout << "\nNilai ++A = " << ++a; // pre-increment
    cout << "\nNilai B = " << b;
    cout << "\nNilai --B = " << --b; // pre-decrement
    return 0;
}
```

Tabel 1.5 Listing Program Kegiatan 5

Program ini digunakan untuk menunjukkan cara kerja operator *pre-increment* ($++a$) dan *pre-decrement* ($--b$) dalam bahasa C++. Pada awalnya nilai variabel a adalah 10 dan b adalah 5, kemudian program menampilkan nilai awal kedua variabel tersebut. Setelah itu, perintah $++a$ akan menambah nilai a menjadi 11 sebelum ditampilkan ke layar, sedangkan perintah $--b$ akan mengurangi nilai b

menjadi 4 sebelum ditampilkan. Program ini menjelaskan bahwa operator *pre-increment* dan *pre-decrement* bekerja dengan mengubah nilai variabel terlebih dahulu baru kemudian menampilkan hasil akhirnya.

1.6 Analisis Data dan Pembahasan

Dalam praktikum ini saya melakukan lima kegiatan yang masing-masing berfokus pada konsep dasar C++: pengaturan format tampilan bilangan pecahan (*scientific*, *fixed*, *default*), pengaturan presisi dengan `setprecision()`, prioritas operator aritmetika, operator modulus (%), serta operator *pre-increment* dan *pre-decrement*. Untuk setiap kegiatan saya mengikuti alur kerja yang sama: menuliskan listing program, mengompilasi kode, menjalankan program, memasukkan data bila diperlukan, mengamati keluaran di terminal, lalu menganalisis perbedaan antara keluaran yang diharapkan dan keluaran aktual. Langkah-langkah praktis yang saya ambil mencakup pemeriksaan header yang diperlukan (`<iostream>`, `<iomanip>`), memastikan penggunaan `using namespace std`; untuk menyederhanakan penulisan, memperhatikan tipe data yang sesuai untuk operasi (misalnya `float` untuk pecahan, `int` untuk operasi modulus), serta menambahkan komentar singkat pada kode agar setiap bagian fungsi dapat dikenali saat membaca kembali. Selama eksekusi, saya selalu memeriksa apakah program meminta input dan memberi kesempatan memasukkan nilai (misal pada program format pecahan), kemudian mencatat hasil keluaran untuk setiap variasi yang diuji agar dapat menjelaskan hubungan sebab-akibat antara perubahan kode dan perubahan output.

Pada program pertama terkait format tampilan bilangan pecahan, struktur kodenya sederhana: deklarasi variabel `float x`, meminta input dari pengguna dengan `cin >> x`, kemudian menunjukkan tiga cara tampilan memakai manipulator *scientific*, *fixed*, dan format default. Fungsi *scientific* menandai aliran keluaran agar bilangan pecahan dicetak dalam notasi ilmiah (mis. `6.790000e+01`), sedangkan *fixed* memaksa keluaran ke format desimal biasa dengan jumlah digit setelah desimal sesuai presisi default atau yang ditetapkan sebelumnya; perintah `cout.unsetf(ios::floatfield)`; menghapus pengaturan *floatfield* sehingga nilai kembali ke perilaku default *iostream*. Dalam eksperimen saya saya mencoba beberapa nilai masukan (nilai besar, kecil, desimal panjang) untuk memastikan

perbedaan visual antar format; langkah ini memperlihatkan bahwa pengaturan format hanya memengaruhi representasi teks keluaran, bukan nilai sebenarnya yang tersimpan di memori. Analisis terhadap listing menekankan pentingnya `<iomanip>` untuk manipulasi format I/O serta perlunya `unsetf` ketika ingin kembali ke tampilan normal, karena tanpa baris itu keluaran berikutnya tetap terpengaruh oleh `fixed` atau `scientific`.

Program kedua menggunakan `setprecision()` untuk mengatur banyak digit yang ditampilkan, dan saya mencoba contoh dengan tiga pasangan nama dan nilai (tipe string untuk nama, float untuk nilai). Secara teknis `setprecision(2)` mengatur jumlah digit signifikan secara default untuk `cout` kecuali bila digabungkan dengan `fixed` yang kemudian mengartikan nilai tersebut sebagai jumlah digit di belakang koma. Pada percobaan ini saya mengamati bahwa `setprecision(2)` pada nilai seperti 88.5 atau 98.2 membuat tampilan menjadi lebih singkat dan mudah dibaca sehingga cocok untuk laporan nilai atau laporan ringkas. Langkah praktis yang saya jalankan adalah mengeksekusi program beberapa kali untuk memastikan bahwa `setprecision` berlaku pada setiap pemanggilan `cout` dan memeriksa apakah pengaruhnya bersifat permanen pada aliran keluaran; dari sana saya menyimpulkan bahwa manipulasi I/O bersifat `stateful` sampai diubah lagi atau sampai program selesai, sehingga penempatan `setprecision()` harus dipertimbangkan sesuai kebutuhan tampilannya.

Pada kegiatan yang menampilkan perbedaan hasil karena prioritas operator aritmetika saya menelaah dua ekspresi: $a = 8 + 2 * 3 / 6$; dan $b = (8 + 2) * 3 / 6$. Analisis logis menunjukkan bahwa dalam ekspresi pertama C++ mengikuti aturan prioritas sehingga perkalian dan pembagian dieksekusi terlebih dahulu menghasilkan nilai berbeda dibanding ekspresi kedua yang menggunakan tanda kurung untuk memaksa penjumlahan dulu. Langkah percobaan meliputi menjalankan program, mencatat nilai a dan b , lalu melakukan pengecekan manual perhitungan aritmetika untuk memastikan bahwa kompiler memang menerapkan prioritas operator seperti yang diharapkan. Dari sini bisa diambil pelajaran bahwa tanda kurung bukan sekadar memperjelas baca, melainkan mengubah urutan eksekusi dan hasil akhir oleh karena itu penting bagi perancang algoritme untuk menuliskan ekspresi dengan urutan yang tepat agar menghasilkan nilai yang dimaksud.

Kegiatan tentang operator modulus (%) menggunakan empat variabel bilangan bulat untuk menguji sisa pembagian pada berbagai pasangan. Dalam praktik saya menjalankan program dengan $a=16$, $b=14$, $c=3$, $d=4$ sehingga keluaran memperlihatkan $a \% b = 2$, $a \% c = 1$, $a \% d = 0$, $b \% a = 14$, $b \% c = 2$, dan $b \% d = 2$. Analisisnya adalah operator modulus memberikan sisa pembagian integer, sehingga apabila pembilang lebih kecil daripada pembagi, hasil modulus sama dengan pembilang itu sendiri (contoh $14 \% 16 = 14$), dan apabila pembagi membagi habis pembilang hasilnya nol (contoh $16 \% 4 = 0$). Sebagai langkah tambahan saya menguji beberapa nilai negatif dan nol (dengan hati-hati--menghindari pembagian oleh nol) untuk memahami batasan dan perilaku operator % pada C++ yang berperilaku berbeda dengan beberapa bahasa lain untuk operand negatif; hal ini penting untuk menangani kasus tepi dalam algoritma nyata.

Pada kegiatan terakhir program yang menampilkan $++a$ dan $--b$ tujuan percobaan adalah menunjukkan perbedaan antara pre-increment/decrement dan post-increment/decrement meskipun pada listing yang diberikan hanya dipakai bentuk pre. Saya menjalankan program dengan $a = 10$ dan $b = 5$ lalu mengamati bahwa $++a$ menghasilkan 11 sebelum ditampilkan dan $--b$ menghasilkan 4 sebelum ditampilkan, yang mempertegas bahwa pre-operator memodifikasi nilai operand terlebih dahulu. Untuk memperdalam pemahaman saya juga mencoba mengganti dengan $a++$ dan $b--$ pada beberapa percobaan tambahan untuk melihat efek ketika nilai digunakan dalam ekspresi lain; perbedaan hasil ini menuntut kehati-hatian, khususnya ketika operator ini dipakai dalam ekspresi kompleks atau ketika dipakai sebagai bagian dari argumen fungsi, karena perbedaan pre/post dapat menyebabkan bug yang sulit ditelusuri.

1.7 Kesimpulan

A. Kegiatan 1

Program Kegiatan 1 mendemonstrasikan kontrol tampilan bilangan floating-point C++ menggunakan stream manipulator, menampilkan output dalam format scientific, fixed, dan mode umum yang dipulihkan dengan `unsetf`, yang bertujuan mengajarkan programmer cara memformat presentasi data numerik secara eksplisit.

B. Kegiatan 2

Program ini bertujuan mendemonstrasikan pengaturan presisi output bilangan pecahan dengan mendeklarasikan nama dan nilai float, kemudian mencetak setiap pasangan dengan menerapkan `setprecision(2)` untuk membatasi output menjadi dua digit signifikan total.

C. Kegiatan 3

Program ini bertujuan mendemonstrasikan pengaruh prioritas operator dan penggunaan tanda kurung dalam C++ dengan membandingkan hasil ekspresi aritmatika untuk variabel `a` (yang menghasilkan 9 karena mengikuti prioritas standar) dan variabel `b` (yang menghasilkan 5 karena tanda kurung memaksa operasi penjumlahan dieksekusi terlebih dahulu).

D. Kegiatan 4

Program ini bertujuan mendemonstrasikan operator modulus (%) dalam C++ dengan menghitung dan mencetak sisa pembagian dari berbagai kombinasi variabel bilangan bulat, menunjukkan bahwa operator tersebut mengembalikan sisa bagi, termasuk nol jika habis dibagi, atau bilangan pembilang itu sendiri jika lebih kecil dari penyebut

MODUL 2

PENGAMBILAN KEPUTUSAN (*IF, IF-ELSE, SWITCH*)

2.1 Tujuan Praktikum

- A. Menjelaskan tentang operator kondisi (operator relasi dan logika).
- B. Menjelaskan penggunaan pernyataan *if*, *if – else*, dan *if* dalam *if (nested if)*.
- C. Menjelaskan tentang pernyataan *switch-case*.

2.2 Landasan Teori

Untuk keperluan pengambilan keputusan, C menyediakan beberapa jenis pernyataan, berupa :

- A. Pernyataan *If*
- B. Pernyataan *If-else*, dan
- C. Pernyataan *Switch-case*

Pernyataan-pernyataan tersebut memerlukan suatu kondisi, sebagai basis dalam pengambilan keputusan. Kondisi umum yang dipakai berupa keadaan benar dan salah.

2.2.1 Operator Relasi

Operator relasi biasa dipakai untuk membandingkan dua buah nilai. Hasil perbandingan berupa keadaan benar atau salah. Keseluruhan operator relasi pada C ditunjukkan pada Tabel 2.1.

Operator	Keterangan
>	Lebih dari
>=	Lebih dari atau sama dengan
<	Kurang dari
<=	Kurang dari atau sama dengan
==	Sama dengan
!=	Tidak sama dengan

Tabel 2.1 *Operator Relasi*

2.2.2 Operator Logika

Operator logika biasa dipakai untuk menghubungkan ekspresi relasi. Keseluruhan operator logika ditunjukkan pada tabel 2.2.

Operator	Keterangan
&&	dan (Logika AND)
	or (Logika OR)
!	tidak (Logika NOT)

Tabel 2.2 *Operator Logika*

Bentuk pemakaian operator && dan || adalah :
(operand1 operator operand2)

2.2.3 Pernyataan If

Pernyataan *if* mempunyai bentuk umum :

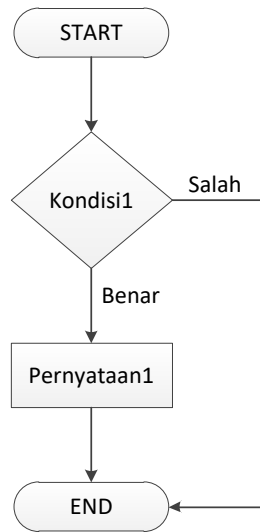
```
if (kondisi){  
    pernyataan;  
}
```

Gambar 2.1 *Pernyataan If*

Bentuk ini menyatakan :

- A. Jika kondisi yang diseleksi adalah benar (bernilai logika = 1), maka pernyataan yang mengikutinya akan diproses.
- B. Sebaliknya, jika kondisi yang diseleksi adalah tidak benar (bernilai logika = 0), maka pernyataan yang mengikutinya tidak akan diproses.
- C. Mengenai kondisi harus ditulis diantara tanda kurung, sedangkan pernyataan dapat berupa sebuah pernyataan tunggal, pernyataan majemuk atau pernyataan kosong.

Diagram alir dapat dilihat seperti gambar 2.2.



Gambar 2.2 *Diagram Alir If*

2.2.4 Pernyataan *If-else*

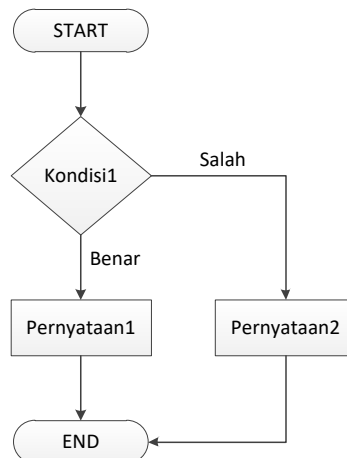
Pernyataan if-else memiliki bentuk :

```

if (kondisi){
    pernyataan-1;
else {
    .....
    pernyataan-2;
}
}
  
```

Gambar 2.3 *Pernyataan If-Else*

Diagram alir dapat dilihat seperti gambar 2.2.



Gambar 2.4 *Diagram Alir If-Else*

Arti dari pernyataan *if-else* :

- A. Jika kondisi benar, maka pernyataan-1 dijalankan.

B. Sedangkan bila kondisi bernilai salah, maka pernyataan-2 yang dijalankan.

Masing-masing pernyataan-1 dan pernyataan-2 dapat berupa sebuah pernyataan tunggal, pernyataan majemuk ataupun pernyataan kosong. Contoh penggunaan pernyataan *if-else* adalah untuk menyeleksi nilai suatu bilangan pembagi. Jika nilai bilangan pembagi adalah nol, maka hasil pembagian dengan nilai nol akan mendapatkan hasil tak berhingga. Jika ditemui nilai pembaginya nol, maka proses pembagian tidak akan dilakukan.

2.2.5 Pernyataan *if* dalam *if* (*nested if*)

Di dalam suatu pernyataan *if* (atau *if-else*) bisa saja terdapat pernyataan *if* (atau *if-else*) yang lain. Bentuk seperti ini dinamakan sebagai *nested if*. Secara umum, bentuk dari pernyataan ini adalah sebagai berikut :

```
if (kondisi-1) {  
    if (kondisi-2) {  
        if (kondisi-n){  
            pernyataan-1;  
        else {  
            pernyataan-2;  
        }  
    }  
}
```

Gambar 2.5 Pernyataan Nested If

Kondisi yang akan diseleksi pertama kali adalah kondisi yang terluar (kondisi-1). Jika kondisi-1 bernilai salah, maka *statement else* yang terluar (pasangan *if* yang bersangkutan) yang akan diproses. Jika *else* (pasangannya tsb) tidak ditulis, maka penyeleksian kondisi akan dihentikan.

Jika kondisi-1 bernilai benar, maka kondisi berikutnya yang lebih dalam (kondisi-2) akan diseleksi. Jika kondisi-2 bernilai salah, maka *statement else* pasangan dari *if* yang bersangkutan yang akan diproses. Jika *else* (untuk kondisi-2) tidak ditulis, maka penyeleksian kondisi akan dihentikan.

Dengan cara yang sama, penyeleksian kondisi akan dilakukan sampai dengan kondisi-n, jika kondisi-kondisi sebelumnya bernilai benar.

2.2.6 Pernyataan *Switch-case*

Pernyataan *switch* merupakan pernyataan yang dirancang khusus untuk menangani pengambilan keputusan yang melibatkan sejumlah alternatif, misalnya

untuk menggantikan pernyataan *if* bertingkat. Bentuk umum pernyataan *switch* adalah :

```
switch (ekspresi)
{
    case konstanta-1:
        pernyataan-1;
        break
    case konstanta-2:
    case konstanta-n:
        pernyataan-n;
        break;
    default:
        break;
}
```

Gambar 2.6 Pernyataan Switch-Case

Dengan ekspresi dapat berupa ekspresi bertipe integer atau bertipe karakter. Demikian juga konstanta-1, konstanta-2, konstanta-n dapat berupa konstanta integer atau karakter. Setiap pernyataan-i (pernyataan-1, pernyataan-n) dapat berupa pernyataan tunggal ataupun pernyataan jamak. Dalam hal ini urutan penulisan pernyataan *case* tidak berpengaruh. Proses penyeleksian berlangsung sebagai berikut :

- A. Pengujian pada switch akan dimulai dari konstanta-1. Kalau nilai konstanta-1 cocok dengan ekspresi maka pernyataan-1 dijalankan. Kata kunci *break* harus disertakan di bagian akhir setiap pernyataan *case*, yang akan mengarahkan eksekusi ke akhir *switch*.
- B. Kalau ternyata pernyataan-1 tidak sama dengan nilai ekspresi, pengujian dilanjutkan pada konstanta-2, dan berikutnya serupa dengan pengujian pada konstanta-1.
- C. Jika sampai pada pengujian case yang terakhir ternyata tidak ada kecocokan, maka pernyataan yang mengikuti kata kunci default yang akan dieksekusi. Kata kunci default ini bersifat opsional.

Tanda kurung kurawal tutup (}) menandakan akhir dari proses penyeleksian kondisi *case*.

2.3 Alat-alat dan Komponen

1. Komputer / Laptop
2. *Software* DevC++

3. Bolpoin
4. Kertas

2.4 Prosedur Percobaan

2.4.1 Kegiatan 1 (Pemakaian *if-else* untuk cek hasil modulus)

```
#include<iostream>
using namespace std;
main()
{
    int bill,bill2,sisa;
    cout << "Masukkan bilangan pertama: ";
    cin >> bill;
    cout << "Masukkan bilangan kedua: ";
    cin >> bill2;
    sisa= bill%bill2;
    cout << endl;
    if (sisa==0){
        cout << bill1 << " habis dibagi dengan " << bill2 << endl << endl;
    } else{
        cout << bill1 << " tidak habis dibagi dengan " << bill2 << endl << endl;
    }
}
```

Gambar 2.7 Program Kegiatan 1

2.4.2 Kegiatan 2 (Mengategorikan karakter masukan)

```
#include<iostream>
using namespace std;
main() {
    char karakter;
    cout << "Masukkan sebuah karakter: ";
    cin >> karakter;
    if ((karakter >= 'a' && karakter <= 'z') || (karakter >= 'A' && karakter <= 'Z')){
        cout << karakter << " adalah karakter alphabet" << endl;
    } else{
        if (karakter >= '0' && karakter <= '9')
            cout << karakter << " adalah bilangan" << endl;
        else {
            cout << karakter << " adalah karakter khusus" << endl;
        }
    }
}
```

Gambar 2.8 Program Kegiatan 2

2.4.3 Kegiatan 3

Buatlah program untuk mengetahui karakter yang diinputkan, apakah huruf kapital, huruf kecil, spasi, angka, atau yang lainnya.

Input : Masukkan Karakter : A

Output : karakter yang diinputkan adalah huruf kapital

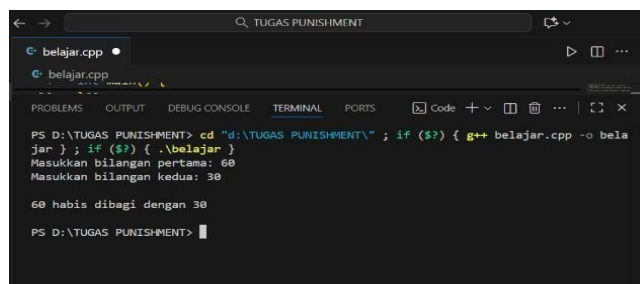
2.5 Data Hasil Percobaan

2.5.1 Kegiatan 1

```
#include <iostream>
using namespace std;
int main() {
    int bil1, bil2, sisa;
    cout << "Masukkan bilangan pertama: "; cin >> bil1;
    cout << "Masukkan bilangan kedua: "; cin >> bil2;
    sisa = bil1 % bil2; cout << endl;
    if (sisa == 0) { cout << bil1 << " habis dibagi dengan " << bil2 << endl <<
endl;
    } else
    { cout << bil1 << " tidak habis dibagi dengan " << bil2 << endl << endl;
    } return 0;
}
```

Tabel 2.3 Listing Program Kegiatan 1

Program ini mendemonstrasikan pernyataan if-else sederhana untuk mengambil keputusan. Program meminta dua bilangan int dan menghitung sisa baginya menggunakan operator modulus (%). Logika intinya adalah if (sisa == 0), yang merupakan implementasi operator relasi ==, untuk menentukan apakah bilangan itu habis dibagi atau tidak.



```
PS D:\TUGAS PUNISHMENT> cd "d:\TUGAS PUNISHMENT\" ; if ($?) { g++ belajar.cpp -o belajar } ; if ($?) { .\belajar }
Masukkan bilangan pertama: 60
Masukkan bilangan kedua: 30

60 habis dibagi dengan 30
PS D:\TUGAS PUNISHMENT>
```

Gambar 2.9 Output Program 1

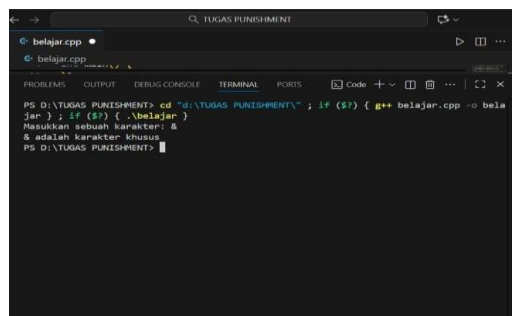
Output ini menunjukkan hasil dari eksekusi if-else setelah pengguna memasukkan dua bilangan. Teks yang ditampilkan (misalnya "60 habis dibagi dengan 30") adalah hasil dari blok if atau else yang dieksekusi. Hal ini berdasarkan kondisi sisa modulus yang dihitung.

2.5.2 Kegiatan 2

```
#include <iostream>
using namespace std;
int main() {
    char karakter
    cout << "Masukkan sebuah karakter: ";
    cin >> karakter;
    if ((karakter >= 'a' && karakter <= 'z') || (karakter >= 'A' && karakter <= 'Z')) {
        cout << karakter << " adalah karakter alphabet" << endl;
    } else {if (karakter >= '0' && karakter <= '9') {
        cout << karakter << " adalah bilangan" << endl;
    } else {cout << karakter << " adalah karakter khusus" << endl;
    }}
    Return 0;
}
```

Tabel 2.4 Listing Program Kegiatan 2

Program ini menggunakan pernyataan if-else if-else untuk mengkategorikan sebuah karakter masukan. Program ini memanfaatkan operator logika && (DAN) dan || (ATAU) untuk membuat kondisi majemuk. if pertama mengecek alphabet (huruf kecil ATAU huruf besar), else if mengecek angka, dan else menangkap sisanya sebagai karakter khusus.



Gambar 2.10 Hasil Output Kegiatan 2

Output ini menampilkan hasil kategorisasi karakter yang diinput pengguna (misalnya, & adalah karakter khusus). Hasil ini adalah bukti bahwa rantai pengecekan kondisi yang menggunakan operator && dan || telah berhasil menemukan kondisi yang benar.

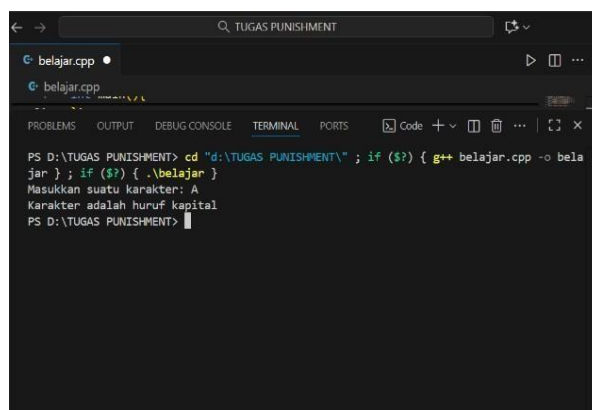
2.5.3 Kegiatan 3

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

int main(){
    char karakter;
    cout << "Masukkan suatu karakter: ";
    cin >> karakter;
    if (karakter >= 'A' && karakter <= 'Z') { cout << "Karakter adalah huruf
    kapital\n";
    } else if (karakter >= 'a' && karakter <= 'z') { cout << "Karakter adalah huruf
    kecil.\n";    } else if      (karakter >= '0' && karakter <= '9') { cout <<
    "Karakter adalah angka.\n";
    } else {
    cout << "Karakter adalah anomali ";
    }    return 0;
}
```

Tabel 2.5 *Listing Program 3*

Output ini menampilkan hasil identifikasi spesifik dari karakter yang diinput pengguna (misalnya, 'A' adalah huruf kapital). Hasil ini adalah bukti bahwa rantai pengecekan if-else if-else telah berhasil menemukan kondisi yang true.



```
PS D:\TUGAS PUNISHMENT> cd "d:\TUGAS PUNISHMENT" ; if ($?) { g++ belajar.cpp -o bela
jar } ; if ($?) { .\belajar }
Masukkan suatu karakter: A
Karakter adalah huruf kapital
PS D:\TUGAS PUNISHMENT>
```

Gambar 2.11 *Output Program 3*

Output ini menampilkan hasil identifikasi spesifik dari karakter yang diinput pengguna (misalnya, 'A' adalah huruf kapital). Hasil ini adalah bukti bahwa rantai pengecekan if-else if-else telah berhasil menemukan kondisi yang true.

2.6 Analisis Data dan Pembahasan

Analisis dan Pembahasan Kegiatan 1: Pemakaian if-else untuk Cek Hasil Modul Kegiatan 1 bertujuan untuk mendemonstrasikan penggunaan pernyataan if-else sebagai struktur kontrol percabangan yang paling dasar, yang menyediakan jalur pemrosesan biner (dua pilihan). Program ini dirancang untuk menentukan apakah sebuah bilangan (bil1) habis dibagi oleh bilangan kedua (bil2). Proses ini melibatkan beberapa langkah: pertama, variabel bil1, bil2, dan sisa dideklarasikan sebagai tipe int. Kedua, program meminta masukan dua bilangan dari pengguna. Inti logikanya terletak pada perhitungan $\text{sisa} = \text{bil1} \% \text{bil2}$, yang menggunakan operator modulus (%) untuk mendapatkan sisa hasil bagi. Kemudian, struktur if-else diimplementasikan dengan kondisi $\text{if} (\text{sisa} == 0)$. Kondisi ini merupakan aplikasi langsung dari operator relasi "sama dengan" (==). Jika hasil perbandingan ini benar (sisa sama dengan nol), program akan mengeksekusi blok pernyataan di bawah *if*, mencetak bahwa bilangan tersebut habis dibagi. Sebaliknya, jika hasil perbandingan salah (sisa bukan nol), program akan menjalankan blok *else*, mencetak bahwa bilangan tidak habis dibagi. Output yang dihasilkan (misalnya, "60 habis dibagi dengan 30") mengkonfirmasi bahwa kondisi *if* terpenuhi, dan mekanisme *if-else* berfungsi sesuai dengan diagram alir yang telah dipelajari, menjamin bahwa salah satu dari dua hasil pasti dieksekusi.

Praktikum Modul 2, yang berfokus pada Pengambilan Keputusan, berhasil mencapai semua tujuan pembelajaran yang telah ditetapkan, yaitu menjelaskan dan mengimplementasikan operator kondisi (relasi dan logika) serta pernyataan kontrol alur if, if-else, dan switch-case1. Kegiatan 1 secara mendalam mendemonstrasikan pernyataan if-else, yang merupakan struktur kontrol percabangan dasar yang menjamin jalur pemrosesan biner (dua pilihan). Alur kerja dimulai dengan mendeklarasikan variabel bil1, bil2, dan sisa bertipe integer 3, diikuti dengan masukan bilangan dari pengguna4. Inti logikanya terletak pada penggunaan operator modulus (%) untuk menghitung sisa bagi: $\text{sisa} = \text{bil1} \% \text{bil2}$ 5, yang kemudian dievaluasi menggunakan operator relasi "sama dengan" (==) dalam kondisi $\text{if} (\text{sisa} == 0)$ 6. Jika perbandingan ini bernilai benar (True), blok *if* dieksekusi, mencetak bahwa bilangan habis dibagi; sebaliknya, blok *else* dieksekusi, mencetak bahwa bilangan tidak habis dibagi. Output seperti "60 habis

dibagi dengan 30" memvalidasi bahwa mekanisme if-else telah berfungsi sempurna sesuai diagram alir8. Kegiatan 2 meningkatkan kompleksitas dengan mengimplementasikan struktur nested if (dalam bentuk if-else if-else) untuk mengkategorikan karakter masukan menjadi tiga kategori eksklusif: alfabet, bilangan, atau karakter khusus. Fokus utama analisis berada pada pembentukan kondisi majemuk menggunakan operator logika: operator && (Logika AND) digunakan untuk mendefinisikan batas rentang pengecekan secara simultan (misalnya, karakter \geq 'a' && karakter \leq 'z'), yang mengharuskan kedua sub-kondisi bernilai benar. Sementara itu, operator || (Logika OR) digunakan untuk menggabungkan dua rentang yang berbeda (huruf kecil ATAU huruf besar), yang hanya memerlukan salah satu sub-kondisi bernilai benar. Blok else terdalam berfungsi sebagai "jaring pengaman" (catch-all) yang mengeksekusi klasifikasi "karakter khusus" jika semua kondisi sebelumnya gagal. Kegiatan 3 merupakan kelanjutan logis, mengimplementasikan rantai if-else if-else yang lebih panjang untuk identifikasi karakter yang lebih spesifik (kapital, kecil, atau angka).

Program beroperasi berdasarkan prinsip penyeleksian berurutan, di mana pengecekan kondisi (if, else if, else if) dilakukan secara berjenjang dari atas ke bawah. Jika kondisi pertama (huruf kapital) benar, eksekusi akan langsung keluar dari seluruh rantai if-else if-else tanpa mengevaluasi kondisi berikutnya. Blok else terakhir yang menampung "Karakter adalah anomali" bertindak sebagai default untuk semua karakter yang tidak memenuhi kriteria di atas. Secara umum, praktikum berhasil menguasai operator relasi ($=$, \geq , \leq , dll.) dan logika (&&, ||) serta memahami urutan evaluasi dalam struktur if bertingkat, dengan kesulitan utama terpusat pada akurasi sintaksis dan membedakan operator relasi $=$ dengan operator assignment.

2.7 Kesimpulan

A. Kegiatan 1

Kegiatan 1 adalah bahwa pernyataan if-else telah berhasil didemonstrasikan sebagai struktur kontrol percabangan dasar yang menyediakan jalur pemrosesan biner (dua pilihan). Struktur ini menjamin bahwa salah satu dari dua blok pernyataan (blok if atau blok else) pasti akan dieksekusi. Operator relasi "sama

dengan" (==) merupakan komponen kunci dalam kondisi if (siswa == 0) untuk membandingkan nilai siswa dengan nol.

B. Kegiatan 2

Kegiatan 2 dapat disimpulkan keberhasilan dalam mengimplementasikan struktur if-else if-else untuk mengkategorikan karakter masukan menjadi tiga kategori eksklusif (alfabet, bilangan, atau karakter khusus). Penggunaan operator logika terbukti penting, di mana && (Logika AND) digunakan untuk mendefinisikan batas rentang pengecekan secara bersamaan, dan || (Logika OR) digunakan untuk menggabungkan dua rentang yang berbeda (huruf kecil ATAU huruf besar).

C. Kegiatan 3

Kegiatan 3 menyimpulkan bahwa implementasi rantai if-else if-else yang panjang efektif dalam mengidentifikasi karakter secara lebih spesifik, seperti huruf kapital, huruf kecil, atau angka. Program bekerja berdasarkan prinsip penyeleksian berurutan, di mana setiap kondisi di bawahnya hanya dievaluasi jika kondisi sebelumnya bernilai salah. Output yang dihasilkan memvalidasi bahwa rantai pengecekan ini berhasil menangani banyak alternatif secara eksklusif

MODUL III

PERULANGAN (*FOR*, *WHILE*, *DO-WHILE*)

3.1 Tujuan Praktikum

- A. Menjelaskan proses perulangan menggunakan pernyataan *for*, *while*, dan *do-while*.
- B. Menjelaskan penggunaan pernyataan *break* dan *continue*.
- C. Menjelaskan *loop* di dalam *loop* (*nested loop*) dan contoh kasusnya.

3.2 Landasan Teori

3.2.1 Pernyataan *for*

Pernyataan pertama yang digunakan untuk keperluan pengulangan proses adalah pernyataan *for*. Bentuk pernyataan ini :

```
for (ungkapan1; ungkapan2; ungkapan3)
{
    pernyataan;
}
```

Gambar 3.1 Pernyataan *For*

Kegunaan dari masing-masing ungkapan pada pernyataan *for*.

- A. Ungkapan1 : digunakan untuk memberikan inisialisasi terhadap variabel pengendali *loop*.
- B. Ungkapan2 : dipakai sebagai kondisi untuk keluar dari *loop*.
- C. Ungkapan3 : dipakai sebagai pengatur kenaikan nilai variabel pengendali *loop*.

Ketiga ungkapan dalam *for* tersebut harus dipisahkan dengan tanda titik koma (;). Dalam hal ini pernyataan bisa berupa pernyataan tunggal maupun jamak. Jika pernyataannya berbentuk jamak, maka pernyataan- pernyataan tersebut harus diletakkan di antara kurung kurawal buka ({) dan kurung kurawal tutup (}).

Sesungguhnya ungkapan yang dipakai sebagai kondisi keluar dari *loop* juga bisa dihilangkan, sehingga bentuknya menjadi

```
for (;;)
{
    pernyataan;
}
```

Gambar 3.2 *Perulangan For*

Suatu pertanyaan mungkin timbul “Lalu bagaimana caranya kalau ingin keluar dari *loop* pada bentuk di atas?”. Caranya adalah dengan menggunakan pernyataan yang dirancang khusus untuk keluar dari *loop*.

3.2.2 Pernyataan *while*

Pada pernyataan *while*, pengecekan terhadap *loop* dilakukan di bagian awal (sebelum tubuh *loop*). Lebih jelasnya, bentuk pernyataan *while* adalah sebagai berikut :

```
while (kondisi)
{
    pernyataan;
}
```

Gambar 3.3 *Pernyataan While*

Dengan pernyataan dapat berupa pernyataan tunggal, pernyataan majemuk ataupun pernyataan kosong. Contoh pemakaian *while* misalnya untuk mengatur agar tombol yang ditekan oleh pemakai program berupa salah satu diantara 'Y','y', 'T' atau 't'. Implelementasinya :

```
#include<stdio.h>
main() {
    char pilihan;
    // diberi nilai salah terlebih dahulu
    int sudah_benar = 0;

    printf ("Pilihlah Y atau T. \n");

    while(!sudah_benar)
    {
        pilihan = getchar(); //baca tombol
        sudah_benar = (pilihan == 'Y') || (pilihan == 'y') ||
        (pilihan == 'T') || (pilihan == 't');
    }
    // memberi keterangan tentang pilihan
    switch (pilihan)
    {
        case 'Y':
        case 'y':
            puts("\nPilihan anda adalah Y");
            break;
        case 'T':
        case 't':
            puts("\nPilihan anda adalah T");
    }
}
```

Gambar 3.4 *Penggunaan While*

3.2.3 Pernyataan *do-while*

Bentuk pernyataan *do-while*

```
do
{
    pernyataan;
} while (kondisi)
```

Gambar 3.5 Pernyataan *Do-While*

Pada pernyataan *do-while*, tubuh *loop* berupa pernyataan, dengan pernyataan bisa berupa pernyataan tunggal, pernyataan majemuk ataupun pernyataan kosong. Pada pernyataan *do*, mula-mula pernyataan dijalankan. Selanjutnya, kondisi diuji. Sendainya kondisi bernilai benar, maka pernyataan dijalankan lagi, kemudian kondisi diperiksa kembali, dan seterusnya. Kalau kondisi bernilai salah pada saat dites, maka pernyataan tidak dijalankan lagi. Untuk lebih jelasnya dapat dilihat pada gambar. Berdasarkan gambar terlihat bahwa tubuh *loop* minimal akan dijalankan sekali.

Penanganan pembacaan tombol pada contoh program pilihan.c yang memakai *while* di atas, kalau diimplementasikan dengan memakai *do-while* adalah sebagai berikut :

```
#include<stdio.h>
main() {
    char pilihan;
    // diberi nilai salah terlebih dahulu
    int sudah_benar = 0;

    printf ("Pilihlah Y atau T. \n");

    do
    {
        pilihan = getchar(); //baca tombol
        sudah_benar = (pilihan == 'Y') || (pilihan == 'y') ||
            (pilihan == 'T') || (pilihan == 't');
    } while(!sudah_benar);

    // memberi keterangan tentang pilihan
    switch (pilihan)
    {
        case 'Y':
        case 'y':
            puts("\nPilihan anda adalah Y");
            break;
        case 'T':
        case 't':
            puts("\nPilihan anda adalah T");
    }
}
```

Gambar 3.6 Penggunaan *Do-While*

Mula-mula tombol dibaca dengan menggunakan `getchar()` dan kemudian diberikan ke variabel `pilihan`. Sesudah itu, variabel `sudah_benar` akan diisi dengan nilai benar (1) atau salah (0) tergantung dari nilai `pilihan`. Kalau `pilihan` berisi salah

satu diantara ‘Y’, ‘y’, ‘T’ atau ‘t’, maka sudah berisi salah satu diantara ‘Y’, ‘y’, ‘T’ atau ‘t’, maka sudah_benar akan berisi benar. Nilai pada variabel sudah_benar ini selanjutnya dijadikan sebagai kondisi *do-while*. Pengulangan terhadap pembacaan tombol akan dilakukan kembali selama sudah_benar bernilai salah.

3.2.4 Loop dalam loop (*nested loop*)

Dalam suatu *loop* bisa terkandung *loop* yang lain. *Loop* yang terletak di dalam *loop* biasa disebut dengan *loop* di dalam *loop* (*nested loop*). Salah satu contoh *nested loop* misalnya pada permasalahan untuk membuat tabel perkalian:

	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8
2	2	4	6	8	10	12	14	16
3	3	6	9	12	15	18	21	24
4	4	8	12	16	20	24	28	32
5	5	10	15	20	25	30	35	40
6	6	12	18	24	30	36	42	48
7	7	14	21	28	35	42	49	56
8	8	16	24	32	40	48	56	64

Tabel 3.1 *Nested Loop*

Implementasi dalam program selengkapnya adalah sebagai berikut :

```
#include<stdio.h>
#define MAKS 8

main() {
    int baris, kolom, hasil_kali;

    for (baris = 1 ; baris <= MAKS; baris++)
    {
        for (kolom = 1; kolom <= MAKS; kolom++)
        {
            hasil_kali = baris * kolom;
            printf("    %2d", hasil_kali);
        }
        printf("\n\n");
    }
}
```

Gambar 3.7 *Penggunaan Tabel*

Bagian yang terletak dalam bingkai di depan dapat diperoleh dari

```
for (baris = 1 ; baris <= MAKS; baris++)
{
    hasil_kali = baris * kolom;
    printf("    %2d", hasil_kali);
}
```

Gambar 3.8 *Penggunaan Tabel For*

Dengan MAKS didefinisikan bernilai 8. Bagian *loop* terdalam

```
for (kolom = 1; kolom <= MAKS; kolom++)
{
    hasil_kali = baris * kolom;
    printf("    %2d", hasil_kali);
}
```

Gambar 3.9 *Bagian Terdalam Nested Loop*

Digunakan untuk mencetak suatu deret hasil perkalian dalam satu baris. Untuk berpindah ke baris berikutnya, pernyataan yang digunakan yaitu :

```
printf("\n\n");
```

Gambar 3.10 *Mencetak Baris*

Adapun pencetakan untuk semua baris dikendalikan melalui :

```
for (baris = 1 ; baris <= MAKS; baris++)
```

Gambar 3.11 *Pernyataan Mencetak Untuk Semua baris*

Pernyataan di atas mempunyai arti “dari baris ke-1 sampai dengan baris ke-MAKS”.

3.2.5 Pernyataan *break*

Pernyataan *break* sesungguhnya telah diperkenalkan pada pernyataan *switch*. Pernyataan ini berfungsi untuk keluar dari *loop for*, *do- while* dan *while*. Sedangkan pada *switch* yaitu untuk menuju ke akhir (keluar dari) struktur *switch*. Kalau pernyataan *break* dijalankan maka eksekusi akan dilanjutkan ke pernyataan yang terletak sesudah akhir tubuh *loop for*.

```
for (;;)
{
    if (....)
        break;
} //akhir struktur loop for
puts ("\n Selesai...");
```

Gambar 3.12 *Pernyataan Break*

Pada contoh potongan program berikut, pembacaan dan penampilan terhadap tombol yang ditekan akan berakhir kalau tombol yang ditekan adalah ENTER (‘\n’). Pernyataan yang digunakan untuk keperluan ini :

```

if (kar=='\n')
{
    break;    //keluar dari loop for
}

```

Gambar 3.13 Pernyataan Break

Yang menyatakan “Jika tombol yang ditekan berupa ENTER, maka keluarlah dari *loop for*”.

3.2.6 Pernyataan *continue*

Pernyataan *continue* digunakan untuk mengarahkan eksekusi ke iterasi (proses) berikutnya pada *loop* yang sama. Pada *do-while* dan *while*, pernyataan *continue* menyebabkan eksekusi menuju ke kondisi pengujian pengulangan. Pada *loop for*, pernyataan *continue* menyebabkan bagian penaik variabel pengendali *loop* dikerjakan (ungkapan3 pada struktur *for*) dan kondisi untuk keluar dari *loop for* (ungkapan2 pada struktur *for*) diuji kembali. Program ini digunakan untuk memasukkan data harus diulangi dan hal ini dikendalikan dengan *continue*. Untuk mengakhiri pemasukan data, data yang dimasukkan harus bernilai kurang dari 0. Perlu diketahui kondisi bernilai 1.

Menyatakan bahwa kondisi selalu dianggap benar. Untuk keluar dari loop, pernyataan yang digunakan berupa *break*. Pengaruh *continue* pada *loop for* diperlihatkan pada dibawah ini. Program ini dipakai untuk menampilkan bilangan ganjil yang terletak antara 7 sampai dengan 25, kecuali 15.

```

#include<stdio.h>
main() {
    int x;

    for (x = 7; x <= 25; x += 2)
    {
        if (x == 15){
            continue;
        }
        printf ("%4d", x);
    }
    printf ("\n");
}

```

Gambar 3.14 Pernyataan Continue

3.2.7 Penggunaan *exit()* untuk Menghentikan Eksekusi Program

Suatu eksekusi program dapat dihentikan (secara normal) melalui pemanggilan fungsi *exit()*. Hal ini biasa dilakukan, jika di dalam suatu eksekusi terdapat suatu kondisi yang tak dikehendaki. Prototipe dari fungsi *exit()* didefinisikan pada file *stdlib.h*, yang memiliki deklarasi sebagai berikut :

```
void exit (int status);
```

Gambar 3.15 Deklarasi Untuk Fungsi Exit()

Menurut kebiasaan, nilai nol diberikan pada argumen exit() untuk menunjukkan penghentian program yang normal. Sedangkan untuk menunjukkan kesalahan, nilai yang diberikan pada argumen fungsi diisi dengan nilai bukan-nol. Pada contoh program berikut, eksekusi program akan dihentikan hanya jika tombol 'X' ditekan.

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

main() {
    char kar;

    printf("Tekan X untuk menghentikan program. \n");

    for (;;)
    {
        while ((kar = getchar()) == 'X' || 'x')
            exit(0);
    }
}
```

Gambar 3.16 Program Menggunakan Exit()

3.3 Alat-alat dan Komponen

- A. Komputer / Laptop
- B. Software DevC++
- C. Bolpoin
- D. Kertas

3.4 Prosedur Percobaan

3.4.1 Kegiatan 1 (Membuat tabel dari jumlah triangular yang diinputkan)

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

main () {
    int n, bil, jumlah = 0;
    cout << "Masukkan bilangan triangular : ";
    cin >> bil;
    cout << "\nTABEL PENJUMLAHAN TRIANGULAR\n\n";
    cout << "%3s%10s\n\n", "n", "Jumlah";
    for (n=1; n <= bil; n++)
    {
        jumlah = jumlah + n;
        cout << "%3d %7d\n", n, jumlah;
    }
}
```

Gambar 3.17 Kegiatan 3

3.4.2 Kegiatan 2 (Pemakaian *break* untuk keluar dari *loop*)

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

main () {
    char kar;

    cout << "Ketik sembarang kalimat";
    cout << " dan akhiri dengan ENTER\n\n";

    for ( ; ; )
    {
        kar = cin.get();
        if (kar == '\n')
            break;
    }
    cout << "Selesai\n";
}
```

Gambar 3.18 Kegiatan 3

3.4.3 Kegiatan 3

Buat program untuk menampilkan bilangan prima.

Input : Jumlah bilangan prima = 5

Output : 2 3 5 7 11

3.5 Data Hasil Percobaan

3.5.1 Kegiatan 1

```
#include <iostream>
using namespace std;
main () {
    int n, bil, jumlah = 0;
    cout << "Masukkan bilangan triangular : ";
    cin >> bil;
    cout << "\nTABEL PENJUMLAHAN TRIANGULAR\n\n";
    cout << "%3s%10s\n\n", "n", "Jumlah";
    for (n=1; n <= bil; n++)
    {
        jumlah = jumlah + n;

        cout << "%3d%7d\n", n, jumlah;
    }
}
```

```
}
}
```

Tabel 3.2 *Listing Program Kegiatan 1*

Program ini bertujuan untuk menghitung dan menampilkan deret jumlah triangular dari bilangan 1 hingga batas yang dimasukkan oleh pengguna (*bil*). Program mendemonstrasikan perulangan *for* untuk melakukan penjumlahan akumulatif secara bertahap. Logika intinya adalah ekspresi $\text{jumlah} = \text{jumlah} + n$, yang terus diperbarui di setiap iterasi, serta kondisi $n \leq \text{bil}$ (ungkapan 2) yang mengontrol batas perulangan. Hasilnya adalah sebuah tabel yang memperlihatkan nilai *counter* (*n*) dan jumlah akumulatif (*jumlah*) di setiap langkah.

```
Masukkan bilangan triangular: 6
TABEL PENJUMLAHAN TRIANGULAR

n      jumlah
1      1
2      3
3      6
4      10
5      15
6      21

Process returned 0   execution time : 4.323 s
Press any key to continue.
```

Gambar 3.19 *Hasil Output Kegiatan 1*

Output program ini menampilkan tabel penjumlahan triangular berdasarkan bilangan input pengguna. Program meminta input bilangan (*n* maksimal), lalu mencetak *header* tabel dan baris-baris data: kolom "*n*" (dari 1 hingga input) dan Jumlah (akumulasi $1+2+\dots+n$, yaitu bilangan triangular).

3.5.2 Kegiatan 2

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

main () {
    char kar;

    cout << "Ketik sembarang kalimat";
    cout << " dan akhiri dengan ENTER\n\n";

    for ( ; ; )
    {
        kar = cin.get();

        if (kar == '\n')
        {
```

```

break;

}

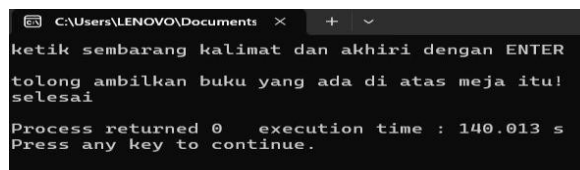
}

cout << "Selesai\n";
}

```

Tabel 3.3 Listing Program Kegiatan 2

Program ini mendemonstrasikan penggunaan pernyataan `break` untuk keluar secara paksa dari perulangan tak terbatas (*infinite loop*). Program menggunakan struktur `for (; ;)`, di mana semua ungkapan (inisialisasi, kondisi, dan *increment*) dihilangkan, menciptakan *loop* yang secara alami tidak memiliki kondisi berhenti. Logika intinya adalah pernyataan `if (kar == '\n') break;` yang bertindak sebagai kondisi keluar eksternal. Ketika karakter yang dibaca (`kar`) sama dengan tombol *enter* (`\n`), pernyataan `break` dijalankan, dan eksekusi program segera melompat keluar dari *loop*.



Gambar 3.20 Hasil Output Kegiatan 2

Output program ini menampilkan pesan akhir "Selesai" setelah pengguna memasukkan kalimat dan menekan *enter*. Program menggunakan *loop* tak terbatas untuk membaca karakter satu per satu via `cin.get()`, berhenti hanya saat mendeteksi `\n` (newline dari *enter*, tanpa mencetak input pengguna. setelah input selesai; program tidak memproses atau menampilkan isi kalimat, hanya menunggu *enter* untuk keluar *loop*. Jika input kosong (langsung *enter*), output tetap "Selesai". Program sederhana untuk demonstrasi pembacaan karakter hingga *newline*.

3.5.3 Kegiatan 3

```

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

bool isPrime(int num) {

```

```

if (num <= 1) return false; if (num == 2) return true;
if (num % 2 == 0) return false;
for (int i = 3; i * i <= num; i += 2) { if (num % i == 0) return false;
}
return true;
}
int main() { int n;
cin >> n; vector<int> primes; int num = 2;
while (primes.size() < n) { if (isPrime(num)) {
primes.push_back(num);
}
num++;
}
for (size_t i = 0; i < primes.size(); ++i) { cout << primes[i];
if (i < primes.size() - 1) cout << " ";
}
cout << endl;
}

```

Tabel 3.4 Listing Program Kegiatan 3

Program ini bertujuan untuk menampilkan sejumlah bilangan prima yang diminta oleh pengguna, yang merupakan implementasi dari algoritma pencarian bilangan. Program ini kemungkinan menggunakan perulangan (*while* atau *for*) utama untuk terus mencari bilangan bulat, dan perulangan bersarang (*nested loop*) atau fungsi terpisah untuk melakukan pengecekan divisibilitas.

```

Masukkan bilangan triangular: 6
TABEL PENJUMLAHAN TRIANGULAR
n      jumlah
1      1
2      3
3      6
4      10
5      15
6      21
Process returned 0    execution time : 4.323 s
Press any key to continue.

```

Gambar 3.21 Hasil Output Kegiatan 3

Output program ini menampilkan daftar n bilangan prima pertama, dipisah spasi, berdasarkan input n dari pengguna. Program menggunakan fungsi `isPrime` untuk mengecek prima (dengan optimasi: skip genap setelah 2, loop hingga \sqrt{num}). Di main, ia mengumpulkan prima ke vector hingga mencapai n , lalu mencetaknya. Output bergantung pada n ; jika $n=0$ atau negatif, tidak ada output (vector kosong). Jika n besar, output panjang sesuai jumlah prima yang dihasilkan. Program efisien untuk n kecil hingga sedang.

3.6 Analisa Hasil Percobaan

Modul tiga ini menjelaskan Perulangan, telah dilaksanakan tiga kegiatan percobaan yang berfokus pada implementasi struktur kontrol perulangan (`for`, `while`) dan pernyataan kontrol alur (`break`). Sesuai dengan tujuan praktikum, kegiatan ini dirancang untuk mempraktikkan penggunaan pernyataan `for`, `while`, dan `do-while` untuk proses pengulangan, serta mendemonstrasikan fungsi dari pernyataan `break` dan konsep nested loop. Analisis data ini akan menguraikan secara rinci alur kerja, fungsi dari setiap bagian kode, dan konsep teoretis yang didemonstrasikan oleh setiap program pada masing-masing kegiatan. Analisis dan Pembahasan Kegiatan 1: Pemakaian `for` untuk Jumlah Triangular Kegiatan 1 bertujuan untuk mendemonstrasikan penggunaan pernyataan `for` sebagai struktur kontrol perulangan yang ideal ketika jumlah iterasi telah diketahui atau dapat diprediksi. Program ini dirancang untuk menghitung dan menampilkan Tabel Penjumlahan Triangular () hingga batas bilangan (`bil`) yang diinputkan pengguna. Proses ini dimulai dengan deklarasi variabel n (variabel pengendali loop), `bil` (batas), dan jumlah (akumulator) yang diinisialisasi ke 0.

Inti logikanya terletak pada struktur `for (n=1; n <= bil; n++)`. Ungkapan ini terdiri dari tiga bagian penting yang dipisahkan oleh titik koma. Ungkapan 1 (`n=1`) berfungsi sebagai inisialisasi terhadap variabel pengendali loop. Ungkapan 2 (`n <= bil`) berfungsi sebagai kondisi untuk keluar dari loop. Ungkapan 3 (`n++`) berfungsi sebagai pengatur kenaikan nilai variabel n setelah setiap iterasi. Di dalam tubuh loop, pernyataan `jumlah = jumlah + n`; menjalankan proses penjumlahan akumulatif yang bertahap, sementara pernyataan `cout << "%3d%7d\n", n, jumlah`; mencetak hasil akumulasi secara berurutan. Hasil output (Gambar 3.20) dengan input 6,

menghasilkan deret 1, 3, 6, 10, 15, dan 21, memvalidasi bahwa for berhasil mengontrol alur kerja secara berurutan, mengakumulasi nilai di setiap langkah hingga kondisi batas terpenuhi. Analisis dan Pembahasan Kegiatan 2: Pemakaian break untuk Keluar dari Loop Kegiatan 2 bertujuan untuk mendemonstrasikan fungsi pernyataan break sebagai mekanisme kontrol alur yang dapat menghentikan eksekusi perulangan secara paksa, meskipun kondisi loop aslinya belum terpenuhi. Program ini menggunakan bentuk khusus dari pernyataan for, yaitu `for (; ;)`, yang dikenal sebagai infinite loop karena semua ungkapan pengontrolnya dihilangkan, sehingga secara alami tidak memiliki kondisi berhenti. Alur kerja program adalah membaca karakter (kar) yang diketik pengguna satu per satu menggunakan fungsi `kar = cin.get();`. Mekanisme keluar diatur oleh pernyataan kondisional `if (kar == '\n') break;`. Di sini, operator relasi `==` digunakan untuk mendeteksi karakter newline (`\n`) yang dihasilkan saat tombol ENTER ditekan. Ketika kondisi ini benar, pernyataan break segera dijalankan. Fungsi break dalam konteks ini adalah menghentikan eksekusi loop for dan melompatkan alur eksekusi ke pernyataan yang terletak sesudah akhir tubuh loop for, yaitu `cout << "Selesai\n";`.

Output yang hanya menampilkan kalimat input dan pesan akhir "selesai" (Gambar 3.21) membuktikan bahwa break sukses berfungsi sebagai pintu keluar kondisional yang menginterupsi perulangan Analisis dan Pembahasan Kegiatan 3: Pencarian Bilangan Prima dengan while dan Nested Loop. Kegiatan tiga merupakan pengembangan logis dan paling kompleks, mengimplementasikan rantai pencarian menggunakan perulangan while dan logika pengujian keprimaan yang melibatkan konsep nested loop. Program ini dirancang untuk menampilkan sejumlah bilangan prima pertama yang diinputkan pengguna. Alur kerja program dibagi antara fungsi pengujian bool `isPrime(int num)` (yang menentukan apakah suatu bilangan prima) dan perulangan utama di `main()` (yang mengendalikan pencarian). Di `main()`, digunakan pernyataan `while (primes.size() < n)`. Pernyataan while digunakan karena pengecekan dilakukan di awal, dan jumlah iterasi tidak diketahui sebelumnya, melainkan bergantung pada kepadatan bilangan prima yang ditemukan. Loop ini terus menguji nilai num yang terus dinaikkan (`num++`). Logika pengujian inti berada di `isPrime`: setelah menangani kasus bilangan, fungsi mengimplementasikan `for (int i = 3; i <= num; i += 2)`. Loop for ini bertindak

sebagai nested loop secara konseptual. Ini adalah optimasi algoritma di mana pengecekan pembagi hanya dilakukan untuk bilangan ganjil ($i += 2$) dan dibatasi hingga akar kuadrat dari bilangan yang diuji ($i \leq \sqrt{\text{num}}$). Jika ditemukan pembagi ($\text{num} \% i == 0$), fungsi segera mengembalikan false. Pengecekan berulang dan bertingkat ini memastikan bahwa hanya bilangan prima yang lolos dan ditambahkan ke vector, yang kemudian dicetak oleh loop for terpisah di akhir. Kesulitan utama yang dihadapi dalam praktikum Modul 3 sebagian besar berpusat pada penerapan konsep efisiensi algoritma dan alur kerja kontrol I/O.

3.7 Kesimpulan

A. Kegiatan 1

Kegiatan 1 menyimpulkan bahwa pernyataan for sangat efektif digunakan ketika jumlah iterasi yang diperlukan untuk menjalankan serangkaian pernyataan telah diketahui atau dapat diprediksi pada awal eksekusi program. Struktur for (terdiri dari ungkapan inisialisasi, kondisi keluar, dan pengatur kenaikan variabel, dipisahkan oleh tanda titik koma `;`) berhasil diterapkan untuk melakukan penjumlahan akumulatif ($\text{jumlah} = \text{jumlah} + n$) secara berulang dan bertahap, yang dibuktikan dengan penghitungan deret bilangan triangular yang akurat hingga batas yang ditentukan pengguna.

B. Kegiatan 2

Kegiatan 2 menyimpulkan fungsi penting dari pernyataan break sebagai mekanisme kontrol alur yang dirancang khusus untuk keluar dari loop (for, while, atau do-while). Dalam konteks percobaan ini, break digunakan untuk menghentikan secara paksa infinite loop for (; ;) segera setelah kondisi internal (if (kar == '\n')) terpenuhi. Ketika break dijalankan, eksekusi program seketika melompat keluar dari tubuh loop for dan melanjutkan eksekusi ke pernyataan yang terletak sesudah akhir struktur loop, menunjukkan kemampuan break untuk menyediakan kondisi keluar eksternal.

C. Kegiatan 3

Kesimpulan kegiatan 3 bahwa pernyataan while sangat cocok digunakan ketika jumlah iterasi tidak diketahui secara pasti di awal, dan perulangan harus terus berjalan selama kondisi pengujian yang terletak di bagian awal loop masih bernilai

benar. Selain itu, kegiatan ini memverifikasi konsep *nested loop* (*loop* di dalam *loop*), yang diimplementasikan secara konseptual dalam fungsi pengujian `isPrime` untuk menguji keprimaan bilangan secara efisien (menguji pembagian hingga \sqrt{n}). Percobaan ini menunjukkan bagaimana *while* mengendalikan proses pencarian berkelanjutan hingga target (*n* bilangan prima) tercapai.

D. Kesimpulan Umum

Kesimpulan umum dari Modul 3 adalah bahwa penguasaan struktur kontrol perulangan sangat penting untuk memampukan program menjalankan serangkaian perintah secara berulang dan efisien. Pernyataan `for` digunakan idealnya untuk iterasi terhitung, `while` dan `do-while` digunakan ketika kondisi adalah faktor penentu utama (dengan `do-while` menjamin tubuh loop dijalankan minimal sekali. Selain itu, pernyataan kontrol alur seperti `break` (untuk keluar dari loop) dan `continue` (untuk mengarahkan eksekusi ke iterasi berikutnya) adalah alat esensial untuk memodifikasi perilaku loop secara dinamis, sementara konsep *nested loop* memungkinkan penyelesaian masalah yang memerlukan perulangan berlapis.

MODUL IV

ARRAY

4.1 Tujuan Praktikum

- A. Menjelaskan konsep dasar array dalam bahasa C.
- B. Menjelaskan perbedaan array satu dimensi dan array multidimensi.
- C. Menerapkan penggunaan array dalam pemrograman sederhana.
- D. Membuat program untuk mengolah data menggunakan array.

4.2 Landasan Teori

4.2.1 Pengertian Array

Array adalah sekumpulan elemen dengan tipe data yang sama (misalnya int, float, atau char) yang disimpan secara kontigu dalam memori. Setiap elemen array dapat diakses menggunakan indeks, yaitu nomor urut yang menunjukkan posisi elemen dalam array.

- A. Indeks array di C dimulai dari 0, bukan dari 1.
- B. Jika sebuah array memiliki ukuran n , maka elemen pertama terletak pada indeks 0, sedangkan elemen terakhir berada pada indeks $n-1$.
- C. Array sangat berguna untuk menyimpan data dalam jumlah banyak, misalnya daftar nilai mahasiswa, data sensor, atau elemen matriks.

Ilustrasi array satu dimensi dengan 5 elemen:

Indeks	0		1		2		3		4	
-----	---		---		---		---		---	
Nilai	70		80		65		90		75	

Tabel 4.1 *Elemen Array*

Pada contoh di atas, $\text{nilai}[0] = 70$, $\text{nilai}[4] = 75$.

4.2.2 Array Satu Dimensi

Array satu dimensi dapat dianggap sebagai deretan data linear. Bentuk deklarasi umum array satu dimensi adalah:

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

Gambar 4.1 *Bentuk Deklarasi Array*

- A. tipe_data → tipe data elemen array (misalnya int, float, char).
- B. nama_array → nama variabel array.
- C. ukuran → jumlah elemen yang bisa ditampung.

Contoh deklarasi dan inisialisasi:

```
int nilai[5] = {70, 80, 65, 90, 75};
```

Gambar 4.2 *Contoh Deklarasi Dan Inisialisasi Array*

Jika tidak semua elemen diinisialisasi, maka sisanya otomatis bernilai 0.

Operasi umum pada array satu dimensi:

1. Mengisi array dengan input dari user.
2. Menelusuri array dengan perulangan untuk menampilkan seluruh elemen.
3. Menghitung nilai agregat, seperti jumlah total atau rata-rata.
4. Mencari elemen tertentu dalam array.

4.2.3 Array Multidimensi

LED atau singkatan dari *Light Emitting Diode* adalah salah satu komponen elektronika yang terbuat dari bahan semi konduktor jenis dioda yang mampu mengeluarkan cahaya.

Array multidimensi adalah array yang memiliki lebih dari satu indeks. Bentuk yang paling umum digunakan adalah array dua dimensi, yang dapat dianggap sebagai tabel dengan baris dan kolom.

Bentuk deklarasi:

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

Gambar 4.3 *Deklarasi Array Multimedia*

Contoh deklarasi dan inisialisasi:

```
int matriks[3][3] = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9}  
};
```

Gambar 4.4 *Deklarasi & Inisialisasi Array Multimedia*

Indeks (baris, kolom)	Nilai
matriks[0][0]	1
matriks[0][1]	2
matriks[0][2]	3
matriks[1][0]	4
matriks[1][1]	5
matriks[1][2]	6
matriks[2][0]	7
matriks[2][1]	8
matriks[2][2]	9

Tabel 4.2 *Ilustrasi Array 2D*

Akses data dalam array 2D:

- A. Matriks[0][1] = 2 → elemen baris ke-1 kolom ke-2.
- B. Matriks[2][2] = 9 → elemen baris ke-3 kolom ke-3.

Array multidimensi banyak digunakan untuk menyimpan data tabel, grafik, atau citra digital.

4.2.4 Operasi pada Array

Array mendukung berbagai operasi dasar dalam pemrograman, di antaranya:

- A. Traversing (menelusuri array)

Proses mengakses semua elemen array satu per satu menggunakan perulangan.

Contoh: mencetak semua nilai mahasiswa.

- B. Searching (pencarian)

- 1. Linear Search → memeriksa satu per satu hingga data ditemukan.
- 2. Binary Search → mencari lebih cepat pada array yang sudah diurutkan.

- C. Sorting (pengurutan)

Mengatur elemen array dalam urutan tertentu (ascending/descending).

Beberapa metode sorting sederhana:

- 1. Bubble Sort → membandingkan elemen berurutan dan menukar jika salah urut.

2. Selection Sort → mencari nilai terkecil/terbesar, lalu menukar dengan elemen di depan.

3. Insertion Sort → menyisipkan elemen pada posisi yang sesuai dalam array yang sudah terurut.

D. Agregasi Data

1. Menghitung jumlah (sum) semua elemen array.

2. Menghitung rata-rata (average).

Menentukan nilai maksimum (max) dan minimum (min).

4.3 Alat-alat dan Komponen

A. Komputer / Laptop

B. *Software* DevC++

C. Bolpoin

D. Kertas

4.4 Prosedur Percobaan

4.4.1 Kegiatan 1 (Deklarasi dan Penelusuran Array)

Buatlah program berikut untuk menampilkan isi array 2D berbentuk matriks 5x2.

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

int main() {
    int matrix[5][2];
    cout << "Masukkan elemen matriks 5x2:" << endl;
    for(int i = 0; i < 5; i++) {
        for(int j = 0; j < 2; j++) {
            cout << "M[" << i << "][" << j << "] = ";
            cin >> matrix[i][j];
        }
    }
    cout << "\nMatriks 5x2:" << endl;
    for(int i = 0; i < 5; i++) {
        for(int j = 0; j < 2; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Gambar 4.5 Program Kegiatan 1

4.4.2 Kegiatan 2 (Mencari Nilai Terbesar dan Terkecil)

Perbaiki program berikut agar dapat menghitung rata-rata nilai mahasiswa yang disimpan dalam array. Dimana program akan meminta user untuk menginput

jumlah mahasiswa, dan akan meminta user untuk menginput nilai dari masing-masing mahasiswa se banyak jumlah mahasiswa yang sudah di inputkan.

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

int main() {
    int n;
    cout << "Masukkan jumlah mahasiswa: ";
    cin >> n;
    int nilai[n];
    float total = 0;
    for(int i = 0; i < n; i++) {
        cout << "Nilai mahasiswa ke-" << i+1 << ": ";
        cin >> nilai[i];
        total += nilai[i];
    }
    float rata = total / n;
    cout << "\nRata-rata nilai = " << fixed << setprecision(2) << rata << endl;
    return 0;
}
```

Gambar 4.6 Program Kegiatan 2

4.4.3 Kegiatan 3 (Array Dua Dimensi) Data Hasil Percobaan

Buat program untuk mengurutkan data integer dalam array menggunakan metode bubble sort.

```
Masukkan jumlah data: 5
Data ke-1: 12
Data ke-2: 34
Data ke-3: 56
Data ke-4: 91
Data ke-5: 1

Data setelah diurutkan: 1 12 34 56 91

-----
Process exited after 9.869 seconds with return value 0
Press any key to continue . . .
```

Gambar 4.7 Program Kegiatan 3

4.5 Data Hasil Percobaan

4.5.1 Kegiatan 1

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

int main() {
    int matrix[5][2];

    cout << "Masukkan elemen matriks 5x2:" << endl;

    for (int i = 0; i < 5; i++) { for (int j = 0; j < 2; j++) {
        cout << "M[" << i << "][" << j << "] = ";
        cin >> matrix[i][j];
    }
```

```

}
}

cout << "\nMatriks 5x2:" << endl;

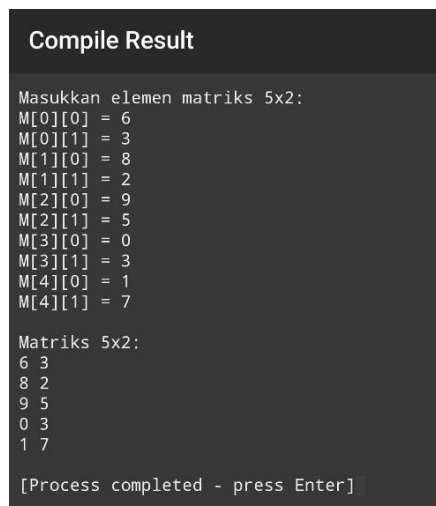
for (int i = 0; i < 5; i++) { for (int j = 0; j < 2; j++) {
cout << matrix[i][j] << " ";
}
cout << endl;
}

return 0;
}

```

Tabel 4.3 Listing Program Kegiatan 1

Kegiatan 1 berfungsi sebagai demonstrasi fundamental dari manipulasi matriks dua dimensi berukuran 5x2 yang bertipe data integer, di mana ia mengimplementasikan dua rangkaian loop for bersarang; rangkaian pertama secara sistematis mengiterasi melalui 10 elemen matriks untuk menerima input data dari pengguna (menggunakan cin), mengisi matriks dari indeks [0][0] hingga [4][1], sementara rangkaian loop bersarang kedua bertanggung jawab untuk melakukan output data yang telah tersimpan (menggunakan cout), dengan penambahan perintah cout << endl; pada loop luar untuk memastikan data tersebut dicetak ke konsol dalam format visual matriks 5 baris dan 2 kolom.



```

Compile Result

Masukkan elemen matriks 5x2:
M[0][0] = 6
M[0][1] = 3
M[1][0] = 8
M[1][1] = 2
M[2][0] = 9
M[2][1] = 5
M[3][0] = 0
M[3][1] = 3
M[4][0] = 1
M[4][1] = 7

Matriks 5x2:
6 3
8 2
9 5
0 3
1 7

[Process completed - press Enter]

```

Gambar 4.8 Hasil Output Kegiatan 1

Output program ini menunjukkan proses dasar pengelolaan *matriks* (5 baris, 2 kolom) di mana pengguna diminta untuk menginput 10 nilai integer secara berurutan, dimulai dari elemen hingga . Setelah semua nilai diterima, program

kemudian berfungsi mencetak ulang data tersebut dalam format *matriks* yang memvalidasi bahwa data input telah berhasil disimpan dan ditampilkan sesuai dengan struktur tabel dua dimensi tersebut.

4.5.2 Kegiatan 2

```
#include <iostream>
#include <iomanip>
using namespace std;
int main() {
    int n;
    cout << "Masukkan jumlah mahasiswa: "; cin >> n;

    int nilai[n]; float total = 0;

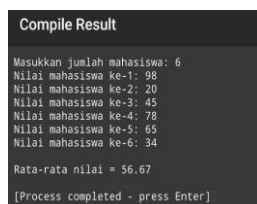
    for (int i = 0; i < n; i++) {
        cout << "Nilai mahasiswa ke-" << i + 1 << ": "; cin >> nilai[i];
        total += nilai[i];
    }

    float rata = total / n;
    cout << "\nRata-rata nilai = "
    << fixed << setprecision(2)
    << rata << endl;

    return 0;
}
```

Tabel 4.4 Listing Program Kegiatan 2

Tujuan listing program kegiatan dua untuk menghitung nilai rata-rata dari sekumpulan nilai mahasiswa. Program diawali dengan meminta pengguna memasukkan jumlah mahasiswa untuk menentukan ukuran *array*, kemudian menggunakan sebuah *loop (for)* untuk meminta input nilai satu per satu, sambil secara simultan mengakumulasi total semua nilai tersebut dalam variabel total.



```
Compile Result
Masukkan jumlah mahasiswa: 6
Nilai mahasiswa ke-1: 98
Nilai mahasiswa ke-2: 20
Nilai mahasiswa ke-3: 45
Nilai mahasiswa ke-4: 78
Nilai mahasiswa ke-5: 65
Nilai mahasiswa ke-6: 34
Rata-rata nilai = 56.67
[Process completed - press Enter]
```

Gambar 4.9 Hasil Output Kegiatan 2

Output ini adalah hasil eksekusi program penghitung rata-rata nilai, di mana prosesnya diawali dengan input dari pengguna yang menetapkan jumlah data yang

akan dimasukkan, yaitu 6 mahasiswa. Program kemudian meminta dan menerima enam nilai spesifik (98, 20, 45, 78, 65, dan 34), yang dijumlahkan seluruhnya menjadi 340. Akhirnya, program melakukan perhitungan rata-rata ($340 / 6$) dan menampilkan hasilnya yang telah diformat, yaitu Rata-rata nilai = 56.67, mengonfirmasi keberhasilan program dalam memproses input dan menyajikan hasil statistik akhir.

4.5.3 Kegiatan 3

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

int main() { int n;
cout << "Masukkan jumlah data: ";
cin >> n;
int data[n];
for (int i = 0;
i < n; i++) {
cout << "Data ke-" << i+1 << ": ";
cin >> data[i];
}
for (int i = 0; i < n - 1; i++) {
for (int j = 0; j < n - i - 1; j++) { if (data[j] > data[j + 1]){
int temp = data[j]; data[j] = data[j + 1]; data[j + 1] = temp;
}
}
}
cout << "\nData setelah diurutkan: "; for (int i = 0; i < n; i++) {
cout << data[i] << " ";
}
cout << endl;

return 0;
}
```

Tabel 4.5 Listing Program Kegiatan 3

Program kegiatan 3 bertujuan untuk mengurutkan (*sorting*) sekumpulan data angka integer yang dimasukkan oleh pengguna dari nilai terkecil ke terbesar (*ascending*) menggunakan algoritma *Bubble Sort*. Pertama, program meminta input jumlah data (N) dan kemudian menerima nilai-nilai data tersebut satu per satu dan menyimpannya ke dalam sebuah array. Inti dari program ini adalah dua *loop for*

bersarang yang menerapkan *Bubble Sort* dengan terus-menerus membandingkan elemen yang berdekatan dan menukarnya jika urutannya salah, sehingga elemen terbesar secara bertahap "menggelembung" ke posisi akhir.

```
Compile Result

Masukkan jumlah data: 6
Data ke-1: 26
Data ke-2: 43
Data ke-3: 18
Data ke-4: 59
Data ke-5: 50
Data ke-6: 78

Data setelah diurutkan: 18 26 43 50 59 78

[Process completed - press Enter]
```

Gambar 4.10 Hasil Output Kegiatan 3

Output program tersebut menunjukkan hasil dari tiga tahapan yang saling berurutan: pertama, pengguna menentukan bahwa ada 6 data yang akan diproses, lalu program menerima 6 nilai integer yang tidak terurut (26, 43, 18, 59, 50, 78) sebagai data mentah, dan pada akhirnya, setelah menjalankan algoritma *Bubble Sort* secara internal, program mencetak kembali data tersebut ke layar dalam kondisi yang sudah terurut dari terkecil ke terbesar, yaitu 18 26 43 50 59 78.

4.6 Analisa Hasil Percobaan

Ketiga program C++ yang kamu tampilkan sebenarnya merepresentasikan tiga tingkatan kemampuan pemrograman dasar: manipulasi array dua dimensi, perhitungan statistik sederhana, dan penerapan algoritma pengurutan klasik, namun jika dianalisis secara kritis berdasarkan logika, asumsi, kelengkapan desain, dan implikasinya dalam dunia nyata, ketiganya menunjukkan pola pemikiran yang masih berada pada fase “fungsi berjalan”, bukan pada fase “program dirancang dengan ketahanan dan ketepatan struktural”, sehingga menarik untuk dibedah lebih dalam. Program pertama tentang matriks 5×2 menggunakan array berukuran tetap dan keputusan ini menyimpan sejumlah asumsi yang sering tidak disadari pemula yaitu bahwa ukuran data selalu sesuai dengan apa yang telah ditentukan oleh programmer, pengguna tidak mungkin memberikan input berbahaya atau salah

format, dan tidak ada kondisi ketika jumlah elemen berubah; ini adalah asumsi yang jelas rapuh jika dibawa ke konteks aplikasi nyata, karena ukuran matriks dalam banyak kasus seharusnya bersifat dinamis dan bergantung pada kebutuhan pengguna, bukan ditentukan secara statis di dalam kode sumber. Selain itu, program tersebut tidak memiliki validasi input, sehingga ketika pengguna memasukkan karakter bukan angka, program akan mengalami error atau bahkan crash, dan ini menunjukkan bahwa logika program hanya benar selama asumsi bahwa input selalu tepat terpenuhi; seorang programmer profesional akan menyadari bahwa pemrosesan data yang benar bukan hanya tentang membaca dan mencetak nilai, tetapi memastikan bahwa data yang masuk memiliki integritas, reliabilitas, serta tidak menimbulkan error yang tidak diantisipasi. Meski begitu, dari sisi struktur memori, program ini memberi bekal penting tentang bagaimana array dua dimensi diakses menggunakan indeks baris dan kolom, dan bagaimana nested loop bekerja dalam operasi sistematis terhadap struktur data namun ketika ditinjau dari sudut pandang efisiensi dan desain berorientasi skenario nyata, program ini menyiratkan keterbatasan besar: tidak dapat beradaptasi terhadap perubahan ukuran, tidak dapat memproses data lebih lanjut (seperti penjumlahan baris atau kolom), dan tidak memiliki modularitas fungsi, sehingga kurang ideal sebagai fondasi aplikasi berbasis matriks yang serius.

Program kedua, yang menghitung rata-rata nilai mahasiswa, menunjukkan langkah maju dalam penggunaan array dan operasi matematis, namun justru memunculkan isu lebih besar pada aspek standar pemrograman: penggunaan `Variable Length Array int nilai[n]` memang terlihat praktis, tetapi sebenarnya bukan bagian dari standar C++ modern dan hanya didukung oleh sebagian compiler seperti GCC, sehingga program ini tidak portable dan berpotensi gagal di lingkungan lain, menunjukkan bahwa kode tersebut mengandung asumsi implisit bahwa compiler yang digunakan mendukung fitur non-standar; dalam konteks praktik profesional, ini adalah kesalahan desain karena ketergantungan pada fitur tidak standar melemahkan kompatibilitas program. Selain itu, program ini tidak menyediakan validasi nilai, sehingga mahasiswa bisa saja diberikan nilai di luar rentang masuk akal misalnya -50 atau 500 dan program tetap akan menghitung rata-ratanya seolah itu normal; logika matematisnya memang benar, tetapi makna datanya menjadi

keliru, dan ini memperlihatkan bahwa program belum mengakomodasi kontrol semantik yang penting dalam pengolahan data akademik. Program ini juga berpotensi kehilangan presisi dalam kasus dataset besar, karena variabel total bertipe float yang tidak sepenuhnya aman dari rounding *error* untuk penjumlahan berulang, seharusnya digunakan tipe double atau bahkan long *double* agar lebih robust; penggunaan `setprecision(2)` adalah langkah baik dalam estetika output, tetapi dari sudut pandang skeptis, itu hanya memperbaiki tampilan hasil, bukan kualitas proses perhitungan. Di sisi lain, program ini juga menunjukkan peluang pengembangan besar: bisa ditambah pencarian nilai tertinggi-terendah, distribusi kategori nilai, bahkan disusun ulang menjadi sistem penilaian berorientasi objek; tetapi selama masih dalam bentuknya sekarang, program ini masih terjebak pada pola “langsung ke main tanpa fungsi”, yang cenderung membuat kode kurang terstruktur ketika skalanya membesar.

Program ketiga, yaitu Bubble Sort, adalah yang paling menarik secara algoritmik karena memperlihatkan implementasi logika perbandingan dan pertukaran secara iteratif memakai nested loop, tetapi sekaligus merupakan contoh jelas bagaimana sebuah algoritma yang benar secara logika dapat menjadi pilihan yang buruk secara efisiensi Bubble Sort terkenal memiliki kompleksitas $O(n^2)$, sehingga meskipun program ini bekerja baik pada input kecil, kinerjanya akan ambruk pada skala besar, dan ini membuka diskusi penting tentang trade-off pemilihan algoritma dalam pemrograman. Tidak hanya itu, program ini sekali lagi menggunakan VLA, sehingga mengulang kelemahan non-standar yang sama seperti program kedua. Dari sisi logika algoritma, implementasi Bubble Sort di program ini sudah sesuai dengan prinsip dasar: membandingkan elemen bersebelahan dan menukar posisinya jika tidak sesuai urutan, sehingga elemen terbesar secara bertahap “mengapung” ke posisi akhir; tetapi program gagal mengoptimalkan loop dengan mendeteksi apakah pada iterasi tertentu tidak terjadi pertukaran (yang seharusnya menghentikan loop lebih awal), dan juga tidak memberi fleksibilitas pada pengguna untuk memilih urutan ascending atau descending, sehingga secara desain masih sangat minimalis dan jauh dari program sorting yang matang. Ditinjau dari perspektif skeptis, seorang pengembang profesional pasti mempertanyakan mengapa tidak menggunakan std sort yang jauh

lebih efisien dan teruji Bubble Sort dipertahankan lebih karena nilai pedagogisnya, bukan karena keunggulan fungsionalnya.

4.7 Kesimpulan

A. Kegiatan 1

Program mendemonstrasikan input dan output data ke dalam matriks (array 5 times 2). Hal ini menunjukkan penguasaan dalam mengakses data menggunakan indeks baris dan kolom melalui nested loop.

B. Kegiatan 2

Program berhasil menghitung rata-rata sekumpulan nilai mahasiswa. Program menggunakan Array Satu Dimensi dengan ukuran yang ditentukan pengguna (N) dan menggunakan iomanip untuk memformat hasil rata-rata dengan dua angka di belakang koma.

C. Kegiatan 3

Program mengimplementasikan algoritma Bubble Sort untuk mengurutkan data integer dari terkecil ke terbesar (ascending). Algoritma ini bekerja dengan membandingkan dan menukar elemen-elemen yang berdekatan menggunakan nested loop.

D. Kesimpulan Umum

Secara keseluruhan, modul praktikum ini berhasil mendemonstrasikan penerapan konsep dasar Array (larik) di C++, mencakup penggunaan array satu dimensi dan multidimensi. Kegiatan 1 mengimplementasikan array dua dimensi (times 2 atau matriks) untuk menerima input dan menampilkannya kembali dalam format tabel yang terstruktur. Kegiatan 2 menunjukkan pengolahan data dengan array satu dimensi untuk menghitung rata-rata nilai mahasiswa. Akhirnya, Kegiatan 3 menerapkan algoritma Bubble Sort menggunakan array satu dimensi untuk mengurutkan data integer secara ascending, yang menunjukkan pemahaman tentang logika perbandingan dan pertukaran elemen secara iteratif

DAFTAR PUSTAKA

- C / Identifier* — *DevDocs*. (n.d.). Retrieved March 1, 2025, from <https://devdocs.io/c/language/identifier>
- C Arrays* - *GeeksforGeeks*. (n.d.). Retrieved March 1, 2025, from <https://www.geeksforgeeks.org/c/c-arrays/>
- C for Loop* - *GeeksforGeeks*. (n.d.). Retrieved March 1, 2025, from <https://www.geeksforgeeks.org/c/c-for-loop/>
- C Functions*. (n.d.). Retrieved March 1, 2025, from https://www.w3schools.com/c/c_functions.php
- C Pointers* - *GeeksforGeeks*. (n.d.). Retrieved March 1, 2025, from <https://www.geeksforgeeks.org/c/c-pointers/>
- Difference Between Constants and Variables in C* - *GeeksforGeeks*. (n.d.). Retrieved March 1, 2025, from <https://www.geeksforgeeks.org/c/constants-vs-variables-in-c-language/>
- Difference Between if-else and switch in C* - *GeeksforGeeks*. (n.d.). Retrieved March 1, 2025, from <https://www.geeksforgeeks.org/cpp/difference-between-if-else-and-switch-in-c/>
- Operators in C* - *GeeksforGeeks*. (n.d.). Retrieved March 1, 2025, from <https://www.geeksforgeeks.org/c/operators-in-c/>