

VERSI 1.0
AGUSTUS, 2024



[PEMROGRAMAN DASAR]

MODUL 3 - MEMORY ADDRESS, CONDITIONS, ENUMS

DISUSUN OLEH:

WEMPY ADITYA WIRYAWAN

MUHAMMAD ZAKY DARAJAT

DIAUDIT OLEH:

HARDIANTO WIBOWO, S.KOM, M.T

LAB. INFORMATIKA

UNIVERSITAS MUHAMMADIYAH MALANG

[PEMROGRAMAN DASAR]

PERSIAPAN MATERI

Mahasiswa diharap sudah mempelajari materi berikut sebelum mengerjakan tugas :

- Memory address
- If - Else
- Switch
- Enums

TUJUAN

- Mahasiswa memahami konsep dan pentingnya memory address dalam pemrograman, serta mampu menggunakan pointer untuk mengakses dan memanipulasi data di alamat memori.
- Mahasiswa menguasai struktur kontrol if-else untuk mengambil keputusan dalam program berdasarkan kondisi tertentu, sehingga program dapat berjalan dengan logika yang tepat.
- Mahasiswa memahami dan menguasai konstruksi switch untuk menyederhanakan pemilihan berdasarkan nilai ekspresi tertentu, sehingga kode program lebih efisien dan mudah dibaca.
- Mahasiswa memahami penggunaan enums untuk membuat simbolik nama untuk konstan terkait, sehingga program lebih mudah dipahami, diubah, dan dipelihara.

TARGET MODUL

- Memahami konsep dasar memory address dan pointer, serta dapat mengidentifikasi alamat memori dari variabel dan objek dalam program.
- Mampu menggunakan if-else untuk mengendalikan alur eksekusi program berdasarkan kondisi yang terpenuhi atau tidak terpenuhi.
- Mampu menggunakan switch untuk mengimplementasikan pemilihan dengan beberapa pilihan kasus berdasarkan nilai ekspresi.
- Mampu mendeklarasikan dan menggunakan enums untuk menggantikan penggunaan angka atau konstanta dalam program dengan simbolik nama yang lebih bermakna.

PERSIAPAN SOFTWARE/APLIKASI

- Komputer/Laptop
- Software IDE (Falcon/Dev C++)

MATERI PRAKTIKUM

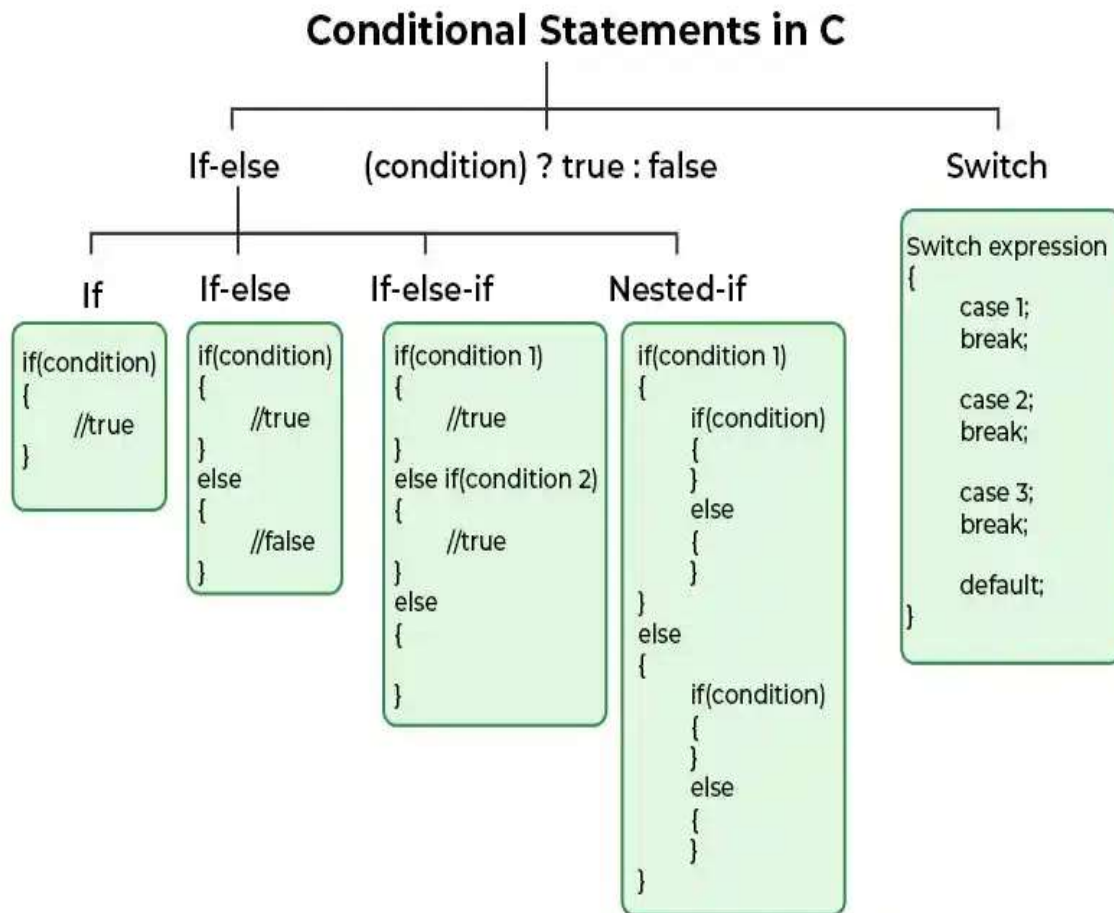
CONDITIONS

Conditions atau pengkondisian merujuk pada ekspresi atau pernyataan yang dievaluasi sebagai benar (true) atau salah (false). Conditions digunakan untuk mengontrol alur program, di mana bagian tertentu dari kode akan dieksekusi hanya jika kondisi tertentu. Ibarat berada di perempatan jalan dengan tujuan ke Surabaya, kita harus memilih jalan yang sesuai agar bisa mencapai tujuan tersebut. Misalnya, jika jalan utara adalah jalan yang tepat untuk mencapai Surabaya, maka kita akan melewati jalan tersebut. Sama halnya dengan conditions dalam pemrograman, jika salah satu kondisi memenuhi syarat (true), maka kondisi tersebut akan dieksekusi.

Untuk lebih memahami apa itu conditions, konsepnya bisa dibandingkan dengan berbagai skenario dalam kehidupan sehari-hari:

1. Pengkondisian dalam keputusan : misal kalian ingin mengecek apakah kalian boleh masuk ke bioskop atau tidak. Jika kalian berumur di atas 18 tahun, kalian boleh masuk. Kalau tidak berusia di atas 18 tahun, maka kalian harus menunggu.
2. Pengkondisian dalam Pengulangan: kalian mungkin ingin menghitung berapa kali Anda harus berlari mengelilingi lapangan agar mencapai target tertentu. Jika kalian belum mencapai target tersebut, kalian akan terus berlari.
3. Pengkondisian dalam Permainan: Dalam sebuah permainan, jika skor kalian lebih tinggi dari lawan, mungkin mendapatkan poin ekstra. Namun, jika skornya lebih rendah, Kalian tidak mendapatkan poin tambahan.
4. Pengkondisian dalam Aplikasi Login: Pada aplikasi atau situs web, jika Kalian memasukkan kata sandi yang benar, Kalian akan diarahkan ke halaman utama. Tetapi jika kata sandi salah, mungkin harus mencoba lagi.

Tipe conditional statement di C:



Dalam bahasa C, terdapat beberapa operator dan pernyataan yang dapat digunakan untuk membuat conditions, diantaranya :

➤ IF

Pernyataan `if` adalah salah satu fungsi dalam bahasa pemrograman untuk mengontrol alur program berdasarkan evaluasi logika tertentu. Ketika kita ingin melakukan pengecekan kondisi atau logika tertentu dalam program, `if` dapat digunakan untuk mengarahkan jalannya eksekusi program. Jika kondisi yang diberikan dalam pernyataan `if` dievaluasi sebagai benar (`true`), maka blok kode di dalamnya akan dieksekusi. Namun, jika kondisi yang diberikan bernilai salah (`false`), maka blok kode di dalam pernyataan `if` tidak akan dieksekusi.

Melalui penggunaan pernyataan `if`, kita dapat mengontrol bagaimana program kita berperilaku berdasarkan situasi yang berbeda. Misalnya, dalam suatu program yang meminta input dari pengguna, kita dapat menggunakan pernyataan `if` untuk memeriksa apakah input yang diberikan sesuai dengan kriteria tertentu. Jika input memenuhi kriteria, program dapat melanjutkan dengan langkah-langkah tertentu; jika tidak, program dapat memberikan tanggapan atau tindakan alternatif.

```
int num = 10;
if (num > 0) {
    printf("Angka adalah bilangan positif.\n");
}
```

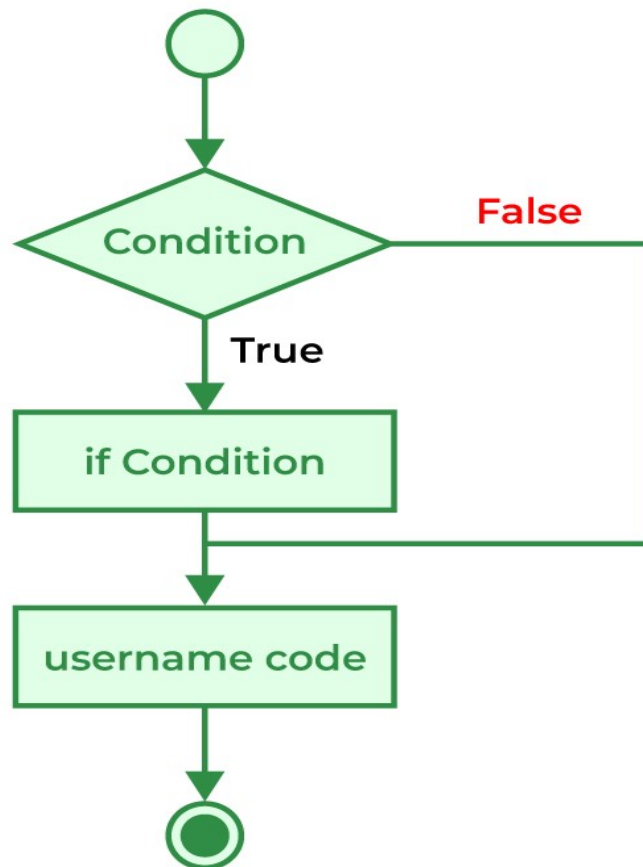
Dari kode di atas, kita mendeklarasikan variabel `num` dengan tipe data `int` dan memberi inisialisasi nilai 10. Kemudian pada bagian condition `if`, kita menuliskan suatu kondisi yaitu `num > 0`. Hal ini memiliki arti bahwa **jika num lebih dari 0** maka hasilnya adalah `true` dan akan mencetak output “angka adalah bilangan positif”.

Penting untuk diingat bahwa pernyataan `if` hanya akan mengevaluasi ekspresi atau kondisi yang diberikan di dalamnya. Jika ekspresi tersebut benar, maka eksekusi program akan masuk ke dalam blok kode `if`. Namun, jika ekspresi tersebut salah, eksekusi program akan melanjutkan ke bagian selanjutnya tanpa menjalankan blok kode di dalam `if`.

Syntax If:

```
if(condition)
{
    // Statements to execute if
    // condition is true
}
```

Di sini, kondisi setelah evaluasi akan bernilai benar atau salah. Pernyataan if menerima nilai boolean - jika nilainya benar maka akan mengeksekusi blok pernyataan di bawahnya, jika tidak maka tidak. Jika kita tidak memberikan tanda kurung kurawal '{' dan '}' setelah if(kondisi) maka secara default if statement akan menganggap pernyataan pertama di bawahnya sebagai bagian dari bloknya.

Flowchart If Statement:

Contoh Implementasi If:

```
// C Program to demonstrate the syntax of if statement
#include <stdio.h>

int main()
{
    int gfg = 9;
    // if statement dengan kondisi true
    if (gfg < 10) {
        printf("%d lebih kecil dari 10", gfg);
    }

    // if statement dengan kondisi false
    if (gfg > 20) {
        printf("%d lebih besar dari 20", gfg);
    }

    return 0;
}
```

Output:

```
9 lebih kecil dari 10
```

Cara kerja pernyataan if dalam bahasa C adalah sebagai berikut:

Langkah 1: Ketika kontrol program sampai pada pernyataan if, ekspresi pengujian dievaluasi.

Langkah 2A: Jika kondisinya benar, pernyataan di dalam blok if dieksekusi.

Langkah 2B: Jika ekspresi salah, pernyataan di dalam badan if tidak dieksekusi.

Langkah 3: Kontrol program berpindah dari blok if dan kode setelah blok if dieksekusi.

Kelebihan dari Pernyataan if

Berikut ini adalah kelebihan utama dari pernyataan if dalam bahasa C:

- Merupakan pernyataan pengambilan keputusan yang paling sederhana.
- Mudah digunakan dan dipahami.
- Dapat mengevaluasi ekspresi dari semua jenis seperti int, char, bool, dll.

Kerugian dari Pernyataan if

- Batasan utama dari blok if tercantum di bawah ini: Hanya berisi satu blok. Jika ada beberapa blok if yang terkait, semua blok akan diuji bahkan ketika blok if yang cocok ditemukan di awal
- Ketika ada sejumlah besar ekspresi, kode blok if menjadi kompleks dan tidak dapat dibaca.
- Lebih lambat untuk sejumlah besar kondisi.

Untuk latihan, coba kalian lengkapi kodingan di bawah ini :

```
#include <stdio.h>

int main() {
    int genap = 2;
    if () {
        printf("Bilangan Genap");
    }
}
```

➤ IF – ELSE

Pernyataan if-else adalah salah satu fungsi dalam bahasa C untuk mengontrol alur program dengan lebih kompleks. Dengan **if-else**, kita dapat mengevaluasi lebih dari satu kondisi dan mengambil tindakan yang sesuai berdasarkan hasil evaluasi tersebut.

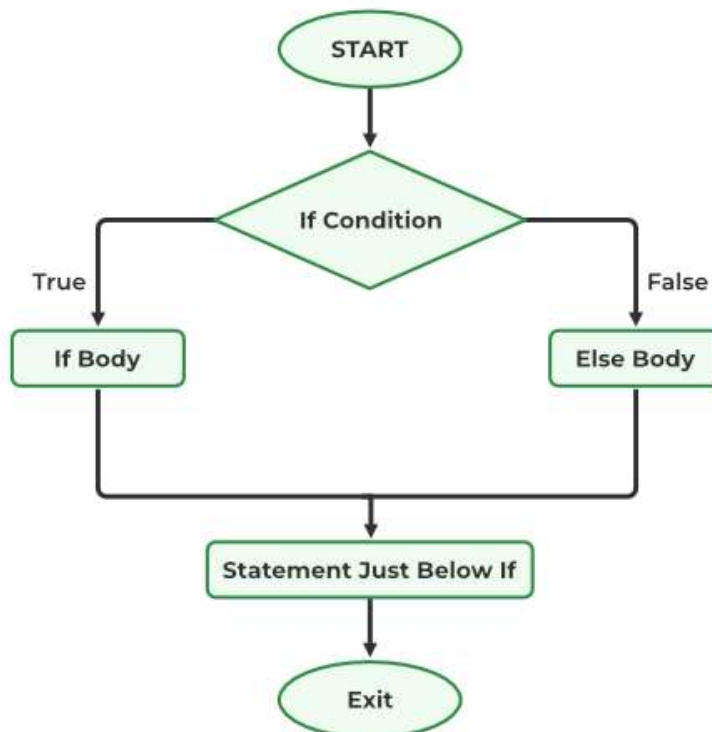
Ketika kita menggunakan pernyataan if-else, program akan mengevaluasi kondisi pertama yang diberikan dalam blok if. Jika kondisi tersebut bernilai benar (true), maka blok kode di dalam if akan dieksekusi. Namun, jika kondisi tersebut bernilai salah (false), maka program akan melanjutkan ke blok else dan mengeksekusi blok kode di dalamnya.

Penggunaan if-else memungkinkan kita untuk membuat pilihan antara dua jalur eksekusi yang berbeda berdasarkan hasil evaluasi kondisi. Ini sangat bermanfaat saat kita ingin menangani situasi di mana kondisi pertama tidak terpenuhi.

Contohnya, dalam sebuah program pendaftaran online, kita dapat menggunakan pernyataan if-else untuk memeriksa apakah seorang pengguna memasukkan usia yang memenuhi syarat (misalnya, usia di atas 18 tahun) atau tidak. Jika usianya memenuhi syarat, program akan memberikan akses penuh; jika tidak, program akan memberikan pesan bahwa pengguna tidak memenuhi kriteria usia..

Syntax If-Else di C:

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```

Flowchart If-else statement:

```

#include <stdio.h>

int main() {
    int num = -5;

    if (num>0) {
        printf("Angka adalah bilangan positif.");
    } else {
        printf("Angka adalah bilangan negatif atau nol.");
    }
}

```

Contoh, pada kode diatas, kita menginisialisasi variabel **num** dengan nilai -5. Pada condition **if** kita memiliki kondisi **jika num lebih dari 0** akan mencetak output “angka adalah bilangan positif” apabila kondisi tersebut adalah **true**. Namun, jika tidak memenuhi kondisi atau **false** maka program akan dilanjutkan ke kondisi selanjutnya yaitu **else**. Dalam kondisi **else**, kita tidak perlu menuliskan kondisi yang dibutuhkan, karena **else** hanya akan mengeksekusi apabila suatu pernyataan tidak memenuhi kondisi. Sehingga, kode program di atas akan langsung dilanjutkan ke kondisi **else** dan mencetak output “Angka adalah bilangan negatif atau nol”.

Contoh lain:

```

// C program to illustrate If statement
#include <stdio.h>

int main()
{
    int i = 20;

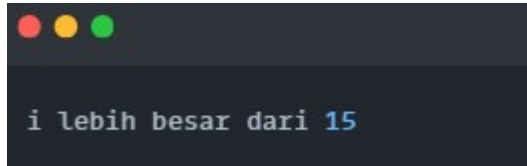
    if (i < 15) {

        printf("i lebih kecil dari 15");
    }
    else {

        printf("i lebih besar dari 15");
    }
    return 0;
}

```

Output:

A screenshot of a terminal window with a dark background. At the top, there are three colored window control buttons (red, yellow, green). Below them, the text "i lebih besar dari 15" is displayed in a light blue monospaced font.

Program C di atas menunjukkan bagaimana kita dapat menggunakan pernyataan `if-else` untuk membuat keputusan berdasarkan kondisi. Kita mulai dengan mendefinisikan variabel `i` dengan nilai 20. Kemudian, kita menggunakan pernyataan `if` untuk memeriksa apakah nilai `i` kurang dari 15. Jika kondisi ini benar, kita menampilkan "i lebih besar dari 15" menggunakan fungsi `printf`. Namun, jika kondisi tersebut salah (yang memang salah karena `i` adalah 20), kita akan masuk ke bagian `else` dan menampilkan "i lebih kecil dari 15". Ada kesalahan pada pesan yang ditampilkan; seharusnya adalah "i lebih kecil dari 15" dalam blok `if` dan "i lebih besar dari 15" dalam blok `else`. Program ini menunjukkan kepada kita bagaimana menggunakan logika kondisional sederhana dalam bahasa C, di mana kita dapat mengeksekusi blok kode berbeda tergantung pada hasil dari evaluasi kondisi.

Cara kerja if-else statement:

1. Ketika kontrol program pertama kali masuk ke blok if-else, kondisi pengujian diperiksa.
2. Jika kondisi pengujian benar:
 - a. Blok if dieksekusi.
3. Jika kondisi pengujian salah:
 - a. Blok else dieksekusi
4. Setelah itu, kontrol program berlanjut ke pernyataan-pernyataan di bawah pernyataan if-else.

Kelebihan dari Pernyataan if-else

- Pernyataan if-else memungkinkan pengguna untuk mengeksekusi pernyataan yang berbeda berdasarkan kondisi yang berbeda.
- Pernyataan ini dapat mengevaluasi ekspresi pengujian tipe int, char, boolean, dan banyak lagi.
- Ini membantu dalam memodifikasi aliran program.
- Sederhana, efisien, dan lebih mudah dibaca ketika jumlah kondisi lebih sedikit.

Kekurangan dari Pernyataan if-else

- Jika ada banyak pernyataan if, kode menjadi tidak dapat dibaca dan kompleks.
- Ini juga menjadi lebih lambat dibandingkan dengan pernyataan switch.

Untuk Latihan, coba kalian lengkapi/perbaiki kode program dibawah ini

```
#include <stdio.h>

int main() {
    int age =
    20 ; if () {
        printf("Umur Tidak Valid");
    } else {
        printf("Umur Valid");
    }
}
```

➤ IF – ELSE IF

Pernyataan if-else if adalah alat yang sangat berguna dalam pemrograman ketika kita perlu memeriksa beberapa kondisi secara berurutan. Dalam banyak kasus, program harus melakukan evaluasi dari satu kondisi ke kondisi lainnya, dan pernyataan ini memungkinkan kita untuk melakukan hal tersebut dengan efisien.

Ketika kita menggunakan pernyataan if-else if, program akan mengevaluasi kondisi pertama yang diberikan dalam blok if. Jika kondisi tersebut bernilai benar (true), maka blok kode di dalam if akan dieksekusi, dan program akan keluar dari blok if-else if. Namun, jika kondisi pertama bernilai salah (false), program akan melanjutkan ke kondisi berikutnya dalam blok else if. Pernyataan if-else if akan terus mengevaluasi kondisi-kondisi berikutnya secara berurutan hingga salah satu kondisi yang bernilai benar ditemukan atau hingga mencapai kondisi terakhir yang ditangani oleh blok else. Jika tidak ada satu pun kondisi yang benar, maka blok else akan dieksekusi.

```
#include <stdio.h>

int main() {
    int age;

    printf("Masukkan usia Anda: ");
    scanf("%d", &age);

    if (age < 0) {
        printf("Usia tidak boleh negatif.\n");
    } else if (age < 18) {
        printf("Anda masih di bawah umur.\n");
    } else if (age < 60) {
        printf("Anda adalah orang dewasa.\n");
    } else {
        printf("Anda sudah lansia.\n");
    }

    return 0;
}
```

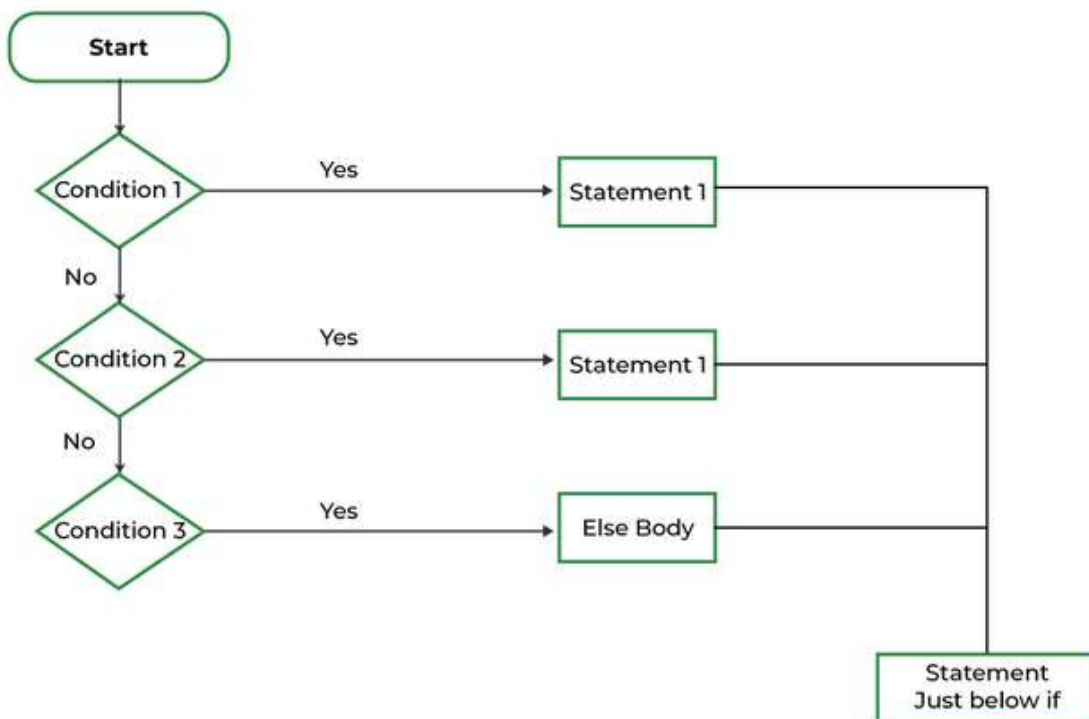
Contoh pada kode diatas memiliki 3 kondisi yaitu kondisi **jika age kurang dari 0** maka program akan mencetak pesan bahwa usia tidak boleh negatif, **jika age kurang dari 18** akan mencetak pesan bahwa pengguna masih dibawah umur, dan **jika age kurang dari 60** akan mencetak pesan bahwa pengguna adalah orang dewasa. Misal, user menginput nilai 17 maka program akan mengeksekusi pada pernyataan kondisi pertama. Namun bila user menginput nilai 72 maka program akan langsung berjalan ke kondisi else dan mencetak pesan bahwa “anda sudah lansia”.

Syntax If else if:

```
// any if-else ladder starts with an if statement only
if(condition) {

}
else if(condition) {
    // this else if will be executed when condition in if is false and
    // the condition of this else if is true
}
.... // once if-else ladder can have multiple else if
else { // at the end we put else

}
}
```

Flowchart If-else-if:**Alur kerja dari tangga if-else-if:**

1. Alur program masuk ke dalam blok if.
2. Alur melompat ke Kondisi Pertama.
3. Kondisi ke-1 diuji secara berurutan:

- a. Jika Kondisi berikut ini menghasilkan nilai true, lanjutkan ke Langkah 4.
 - b. Jika Kondisi berikut menghasilkan nilai false, lanjutkan ke Langkah 5.
4. Blok sekarang dieksekusi. Pergi ke Langkah 7.
5. Alur melompat ke Kondisi 2.
 - a. Jika Kondisi berikut ini menghasilkan nilai true, lanjutkan ke Langkah 4.
 - b. Jika Kondisi berikut menghasilkan false, lanjutkan ke Langkah 6.
6. Alur melompat ke Kondisi 3.
 - a. Jika Kondisi berikut ini menghasilkan nilai true, lanjutkan ke langkah 4.
 - b. Jika Kondisi berikut menghasilkan false, jalankan blok else. Pergi ke Langkah 7.
7. Keluar dari tangga if-else-if.

Contoh implementasi if-else-if:

```
#include <stdio.h>

int main() {
    int umur;

    // Meminta pengguna memasukkan umur
    printf("Masukkan umur Anda: ");
    scanf("%d", &umur);

    // Menentukan kategori usia menggunakan if-else if
    if (umur < 13) {
        printf("Anda adalah seorang anak-anak.\n");
    } else if (umur ≥ 13 && umur < 20) {
        printf("Anda adalah seorang remaja.\n");
    } else if (umur ≥ 20 && umur < 65) {
        printf("Anda adalah seorang dewasa.\n");
    } else {
        printf("Anda adalah seorang lansia.\n");
    }

    return 0;
}
```

Alur program ini dimulai dengan meminta kita memasukkan umur melalui input standar menggunakan `scanf`. Setelah kita memasukkan umur, program masuk ke tahap pengkategorian usia dengan menggunakan serangkaian pernyataan `if-else if`. Pertama, program memeriksa apakah umur yang kita masukkan kurang dari 13 tahun. Jika benar, program menampilkan pesan bahwa kita adalah seorang anak-anak. Jika tidak, program melanjutkan untuk memeriksa apakah umur kita berada dalam rentang 13 hingga 19 tahun, yang akan mengklasifikasikan kita sebagai remaja. Jika kedua kondisi tersebut tidak terpenuhi, program kemudian memeriksa apakah umur kita berada dalam rentang 20 hingga 64 tahun untuk menentukan apakah kita termasuk kategori dewasa. Jika semua kondisi sebelumnya tidak terpenuhi, maka kita dikategorikan sebagai lansia, yaitu mereka yang berumur 65 tahun atau lebih. Berdasarkan kondisi mana yang terpenuhi, program akan menampilkan pesan yang sesuai, dan kemudian program selesai dijalankan dengan mengembalikan nilai 0 sebagai tanda bahwa program berakhir dengan sukses.

Latihan, cobalah perbaiki code dibawah dengan benar:

```
#include <stdio.h>

int main() {
    int skor;
    char nilai_huruf;

    printf("Masukkan skor Anda (0-100): ");
    scanf("%d", &skor);

    if (skor ≥ 90)
        nilai_huruf = 'A';
    else if (skor ≥ 80)
        nilai_huruf = 'B';
    else if (skor ≥ 70)
        nilai_huruf = 'C';
    else if (skor ≥ 60)
        nilai_huruf = 'D';
    else
        nilai_huruf = 'F';

    printf("Nilai huruf Anda adalah: %c\n", nilai_huruf);

    return 0;
}
```


➤ Nested IF

Pernyataan nested if adalah konsep dalam pemrograman di mana kita menempatkan satu pernyataan if di dalam pernyataan if lainnya. Dengan demikian, kita dapat membuat struktur kondisional yang lebih kompleks dan berlapis.

Konsep ini sangat bermanfaat ketika kita perlu menguji beberapa kondisi secara berturut-turut dan beberapa kondisi hanya relevan jika kondisi yang lebih umum benar. Pada dasarnya, kita membuat pemilihan lebih rinci dalam konteks kondisi yang lebih umum.

```
int num = 10;

if (num > 0) {
    printf("Angka adalah bilangan positif.\n");
    if (num % 2 == 0) {
        printf("Angka adalah bilangan genap.\n");
    } else {
        printf("Angka adalah bilangan ganjil.\n");
    }
} else {
    printf("Angka adalah bilangan negatif atau nol.\n");
}
```

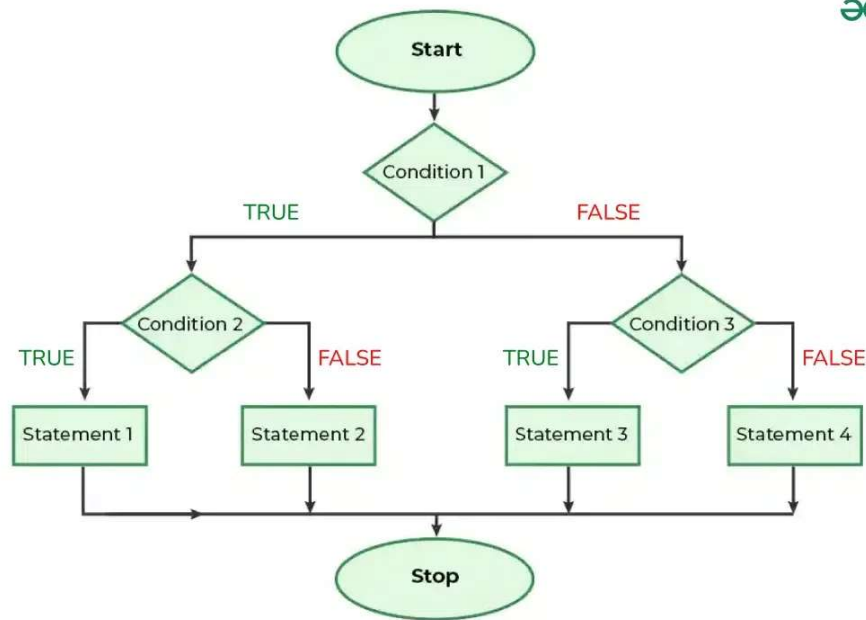
Program di atas adalah contoh dari sebuah struktur percabangan dalam bahasa C menggunakan if dan else. Program ini digunakan untuk mengecek nilai dari variabel num dan memberikan pesan sesuai dengan kriteria yang terpenuhi. Pertama, program akan memeriksa **apakah num lebih besar dari 0**. Jika iya, maka program akan mencetak pesan bahwa angka tersebut adalah bilangan positif. Selanjutnya, di dalam blok if pertama, program akan memeriksa **apakah num adalah bilangan genap atau ganjil** dengan menggunakan operasi modulo ($\text{num} \% 2$). Jika hasil modulo adalah 0, maka angka tersebut adalah bilangan genap, dan program akan mencetak pesan yang sesuai. **Jika hasil modulo bukan 0**, maka angka tersebut adalah bilangan ganjil, dan program juga akan mencetak pesan yang sesuai.

Namun, **jika nilai num tidak lebih besar dari 0**, program akan masuk ke blok else dan mencetak pesan bahwa angka tersebut adalah bilangan negatif atau nol. Dengan struktur percabangan ini, program dapat memberikan respon yang sesuai berdasarkan nilai num, sehingga kita dapat dengan mudah mengetahui sifat dari angka yang dimasukkan.

Syntax Nested If:

```
if (condition1)
{
    // Executes when condition1 is true
    if (condition_2)
    {
        // statement 1
    }
    else
    {
        // Statement 2
    }
}
else {
    if (condition_3)
    {
        // statement 3
    }
    else
    {
        // Statement 4
    }
}
```

Flowchart nested if:



Contoh implementasi nested if:

```

#include <stdio.h>

int main() {
    int umur;
    char jenis_kelamin;

    // Meminta pengguna untuk memasukkan jenis kelamin
    printf("Masukkan jenis kelamin Anda (L/P): ");
    scanf(" %c", &jenis_kelamin);

    // Meminta pengguna untuk memasukkan umur
    printf("Masukkan umur Anda: ");
    scanf("%d", &umur);

    // Menentukan kategori usia berdasarkan jenis kelamin dan umur
    if (jenis_kelamin == 'L' || jenis_kelamin == 'l') { // Jika laki-laki
        if (umur < 13) {
            printf("Anda adalah anak laki-laki.\n");
        } else if (umur >= 13 && umur < 20) {
            printf("Anda adalah remaja laki-laki.\n");
        } else if (umur >= 20 && umur < 65) {
            printf("Anda adalah pria dewasa.\n");
        } else {
            printf("Anda adalah pria lansia.\n");
        }
    } else if (jenis_kelamin == 'P' || jenis_kelamin == 'p') { // Jika perempuan
        if (umur < 13) {
            printf("Anda adalah anak perempuan.\n");
        } else if (umur >= 13 && umur < 20) {
            printf("Anda adalah remaja perempuan.\n");
        } else if (umur >= 20 && umur < 65) {
            printf("Anda adalah wanita dewasa.\n");
        } else {
            printf("Anda adalah wanita lansia.\n");
        }
    } else {
        printf("Jenis kelamin tidak valid.\n");
    }

    return 0;
}
  
```

Program di atas adalah contoh penggunaan nested if untuk mengkategorikan pengguna berdasarkan jenis kelamin dan umur. Pertama, program meminta pengguna untuk memasukkan jenis kelamin (L untuk laki-laki atau P untuk perempuan) dan umur. Kemudian, program memeriksa jenis kelamin pengguna. Jika pengguna memasukkan L (laki-laki), program masuk ke dalam blok if pertama, dan di dalamnya terdapat nested if yang mengevaluasi umur pengguna untuk menentukan apakah mereka adalah anak laki-laki, remaja laki-laki, pria dewasa, atau pria lansia. Jika jenis kelamin adalah P (perempuan), program masuk ke blok else if kedua dan melakukan evaluasi serupa untuk kategori perempuan. Jika input jenis kelamin tidak valid, program menampilkan pesan kesalahan. Program ini menunjukkan bagaimana kita dapat menggunakan nested if untuk melakukan evaluasi kondisi yang lebih kompleks, memungkinkan kita untuk membuat keputusan bertingkat berdasarkan beberapa kriteria.

Latihan, perbaiki kode ini dengan benar:

```
#include <stdio.h>

int main() {
    int umur;
    char jenis_kelamin;

    printf("Masukkan jenis kelamin Anda (L/P): ");
    scanf(" %c", &jenis_kelamin);

    printf("Masukkan umur Anda: ");
    scanf("%d", &umur);

    if (jenis_kelamin == 'L') {
        if (umur < 13)
            printf("Anak laki-laki\n");
        else if (umur ≤ 19)
            printf("Remaja laki-laki\n");
        else if (umur < 65)
            printf("Pria dewasa\n");
        else
            printf("Pria lansia\n");
    } else if (jenis_kelamin == 'P') {
        if (umur < 13)
            printf("Anak perempuan\n");
        else if (umur ≥ 13 || umur ≤ 19)
            printf("Remaja perempuan\n");
        else if (umur < 65)
            printf("Wanita dewasa\n");
        else
            printf("Wanita lansia\n");
    } else {
        printf("Jenis kelamin tidak valid\n");
    }

    return 0;
}
```

➤ Switch

Switch adalah pernyataan yang digunakan untuk memeriksa nilai dari ekspresi tertentu dan menjalankan blok kode yang sesuai dengan nilai ekspresi tersebut. Kondisi switch digunakan ketika kalian ingin membandingkan ekspresi dengan beberapa nilai yang berbeda dan melakukan tindakan yang berbeda berdasarkan nilai tersebut.

Alur evaluasi nilai dari kondisi switch adalah jika ekspresi sama dengan nilai dari setiap case, maka akan dieksekusi. Jika ekspresi tidak sama dengan nilai apapun dari case diatas, maka blok kode **default** akan dieksekusi. Perlu diingat bahwa setiap blok kode dalam case harus diakhiri dengan pernyataan **break**. Ini untuk memastikan bahwa setelah satu blok kode dieksekusi, program keluar dari switch dan melanjutkan eksekusi setelah switch. Kalian bisa menggabungkan beberapa case jika kalian ingin menjalankan blok kode yang sama untuk beberapa nilai.

Berikut contoh kode programnya :

```
#include <stdio.h>

int main() {
    int choice;

    printf("Pilih salah satu dari berikut:\n");
    printf("1. Profil Jaemin\n");
    printf("2. Nickname Jaemin\n");
    printf("3. Quote Jaemin\n");
    printf("\nPilihan Anda: ");
    scanf("%d", &choice);

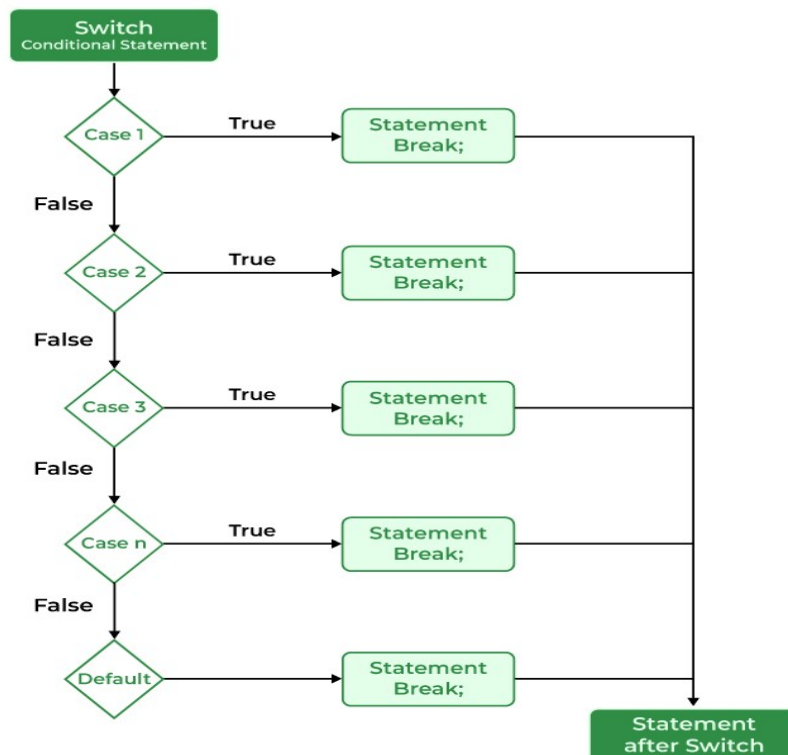
    switch (choice) {
        case 1:
            printf("Nama: Na Jae-min\n, Tanggal lahir: 13\nAgustus 2000");
            break;
        case 2:
            printf("Nana <3");
            break;
        case 3:
            printf("If what you want to do doesn't work,\ncontinue until you can.");
            break;
        default:
            printf("Pilihan tidak valid");
    }
    return 0;
}
```

Dari contoh program di atas, apabila user menginput angka 1, maka program akan mengeksekusi case 1. Begitu juga apabila user menginput angka 2 maka akan mengeksekusi case 2. Namun, jika user menginput angka selain 1/2/3 maka program akan dilanjutkan ke **default** dan menampilkan pesan bahwa “pilihan tidak valid”.

Syntax Switch:

```
switch(expression)
{
case value1: statement_1;
            break;
case value2: statement_2;
            break;
.
.
.
case value_n: statement_n;
            break;
default: default_statement;
}
```

Flowchart Switch:



Aturan dari pernyataan switch case

Berikut ini adalah beberapa aturan yang perlu kita ikuti saat menggunakan pernyataan switch:

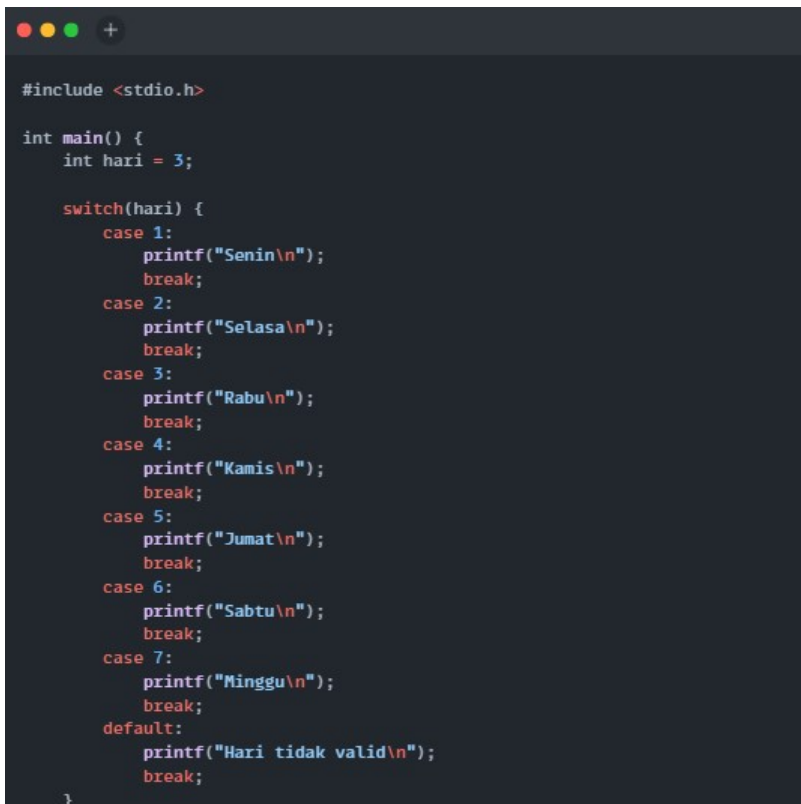
1. Dalam pernyataan switch, "case value" harus bertipe "char" dan "int".
2. Bisa ada satu atau N jumlah case.
3. Nilai-nilai dalam case harus unik.
4. Setiap pernyataan kasus dapat memiliki pernyataan break. Ini bersifat opsional.
5. Pernyataan default juga bersifat opsional.

Bagaimana Cara Kerja Pernyataan Switch?

Cara kerja pernyataan switch dalam C adalah sebagai berikut:

1. Langkah 1: Variabel switch dievaluasi.
2. Langkah 2: Nilai yang dievaluasi dicocokkan dengan semua kasus yang ada.
3. Langkah 3A: Jika nilai kasus yang cocok ditemukan, kode yang terkait dieksekusi.
4. Langkah 3B: Jika kode yang cocok tidak ditemukan, maka kasus default dieksekusi jika ada.
5. Langkah 4A: Jika kata kunci break ada dalam kasus, maka kontrol program akan keluar dari pernyataan switch.
6. Langkah 4B: Jika kata kunci break tidak ada, maka semua kasus setelah kasus yang cocok akan dieksekusi.
7. Langkah 5: Pernyataan setelah pernyataan switch dieksekusi.

Contoh implementasi Switch case:

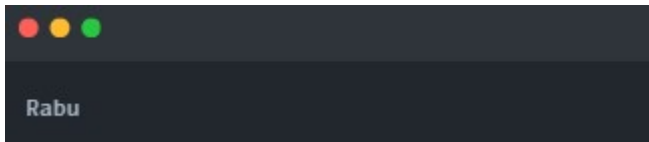


```
#include <stdio.h>

int main() {
    int hari = 3;

    switch(hari) {
        case 1:
            printf("Senin\n");
            break;
        case 2:
            printf("Selasa\n");
            break;
        case 3:
            printf("Rabu\n");
            break;
        case 4:
            printf("Kamis\n");
            break;
        case 5:
            printf("Jumat\n");
            break;
        case 6:
            printf("Sabtu\n");
            break;
        case 7:
            printf("Minggu\n");
            break;
        default:
            printf("Hari tidak valid\n");
            break;
    }
}
```

Output:



Pada kode di atas, variabel `hari` diatur dengan nilai 3. Pernyataan `switch` memeriksa nilai `hari` dan mencocokkannya dengan salah satu dari beberapa `case`. Dalam hal ini, `hari` cocok dengan `case 3`, sehingga blok kode di dalam `case 3` dieksekusi dan mencetak "Rabu" ke layar. Setiap `case` diakhiri dengan `break` untuk mencegah eksekusi berlanjut ke `case` berikutnya. Jika tidak ada kasus yang cocok, blok `default` akan dieksekusi, tetapi dalam contoh ini, nilai 3 cocok dengan `case 3`, sehingga "Rabu" adalah output yang dihasilkan.

Selain itu, kalian juga bisa menambahkan operasi Boolean pada conditions, yaitu menggunakan true atau false. Contohnya, perhatikan kode dibawah ini :

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

int main() {
    bool isRainy = false;

    if (isRainy) {
        printf("It's rainy outside.\n");
    } else {
        printf("It's not rainy outside.\n");
    }

    return 0;
}
```

Dalam program ini, kita menggunakan tipe data boolean dari library <stdbool.h> untuk membuat variabel isRainy. Tipe data boolean memungkinkan kita menyimpan nilai true atau false.

Kemudian kita menggunakan pernyataan if untuk melakukan pengecekan kondisi. Jika nilai dari isRainy adalah true, maka blok kode dalam pernyataan if akan dieksekusi, dan pesan "It's rainy outside." akan dicetak menggunakan printf(). Jika nilai dari isRainy adalah false, maka blok kode dalam pernyataan else akan dieksekusi, dan pesan "It's not rainy outside." akan dicetak menggunakan printf().

Dalam contoh ini, nilai isRainy diatur sebagai false, sehingga blok kode dalam pernyataan else yang akan dieksekusi. Oleh karena itu, pesan "It's not rainy outside." akan dicetak. Jika kita mengubah nilai isRainy menjadi true, maka pesan yang dicetak akan menjadi "It's rainy outside."

MEMORY ADDRESS

Ketika kita menambahkan nilai pada suatu variabel, maka nilai tersebut akan disimpan didalam memory address. Setiap variable dalam bahasa C memiliki alamat memori yang unik. Memory address adalah konsep kunci dalam bahasa pemrograman, terutama dalam pemrograman rendah (*low-level programming*) seperti bahasa C. Alamat ini digunakan oleh program untuk mengakses nilai yang tersimpan dalam variabel atau untuk melakukan manipulasi data. dengan memahami memory address, kita bisa mengelola variabel, alokasi memori, dan bahkan keamanan dalam pengembangan perangkat lunak.

Setiap kali kita mendeklarasikan variabel dalam program, kita memberikan petunjuk kepada sistem komputer untuk mengalokasikan ruang memori yang dibutuhkan untuk variabel tersebut. Misalnya, ketika kita mendeklarasikan variabel dengan tipe data **int**, system akan mengalokasikan sejumlah byte tertentu dalam memori untuk menyimpan nilai integer. Alokasi memori ini memungkinkan kita untuk menyimpan dan mengakses nilai variabel.

Tipe data yang kita deklarasikan dalam program memiliki pengaruh langsung terhadap ukuran alokasi memori. Tipe data yang berbeda memiliki ukuran yang berbeda. Sebagai contoh, **int** biasanya menggunakan 4 byte, **float** menggunakan 4 byte, dan **double** menggunakan 8 byte. Ukuran alokasi memori ini penting karena mempengaruhi berapa banyak data yang dapat kita simpan dan bagaimana data tersebut akan diinterpretasikan oleh program.

Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

Data dalam memori diorganisasi dalam unit kecil yang disebut **byte**. Byte adalah unit dasar penyimpanan dalam komputer dan memiliki alamat memori yang unik. Selain byte, konsep lain yang penting adalah word. Word adalah sejumlah byte yang dianggap sebagai unit tunggal oleh prosesor. Misalnya, pada arsitektur 32-bit, sebuah word biasanya terdiri dari 4 byte. Penyusunan data dalam word memungkinkan prosesor untuk mengambil data secara efisien.

Mengukur Ukuran Memori dengan sizeof

Operator **sizeof** dalam C digunakan untuk mendapatkan ukuran dari tipe data atau variabel dalam byte.

Contoh Penggunaan sizeof

```
#include <stdio.h>

int main() {
    printf("Ukuran char: %lu byte\n", sizeof(char));
    printf("Ukuran int: %lu byte\n", sizeof(int));
    printf("Ukuran float: %lu byte\n", sizeof(float));
    printf("Ukuran double: %lu byte\n", sizeof(double));
    printf("Ukuran pointer: %lu byte\n", sizeof(int*));

    return 0;
}
```

Hasil dari program di atas:

```
Ukuran char: 1 byte
Ukuran int: 4 byte
Ukuran float: 4 byte
Ukuran double: 8 byte
Ukuran pointer: 8 byte
```

```
=== Code Execution Successful ===
```

Coba perhatikan kode program dibawah ini :

```
#include <stdio.h>

int main(){
    char nama [] = "omen";
    int umur = 25;

    printf("Alamat memori nama : %p\n", &nama);
    printf("Alamat memori umur : %p\n", &umur);

    return 0;
}
```

Program di atas adalah contoh program sederhana dalam bahasa C yang menampilkan alamat memori dari dua variabel, yaitu nama dan umur. Pada deklarasi variabel nama dengan inisialisasi nilai "omen", karakter-karakter ini akan disimpan secara berurutan dalam memori. Begitu juga dengan variabel umur yang dari inisialisasi nilainya akan disimpan secara berurutan dalam memori. Kemudian, pada baris 7 dan 8, digunakan fungsi printf dengan format specifier **%p** untuk mencetak alamat memori dari masing-masing variabel menggunakan operator **&** yang digunakan untuk mengambil alamat memori dari variabel tersebut.

Maka, output program diatas akan menghasilkan :

```
Alamat memori nama : 0060FEFF
Alamat memori umur : 0060FEF8

Process returned 0   execution time : 0.038 s
Press any key to continue.
```

Selain itu, dengan menggunakan pointer, kita dapat secara langsung berinteraksi dengan alamat memori dan melakukan operasi terhadap data yang tersimpan di lokasi memori tersebut. Ini memungkinkan kita untuk mengakses dan memanipulasi data secara efisien, serta mentransfer data di antara fungsi atau prosedur dalam program.

```
#include <stdio.h>

int main() {
    int num = 10;
    int *ptr; // Deklarasi pointer

    ptr = &num; // Inisialisasi pointer dengan alamat memori
                // dari variabel 'num'

    printf("Nilai dari variabel num: %d\n", num);
    printf("Alamat memori dari variabel num: %p\n", &num);
    printf("Nilai yang ditunjuk oleh pointer: %d\n", *ptr);
    printf("Alamat memori yang ditunjuk oleh pointer: %p\n",
           ptr);

    return 0;
}
```

Output:

```

Nilai dari variabel num: 10
Alamat memori dari variabel num: 0x7ffdd7238f74
Nilai yang ditunjuk oleh pointer: 10
Alamat memori yang ditunjuk oleh pointer: 0x7ffdd7238f74

=== Code Execution Successful ===

```

Kode diatas merupakan contoh sederhana tentang penggunaan memory address yang menggunakan pointer. Pointer adalah variabel khusus yang menyimpan alamat memori. Dalam contoh ini, kita mendeklarasikan variabel **num** yang menyimpan nilai 10. Kemudian, kita mendeklarasikan pointer **ptr** dengan operator “*”. Selanjutnya, kita menginisialisasi pointer **ptr** dengan alamat memori dari variabel **num** menggunakan operator **&**. Dalam printf, **%p** digunakan untuk mencetak alamat memori dan ***ptr** digunakan untuk mendapatkan nilai yang ditunjuk oleh pointer.

Pointer bisa digunakan untuk :

- Alokasi memori dinamis : dalam bahasa C, kalian dapat mengalokasikan memori saat program berjalan menggunakan fungsi ‘**malloc()**’ atau ‘**calloc()**’ dan mengaksesnya menggunakan pointer. **malloc()** digunakan untuk mengalokasikan sejumlah tertentu byte memori dari heap (area memori yang digunakan untuk alokasi dinamis). Fungsi ini memiliki satu parameter, yaitu ukuran memori yang ingin dialokasikan dalam byte. Jika alokasi gagal, malloc akan mengembalikan null. Sedangkan fungsi **calloc()** digunakan untuk mengalokasikan memori untuk sejumlah elemen dari suatu tipe data dengan ukuran yang diberikan. Fungsi calloc memiliki dua parameter, yaitu jumlah elemen dan ukuran elemen dalam byte.
- Pengembalian nilai dari fungsi : kalian bisa mengembalikan beberapa nilai dari fungsi menggunakan pointer.
- Perubahan variabel dalam fungsi : jika kalian ingin mengubah variabel dalam fungsi, kalian bisa mengirimkan alamat memori variabel tersebut melalui pointer.

Contoh Kasus Alokasi Memori Dinamis

Berikut adalah contoh program yang menunjukkan alokasi memori dinamis dengan **malloc()**:

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

int main() {
    int n, i;
    int *arr;

    printf("Masukkan jumlah elemen: ");
    scanf("%d", &n);

    // Alokasi memori dinamis
    arr = (int*)malloc(n * sizeof(int));

    // Memeriksa apakah alokasi berhasil
    if (arr == NULL) {
        printf("Alokasi memori gagal!\n");
        return 1;
    }

    // Menginisialisasi dan menampilkan nilai array
    for (i = 0; i < n; ++i) {
        arr[i] = i + 1;
        printf("arr[%d] = %d, Alamat memori = %p\n", i, arr[i],
(void*)&arr[i]);
    }

    // Membebaskan memori
    free(arr);

    return 0;
}
```

Penjelasan Program

Pada contoh di atas:

- Pengguna diminta untuk memasukkan jumlah elemen **n**.
- Memori sebesar **n * sizeof(int)** dialokasikan secara dinamis menggunakan fungsi **malloc()**.
- Jika alokasi memori gagal, program akan menampilkan pesan kesalahan dan terminasi.
- Jika alokasi berhasil, program akan menginisialisasi array dengan nilai-nilai dan menampilkan elemen-elemen beserta alamat memori masing-masing.
- Memori yang dialokasikan kemudian dibebaskan menggunakan fungsi **free()**.

Berikut adalah contoh program yang menunjukkan alokasi memori dinamis dengan **calloc()**:

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

int main() {
    int n, i;
    int *arr;

    printf("Masukkan jumlah elemen: ");
    scanf("%d", &n);

    // Alokasi memori dinamis
    arr = (int*)calloc(n, sizeof(int));

    // Memeriksa apakah alokasi berhasil
    if (arr == NULL) {
        printf("Alokasi memori gagal!\n");
        return 1;
    }

    // Menampilkan dan menginisialisasi nilai array
    for (i = 0; i < n; i++) {
        printf("arr[%d] = %d, Alamat memori = %p\n", i, arr[i],
(void*)&arr[i]);
    }

    // Membebaskan memori
    free(arr);

    return 0;
}
```

Perbedaan malloc dan calloc

- **malloc** hanya mengalokasikan memori dan tidak menginisialisasi nilai.
- **calloc** mengalokasikan memori dan menginisialisasi semua elemen dengan nilai nol.

➤ TUJUAN PENGGUNAAN MEMORY ADDRESS

Memory address bisa digunakan untuk penyimpanan data. Ketika kalian mendeklarasikan variabel dalam program, komputer mengalokasikan ruang memori untuk menyimpan nilai dari variabel tersebut, dan variabel diakses melalui alamat memori yang ditentukan. Instruksi dari program juga disimpan memori komputer dan diakses melalui alamat memori. Ketika program dieksekusi, CPU akan membaca instruksi dari alamat memori tertentu dan menjalankan tindakan sesuai dengan instruksi tersebut.

Selain itu, memory address dapat digunakan untuk mengamankan perangkat lunak dari berbagai jenis serangan yang mungkin terjadi. Salah satu serangan umum yang sering terjadi adalah *buffer overflow*, dimana penyerang mencoba menulis data di luar batas alokasi memori yang sah, yang dapat mengakibatkan kode berbahaya dieksekusi atau data sensitive terekspos.

Dengan memahami cara alamat memori digunakan dan diakses, Anda dapat mengambil langkah-langkah pengamanan seperti:

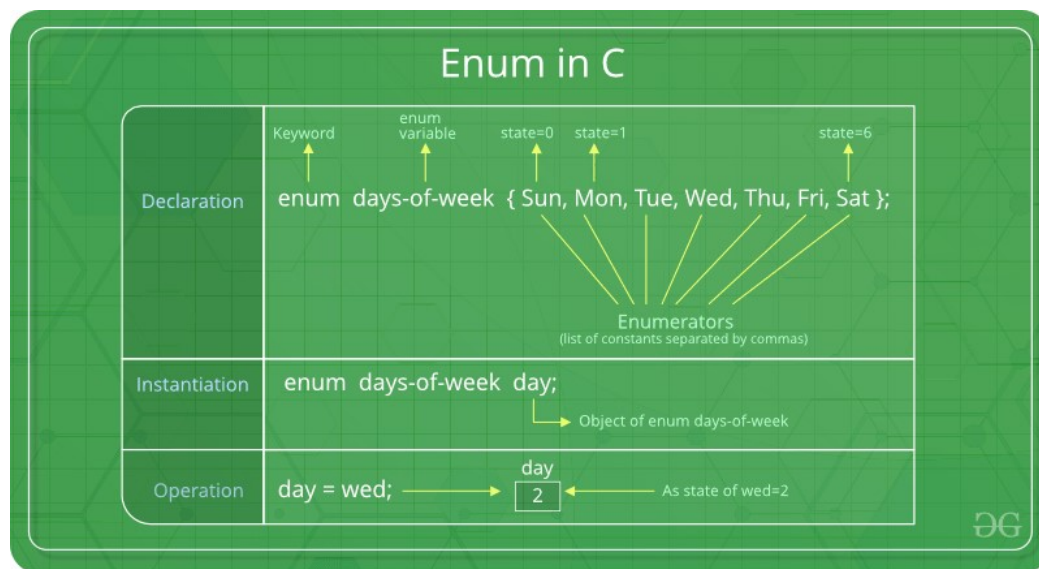
- Memvalidasi input dan memastikan bahwa data yang dimasukkan tidak melebihi batas alokasi memori yang telah ditetapkan.
- Menggunakan fungsi aman seperti **strncpy** atau **snprintf** untuk menghindari buffer overflow.
- Memeriksa pointer sebelum mengakses alamat memori yang ditunjuk oleh mereka.
- Memastikan izin akses yang tepat untuk area memori, sehingga kode berbahaya tidak dapat dieksekusi dari area tersebut.

Dengan demikian, pemahaman tentang memory address tidak hanya penting dalam penyimpanan dan pengaksesan data, tetapi juga dalam pengamanan perangkat lunak dari ancaman serangan. Dengan melibatkan praktik terbaik dalam manajemen memori dan pengelolaan alamat, kalian dapat meningkatkan keamanan dan kualitas perangkat lunak yang kalian kembangkan.

Penggunaan memory address juga memungkinkan kita untuk secara langsung mengakses dan memanipulasi data dalam program. Ini penting untuk operasi seperti pindah data dari satu variabel ke variabel lain, pengurutan data, dan transformasi data.

ENUMS

Enum adalah salah satu fitur dalam bahasa pemrograman C yang memungkinkan kalian mendefinisikan tipe data khusus yang terdiri dari sekumpulan konstanta. Enum digunakan untuk menyederhanakan kode dan membuatnya lebih mudah dibaca dan dimengerti oleh programmer. Enums memungkinkan kita untuk memberi nama pada nilai-nilai yang berhubungan dengan suatu konsep tertentu, sehingga memudahkan pemahaman dan pemrograman.



Dalam C, enum dideklarasikan dengan menggunakan kata kunci **“enum”**, diikuti oleh nama enumnya, dan di dalamnya berisi konstanta-konstanta yang dipisahkan oleh koma. Setiap konstanta diberi nama dan nilainya secara otomatis bertambah 1 dari konstanta sebelumnya.

```

#include <stdio.h>

enum Member {
    ROSE,        // Nilai default: 0
    JENNIE,      // Nilai default: 1
    JISOO,       // Nilai default: 2
    LALISA       // Nilai default: 3
};

int main() {
    enum Member memberFavorit = JENNIE;
    printf("Member favorit saya adalah %d\n", memberFavorit);
    return 0;
}

```

Untuk membuat enum, kalian bisa menuliskannya seperti contoh kode program diatas. Enum ditulis sebelum membuat fungsi main() dan diikuti dengan nama enumnya. Contoh diatas adalah dengan nama enum **Member**. Kemudian jangan lupa untuk menuliskan kurung kurawal ({ }) sebagai pembatas dalam suatu fungsi, didalamnya kalian tuliskan data yang kalian butuhkan untuk program. Secara otomatis, setiap anggota enum diberi nama dan diberi nilai konstan yang berhubungan dengan konsep yang diwakili.

Hasil output program diatas adalah :

```

Member favorit saya adalah 1

Process returned 0    execution time : 0.008 s
Press any key to continue.

```

TUJUAN PENGGUNAAN ENUMS

Tujuan penggunaan enums adalah untuk memberikan nama yang bermakna kepada nilai-nilai tersebut sehingga memudahkan pemahaman dan penggunaan dalam kode program. Berikut beberapa tujuan penggunaan enums dalam pemrograman :

1. Kode yang Lebih Mudah Dibaca: Dengan enums, kalian dapat memberikan nama yang deskriptif kepada nilai-nilai konstan, membuat kode program lebih mudah dipahami oleh pengembang lain atau bahkan diri kalian sendiri di masa depan.
2. Menghindari Kode Sulit Dibaca: Tanpa enums, kalian mungkin perlu menggunakan konstanta angka atau karakter sebagai nilai konstan. Hal ini dapat membuat kode sulit dipahami jika seseorang yang membaca kode tidak tahu apa yang angka tersebut mewakili.
3. Kode yang Lebih Kuat dan Terstruktur: Enums membantu mengorganisasi nilai-nilai konstan menjadi kelompok yang terstruktur. Ini membantu menghindari konflik nama dan menyediakan cara yang lebih baik untuk mengelompokkan konstanta yang terkait.
4. Mengurangi Kesalahan: Dengan enums, kalian mengurangi risiko kesalahan dalam mengetik angka atau karakter yang salah dalam kode. Enums membantu mencegah kesalahan yang mungkin terjadi saat menggunakan konstanta angka atau karakter.
5. Pemeliharaan Lebih Mudah: Jika kalian perlu mengubah nilai konstan, kalian hanya perlu mengubahnya dalam definisi enum, dan perubahan ini akan diterapkan di seluruh kode yang menggunakan enum tersebut.
6. Kode yang Lebih Portabel: Dalam beberapa situasi, nilai-nilai konstan dalam enums dapat membuat kode lebih portabel, karena nilai-nilai tersebut tetap konsisten di berbagai platform.
7. Keterbacaan yang Lebih Tinggi dalam Dokumentasi: Saat kalian mendokumentasikan kode, enums membantu dalam memberikan penjelasan yang lebih jelas tentang nilai-nilai yang digunakan dalam kode.

CARA MENGAKSES ENUMS

Setelah enum didefinisikan, kalian bisa mengakses nilainya dengan menyebutkan nama enum diikuti oleh nilai yang ingin kalian akses. Misalnya, jika Anda memiliki enum bernama Day, kalian dapat mengakses nilainya seperti ini:

```
#include <stdio.h>

enum Day { Sunday, Monday, Tuesday, Wednesday, Thursday, Friday,
Saturday }; int main() {
    enum Day today = Wednesday;

    if (today == Wednesday)
    { printf("Today is
Wednesday.\n");
    } else {
    printf("Today is not Wednesday.\n");
    }
}
```

Output:

```
Today is Wednesday.

=== Code Execution Successful ===
```

Dari contoh kode di atas, enum Day didefinisikan, dan kemudian nilai Wednesday diakses dan digunakan dalam program. Enum sangat berguna dalam situasi di mana kalian perlu mengelompokkan beberapa nilai terkait menjadi satu kelompok dan memberi nama yang lebih bermakna pada nilai-nilai tersebut.

Catatan Penting: Enum dalam C secara default diindeks mulai dari 0. Kalian juga dapat mengatur nilai awal dari enum jika kalian ingin indeks dimulai dari nilai tertentu.

Dalam enum, kalian tidak perlu memasukkan tanda kutip seperti pada string atau karakter. Kalian dapat langsung menggunakan nilai-nilainya seperti Sunday, Monday, dsb.

Mengatur Nilai Custom pada Enum

Anda dapat menetapkan nilai custom pada anggota enum saat mendefinisikannya untuk lebih sesuai dengan kebutuhan aplikasi Anda.

Contoh Nilai Custom

```
#include <stdio.h>

enum Bulan {
    JANUARI = 1, FEBRUARI, MARET, APRIL, MEI, JUNI, JULI, AGUSTUS,
    SEPTEMBER, OKTOBER, NOVEMBER, DESEMBER
};

int main() {
    enum Bulan bulanIni;
    bulanIni = OKTOBER;

    printf("Bulan ini adalah bulan ke-%d dalam setahun.\n", bulanIni);
    return 0;
}
```

Output:

```
Bulan ini adalah bulan ke-10 dalam setahun.

=== Code Execution Successful ===
```

Penjelasan Contoh

- **enum Bulan** menetapkan nilai starting point sebesar 1 untuk JANUARI. Nilai berikutnya adalah FEBRUARI=2, MARET=3, dan seterusnya.
- Variabel **bulanIni** dideklarasikan dengan tipe **enum Bulan** dan diberikan nilai **OKTOBER**.
- Pada pemanggilan **printf**, nilai integer dari **OKTOBER** yang adalah 10 ditampilkan.

Contoh lain mengakses enums menggunakan **switch** bisa kalian perhatikan pada kode dibawah

```
#include <stdio.h>

enum Member {
    ROSE = 1,
    JENNIE,
    JISOO,
    LALISA
};

int main() {
    enum Member who;
    who = JENNIE;

    printf("She is a ");
    switch (who) {
        case 1:
            printf("Singer");
            break;
        case 2:
            printf("Rapper");
            break;
        case 3:
            printf("Actress");
            break;
        case 4:
            printf("Dancer");
            break;
        default:
            printf("Unknown");
    }
    printf(".\n");

    return 0;
}
```

Output:

```
She is a Rapper.
```

```
=== Code Execution Successful ===
```

Pada contoh kode di atas, kita menggunakan enum dengan nama "Member" yang berisi beberapa anggota untuk mewakili nama-nama dalam suatu kelompok. Enum tersebut diinisialisasi mulai dari nilai 1 karena kita nantinya akan menggunakan switch case untuk mengecek nilai enum. Selanjutnya, kita mendeklarasikan variabel "who" dengan tipe enum "Member" dan memberikan nilai JENNIE ke variabel tersebut, sehingga variabel "who" memiliki nilai yang secara default setara dengan 2 berdasarkan inisialisasi enum.

Kemudian, pada bagian switch case, program akan mengevaluasi nilai dari variabel "who". Karena pada enum setiap anggota secara otomatis memiliki nilai default yang berurutan, maka program akan menjalankan kode pada case yang memiliki nilai yang sama dengan nilai dari variabel "who". Dalam contoh ini, program akan mencetak output yang sesuai dengan case 2 karena variabel "who" memiliki nilai JENNIE yang setara dengan 2 dalam enum "Member".

TUGAS PRAKTIKUM

CODELAB 1

Instruksi:

1. Copy dan jalankan kode program di bawah ini pada IDE kalian.
2. Lengkapi kode program yang masih belum lengkap.
3. Tunjukkan hasilnya kepada asisten dan jelaskan apa yang telah kalian perbaiki/implementasikan.

Studi Kasus:

Pada kasus ini, kalian diminta untuk membuat program yang mengklasifikasikan harga tiket pesawat berdasarkan kategori usia pengguna.

Kode Program:

```
#include <stdio.h>

int main() {
    int umur;

    printf("Masukkan umur penumpang: ");
    scanf("%d", &umur);

    if (umur >= 0 && umur <= 120) {
        if (umur < 2) {
            printf("Penumpang mendapatkan tiket gratis.\n");
        } else if (umur >= 2 && umur <= 12) {
            printf("Harga tiket untuk penumpang anak-anak adalah 50%%\ndari harga normal.\n");
        } else if (umur > 12 && umur <= 60) {
            printf("Harga tiket untuk penumpang dewasa adalah harga\nnormal.\n");
        } else {
            printf("Harga tiket untuk penumpang lanjut usia adalah\n75%% dari harga normal.\n");
        }
    } else {
        printf("Input tidak valid. Umur harus berada dalam rentang 0\ningga 120.\n");
    }

    return 0;
}
```


CODELAB 2**Kombinasi Enums dan Control Flow (If-Else dan Switch)****Instruksi:**

Buatlah program sederhana menggunakan enums dan condition di mana program akan dieksekusi berdasarkan inputan user, dengan ketentuan sebagai berikut:

1. Enums memiliki 3 anggota yaitu:
 - a. Economy (Kelas Ekonomi)
 - b. Business (Kelas Bisnis)
 - c. FirstClass (Kelas Utama)
2. Gunakan conditions untuk mengakses enums berdasarkan pilihan dari masing-masing kelas tiket.

Contoh output yang diharapkan:

```
Pilih Kelas Tiket:  
1. Economy  
2. Business  
3. First Class  
Masukkan pilihan (1-3): 2  
Harga tiket kelas bisnis: $500  
  
=== Code Execution Successful ===
```

Jangan lupa untuk menunjukkan kode program kepada asisten masing-masing sebagai bagian dari penilaian kalian.

KEGIATAN 1

Buatlah program yang berfungsi sebagai sistem pemesanan tiket bioskop dengan ketentuan sebagai berikut:

1. Terdapat dashboard untuk keterangan bioskop.
2. Di bioskop ini, terdapat beberapa pilihan film yaitu 5 film dan 3 jenis tiket.
 - Untuk jenis tiket dibagi dalam bentuk enum.
 - Film: **enum Film**
 - Jenis Tiket: **enum JenisTiket**
3. Pelanggan akan memilih film dan jenis tiket yang diinginkan. Kalian harus menginput pilihan pelanggan menggunakan angka. Setiap pilihan akan menghasilkan total belanja.
4. Tanyakan pada pelanggan apakah termasuk dalam member atau bukan member dari bioskop.
 - Jika pelanggan adalah member, maka pelanggan akan mendapatkan potongan harga 10%.
 - Jika bukan member maka pelanggan tidak mendapat potongan harga.
5. Setelah pelanggan memilih film dan jenis tiket, dan telah bertanya member atau bukan, kalian harus menghitung total belanja dengan mengakumulasi harga dari film dan jenis tiket yang dipilih.
6. Di samping total harga, kalian harus menampilkan alamat memori variabel yang menyimpan total harga makanan atau minuman tersebut.
7. Setelah total harga terhitung, input jumlah uang yang dibayarkan pelanggan:
 - a) Jika uang yang dibayarkan lebih dari total belanja, tampilkan jumlah kembalian.
 - b) Jika uang yang dibayarkan kurang dari total belanja, tampilkan pesan bahwa uang kurang.
 - c) Jika uang yang dibayarkan sama dengan total belanja, tampilkan pesan terima kasih.

Contoh output yang diharapkan:

```

===== Bioskop =====
Film:
1. Spiderman - Rp.50000
2. Batman - Rp.55000
3. Superman - Rp.60000
4. Avengers - Rp.70000
5. Joker - Rp.45000
Pilih film (1-5): 2

Jenis Tiket:
1. Reguler - Rp.30000
2. VIP - Rp.100000
3. VVIP - Rp.150000
Pilih jenis tiket (1-3): 2
Apakah Anda adalah member? (Y/N): y
Total harga: Rp. 139500.00
Alamat memori variabel total: 0x7ffc27640440
Masukkan uang pembayaran: 200000
Kembalian Anda: Rp. 60500.00

=== Code Execution Successful ===

```

KRITERIA & DETAIL PENILAIAN

Kriteria	Poin
Codelab 1	5
Codelab 2	15
Kegiatan 1	40
Pemahaman	20
Ketepatan Menjawab	20