

VERSI 1.2

JULI, 2023



[PEMROGRAMAN DASAR]

MODUL 6 - FUNCTION, FILE HANDLING

DISUSUN OLEH:

- ANNISA ARTANTI WIDYADHANA

- KIARA AZZAHRA

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.

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.

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.

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. Contohnya seperti ini:

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

FUNCTION – MEMBUAT FUNGSI

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

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.

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.

```
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
```

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.

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 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

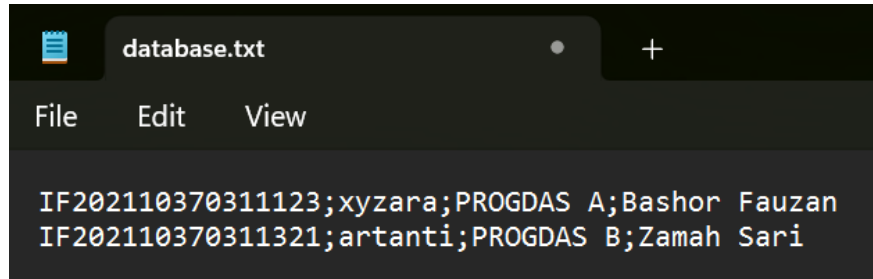
Dalam bahasa C, kalian bisa melakukan operasi *create*, *read*, *write*, dan *close* pada suatu file. Mengapa files perlu digunakan? Karena dengan menggunakan files, kita bisa melakukan beberapa operasi yaitu :

- 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**.

➤ 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



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

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>

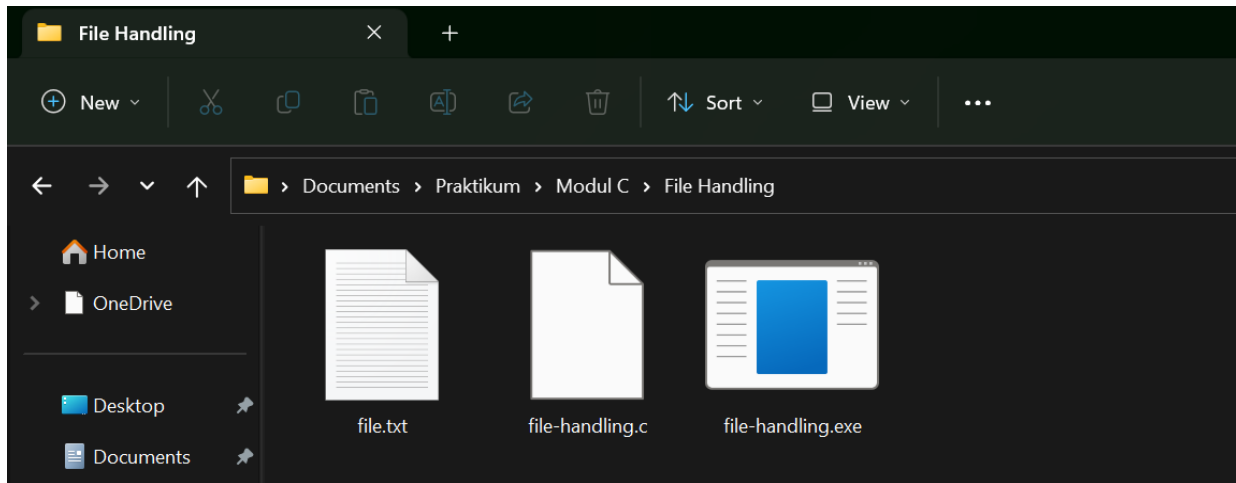
int main()
{
    // file pointer
    FILE* fptr;

    // 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);
        }

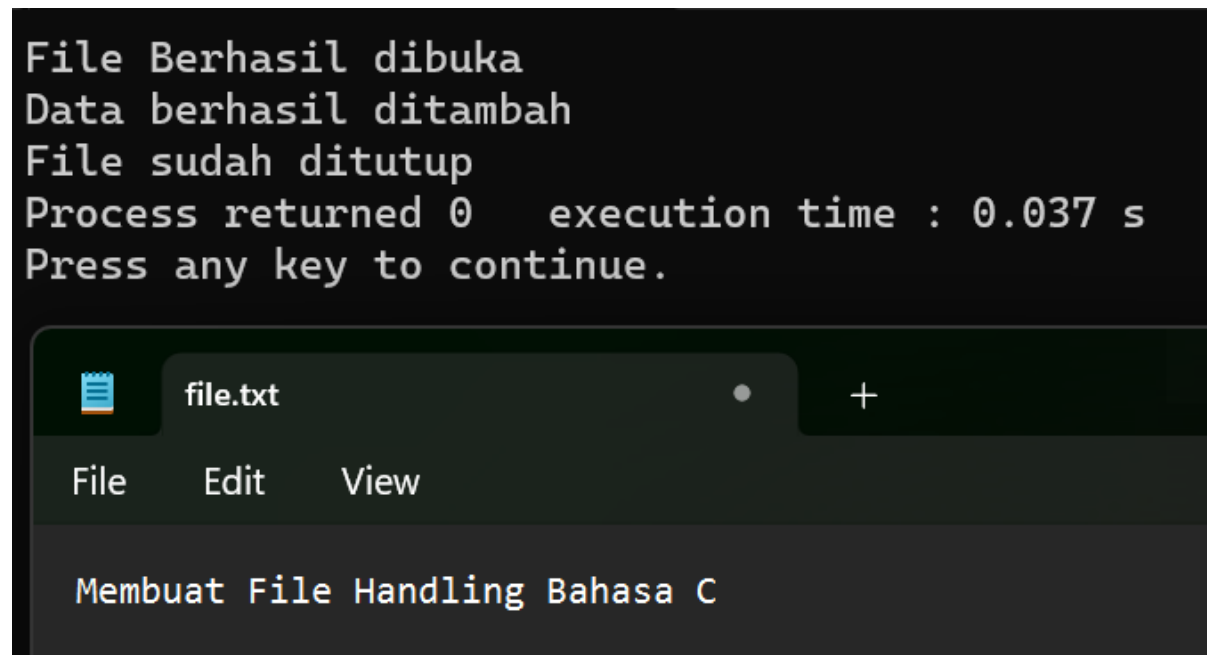
        fclose(fptr);

        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.



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

file.txt

File Edit View

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");

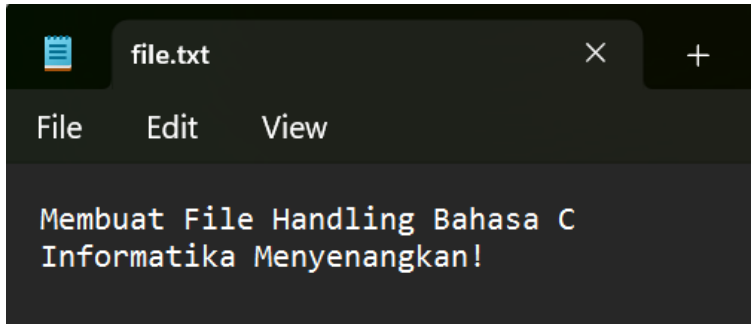
        if (strlen(addData) > 0) {
            fputs(addData, fptr);
            fputs("\n", fptr);
        }

        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

Buatlah program sederhana yang meminta input dari user tentang kalkulator menghitung luas dan volume balok. Gunakan **function** untuk menyimpan rumus luas dan volume balok kemudian panggil fungsi tersebut untuk menghitung balok.

Contoh output :

```
===KALKULATOR MENGHITUNG BALOK===
Masukkan panjang balok: 12
Masukkan lebar balok: 10
Masukkan tinggi balok: 15

Luas permukaan balok: 900 cm^2
Volume balok: 1800 cm^3
Process returned 0   execution time : 6.705 s
Press any key to continue.
```

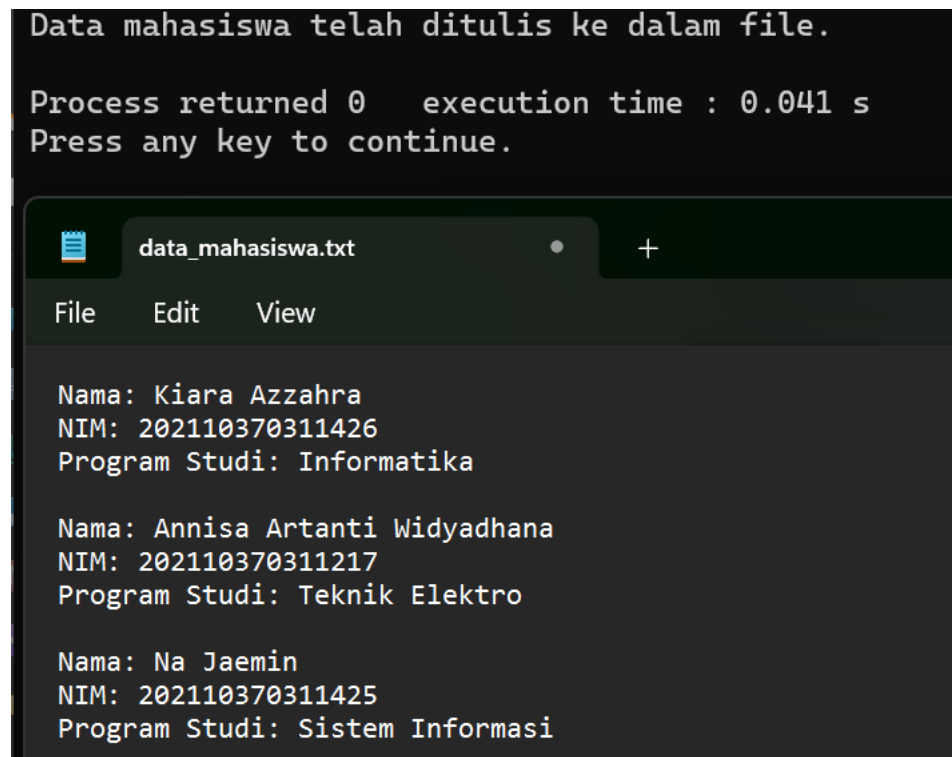
CODELAB 2

Buatlah file dengan nama `data_mahasiswa.txt` dan berikan data berupa :

- Nama
- NIM
- Program Studi

Buat 3 data mahasiswa dan tunjukkan kode program dan hasil output kepada asisten masing-masing.

Contoh output :



```

Data mahasiswa telah ditulis ke dalam file.
Process returned 0   execution time : 0.041 s
Press any key to continue.

```

data_mahasiswa.txt

File Edit View

```

Nama: Kiara Azzahra
NIM: 202110370311426
Program Studi: Informatika

Nama: Annisa Artanti Widyadhana
NIM: 202110370311217
Program Studi: Teknik Elektro

Nama: Na Jaemin
NIM: 202110370311425
Program Studi: Sistem Informasi

```

KEGIATAN 1

Buatlah program Create, Read, Update, Delete (CRUD) dengan ketentuan sebagai berikut :

1. Buatlah file dengan nama **library_books.txt** yang akan digunakan untuk menyimpan informasi buku
2. Buatlah menu interaktif yang akan ditampilkan untuk user yaitu :
 - a. **Create New Book** : untuk menambahkan data apabila terdapat buku baru.
 - b. **Display List of Books** : untuk menampilkan data buku yang terdaftar di data.

- c. **Update Book Information** : untuk mengupdate buku yang sudah tersedia di data. Misal perlu adanya update pergantian ID pada salah satu buku.
 - d. **Delete Book** : untuk meminta user memasukkan judul buku yang ingin dihapus. Jika ditemukan, hapus informasi buku dari file **books.txt**
 - e. **Exit** : keluar dari program dan kembali ke menu awal
3. Data yang harus ditambahkan yaitu :
- a. ID buku : Tidak diizinkan adanya duplikasi ID antara buku-buku. Artinya, setiap buku harus memiliki ID yang unik. Jika ID yang sama sudah ada dalam data, maka akan ditampilkan pesan "**ID SUDAH TERDAFTAR**", dan pengguna harus memasukkan ID buku yang berbeda.
 - b. Judul Buku
 - c. Nama Author
 - d. Jumlah Halaman
 - e. Genre Buku
4. Pastikan programmu memiliki error handling yang baik, misalnya menangani kasus jika buku 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 apa yang program lakukan.

KRITERIA & DETAIL PENILAIAN

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