

VERSI 1.3

JULI, 2023



[PEMROGRAMAN DASAR]

MODUL 3 - MEMORY ADDRESS, CONDITIONS, ENUMS

DISUSUN OLEH:

KIARA AZZAHRA

ANNISA ARTANTI WIDYADHANA

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:

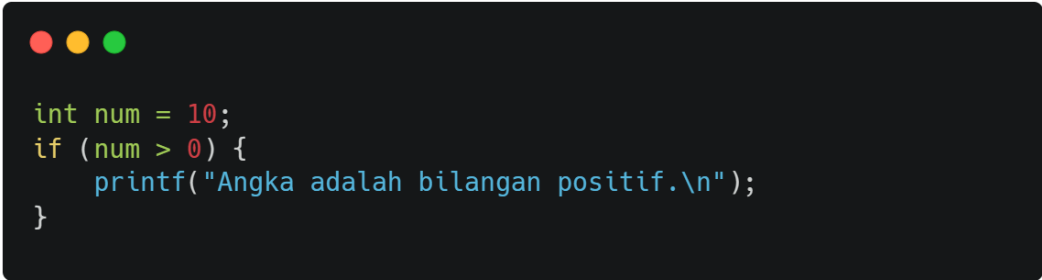
1. Pengkondisian dalam keputusan : misal kalian ingin mengecek apakah kalian boleh masuk ke bioskop atau tidak. Jika kalian berumur di atas 18 tahun, kalian boleh masuk. Kalau tidak berusia di atas 18 tahun, maka kalian harus menunggu.
2. Pengkondisian dalam Pengulangan: kalian mungkin ingin menghitung 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.

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**.

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..

```

#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”.

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”.

➤ 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.

➤ **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”.

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.

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.

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;
}

```

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.

➤ 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");
    }

    return 0;
}
```

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.

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;
}
```

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

Copy paste kode program di bawah ini ke IDE kalian. Cobalah untuk melengkapi dan memperbaikinya, jika sudah tunjukkan ke asisten masing - masing dan jelaskan apa saja yang telah kalian perbaiki.

```
#include <stdio.h>

int main() {
    String nilai;

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

    if (nilai >= 0 && nilai <= 100) {
        if (nilai <= 90) {
            printf("Nilai Anda adalah A.\n")
        } else (nilai >= 80) {
            printf("Nilai Anda adalah B.\n")
        } else (nilai >= 70) {
            printf("Nilai Anda adalah C.\n")
        } else (nilai > 60) {
            printf("Nilai Anda adalah D.\n")
        } else {
            printf("Nilai Anda adalah E.\n")
        } break;
    }
```

```

        default :
            printf("Program Selesai");
    } else if{
        printf("Input tidak valid. Nilai harus berada dalam rentang 0 hingga
100.\n")
    }

    return 0;
}

```

CODELAB 2

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

1. Enums memiliki 3 anggota yaitu :
 - a. Unstoppable (Film Action)
 - b. Insidious (Film Horror)
 - c. Us (Film Thriller)
2. Gunakan conditions untuk mengakses enums berdasarkan genre dari masing-masing film di atas.

Contoh output :

```

Masukkan Nama Film:
1. Unstoppable
2. Insidious
3. Us
Pilihan anda: 1

Film Action
Process returned 0   execution time : 2.345 s
Press any key to continue.

```

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

KEGIATAN 1

Buat program kasir untuk rumah makan dengan ketentuan sebagai berikut :

1. Terdapat dashboard untuk keterangan rumah makan
2. Di rumah makan ini, terdapat beberapa pilihan yaitu 5 menu makanan dan 5 menu minuman.

Untuk menu makan dan minuman dipisah dalam bentuk enum.

Menu makanan : **enum Makanan**

Menu minuman : **enum Minuman**

3. Pelanggan akan memilih menu makanan dan minuman 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 rumah makan. Jika pelanggan adalah member, maka pelanggan akan mendapatkan potongan 10%. Jika bukan member maka pelanggan tidak mendapat potongan harga.
5. Setelah pelanggan memilih menu makanan, minuman dan bertanya member atau bukan, kalian harus menghitung total belanja dengan mengakumulasi harga dari menu 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.

Berikut contoh output yang dihasilkan :

```

===== Rumah Makan =====
Menu Makanan:
1. Nasi Goreng    Rp. 50000
2. Mie Goreng     Rp. 45000
3. Ayam Goreng    Rp. 65000
Pilih menu makanan (1-3): 3

Menu Minuman:
1. Air Mineral    Rp. 13000
2. Es Teh         Rp. 15000
3. Jus Jeruk      Rp. 20000
Pilih menu minuman (1-3): 2
Apakah Anda adalah member? (Y/N): y

Total harga: Rp. 72000.00
Alamat memori variabel total: 0060FEFC
Masukkan uang pembayaran: 80000
Kembalian Anda: Rp. 8000.00

Process returned 0   execution time : 23.933 s
Press any key to continue.

```

KRITERIA & DETAIL PENILAIAN

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