UNIT 5

Dynamic Memory Allocation: Introduction, Library functions – malloc, calloc, realloc and free.
File Handling: Basics, File types, File operations, File er, File opening modes, File handling functions, File handling through command line argument, Record I/O in files.
Graphics: Introduction, Constant, Data types and global variables used in graphics, Library functions used in drawing, Drawing and filling images, GUI interaction within the program.

Dynamic Memory Allocation: Covered in Unit 3.

File Handling: File Handling is the storing of data in a file using a program. In C programming language, the programs store results, and other data of the program to a file using *file handling* in C. Also, we can extract/fetch data from a file to work with it in the program.

The operations that you can perform on a File in C are −

- Creating a new file
- Opening an existing file
- Reading data from an existing file
- Writing data to a file
- Moving data to a specific location on the file
- Closing the file

Creating or opening file using fopen()

The fopen() function is used to create a new file or open an existing file in C. The fopen function is defined in the **stdio.h** header file.

Now, lets see the syntax for creation of a new file or opening a file

file = fopen(“file_name”, “mode”)

This is a common syntax for both opening and creating a file in C.

Parameters

file_name − It is a string that specifies the name of the file that is to be opened or created using the fopen method. mode: It is a string (usually a single character) that specifies the mode in which the file is to be opened. There are various modes available to open a file in C, we will learn about all of them later in this article.

When will a file be created?

The fopen function will create a new file when it will not find any file of the specified name in the specified location. Else, if the file is found it will be opened with the mode specified.

Let's see can example which will make the concept clear, Suppose we are opening a file named hello.txt using the fopen function. The following will be the statement,

file = fopen("hello.txt", "w")

This will search for a file named hello.txt in the current directory. If the file exists, it will open the file otherwise it will create a new file named "hello.txt" and open it with write mode (specified using "w").

Now, let's see all types of modes that are available for us to read or write a file in C, and see code snippets that will show sample runs of the code.

Mode = "r" − open for reading, this mode will open the file for reading purpose only, i.e. the contents can only be viewed, nothing else like edits can be done to it.

This mode cannot create a new file and **open() returns NULL**, if we try to create a new file using this mode.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   if (file = fopen("hello.txt", "r")){
      printf("File opened successfully in read mode");
   }
   else
   printf("The file is not present! cannot create a new file using r mode");
   fclose(file);
   return 0;
}
```

Output

File opened successfully in read mode

We have created a file named hello.txt in our current directory but if we try to access other file then we will get "The file is not present! cannot create a new file using r mode" as output.

Mode = "rb" − open for reading in binary mode, this mode will open the file for reading in binary mode only, i.e. the contents can only be viewed and nothing else like edits can be done to it.

This mode cannot create a new file and **open() returns NULL**, if we try to create a new file using this mode.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   if (file = fopen("program.txt", "rb")){
      printf("File opened successfully in read mode");
   }
   else
   printf("The file is not present! cannot create a new file using rb mode");
   fclose(file);
   return 0;
}
```

Output

The file is not present! cannot create a new file using rb mode

Mode = "w" − open for writing only, this mode will open the file if present in the current directory for writing only i.e. reading operation cannot be performed. If the file is not present in the current directory, the program will create a new file and open it for writing.

If we open a file that contains some text in it, the contents will be overwritten.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   if (file = fopen("helo.txt", "w")){
      printf("File opened successfully in write mode or a new file is created");
   }
   else
   printf("Error!");
   fclose(file);
   return 0;
}
```

Output

File opened successfully in write mode or a new file is created

You can see here, we have tried to open file “helo.txt” which is not present in the directory, still the function returned the success message because it has create a file named “helo.txt”.

Mode = “wb” − open for writing in binary mode, this mode will open the file if present in the current directory for writing in binary mode i.e. reading operation cannot be performed. If the file is not present in the current directory, the program will create a new file and open it for writing in binary mode.

If we open a file that contains some text in it, the contents will be overwritten.

Example

```
#include <stdio.h>
int main(){
  FILE * file;
  if (file = fopen("hello.txt", "wb")){
    printf("File opened successfully in write in binary mode or a new file is created");
  }
  else
  printf("Error!");
  fclose(file);
  return 0;
}
```

Output

File opened successfully in write in binary mode or a new file is created

Mode = “a” − open for append only, this mode will open the file if present in the current directory for writing only i.e. reading operation cannot be performed. If the file is not present in the current directory, the program will create a new file and open it for writing. If we open a file that contains some text in it, the contents will not be overwritten; instead the new text will be added after the existing text in the file.

Example

```
#include <stdio.h>
int main(){
  FILE * file;
  if (file = fopen("hello.txt", "a")){
    printf("File opened successfully in append mode or a new file is created");
  }
  else
```

```
  printf("Error!");
  fclose(file);
  return 0;
}
```

Output

File opened successfully in append mode or a new file is created

Mode = "ab" − open for append in binary, this mode will open the file if present in the current directory for writing in binary only i.e. reading operation cannot be performed. If the file is not present in the current directory, the program will create a new file and open it for writing in binary.

If we open a file that contains some text in it, the contents will not be overwritten; instead the new text will be added after the existing text in the file.

Example

```
#include <stdio.h>
int main(){
  FILE * file;
  if (file = fopen("hello.txt", "ab")){
    printf("File opened successfully in append in binary mode or a new file is created");
  }
  else
  printf("Error!");
  fclose(file);
  return 0;
}
```

Output

File opened successfully in append in binary mode or a new file is created

Mode = "r+" − open for reading and writing both, this mode will open the file for both reading and writing purposes i.e. both read and write operations can be performed to the file.

This mode cannot create a new file and **open() returns NULL**, if we try to create a new file using this mode.

If we open a file that contains some text in it and write something, the contents will be overwritten.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   if (file = fopen("hello.txt", "r+")){
      printf("File opened successfully in read and write both");
   }
   else
   printf("The file is not present! cannot create a new file using r+ mode");
   fclose(file);
   return 0;
}
```

Output

File opened successfully in read and write both

We have created a file named hello.txt in our current directory but if we try to access another file then we will get “The file is not present! cannot create a new file using r+ mode” as output.

Mode = “rb+” − open for reading in binary mode, this mode will open the file for reading in binary mode only, i.e. the contents can only be viewed and nothing else like edits can be done to it.

This mode cannot create a new file and **open() returns NULL,** if we try to create a new file using this mode.

If we open a file that contains some text in it and write something, the contents will be overwritten.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   if (file = fopen("program.txt", "rb+")){
      printf("File opened successfully in read mode");
   }
   else
   printf("The file is not present! cannot create a new file using rb+ mode");
   fclose(file);
   return 0;
}
```

Output

The file is not present! cannot create a new file using rb+ mode

Mode = “w” − open for writing and reading, this mode will open the file if present in the current directory for writing and reading operation both. If the file is not present in the current directory, the program will create a new file and open it for reading and writing.

If we open a file that contains some text in it, the contents will be overwritten.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   if (file = fopen("helo.txt", "w+")){
      printf("File opened successfully in read-write mode or a new file is created");
   }
   else
   printf("Error!");
   fclose(file);
   return 0;
}
```

Output

File opened successfully in read-write mode or a new file is created

You can see here, we have tried to open file “helo.txt” which is not present in the directory, still the function returned the success message because it has create a file named “helo.txt”.

Mode = “wb+” : open for writing and reading in binary mode, this mode will open the file if present in the current directory for writing and reading in

binary mode. If the file is not present in the current directory, the program will create a new file and open it for reading and writing in binary mode. If we open a file that contains some text in it, the contents will be overwritten.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   if (file = fopen("hello.txt", "wb+")){
      printf("File opened successfully in read-write in binary mode or a new file is created");
   }
   else
   printf("Error!");
   fclose(file);
   return 0;
}
```

Output

File opened successfully in read-write in binary mode or a new file is created

Mode = "a+" − open for read and append, this mode will open the file if present in the current directory for both reading and writing. If the file is not present in the current directory, the program will create a new file and open it for reading and writing.

If we open a file that contains some text in it, the contents will not be overwritten; instead the new text will be added after the existing text in the file.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   if (file = fopen("hello.txt", "a+")){
      printf("File opened successfully in read-append mode or a new file is created");
   }
   else
   printf("Error!");
   fclose(file);
   return 0;
}
```

Output

File opened successfully in read-append mode or a new file is created

Mode = "ab+" − open for read and append in binary, this mode will open the file if present in the current directory for both reading and writing in binary. If the file is not present in the current directory, the program will create a new file and open it for reading and writing in binary. If we open a file that contains some text in it, the contents will not be overwritten; instead the new text will be added after the existing text in the file.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   if (file = fopen("hello.txt", "ab+")){
      printf("File opened successfully in read-append in binary mode or a new file is created");
   }
   else
   printf("Error!");
   fclose(file);
   return 0;
}
```

Output

File opened successfully in read-append mode or a new file is created

Reading Data for from an existing file

We can read content of a file in c using the fscanf() and fgets() and fgetc() functions. All are used to read contents of a file. Let's see the working of each of the function −

fscanf()

The fscanf() function is used to read character set i.e strings from the file. It returns the EOF, when all the content of the file are read by it.

Syntax

int fscanf(FILE *stream, const char *charer[])

Parameters

FILE *stream: the er to the opened file.
const char *charer[]: string of character.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   char str[500];
   if (file = fopen("hello.txt", "r")){
        while(fscanf(file,"%s", str)!=EOF){
        printf("%s", str);
      }
   }
   else
   printf("Error!");
   fclose(file);
   return 0;
}
```

Output

LearnprogrammingatIMSEC

fgets()

The fget() function in C is used to read string from the stream.

Syntax

char* fgets(char *string, int length, FILE *stream)

Parameter

char *string: It is a string which will store the data from the string.

int length: It is an int which gives the length of string to be considered.

FILE *stream: It is the er to the opened file.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   char str[500];
   if (file = fopen("hello.txt", "r")){
      printf("%s", fgets(str, 50, file));
   }
   fclose(file);
   return 0;
}
```

Output

Learn programming at imsec

fgetc()

The fgetc() function in C is used to return a single character from the file. It gets a character from the file and returns EOF at the end of file.

Syntax

char* fgetc(FILE *stream)

Parameter

FILE *stream: It is the er to the opened file.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   char str;
   if (file = fopen("hello.txt", "r")){
      while((str=fgetc(file))!=EOF)
      printf("%c",str);
   }
   fclose(file);
   return 0;
}
```

Output

Learn programming at imsec

Writing Data to a file in C

We can write data to a file in C using the fprintf(), fputs(), fputc() functions. All are used to write contents to a file.

Let’s see the working of each of the function −

fprintf()

The fprintf() function is used to write data to a file. It writes a set of characters in a file.

Syntax

int fprintf(FILE *stream, char *string[])

Parameters

FILE for *stream: It is the er to the opened file.
char *string[]: It is the character array that we want to write in the file.

Example

```
#include <stdio.h>
int main(){
  FILE * file;
  if (file = fopen("hello.txt", "w")){
    if(fprintf(file, " imsec ”) >= 0)
    printf("Write operation successful");
  }
  fclose(file);
  return 0;
}
```

Output

Write operation successful

fputf()

The fputf() function in C can be used to write to a file. It is used to write a line (character line) to the file.

Syntax

int fputs(const char *string, FILE *stream)

Parameters

Constant char *string[]: It is the character array that we want to write in the file.
FILE for *stream: It is the er to the opened file.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   if (file = fopen("hello.txt", "w")){
      if(fputs(" imsec ", file) >= 0)
      printf("String written to the file successfully...");
   }
   fclose(file);
   return 0;
}
```

Output

String written to the file successfully…

fputc()

The fputc() function is used to write a single character to the file.

Syntax

int fputc(char character , FILE *stream)

Parameters

char character : It is the character that we want to write in the file.
FILE for *stream: It is the er to the opened file.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   if (file = fopen("hello.txt", "w")){
      fputc('T', file);
   }
   fclose(file);
   return 0;
}
```

Output

'T' is written to the file.

fclose()

The fclose() function is used to close the open file. We should close the file after performing operations on it to save the operations that we have applied to it.

Syntax

fclose(FILE *stream)

Parameters

FILE for *stream: It is the er to the opened file.

Example

```
#include <stdio.h>
int main(){
   FILE * file;
   char string[300];
   if (file = fopen("hello.txt", "a+")){
      while(fscanf(file,"%s", string)!=EOF){
         printf("%s", string);
      }
      fputs("Hello", file);
   }
   fclose(file);
   return 0;
}
```

Output

Learnprogrammingat imsec
File contains

Learn programming at IMSEC Hello

Graphics: Graphics are defined as any sketch or a drawing or a special network that pictorially represents some meaningful information. Computer Graphics is used where a set of images needs to be manipulated or the creation of the image in the form of pixels and is drawn on the computer. Computer Graphics can be used in digital photography, film, entertainment, electronic gadgets, and all other core technologies which are required. It is a vast subject and area in the field of computer science. Computer Graphics can be used in UI design, rendering, geometric objects, animation, and many more. In most areas, computer graphics is an abbreviation of CG. There are several tools used for the implementation of Computer Graphics. The basic is the <graphics.h> header file in Turbo-C, Unity for advanced and even OpenGL can be used for its Implementation.
The term 'Computer Graphics' was coined by Verne Hudson and William Fetter from Boeing who were pioneers in the field.

Computer Graphics refers to several things:

- The manipulation and the representation of the image or the data in a graphical manner.
- Various technology is required for the creation and manipulation.
- Digital synthesis and its manipulation.

Types of Computer Graphics

- **Raster Graphics:** In raster, graphics pixels are used for an image to be drawn. It is also known as a bitmap image in which a sequence of images is into smaller pixels. Basically, a bitmap indicates a large number of pixels together.
- **Vector Graphics:** In vector graphics, mathematical formulae are used to draw different types of shapes, lines, objects, and so on.

Applications

- **Computer Graphics are used for** an **aided design for engineering and architectural system-** These are used in electrical automobiles, electro-mechanical, mechanical, electronic devices. For example gears and bolts.
- **Computer Art –** MS Paint.
- **Presentation Graphics –** It is used to summarize financial statistical scientific or economic data. For example- Bar chart, Line chart.
- **Entertainment-** It is used in motion pictures, music videos, television gaming.
- **Education and training-** It is used to understand the operations of complex systems. It is also used for specialized system such for framing for captains, pilots and so on.
- **Visualization-** To study trends and patterns.For example- Analyzing satellite photo of earth.

Library functions used in graphics:

The graphics functions and graphics modes are typically part of the <graphics.h> library. To use these functions in your program, you need to include the graphics.h file at the beginning of your code. Here is an example of how you can include it:

```
#include <graphics.h>
```

This line tells the C compiler to include the graphics.h header file, which provides the necessary declarations and definitions for the graphics functions and modes. Without this inclusion, the compiler won't recognize these functions, and you'll get compilation errors.

1. **initgraph():**
    - Purpose: Initializes the graphics system, allowing you to use other graphics functions.
    - Syntax:

    void initgraph(int *graphdriver, int *graphmode, char *pathtodriver);
    - Explanation:
        - graphdriver: Represents the graphics driver to be used (e.g., DETECT, VGA, etc.).
        - graphmode: Specifies the graphics mode (e.g., VGAMAX, VGAMED, etc.).
        - pathtodriver: Optional parameter specifying the path to the graphics driver file.
2. **closegraph():**
    - Purpose: Closes the graphics system and releases resources.
    - Syntax:

    void closegraph();
    - Explanation:
        - This function is essential to clean up resources after graphics operations. It is typically called at the end of the graphics program.
3. **line():**
    - Purpose: Draws a line between two specified points.
    - Syntax:

    void line(int x1, int y1, int x2, int y2);
    - Explanation:
        - (x1, y1) and (x2, y2) are the coordinates of the starting and ending points of the line, respectively.
4. **circle():**
    - Purpose: Draws a circle with a specified center and radius.
    - Syntax:

    void circle(int x, int y, int radius);
    - Explanation:
        - (x, y) is the center of the circle, and radius is the distance from the center to any point on the circumference.
5. **rectangle():**
    - Purpose: Draws a rectangle with specified coordinates for the top-left and bottom-right corners.
    - Syntax:

    void rectangle(int left, int top, int right, int bottom);
    - Explanation:
        - (left, top) and (right, bottom) represent the coordinates of the top-left and bottom-right corners of the rectangle.
6. **getch():**
    - Purpose: Waits for a key press and returns the ASCII value of the key.
    - Syntax:

    int getch();
    - Explanation:
        - Useful for creating interactive graphics programs where user input is required. It allows the program to wait until a key is pressed**.**

7. **putpixel():**
    - Purpose: Sets the color of a pixel at a specified location.
    - Syntax:

    void putpixel(int x, int y, int color);
    - Explanation:
        - (x, y) is the coordinates of the pixel, and color is the code representing the color to set.
8. **setcolor():**
    - Purpose: Sets the current drawing color.
    - Syntax:

    void setcolor(int color);
    - Explanation:
        - Changes the drawing color for subsequent graphics operations. The color parameter is an integer representing the color code.

Drawing and filling images in C : It can be done using the graphics functions provided by libraries like **<graphics.h>** in Turbo C or Borland C++. Here's a basic example that demonstrates how to draw and fill a rectangle in C using graphics.h:

```
#include <graphics.h>

int main() {
   int gd = DETECT, gm;
   initgraph(&gd, &gm, "");

   // Draw a rectangle
   rectangle(100, 100, 300, 200);

   // Fill the rectangle with color
   floodfill(200, 150, WHITE); // Replace WHITE with the desired color code

   getch();  // Wait for a key press
   closegraph();

   return 0;
}
```

In this example:

- **rectangle(100, 100, 300, 200)** draws a rectangle with top-left corner at (100, 100) and bottom-right corner at (300, 200).

- **floodfill(200, 150, WHITE)** fills the rectangle with color starting from the point (200, 150). The color to fill with is specified by the **WHITE** constant, which you can replace with other color codes.

Remember that the actual color codes may vary depending on the graphics system and compiler you are using. Always refer to the documentation for the specific compiler or graphics library.

GUI interaction within the program:

To incorporate GUI (Graphical User Interface) interaction within a C program, you'll typically need to use external libraries, as the standard C language itself does not provide native support for GUI development. One common library for GUI programming in C is GTK (GIMP Toolkit). GTK is widely used, especially in the Linux environment, but it's also available for other platforms.

1. Introduction to GUI Programming in C:

- GUI (Graphical User Interface) programming involves creating interactive applications with visual elements such as windows, buttons, and text fields.
- Standard C does not have built-in support for GUI development, so external libraries are used to handle graphical components.

2. Choosing a GUI Library:

- GTK (GIMP Toolkit) is one of the popular libraries for GUI programming in C.
- GTK is cross-platform and widely used, especially in the Linux environment.
- Other alternatives include Qt and wxWidgets.

3. Setting Up a Basic GTK Program:

- Include the necessary header files and link against the GTK library during compilation.
- Initialize GTK using **gtk_init(&argc, &argv)**.

4. Creating a Simple Window:

- Use **gtk_window_new(GTK_WINDOW_TOPLEVEL)** to create a main window.
- Customize the window's properties with functions like **gtk_window_set_title()** and **gtk_container_set_border_width()**.

5. Adding Widgets (Components) to the Window:

- Widgets are graphical components like buttons, labels, and text fields.
- Create widgets using functions such as **gtk_button_new_with_label()**.

6. Handling Events:

- Connect signals (events) to callback functions using **g_signal_connect()**.
- Callback functions respond to user interactions, such as button clicks.
- GTK provides many built-in signals for various events.

7. GTK Main Event Loop:

- Start the GTK main event loop with **gtk_main()**.
- This loop handles user input and keeps the GUI responsive.

8. Alternative Libraries:

- Some other libraries like Qt and wxWidgets for cross-platform GUI development.
- These libraries may have different approaches and features.

Here's a basic example using GTK in C to create a simple GUI window:

```
 #include <gtk/gtk.h>
// Callback function for handling button click event
void button_clicked(GtkWidget *widget, gpointer data) {
    g_print("Button Clicked!\n");
}

int main(int argc, char *argv[]) {
    // Initialize GTK
    gtk_init(&argc, &argv);

    // Create the main window
    GtkWidget *window = gtk_window_new(GTK_WINDOW_TOPLEVEL);
    gtk_window_set_title(GTK_WINDOW(window), "Simple GUI");
    gtk_container_set_border_width(GTK_CONTAINER(window), 10);
    gtk_widget_set_size_request(window, 300, 200);

    // Create a button
    GtkWidget *button = gtk_button_new_with_label("Click Me");
    g_signal_connect(button, "clicked", G_CALLBACK(button_clicked), NULL);

    // Add the button to the window
    gtk_container_add(GTK_CONTAINER(window), button);

    // Show all the widgets
    gtk_widget_show_all(window);

    // Set up the main event loop
    g_signal_connect(window, "destroy", G_CALLBACK(gtk_main_quit), NULL);
    gtk_main();

    return 0;
}
```

To compile this program, you'll need to link against the GTK library. For example, with gcc:

```
gcc your_program.c -o your_program `pkg-config --cflags --libs gtk+-3.0`
```

In this example:

- We create a simple GTK window with a button.
- When the button is clicked, the **button clicked** function is called, printing a message to the console.

Please note that GTK requires additional setup and may have platform-specific dependencies. Ensure that you have the GTK development libraries installed on your system.

For cross-platform GUI development in C, you might also consider libraries like Qt or wxWidgets, which provide abstraction layers for GUI programming and support multiple platforms. These libraries often come with their own set of tools and frameworks for designing GUI interfaces visually.