

VERSI 1.0
AGUSTUS, 2024



[PEMROGRAMAN DASAR]

MODUL 6 - FUNCTION, FILE HANDLING

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 diharapkan mempelajari materi sebelum mengerjakan tugas, materi yang tercakup antara lain:

- Pengenalan Fungsi
- Pembuatan Fungsi
- Pemanggilan Fungsi
- Parameter dan Argumen Fungsi
- Multiple Parameters
- Array sebagai Parameter Fungsi
- Pengembalian Nilai
- Deklarasi Fungsi
- Rekursif
- File Handling

TUJUAN

- Mahasiswa dapat memahami konsep dasar fungsi dalam pemrograman dan pentingnya penggunaan fungsi untuk memecah program menjadi bagian-bagian yang lebih kecil dan terorganisir.
- Mahasiswa dapat mengenal dan memahami fungsi bawaan (built-in) yang telah disediakan oleh bahasa pemrograman.
- Mahasiswa dapat membuat fungsi baru yang sesuai dengan kebutuhan program dan memahami struktur dasar pembuatan fungsi.
- Mahasiswa dapat memahami cara memanggil fungsi agar code di dalam fungsi dijalankan dan memahami bagaimana aliran program berjalan saat fungsi dipanggil.
- Mahasiswa dapat memahami konsep parameter dan argumen dalam fungsi serta perbedaan keduanya.
- Mahasiswa dapat menggunakan lebih dari satu parameter dalam definisi fungsi untuk mengoperasikan data yang lebih kompleks.

- Mahasiswa dapat memahami cara menggunakan array sebagai parameter fungsi untuk mengoperasikan sejumlah besar data.
- Mahasiswa dapat memahami konsep nilai kembalian fungsi dan cara menggunakannya untuk mengembalikan hasil dari fungsi.
- Mahasiswa dapat memahami pentingnya deklarasi fungsi dan bagaimana cara mendeklarasikan fungsi sebelum digunakan.
- Mahasiswa dapat memahami konsep rekursif dan cara menggunakan fungsi rekursif untuk menyelesaikan masalah dengan pendekatan berulang.
- Mahasiswa menguasai teknik pembuatan, penulisan, dan pembacaan berkas untuk menyimpan dan mengakses data secara permanen dalam program, meningkatkan fleksibilitas dan kegunaan program.

TARGET MODUL

- Mahasiswa dapat menjelaskan apa itu fungsi dan manfaatnya dalam mengorganisir code program.
- Mahasiswa dapat menggunakan fungsi yang telah ditentukan (built-in) untuk melakukan tugas-tugas tertentu tanpa harus mengimplementasikan dari awal.
- Mahasiswa dapat membuat fungsi sederhana dengan benar dan mengenal bagian-bagian dalam pembuatan fungsi.
- Mahasiswa dapat memanggil fungsi yang telah dibuat dan menjalankan blok code di dalamnya.
- Mahasiswa dapat mendefinisikan fungsi dengan parameter dan memahami cara memberikan argumen saat memanggil fungsi.
- Mahasiswa dapat membuat fungsi dengan beberapa parameter dan mengerti cara mengirimkan nilai argumen yang sesuai.
- Mahasiswa dapat mengimplementasikan fungsi yang menerima array sebagai argumen.
- Mahasiswa dapat menggunakan pernyataan return untuk mengembalikan nilai dari fungsi.
- Mahasiswa dapat mendeklarasikan fungsi secara benar dan mengenali manfaatnya dalam mengorganisir code program.
- Mahasiswa dapat membuat dan memahami implementasi fungsi rekursif dalam pemrograman.
- Mampu membuat dan membuka berkas, menulis data ke dalam berkas, serta membaca data dari berkas menggunakan fungsi File Handling dalam bahasa C.

PERSIAPAN SOFTWARE/APLIKASI

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

MATERI PRAKTIKUM

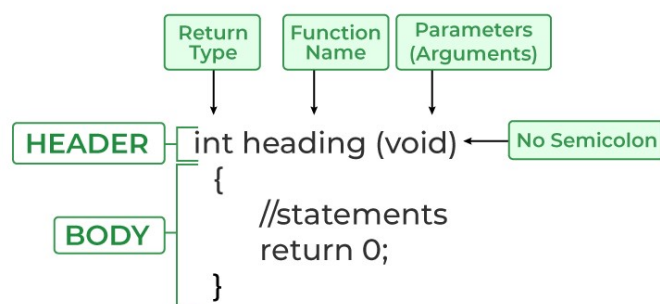
FUNCTION – INTRODUCTION

Dalam dunia pemrograman, "fungsi" atau "prosedur" adalah konsep yang sangat penting untuk membantu mengelola kompleksitas kode. Fungsi memungkinkan kita untuk membagi program menjadi bagian-bagian yang lebih kecil, memisahkan tugas-tugas tertentu ke dalam unit yang dapat dipanggil secara terpisah. Dengan cara ini, program kita tidak hanya menjadi lebih terstruktur, tetapi juga lebih mudah dibaca, dimengerti, dan dikelola.

Bayangkan jika kita harus menulis seluruh kode program dalam satu blok di dalam fungsi `main()`. Ketika program tumbuh dan menjadi lebih besar serta kompleks, hal ini akan menyebabkan kerumitan yang sulit diatasi. Oleh karena itu, kita menggunakan fungsi untuk mengatasi masalah ini.

Dalam pemrograman, fungsi atau prosedur sering digunakan untuk membungkus program menjadi bagian-bagian kecil. Tujuannya agar program tidak menumpuk pada fungsi `main()` saja. Bayangkan saja, kalau program kita tambah besar dan kompleks. Kalau semua code nya ditulis di dalam fungsi `main()`, maka kita akan kesulitan membacanya. Maka dari itu, kita harus menggunakan fungsi.

Function Definition



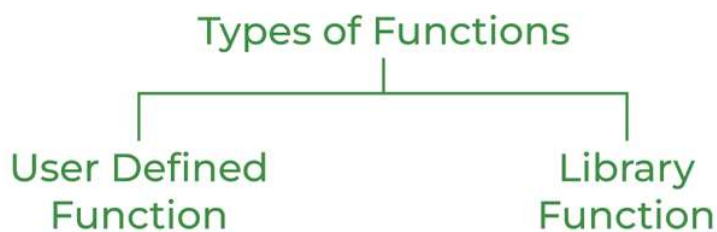
Fungsi adalah blok kode yang hanya berjalan saat dipanggil. Kalian dapat meneruskan data, yang dikenal sebagai parameter, ke dalam suatu fungsi. Fungsi digunakan untuk melakukan tindakan tertentu, dan penting untuk menggunakan kembali code. Tentukan kode satu kali, dan gunakan berkali-kali.

Cara kerja fungsi dalam bahasa C dapat dibagi menjadi langkah-langkah berikut:

1. Mendeklarasikan Fungsi: Ini adalah langkah di mana kita mendeklarasikan sebuah fungsi. Di sini, kita mendefinisikan jenis nilai yang dikembalikan (return type) dan parameter dari fungsi tersebut.
2. Mendefinisikan Fungsi: Ini adalah langkah di mana kita menulis kode sebenarnya dari fungsi tersebut.
3. Memanggil Fungsi: Ini adalah langkah di mana kita memanggil fungsi dengan melewati argumen ke dalam fungsi.
4. Menjalankan Fungsi: Ini adalah langkah di mana kita menjalankan semua pernyataan di dalam fungsi untuk mendapatkan hasil akhir.
5. Mengembalikan Nilai: Ini adalah langkah di mana nilai yang telah dihitung setelah eksekusi fungsi dikembalikan. Keluar dari fungsi adalah langkah terakhir di mana semua memori yang dialokasikan untuk variabel, fungsi, dll., dihancurkan sebelum memberikan kontrol penuh ke fungsi utama.

Jenis-Jenis Fungsi, Ada dua jenis fungsi dalam bahasa C:

1. Fungsi Pustaka: Fungsi yang sudah disediakan oleh bahasa C.
2. Fungsi yang Didefinisikan Pengguna: Fungsi yang dibuat oleh pengguna sesuai kebutuhan.



Manfaat utama dari penggunaan fungsi adalah reusabilitas. Ini berarti kita dapat menulis kode sekali, dan kemudian menggunakan kembali kode tersebut berkali-kali di berbagai bagian program. Ini membantu menghemat waktu dan upaya, serta meningkatkan efisiensi dalam pengembangan perangkat lunak.

Pentingnya Fungsi:

1. Pemisahan Tugas: Fungsi memungkinkan kita untuk memisahkan tugas yang berbeda ke dalam unit yang terpisah. Ini membantu menjaga kode tetap terorganisir dan mudah diatur.
2. Kemudahan Pembacaan: Dengan memisahkan tugas ke dalam fungsi-fungsi terpisah, kode

menjadi lebih mudah dibaca dan dimengerti. Kita dapat fokus pada logika di setiap fungsi tanpa harus khawatir tentang keseluruhan program.

3. Penggunaan Kembali Kode: Fungsi memungkinkan kita untuk menulis kode sekali dan menggunakannya di banyak bagian program. Ini menghindari pengulangan kode yang tidak perlu.

Keuntungan dari Fungsi dalam C

Fungsi dalam bahasa C merupakan fitur yang sangat berguna dengan banyak keuntungan seperti disebutkan di bawah ini:

- Fungsi dapat mengurangi pengulangan pernyataan yang sama dalam program.
- Fungsi membuat kode lebih mudah dibaca dengan memberikan modularitas pada program kita.
- Tidak ada batasan tetap pada jumlah pemanggilan fungsi; fungsi dapat dipanggil sebanyak yang diinginkan.
- Fungsi mengurangi ukuran program.
- Setelah fungsi dideklarasikan, Anda dapat menggunakannya tanpa perlu memikirkan tentang cara kerja internal dari fungsi tersebut.

Kerugian dari Fungsi dalam C

Berikut adalah kerugian utama dari fungsi dalam bahasa C:

- Tidak dapat mengembalikan beberapa nilai sekaligus.
- Memerlukan memori dan waktu tambahan karena alokasi stack frame dan pengalihan kontrol program.

FUNCTION - FUNGSI YANG TELAH DITENTUKAN

Misalnya, `main()` adalah sebuah fungsi, yang digunakan untuk mengeksekusi code, dan `printf()` merupakan sebuah fungsi yang digunakan untuk menampilkan/mencetak teks ke layar.

Fungsi pustaka juga disebut sebagai "fungsi bawaan". Paket kompiler sudah memiliki fungsi-fungsi ini, yang masing-masing memiliki arti tertentu dan sudah termasuk dalam paket tersebut. Fungsi bawaan memiliki keuntungan dapat langsung digunakan tanpa perlu didefinisikan, sedangkan fungsi yang didefinisikan oleh pengguna harus dideklarasikan dan didefinisikan sebelum digunakan.

Contoh:

- `pow()`
- `sqrt()`
- `strcmp()`
- `strcpy()`

Keuntungan Fungsi Pustaka C

- Fungsi pustaka C mudah digunakan dan dioptimalkan untuk kinerja yang lebih baik.
- Fungsi pustaka C menghemat banyak waktu pengembangan fungsi.
- Fungsi pustaka C praktis karena selalu berfungsi dengan baik.

Contohnya seperti ini:

```
int main() {  
    printf("Hello World!");  
    return 0;  
}
```

FUNCTION – MEMBUAT FUNGSI

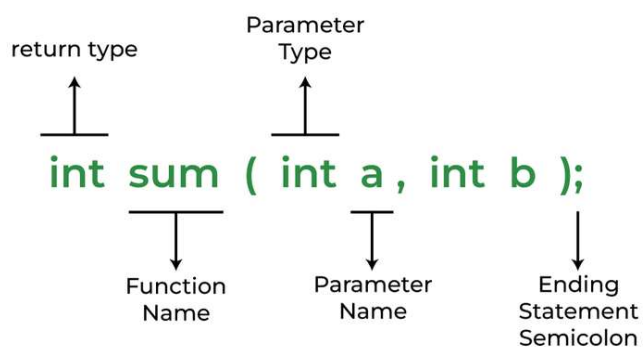
Fungsi yang dibuat oleh pemrogram dikenal sebagai fungsi yang didefinisikan pengguna atau "fungsi khusus". Fungsi ini dapat dimodifikasi sesuai kebutuhan pemrogram. Setiap kali kita menulis fungsi yang spesifik untuk kasus tertentu dan tidak didefinisikan dalam file header apa pun, kita perlu

mendeklarasikan dan mendefinisikan fungsi kita sendiri sesuai dengan sintaksis yang ada.

Keuntungan dari Fungsi yang Didefinisikan Pengguna

- Dapat Diubah: Fungsi ini bisa dimodifikasi sesuai kebutuhan.
- Kode Dapat Digunakan Kembali: Kode dari fungsi ini dapat digunakan kembali dalam program lain.
- Mudah Dipahami: Fungsi ini mudah dipahami, debug, dan dipelihara.

Untuk membuat (sering disebut sebagai declare) fungsi kalian sendiri, tentukan nama fungsinya, diikuti dengan tanda kurung () dan kurung kurawal { }:

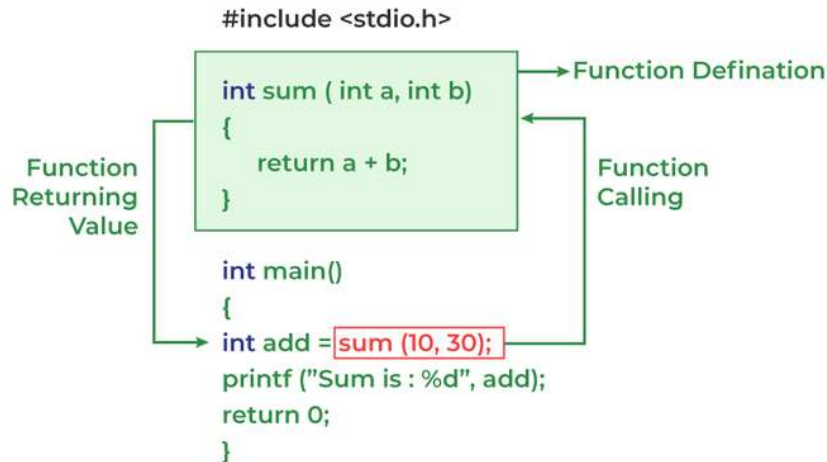


```
void myFunction() {
    // kode yang akan dieksekusi
}
```

myFunction() adalah nama fungsinya. **Void** berarti bahwa fungsi tersebut tidak memiliki nilai kembalian, kemudian tambahkan kode yang menentukan apa yang harus dilakukan fungsi.

FUNCTION – MEMANGGIL FUNGSI

Working of Function in C



Fungsi yang dideklarasikan tidak segera dieksekusi. Fungsi tersebut akan "disimpan untuk digunakan nanti" dan akan dieksekusi saat dipanggil. Untuk memanggil fungsi, tulis nama fungsi nya, kemudian diikuti dengan dua tanda kurung `()` dan titik koma `;`. Dalam contoh berikut, `myFunction()` digunakan untuk mencetak teks (aksi), ketika dipanggil:

```

// Buat fungsi
void myFunction() {
    printf("I just got executed!");
}

int main() {
    myFunction(); // memanggil fungsi
    return 0;
}

// Output "I just got executed!"

```

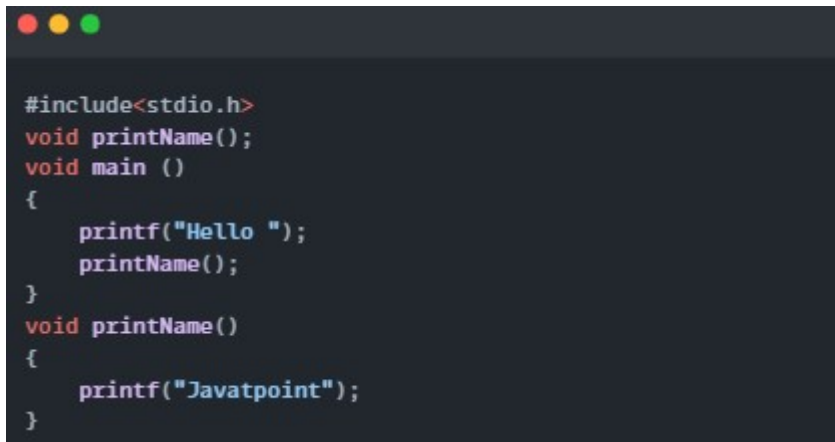
Jadi, pada contoh program di atas, `void myFunction() {`, adalah deklarasi fungsi `myFunction`. Fungsi ini bertipe `void`, yang artinya fungsi tidak mengembalikan nilai apapun. Fungsi ini tidak menerima parameter `()` karena parameter-nya kosong). Kemudian `printf("I just got executed!");`, ini fungsi `printf` akan mencetak pesan "I just got executed!" ke layar. Lalu, `myFunction();`, ini kita memanggil fungsi `myFunction` yang telah kita deklarasikan di atas. Ketika program mencapai pernyataan ini, maka fungsi `myFunction` akan dieksekusi.

Aspek-aspek yang berbeda dari pemanggilan fungsi

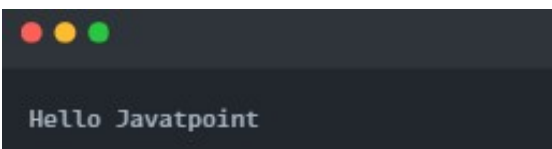
Sebuah fungsi dapat menerima atau tidak menerima argumen apa pun. Fungsi tersebut mungkin atau mungkin tidak mengembalikan nilai apa pun. Berdasarkan fakta-fakta ini, ada empat aspek pemanggilan fungsi yang berbeda.

- Fungsi tanpa argumen dan tanpa return value
- Fungsi tanpa argumen dan dengan return value
- Fungsi dengan argumen dan tanpa return value
- Fungsi dengan argumen dan dengan return value

Contoh Fungsi tanpa argumen dan tanpa return value



```
#include<stdio.h>
void printName();
void main ()
{
    printf("Hello ");
    printName();
}
void printName()
{
    printf("Javatpoint");
}
```



```
Hello Javatpoint
```

Program ini contoh penggunaan fungsi dalam bahasa C yang tidak menerima argumen dan tidak mengembalikan nilai. Program ini terdiri dari dua fungsi: `main` dan `printName`. Fungsi `printName` dideklarasikan di awal program dengan tanda tangan `void printName();`, menandakan bahwa fungsi ini tidak menerima parameter dan tidak mengembalikan nilai (`void`). Dalam fungsi `main`, program pertama-tama mencetak "Hello " ke layar menggunakan `printf("Hello ");`. Setelah itu, fungsi `printName` dipanggil tanpa argumen dengan `printName();`. Ketika fungsi `printName` dieksekusi, ia mencetak "Javatpoint" ke layar dengan `printf("Javatpoint");`. Dengan demikian, output lengkap dari program ini adalah "Hello Javatpoint", di mana "Hello " dicetak dari `main`, dan "Javatpoint" dicetak dari `printName`. Fungsi `main` adalah titik awal eksekusi program, dan `printName` hanya berfungsi untuk menambahkan nama ke output tanpa mengembalikan nilai apapun.

Contoh Fungsi tanpa argumen dan dengan return value

```
#include<stdio.h>
int sum();
void main()
{
    printf("Going to calculate the area of the square\n");
    float area = square();
    printf("The area of the square: %f\n",area);
}
int square()
{
    float side;
    printf("Enter the length of the side in meters: ");
    scanf("%f",&side);
    return side * side;
}
```

```
Going to calculate the area of the square
Enter the length of the side in meters: 10
The area of the square: 100.000000
```

Program ini menunjukkan penggunaan fungsi dalam bahasa C yang tidak menerima argumen tetapi mengembalikan nilai. Terdapat dua fungsi dalam program ini: `main` dan `square`. Fungsi `square` dideklarasikan dengan tipe pengembalian `int`, yang seharusnya diganti dengan `float` agar sesuai dengan nilai yang dikembalikan. Fungsi `main` adalah titik awal eksekusi program dan mencetak pesan ke layar, "Going to calculate the area of the square". Kemudian, fungsi `square` dipanggil, yang meminta pengguna untuk memasukkan panjang sisi persegi dalam meter menggunakan `scanf`. Nilai sisi yang dimasukkan oleh pengguna digunakan untuk menghitung luas persegi dengan mengalikan sisi dengan dirinya sendiri (`side * side`). Hasil perhitungan ini dikembalikan sebagai nilai dari fungsi `square`. Kembali ke fungsi `main`, nilai yang dikembalikan oleh `square` disimpan dalam variabel `area`, dan kemudian program mencetak luas persegi dengan pesan, "The area of the square: ", diikuti oleh nilai `area`. Fungsi `square` mengilustrasikan bagaimana input dari pengguna dapat diproses dan hasilnya dikembalikan ke tempat pemanggil untuk digunakan lebih lanjut dalam program.

Contoh Fungsi dengan argumen dan tanpa return value

```
#include<stdio.h>
void sum(int, int);
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum of two numbers:");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    sum(a,b);
}
void sum(int a, int b)
{
    printf("\nThe sum is %d",a+b);
}
```

```
Going to calculate the sum of two numbers:

Enter two numbers 10
24

The sum is 34
```

Program di atas menunjukkan penggunaan fungsi dalam bahasa C yang menerima argumen tetapi tidak mengembalikan nilai. Terdapat dua fungsi dalam program ini: `main` dan `sum`. Fungsi `sum` dideklarasikan untuk menerima dua argumen bertipe `int`. Fungsi `main` adalah titik awal eksekusi program dan dimulai dengan mencetak pesan ke layar, "Going to calculate the sum of two numbers:". Program kemudian meminta pengguna untuk memasukkan dua bilangan bulat melalui `scanf`, yang disimpan dalam variabel `a` dan `b`. Setelah menerima input dari pengguna, fungsi `sum` dipanggil dengan `sum(a, b)`, dan dua bilangan yang dimasukkan dilewatkan sebagai argumen. Fungsi `sum` menerima dua parameter `a` dan `b`, dan mencetak hasil penjumlahan keduanya dengan `printf("\nThe sum is %d", a + b);`. Fungsi `sum` tidak mengembalikan nilai, melainkan menampilkan hasil langsung ke layar. Program ini mengilustrasikan cara melewatkan nilai ke fungsi untuk diproses tanpa perlu mengembalikan hasil ke tempat pemanggil, karena hasilnya langsung dicetak oleh fungsi `sum`.

Contoh Fungsi dengan argumen dan dengan return value

```
#include<stdio.h>
int even_odd(int);
void main()
{
    int n,flag=0;
    printf("\nGoing to check whether a number is even or odd");
    printf("\nEnter the number: ");
    scanf("%d",&n);
    flag = even_odd(n);
    if(flag == 0)
    {
        printf("\nThe number is odd");
    }
    else
    {
        printf("\nThe number is even");
    }
}
int even_odd(int n)
{
    if(n%2 == 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

```
Going to check whether a number is even or odd
Enter the number: 100
The number is even
```

Program di atas merupakan contoh penggunaan fungsi dalam bahasa C yang menerima argumen dan mengembalikan nilai. Program ini bertujuan untuk menentukan apakah sebuah bilangan bulat adalah genap atau ganjil. Terdapat dua fungsi dalam program ini: `main` dan `even_odd`. Fungsi `even_odd` dideklarasikan untuk menerima satu argumen bertipe `int` dan mengembalikan nilai `int`. Dalam fungsi `main`, program dimulai dengan mendeklarasikan variabel `n` untuk menyimpan bilangan yang dimasukkan pengguna dan `flag` untuk menyimpan hasil dari fungsi `even_odd`. Program meminta pengguna untuk memasukkan bilangan bulat melalui `scanf`, yang disimpan dalam variabel `n`.

Agustus, 2024 [pemrograman dasar] 12

Fungsi `even_odd` dipanggil dengan `even_odd(n)`, di mana `n` adalah bilangan yang dimasukkan pengguna. Fungsi `even_odd` memeriksa apakah bilangan tersebut genap atau ganjil dengan menggunakan operasi modulus `n % 2`. Jika hasil modulus adalah 0, bilangan tersebut genap, dan fungsi mengembalikan `1`; jika tidak, fungsi mengembalikan `0`. Kembali ke fungsi `main`, nilai yang dikembalikan disimpan dalam `flag`. Jika `flag` bernilai `0`, program mencetak "The number is odd"; jika tidak, program mencetak "The number is even". Dengan demikian, program ini mengilustrasikan cara menggunakan fungsi untuk menerima input, memprosesnya, dan mengembalikan hasil untuk menentukan apakah sebuah bilangan genap atau ganjil.

FUNCTION – PARAMETER DAN ARGUMEN FUNGSI

Dalam bahasa pemrograman, fungsi dapat menerima informasi melalui parameter. Parameter berfungsi sebagai variabel yang digunakan di dalam fungsi. Untuk menentukan parameter, kalian hanya perlu menuliskannya di antara tanda kurung setelah nama fungsi, dan jika kalian membutuhkan lebih dari satu parameter, kalian dapat menambahkannya dengan memisahkannya menggunakan koma. Dengan cara ini, kalian dapat memberikan data atau nilai yang diperlukan saat memanggil fungsi, sehingga fungsi tersebut dapat bekerja sesuai dengan informasi yang diberikan.

```
#include <stdio.h>
int sum(int a, int b)
{
    return a + b;
}

int main()
{
    int add = sum(10, 30);
    printf("Sum is: %d", add);
    return 0;
}
```

Diagram illustrating the function call and parameters:

- Formal Parameter:** Points to the parameters `int a, int b` in the function definition `int sum(int a, int b)`.
- Actual Parameter:** Points to the arguments `10, 30` in the function call `int add = sum(10, 30);`.

Syntax dasar function dengan parameter

```
returnType functionName(parameter1, parameter2, parameter3) {
    // kode yang akan dieksekusi
}
```

Berikut ini adalah contoh fungsi yang mengambil serangkaian karakter dengan **nama** sebagai parameter. Saat fungsi dipanggil nanti, kami menjadikan **nama** yang digunakan di dalam fungsi untuk mencetak "Halo" dan nama setiap orang.

```
void myFunction(char nama[]) {  
    printf("Hallow %s\n", nama);  
}  
  
int main() {  
    myFunction("Annisa");  
    myFunction("Artanti");  
    myFunction("Widyadhana");  
    return 0;  
}  
  
// Hallow Annisa  
// Hallow Artanti  
// Hallow Widyadhana
```

```
Hallo Annisa  
Hallo Artanti  
Hallo Widyadhana  
  
=== Code Execution Successful ===
```

Program di atas mendemonstrasikan penggunaan fungsi **myFunction** yang menerima sebuah string (serangkaian karakter) sebagai parameter. Di dalam fungsi **myFunction**, string tersebut digunakan untuk mencetak pesan "**Hallow**" yang diikuti dengan nama yang diberikan sebagai parameter.

Pada bagian **void myFunction(char nama[]) {**, ini adalah definisi fungsi **myFunction**. Fungsi ini memiliki tipe pengembalian **void**, yang berarti fungsi ini tidak mengembalikan nilai apapun. Parameter fungsi ini ditentukan sebagai array karakter **nama[]**, yang akan digunakan untuk menerima nama sebagai input, dan perlu diperhatikan, jika String menjadi sebuah parameter tidak perlu memberikan batas di dalam kurung siku "**[]**".

Kemudian, **printf("Hallow %s\n", nama);**, pada bagian ini, fungsi **printf** digunakan untuk mencetak pesan "**Hallow**" diikuti dengan nama yang diberikan sebagai argumen di dalam array **nama[]**. Setelah itu, **%s** adalah format specifier yang akan digantikan oleh nilai dari variabel **nama[]**.

Dari semua proses di atas, pemanggilan fungsi **myFunction** dengan argumen "**Annisa**" akan mencetak "**Hallow Annisa**" di layar. Pemanggilan fungsi argumen "**Artanti**" akan mencetak "**Hallow Artanti**" di layar. Pemanggilan fungsi dengan argumen "**Widyadhana**" akan mencetak "**Hallow Widyadhana**" di layar.

Ada dua cara untuk mengirim argumen ke fungsi dalam C:

1. **Pass by Value**
2. **Pass by Reference**

Pass by Value

Dalam metode ini, nilai dari parameter yang sebenarnya (di luar fungsi) disalin ke parameter fungsi. Akibatnya, perubahan yang dibuat di dalam fungsi tidak mempengaruhi nilai parameter di luar fungsi.

Contoh:

```
#include <stdio.h>

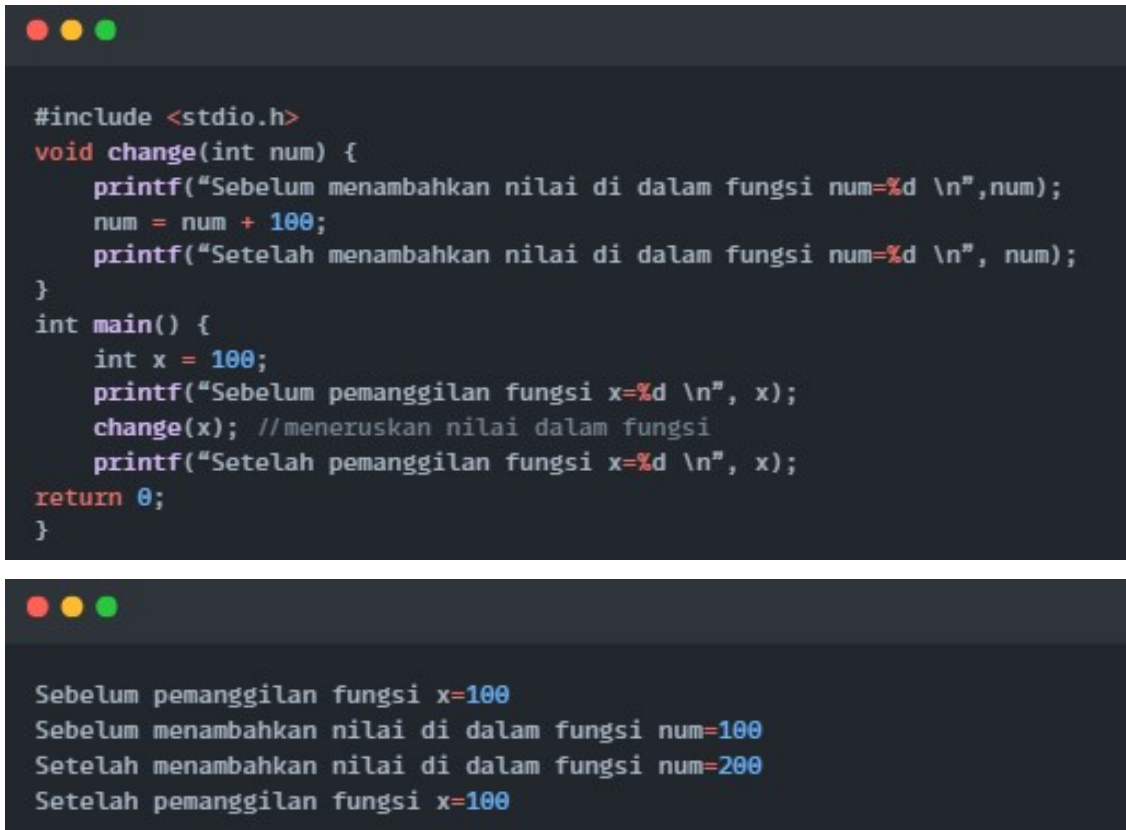
void swap(int var1, int var2)
{
    int temp = var1;
    var1 = var2;
    var2 = temp;
}

// Driver code
int main()
{
    int var1 = 3, var2 = 2;
    printf("Before swap Value of var1 and var2 is: %d, %d\n",
           var1, var2);
    swap(var1, var2);
    printf("After swap Value of var1 and var2 is: %d, %d",
           var1, var2);
    return 0;
}
```

```
Before swap Value of var1 and var2 is: 3, 2
After swap Value of var1 and var2 is: 3, 2

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


Contoh lain:



```
#include <stdio.h>
void change(int num) {
    printf("Sebelum menambahkan nilai di dalam fungsi num=%d \n", num);
    num = num + 100;
    printf("Setelah menambahkan nilai di dalam fungsi num=%d \n", num);
}
int main() {
    int x = 100;
    printf("Sebelum pemanggilan fungsi x=%d \n", x);
    change(x); //meneruskan nilai dalam fungsi
    printf("Setelah pemanggilan fungsi x=%d \n", x);
    return 0;
}
```

```
Sebelum pemanggilan fungsi x=100
Sebelum menambahkan nilai di dalam fungsi num=100
Setelah menambahkan nilai di dalam fungsi num=200
Setelah pemanggilan fungsi x=100
```

Program di atas menunjukkan penggunaan konsep “pass by value” dalam bahasa C. Program ini terdiri dari dua fungsi: `main` dan `change`. Fungsi `change` menerima satu argumen bertipe `int` bernama `num` dan mencetak nilai `num` sebelum dan setelah penambahan 100. Dalam fungsi `main`, variabel `x` diinisialisasi dengan nilai 100. Program mencetak nilai `x` sebelum memanggil fungsi `change`. Kemudian, `change(x)` dipanggil, di mana nilai `x` (100) diteruskan ke fungsi `change`. Di dalam `change`, nilai `num` awalnya adalah 100, dan setelah ditambahkan 100, `num` menjadi 200. Namun, karena bahasa C menggunakan call by value, perubahan ini hanya terjadi pada salinan dari `x` yang ada dalam fungsi `change`, sehingga tidak memengaruhi nilai asli `x` di luar fungsi. Setelah fungsi `change` selesai, program kembali ke fungsi `main` dan mencetak nilai `x` lagi, yang tetap 100. Ini mengilustrasikan bahwa call by value bekerja dengan salinan dari nilai argumen yang diterima oleh fungsi, bukan dengan nilai asli itu sendiri.

Pass by Reference

Dalam metode ini, parameter yang sebenarnya (di luar fungsi) dan parameter fungsi merujuk ke lokasi memori yang sama. Jadi, perubahan yang dibuat di dalam fungsi akan mempengaruhi nilai parameter di luar fungsi.

Contoh:

```
#include <stdio.h>

void swap(int *var1, int *var2)
{
    int temp = *var1;
    *var1 = *var2;
    *var2 = temp;
}

// Driver code
int main()
{
    int var1 = 3, var2 = 2;
    printf("Before swap Value of var1 and var2 is: %d, %d\n",
           var1, var2);
    swap(&var1, &var2);
    printf("After swap Value of var1 and var2 is: %d, %d",
           var1, var2);
    return 0;
}
```

```
Before swap Value of var1 and var2 is: 3, 2
After swap Value of var1 and var2 is: 2, 3
```

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

Contoh lain:

```
#include <stdio.h>
void change(int *num) {
    printf("Sebelum menambahkan nilai di dalam fungsi num=%d \n", *num);
    (*num) += 100;
    printf("Setelah menambahkan nilai di dalam fungsi num=%d \n", *num);
}
int main() {
    int x = 100;
    printf("Sebelum pemanggilan fungsi x=%d \n", x);
    change(&x); //melewatkan referensi dalam fungsi
    printf("Setelah pemanggilan fungsi x=%d \n", x);
    return 0;
}
```

```
Sebelum pemanggilan fungsi x=100
Sebelum menambahkan nilai di dalam fungsi num=100
Setelah menambahkan nilai di dalam fungsi num=200
Setelah pemanggilan fungsi x=200
```

Program di atas mengimplementasikan penggunaan “pass by reference” dalam bahasa C melalui penggunaan pointer. Program ini terdiri dari dua fungsi: `main` dan `change`. Fungsi `change` menerima satu argumen berupa pointer bertipe `int`, yang memungkinkan fungsi ini mengubah nilai asli dari variabel yang dilewatkan. Di dalam fungsi `main`, variabel `x` diinisialisasi dengan nilai 100. Program mencetak nilai `x` sebelum memanggil fungsi `change`. Ketika `change(&x)` dipanggil, alamat dari `x` dilewatkan ke fungsi `change`. Dalam fungsi `change`, pointer `num` digunakan untuk mengakses dan memodifikasi nilai `x` secara langsung. Sebelum menambahkan nilai, fungsi mencetak nilai asli `x` melalui dereferensi pointer `*num`. Fungsi kemudian menambahkan 100 ke nilai yang ditunjuk oleh `num` dengan `(*num) += 100`, yang secara langsung mengubah nilai `x` di `main`. Setelah penambahan, fungsi `change` mencetak nilai baru `x`. Kembali ke fungsi `main`, nilai `x` sekarang telah berubah menjadi 200, dan ini dicetak setelah pemanggilan fungsi `change`. Program ini mengilustrasikan bagaimana penggunaan pointer memungkinkan fungsi untuk mengubah nilai variabel asli dengan memanipulasi alamatnya, berbeda dengan call by value yang hanya bekerja dengan salinan dari nilai tersebut.

FUNCTION – MULTIPLE PARAMETERS

Sebuah fungsi dapat memiliki lebih dari satu parameter. Multiple parameters (parameter ganda) memungkinkan kita untuk mengirimkan beberapa nilai atau informasi ke dalam fungsi untuk diproses atau digunakan di dalamnya. Dengan menggunakan multiple parameters, kita dapat memberikan data yang lebih lengkap dan spesifik saat memanggil fungsi, sehingga fungsi dapat beroperasi dengan lebih fleksibel dan efisien.

Untuk mendefinisikan multiple parameters dalam suatu fungsi, kita hanya perlu menyatakan tipe data dan nama parameter secara berurutan di dalam tanda kurung setelah nama fungsi. Jika ada lebih dari satu parameter, kita cukup memisahkan mereka dengan tanda koma Contohnya seperti ini:

```
void myFunction(char nama[], int umur) {
    printf("Hallow %s. Kamu Berumur %d Tahun.\n", nama, umur);
}

int main() {
    myFunction("Annisa", 17);
    myFunction("Artanti", 18);
    myFunction("Widyadhana", 19);
    return 0;
}

// Hallow Annisa. Kamu berumur 17 Tahun.
// Hallow Artanti. Kamu berumur 18 Tahun.
// Hallow Widyadhana. Kamu berumur 19 Tahun.
```

Program di atas merupakan contoh penggunaan fungsi dengan multiple parameters. Fungsi **myFunction** memiliki dua parameter, yaitu **nama[]** yang bertipe karakter (string) dan **umur** yang bertipe integer. Ketika fungsi ini dipanggil di dalam fungsi **main()**, kita memberikan argumen berupa string nama dan nilai umur.

Pada bagian `void myFunction(char nama[], int umur)` { adalah definisi fungsi **myFunction**. Fungsi ini memiliki dua parameter, yaitu **nama[]** bertipe karakter (string) dan **umur** bertipe integer. Kedua parameter ini akan digunakan di dalam fungsi untuk mencetak pesan yang memuat nama dan umur yang diberikan sebagai argumen. `Printf("Hallow %s. Kamu Berumur %d Tahun.\n", nama, umur);` ini, fungsi `printf` akan mencetak pesan "Hallow " diikuti dengan nama yang diberikan sebagai argumen **nama**, dan "Kamu Berumur" diikuti dengan umur yang diberikan sebagai argumen **umur**. Lalu `%s` dan `%d` adalah format specifier yang akan digantikan oleh nilai dari **nama** dan **umur** masing-masing.

FUNCTION – ARRAY SEBAGAI PARAMETER FUNGSI

```
void myFunction(int myNumbers[5]) {  
    for (int i = 0; i < 5; i++) {  
        printf("%d\n", myNumbers[i]);  
    }  
}  
  
int main() {  
    int myNumbers[5] = {10, 20, 30, 40, 50};  
    myFunction(myNumbers);  
    return 0;  
}
```

Fungsi ini menerima sebuah array **myNumbers** bertipe integer sebagai parameter. Parameter tersebut ditentukan sebagai **int myNumbers[5]** (*Jangan lupa yaa, untuk Array yang dijadikan sebuah parameter bisa menggunakan tanpa batas jumlah Array*), yang berarti fungsi ini menerima array dengan panjang tetap sebanyak 5 elemen. Kemudian, **for (int i = 0; i < 5; i++) { ... }** adalah loop for yang akan berjalan sebanyak 5 kali, sesuai dengan panjang array **myNumbers**. Pada bagian **printf("%d\n", myNumbers[i]);**, setiap iterasi loop, fungsi **printf** akan mencetak nilai dari elemen array **myNumbers** pada indeks **i**. Dengan demikian, program akan mencetak setiap elemen array **myNumbers** dalam baris terpisah.

Pada bagian **int myNumbers[5] = {10, 20, 30, 40, 50};** mendeklarasikan dan inisialisasi array **myNumbers** dengan panjang 5 dan nilai-nilai 10, 20, 30, 40, dan 50. **MyFunction(myNumbers);** adalah pemanggilan fungsi **myFunction** dengan argumen **myNumbers**. Array **myNumbers** akan disalin ke dalam parameter fungsi **myNumbers[5]** dan selanjutnya setiap elemen array akan dicetak oleh fungsi **myFunction**.

FUNCTION – PENGEMBALIAN NILAI

Pada pemrograman, sebuah fungsi dapat mengembalikan nilai sebagai hasil dari operasi yang dilakukan di dalamnya. Sebelumnya, kita menggunakan kata kunci **void** untuk menandakan bahwa fungsi tidak mengembalikan nilai apapun. Namun, jika kita ingin fungsi mengembalikan nilai, kita dapat menggunakan tipe data seperti **int**, **float**, atau **tipe data lainnya** sebagai tipe pengembalian fungsi. Selanjutnya, kita menggunakan kata kunci **return** di dalam fungsi untuk mengembalikan nilai tersebut.

Dengan kata lain, jika sebuah fungsi memiliki tipe pengembalian selain **void**, itu berarti fungsi tersebut akan menghasilkan nilai kembali saat dijalankan, dan kita menggunakan **return** untuk mengirimkan nilai tersebut kembali ke pemanggil fungsi. Dengan menggunakan pendekatan ini, kita dapat membuat fungsi yang berfungsi sebagai "mesin hitung" yang mengolah data dan memberikan hasilnya sebagai output.

```
int myFunction(int x) {  
    return 5 + x;  
}  
  
int main() {  
    printf("Result is: %d", myFunction(3));  
    return 0;  
}  
  
// Outputs 8 (5 + 3)
```

Pada program di atas, terdapat fungsi **myFunction** yang mengambil satu parameter bertipe integer **x**. Fungsi ini mengembalikan hasil dari operasi **5 + x** menggunakan pernyataan **return**. Kemudian, **int myFunction(int x) {** adalah definisi fungsi **myFunction** yang menerima satu parameter bertipe integer **x**. Fungsi ini akan mengembalikan hasil perhitungan.

Pada **return 5 + x;** yang berada di dalam fungsi, dilakukan operasi penambahan antara angka **5** dan nilai **x**. Hasil penambahan tersebut kemudian dikembalikan sebagai nilai hasil dari fungsi menggunakan pernyataan **return**.

Pada `printf("Result is: %d", myFunction(3));` di dalam fungsi `main()`, kita memanggil fungsi `myFunction` dengan argumen `3`. Hasil dari operasi $5 + 3$, yaitu `8`, akan dikembalikan oleh fungsi `myFunction` dan kemudian dicetak menggunakan `printf` dalam kalimat `"Result is: 8"`.

FUNCTION – DEKLARASI FUNGSI

Deklarasi fungsi adalah cara untuk memberi tahu komputer tentang nama fungsi dan jenis masukan yang diterimanya. Ini seperti menyebutkan judul resep dan bahan yang diperlukan sebelum kita menuliskan langkah-langkah rinci. Definisi fungsi, di sisi lain, adalah langkah-langkah rinci dalam resep itu sendiri. Di sinilah kita menuliskan instruksi tentang apa yang harus dilakukan oleh fungsi saat dipanggil.

Untuk pengoptimalan code, disarankan untuk memisahkan deklarasi dan definisi fungsi. Kalian akan sering melihat program C yang memiliki deklarasi fungsi di atas `main()`, dan definisi fungsi di bawah `main()`. Ini akan membuat code lebih terorganisir dan lebih mudah dibaca, misalnya seperti di bawah ini:

```
// Deklarasi Fungsi
void myFunction();

// Main Method
int main() {
    myFunction(); // Pemanggilan Fungsi
    return 0;
}

// Definisi Fungsi
void myFunction() {
    printf("I just got executed!");
}
```

Deklarasi Fungsi:

Deklarasi fungsi `myFunction()` diberikan di awal kode sebelum fungsi `main()`. Deklarasi ini memberi tahu compiler bahwa ada suatu fungsi bernama `myFunction`, tetapi implementasi lengkap dari fungsi ini akan diberikan nanti dalam code.

Dalam deklarasi ini, kita menyebutkan bahwa fungsi **myFunction()** adalah sebuah fungsi yang tidak mengembalikan nilai (**void**) dan tidak menerima parameter. Jadi, ketika fungsi ini dipanggil, ia akan melakukan beberapa tugas tanpa mengembalikan nilai apa pun.

Definisi Fungsi:

Definisi fungsi **myFunction()** diberikan setelah fungsi **main()**. Ini adalah bagian dari code di mana kita memberikan implementasi lengkap dari fungsi **myFunction()**. Di dalam blok code fungsi ini, kita memberikan instruksi tentang apa yang harus dilakukan ketika fungsi **myFunction()** dipanggil.

Dalam definisi ini, fungsi **myFunction()** tidak mengambil argumen apa pun (karena tidak ada parameter dalam definisi). Ketika fungsi ini dipanggil, ia akan mencetak pesan "**I just got executed!**" ke layar.

FUNCTION – REKURSIF

Rekursif adalah teknik membuat pemanggilan fungsi itu sendiri. Teknik ini menyediakan cara untuk memecah masalah rumit menjadi masalah sederhana yang lebih mudah dipecahkan. Rekursif mungkin agak sulit untuk dipahami. Cara terbaik untuk mengetahui cara kerjanya adalah dengan bereksperimen langsung nih. Perhatikan contoh rekursif di bawah ini:

```
int nSum(int n)
{
    if (n==0) {
        return 0;
    }

    int res = n+ nsum(n-1);
    return res;
}
```

Base condition

Recursive case


```

int sum(int k);

int main() {
    int result = sum(10);
    printf("%d", result);
    return 0;
}

int sum(int k) {
    if (k > 0) {
        return k + sum(k - 1);
    } else {
        return 0;
    }
}

```

Saat **sum()** fungsi dipanggil, ia menambahkan parameter **k** ke jumlah semua angka yang lebih kecil dari **k** dan mengembalikan hasilnya. Saat **k** menjadi **0**, fungsi hanya mengembalikan 0. Saat dijalankan, program mengikuti langkah-langkah berikut:

```

10 + jumlah(9)
10 + ( 9 + jumlah(8) )
10 + ( 9 + ( 8 + jumlah(7) ) )
...
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + jumlah(0)
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0

```

Karena fungsi tidak memanggil dirinya sendiri saat $k = 0$, program berhenti di situ dan mengembalikan hasilnya.

Kalian harus sangat berhati-hati dengan rekursif, karena dapat dengan mudah menyelinap ke dalam penulisan fungsi yang tidak pernah berakhir, atau fungsi yang menggunakan memori atau daya prosesor dalam jumlah berlebih. Namun, ketika ditulis dengan benar, rekursif dapat menjadi pendekatan pemrograman yang sangat efisien dan elegan secara matematis.

FILES

Dalam bahasa C, kalian bisa melakukan operasi *create*, *read*, *write*, dan *close* pada suatu file. Mengapa files perlu digunakan? Karena dengan menggunakan files, kita bisa melakukan beberapa operasi yaitu :

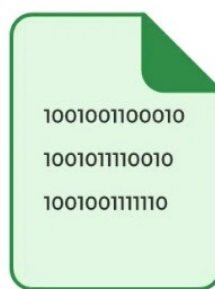
- Membaca data dari bekas eksternal untuk diproses dalam program
- Menulis data dari program ke dalam berkas eksternal sebagai penyimpanan data jangka panjang
- Membuat, mengubah, atau menghapus berkas untuk menyimpan informasi aplikasi
- Mengolah data yang besar dan kompleks, yang tidak praktis disimpan dalam kode program.

Tipe files dalam bahasa C ada dua yaitu ***text files*** dan ***binary files***.

Types of Files in C



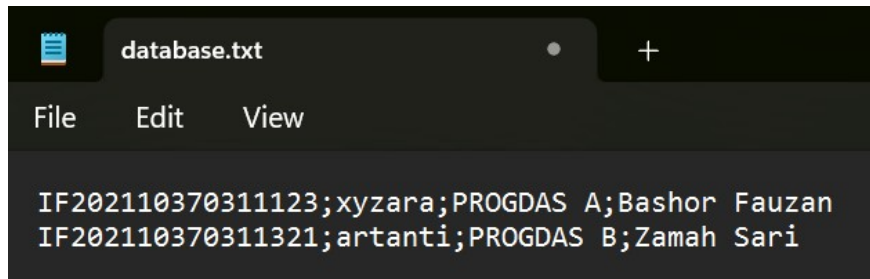
file.txt



file.bin

➤ TEXT FILES

Text files adalah jenis file yang berisi data yang disusun dalam bentuk teks atau karakter yang dapat dibaca oleh manusia. Setiap karakter dalam file diwakili oleh kode ASCII atau



```
database.txt
File Edit View
IF202110370311123;xyzara;PROGDAS A;Bashor Fauzan
IF202110370311321;artanti;PROGDAS B;Zamah Sari
```

Unicode yang sesuai. Biasanya digunakan untuk menyimpan data dalam bentuk teks sederhana seperti huruf, angka, dan symbol, seperti file dengan ekstensi **.txt**

➤ BINARY FILES

Binary files adalah jenis file yang berisi data dalam bentuk biner, yaitu kode angka biner (0 dan 1). File biner tidak dapat dibaca oleh manusia secara langsung, karena data dalam berkas ini diwakili dalam bentuk biner yang lebih kompleks dan terstruktur. File biner sering digunakan untuk menyimpan data yang lebih kompleks seperti gambar, audio, video, program eksekusi, atau struktur data yang rumit.

```
data[0]=1000000000
data[1]=1000000001
data[2]=1000000002
data[3]=1000000003
data[4]=1000000004
data[5]=1000000005
data[6]=1000000006
```

OPERASI FILE

Bahasa C menyediakan beberapa fungsi untuk melakukan operasi pada file handling, berikut fungsi-fungsi yang bisa kalian gunakan dalam kode program :

Operasi File	Deskripsi
fopen()	Membuka file dalam mode tertentu (membaca, menulis, atau menggabungkan)
fclose()	Menutup file yang telah selesai digunakan

fgets()	Membaca teks dari file dan menyimpannya ke dalam string
fputs()	Menulis baris teks ke dalam file dari string yang diberikan
fscanf()	Membaca data dari file dengan format tertentu dan menyimpannya dalam variabel yang sesuai
fprintf()	Menulis data ke dalam file dengan format tertentu dari variabel yang diberikan
fseek()	Memindahkan posisi pointer file ke posisi tertentu dalam file
ftell()	Mengembalikan posisi saat ini dari file pointer
rewind()	Mengatur penunjuk file pointer ke awal file
putw()	Menulis sebuah bilangan bulat (integer) ke dalam file
getw()	Membaca sebuah bilangan bulat (integer) dari file

Dalam membuka file menggunakan **fopen()**, kalian bisa mengaksesnya dengan menggunakan banyak mode, untuk lebih lengkapnya adalah sebagai berikut :

Mode	Deskripsi
r	membaca file dalam mode baca (text mode)
rb	membaca file dalam mode biner (binary mode)
w	menulis ke file dalam mode teks
wb	menulis ke file dalam mode biner
a	menambah data ke file
ab	menambah data ke file dalam mode biner
r+	membaca dan menulis file dalam mode baca dan tulis (text mode)

rb+	membaca dan menulis file dalam mode biner (binary mode)
w+	membaca dan menulis ke file dalam mode teks (text mode)
wb+	membaca dan menulis ke file dalam mode biner
a+	membaca dan menambah data ke file
ab+	membaca dan menambah data ke file dalam mode biner

MEMBUAT FILE

```
fopen("nama file.txt", "w") mode
```



Parameter

- **nama_file:** Nama file ketika berada di direktori yang sama dengan file sumber. Jika tidak, gunakan path lengkap.
- **mode:** Menentukan operasi apa yang akan dilakukan pada file saat dibuka.

Return Value

- Jika file berhasil dibuka, fungsi akan mengembalikan pointer ke file tersebut.
- Jika file tidak dapat dibuka, fungsi akan mengembalikan NULL.

Selain untuk membuka file yang sudah tersedia, **fopen()** juga bisa digunakan untuk membuat file baru. Namun, perlu ditambahkan format kode supaya file bisa terbuat. Contoh :

```
// C Program to create a file
#include <stdio.h>
#include <stdlib.h>

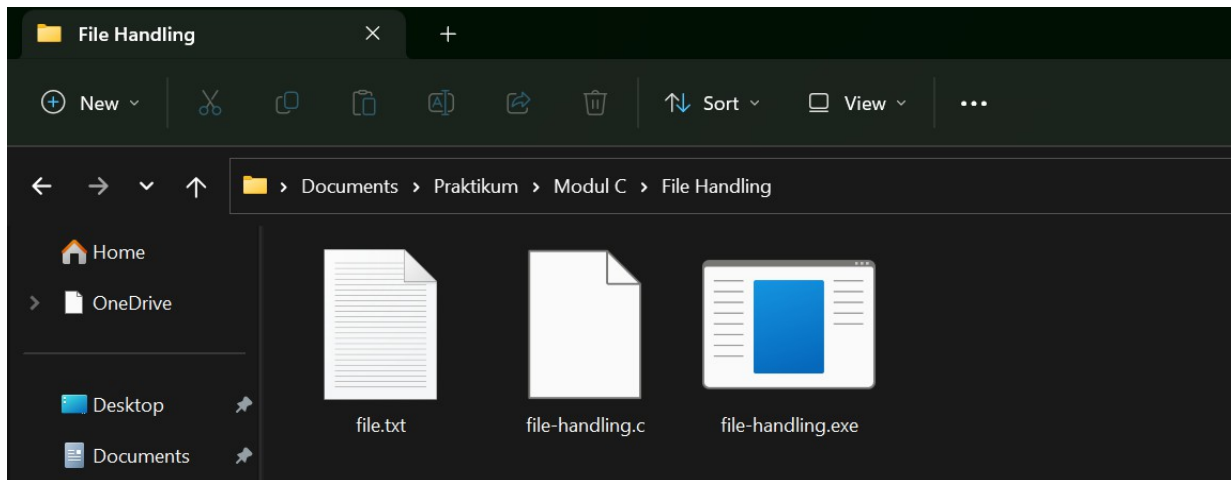
int main()
{
    // file pointer
    FILE* fptr;

    // membuat file dengan mode "w"
    fptr = fopen("file.txt", "w");

    // check apakah file berhasil dibuat
    if (fptr == NULL) {
        printf("File gagal dibuat, program akan ditutup);
        exit(0);
    }
    else {
        printf("File berhasil dibuat");
    }

    return 0;
}
```

Kode diatas akan membuat file dengan mode “w”, yaitu untuk menulis data ke dalam file dengan menggunakan teks. Silahkan kalian coba source code diatas, jika program berhasil maka akan menghasilkan output “File berhasil dibuat” dan akan terbuat file didalam folder sesuai letak dimana file kodingan kalian berada dengan ekstensi **.txt**



Cek di file explorer pada bagian folder dimana kalian menyimpan file kodingan kalian dan jika sudah ada **file.txt**, maka kalian sudah berhasil membuat filenya. Jika kalian ingin supaya file berada di folder yang lebih spesifik, kalian bisa menggunakan kode :

```
fptr = fopen("C:\\directoryname\\filename.txt", "w");
```

"C:\\directoryname\\filename.txt": Ini adalah string yang berisi alamat lengkap (full path) dari file yang ingin dibuka atau dibuat. Di sini, kita menggunakan \\ (backslash) sebagai pemisah direktori. Namun, dalam bahasa C, karakter \\ adalah karakter escape. Oleh karena itu, kita menggunakan \\ untuk merepresentasikan \\.

MENAMBAH DATA

Untuk melakukan penambahan data ke dalam file, bahasa C menyediakan dua fungsi yaitu **fprintf()** dan **fputs()**. Sebelum melakukan penambahan data, langkah pertama yang harus dilakukan adalah membuka file terlebih dahulu menggunakan fungsi **fopen()**, yang memungkinkan program untuk mengakses dan memodifikasi isi dari file. Setelah operasi pada file selesai, sangat penting untuk menutup file menggunakan fungsi **fclose()** guna menyimpan perubahan yang telah dilakukan dan membebaskan sumber daya yang digunakan untuk mengakses file.

Berikut adalah contoh kode untuk menambahkan data ke dalam file:.

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

int main()
{
    // deklarasi file pointer
    FILE* fptr;

    // membuka file dengan mode "w"
    char addData[30] = "Membuat File Handling Bahasa C";
    fptr = fopen("file.txt", "w");

    if (fptr == NULL) {
        printf("File Gagal Dibuka");
    }
    else {
        printf("File Berhasil dibuka");

        //menambahkan data ke dalam file
        if (strlen(addData)>0) {
            fputs(addData, fptr);
            fputs("\n", fptr);
        }

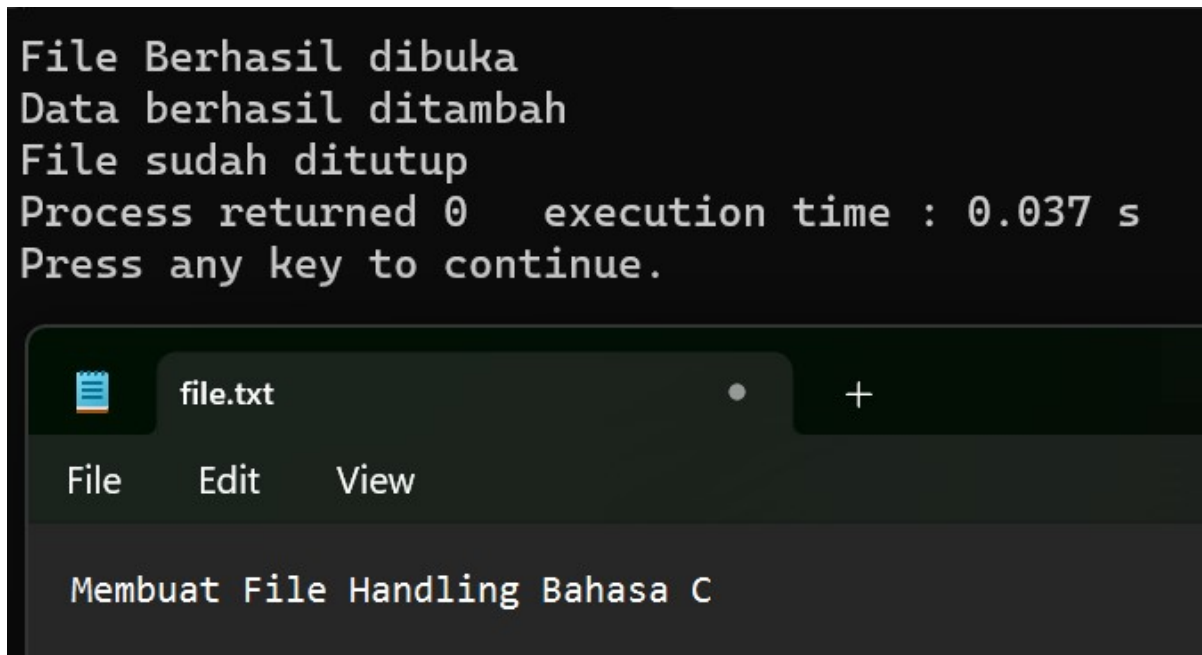
        fclose(fptr);

        printf("\nData berhasil ditambah dan file sudah
ditutup");
    }

    return 0;
}
```


Dalam contoh di atas, kita menggunakan mode 'w' saat membuka file dengan **fopen()**, yang berarti kita akan membuka file dengan tujuan untuk menulis. Jika file sudah ada, mode ini akan menghapus data yang lama dalam file tersebut atau membuat file baru jika file belum ada. Kemudian, kita menggunakan **fputs()** untuk menambahkan data teks ke dalam file dalam bentuk string.

Jika data berhasil ditambahkan, maka di dalam file.txt yang kalian buat, akan otomatis menampilkan text yang sudah kalian tambahkan.



The image shows a terminal window with the following output:

```
File Berhasil dibuka
Data berhasil ditambah
File sudah ditutup
Process returned 0    execution time : 0.037 s
Press any key to continue.
```

Below the terminal window is a text editor window titled 'file.txt'. The editor has a menu bar with 'File', 'Edit', and 'View'. The content of the file is:

```
Membuat File Handling Bahasa C
```

Apabila kalian ingin menambahkan data ke dalam file tanpa menghapus data yang lama, kalian bisa menggunakan mode “a” untuk menambahkan data. Mode “a” (*append*) digunakan ketika ingin menambahkan data ke dalam file tanpa menghapus data yang sudah ada. Jika file sudah ada, mode “a” akan membuka file tersebut dan menempatkan pointer file di akhir (*end-of-file*). Semua data yang ditulis akan ditambahkan ke akhir file tanpa menghapus data lama. Jika file belum ada, mode “a” akan menciptakan file baru.

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

int main()
{
    // file pointer
    FILE* fptr;

    // membuat file dengan mode "a"
    char addData[30] = "Informatika Menyenangkan!";
    fptr = fopen("file.txt","a");

    if (fptr == NULL) {
        printf("File Gagal Dibuka");
    }
    else {
        printf("File Berhasil dibuka");

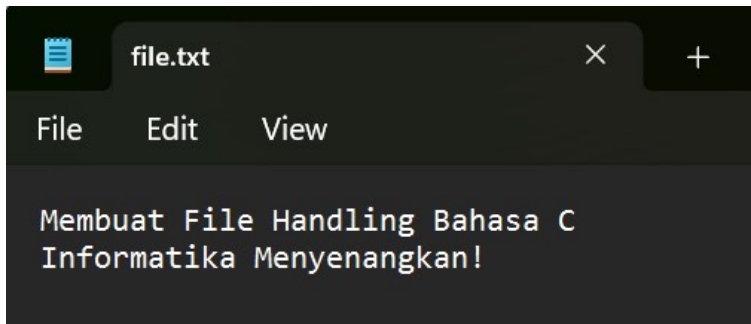
        if (strlen(addData)>0) {
            fputs(addData, fptr);
            fputs("\n", fptr);
        }

        fclose(fptr);

        printf("\nData berhasil ditambah dan file sudah
ditutup");
    }

    return 0;
}
```

Ketika program berhasil dijalankan, maka hasil output pada file.txt akan otomatis menambah data seperti berikut :



READ FILE

Kalian bisa mengambil data dari file eksternal dan memuatnya ke dalam program C. Dengan membaca file, kita dapat mengakses informasi atau data yang tersimpan dalam file untuk diproses oleh program. Kita bisa membukanya dengan menggunakan **fopen()** dan menggunakan mode **"r"**.

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

int main()
{
    FILE* fptr;

    // membuka file dengan mode "r"
    char addData[30];
    fptr = fopen("file.txt","r");

    if (fptr != NULL) {
        printf("Isi data adalah :\n");

        while (fgets(addData, sizeof(addData), fptr) != NULL) {
            printf("%s", addData);
        }

        fclose(fptr);
    }
    else {
        printf("File Gagal Dibuka");
    }
    return 0;
}
```

Pada contoh diatas, kita menggunakan **fgets()** untuk membaca isi dari sebuah file. Fungsi ini akan membaca satu baris teks dari file dan menyimpannya dalam array yang telah kita tentukan sebelumnya. parameter pertama (**addData**) digunakan untuk menyimpan konten file yang baru kita buat. Parameter kedua (**sizeof(addData)**) digunakan untuk menentukan ukuran data yang akan dibaca. Parameter ketiga (**fptr**) memerlukan pointer file yang digunakan untuk membaca file.

Jika program berhasil, maka output yang dihasilkan adalah :

```
Isi data adalah :  
Membuat File Handling Bahasa C  
Informatika Menyenangkan!  
  
Process returned 0    execution time : 0.024 s  
Press any key to continue.
```

TUGAS PRAKTIKUM

CODELAB 1

Deskripsi: Buatlah program sederhana untuk menghitung nilai faktorial dan deret Fibonacci. Gunakan fungsi rekursif untuk keduanya dan tampilkan hasilnya berdasarkan input dari user.

Detail Tugas:

- Buat fungsi rekursif untuk menghitung faktorial dari suatu angka.
- Buat fungsi rekursif untuk menghasilkan deret Fibonacci hingga suku ke-n.
- Ambil input angka dari user untuk digunakan dalam kedua fungsi tersebut.
- Tampilkan hasilnya kepada user.

Contoh output program yang diharapkan:

```
Masukkan angka untuk menghitung faktorial: 5
Faktorial dari 5 adalah: 120
Masukkan angka untuk menghasilkan deret Fibonacci: 10
Deret Fibonacci hingga suku ke-10 adalah: 0 1 1 2 3 5 8 13 21 34

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

CODELAB 2

Deskripsi: Buatlah program yang menggunakan file handling untuk menyimpan dan menampilkan data mahasiswa. Program akan menyimpan Nama, NIM, dan Program Studi dari 3 mahasiswa ke dalam file bernama data_mahasiswa.txt, kemudian membaca kembali data tersebut dari file dan menampilkannya di layar.

Detail Tugas:

- Buat fungsi untuk menulis data mahasiswa ke dalam file.
- Buat fungsi untuk membaca data mahasiswa dari file dan menampilkannya.
- Tampilkan hasil output kepada asisten masing-masing.

Contoh output program yang diharapkan

```
Pilih aksi:
1. Tulis data mahasiswa
2. Baca data mahasiswa
Masukkan pilihan: 1
Masukkan nama: Annisa
Masukkan NIM: 123456
Masukkan Program Studi: Informatika
...
Data mahasiswa berhasil ditulis ke file.

Pilih aksi:
1. Tulis data mahasiswa
2. Baca data mahasiswa
Masukkan pilihan: 2
Nama: Annisa
NIM: 123456
Program Studi: Informatika
```

KEGIATAN 1

Buatlah program Manajemen Toko Elektronik dengan fitur **Create, Read, Update, Delete (CRUD)**. Program ini akan menyimpan informasi tentang produk elektronik di toko dan menyimpannya dalam berkas **produk_toko.txt**. Implementasikan menu interaktif untuk pengguna sehingga mereka bisa melakukan operasi CRUD pada data produk.

Tugas:

1. Buat sebuah file bernama **produk_toko.txt** yang akan digunakan untuk menyimpan informasi produk.
2. Buatlah menu interaktif yang akan ditampilkan untuk pengguna dengan fitur-fitur berikut:
 - **Tambah Produk Baru:** Untuk menambahkan data jika terdapat produk baru di toko.
 - **Tampilkan Daftar Produk:** Untuk menampilkan semua data produk yang terdaftar dalam berkas.
 - **Update Informasi Produk:** Untuk memperbarui data produk yang sudah ada berdasarkan **ID Produk**.
 - **Hapus Produk:** Untuk menghapus data produk berdasarkan **ID Produk**.
 - **Keluar:** Untuk keluar dari program dan kembali ke menu awal.
3. Data yang harus disimpan untuk setiap produk meliputi:
 - **ID Produk:** Harus unik, jika ID yang sama sudah ada dalam data, tampilkan pesan "ID SUDAH TERDAFTAR", dan pengguna harus memasukkan ID produk yang berbeda.
 - **Nama Produk**
 - **Kategori Produk**
 - **Harga**
 - **Stok**
4. Pastikan program Anda memiliki penanganan error (error handling) yang baik, misalnya menangani kasus jika produk tidak ditemukan atau jika ada masalah dalam membuka atau menulis ke file.
5. Setiap kali program dimulai, tampilkan pesan selamat datang dan deskripsi singkat tentang program.

Contoh output program yang diharapkan:

Selamat datang di program Manajemen Toko Elektronik

Menu:

1. Tambah Produk Baru
2. Tampilkan Daftar Produk
3. Update Informasi Produk
4. Hapus Produk
5. Keluar

Masukkan pilihan: 1

Masukkan ID produk: 1234

Masukkan nama produk: televisi

Masukkan kategori produk: elektronik1

Masukkan harga produk: 1000000

Masukkan stok produk: 10

Produk berhasil ditambahkan!

Menu:

1. Tambah Produk Baru
2. Tampilkan Daftar Produk
3. Update Informasi Produk
4. Hapus Produk
5. Keluar

Masukkan pilihan: 2

Daftar Produk:

ID: 1234

Nama: televisi

Kategori: elektronik1

Harga: 1000000.000000

Stok: 10

Menu:

1. Tambah Produk Baru
2. Tampilkan Daftar Produk
3. Update Informasi Produk
4. Hapus Produk
5. Keluar

Masukkan pilihan: 3

Masukkan ID produk yang ingin diupdate: 1234

Masukkan nama baru: ac

Masukkan kategori baru: elektronik2

Masukkan harga baru: 2000000

Masukkan stok baru: 20

Produk berhasil diupdate!

Menu:

1. Tambah Produk Baru
2. Tampilkan Daftar Produk
3. Update Informasi Produk
4. Hapus Produk
5. Keluar

Masukkan pilihan: 2

Daftar Produk:

ID: 1234

Nama: ac

Kategori: elektronik2

Harga: 2000000.000000

Stok: 20

Menu:

1. Tambah Produk Baru
2. Tampilkan Daftar Produk
3. Update Informasi Produk
4. Hapus Produk
5. Keluar

Masukkan pilihan: 4

Masukkan ID produk yang ingin dihapus: 1234

Produk berhasil dihapus!

Menu:

1. Tambah Produk Baru
2. Tampilkan Daftar Produk
3. Update Informasi Produk
4. Hapus Produk
5. Keluar

Masukkan pilihan: 2

Daftar Produk:

Menu:

1. Tambah Produk Baru
2. Tampilkan Daftar Produk
3. Update Informasi Produk
4. Hapus Produk
5. Keluar

Masukkan pilihan: 5

Process exited after 88.61 seconds with return value 0

Press any key to continue . . . |

KRITERIA & DETAIL PENILAIAN

Kriteria	Poin
Codelab 1	10
Codelab 2	10
Kegiatan 1	40
Pemahaman	25
Ketepatan Menjawab	15