

# *PYTHON-BACKEND ASSIGNMENT*

## *Module 2 – Introduction to Programming*

### **1. Overview of C Programming**

- ❖ **THEORY EXERCISE:** Write an essay covering the history and evolution of C programming. Explain its importance and why it is still used today.

→ The C programming language is one of the most influential and enduring languages in the history of computing. Developed in the early 1970s it has shaped the world of software development and continues to be relevant decades after its inception. This essay will explore the history and evolution of C programming.

→ Explain its importance and why it is still used today.

- Efficiency and Performance
- Portability
- Versatility
- Legacy Code and System Maintenance
- Educational Value

### **2. Setting Up Environment**

- ❖ **THEORY EXERCISE:** Describe the steps to install a C compiler (e.g., GCC) and set up an Integrated Development Environment (IDE) like Dev-C++, VS Code, or Code Blocks

→ 1. Installing GCC Compiler on Windows

→ 2. Setting Up DevC++

→ 3. Setting Up Code::Blocks

### **3. Basic Structure of a C Program**

❖ **THEORY EXERCISE:** Explain the basic structure of a C program, including headers, main function, comments, data types, and variables. Provide examples.

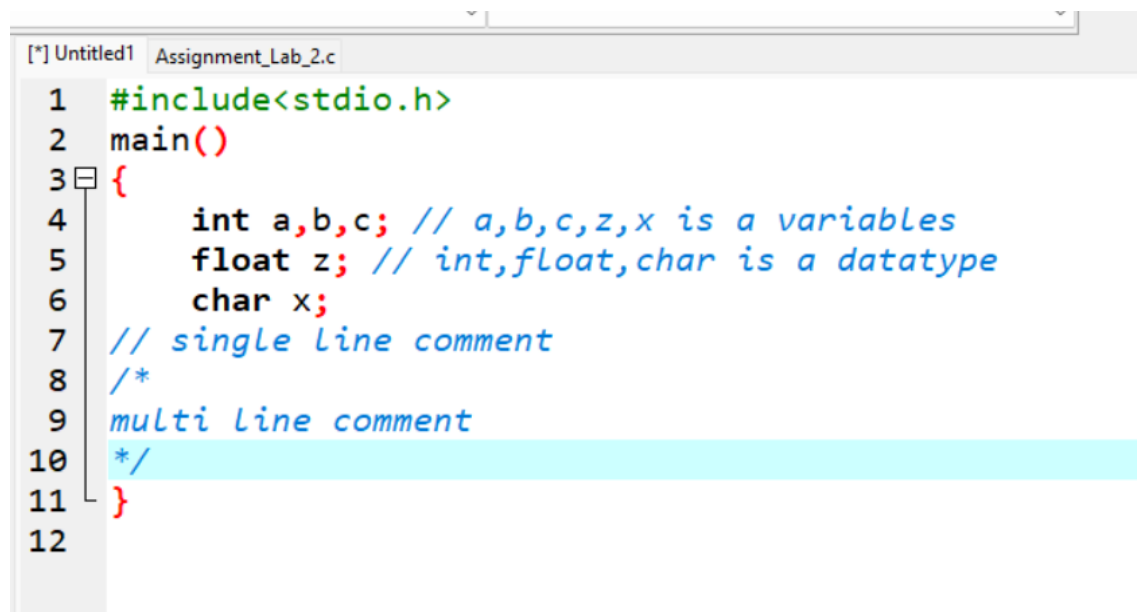
→ 1. Headers : Headers provide declarations for functions and macros. The most commonly used header is <stdio.h>, which allows for input and output functions.

→ 2. Main Function : Every C program must have a main( ). It serves as the entry point for the program execution

→ 3. Comments : single-line (//) and multi-line (/\*...\*/).

→ 4. Data Types : Data types specify the type of data a variable can hold. Common data types include. Int, float, char, long int.

→ 5. Variables : Variables are used to store data. They must be declared with a specific data type before use.



```
1  #include<stdio.h>
2  main()
3  {
4      int a,b,c; // a,b,c,z,x is a variables
5      float z; // int,float,char is a datatype
6      char x;
7      // single line comment
8      /*
9      multi line comment
10     */
11 }
12
```

## 4. Operators in C

❖ **THEORY EXERCISE:** Write notes explaining each type of operator in C: arithmetic, relational, logical, assignment, increment/decrement, bitwise, and conditional operators.

→ **Arithmetic Operators**

These operators are used for basic mathematical operations:

- **Addition (+):** Adds two operands.
- **Subtraction (-):** Subtracts the second operand from the first.

- **Multiplication (\*)**: Multiplies two operands.
- **Division (/)**: Divides the first operand by the second.
- **Modulus (%)**: Finds the remainder after division of one operand by another.

#### →Relational Operators

These operators are used to compare two values:

- **Equal to (==)**: Checks if two operands are equal.
- **Not equal to (!=)**: Checks if two operands are not equal.
- **Greater than (>)**: Checks if the first operand is greater than the second.
- **Less than (<)**: Checks if the first operand is less than the second.
- **Greater than or equal to (>=)**: Checks if the first operand is greater than or equal to the second.
- **Less than or equal to (<=)**: Checks if the first operand is less than or equal to the second.

#### →Logical Operators

These operators are used to combine conditional statements:

- **Logical AND (&&)**: Returns true if both operands are true.
- **Logical OR (||)**: Returns true if at least one of the operands is true.
- **Logical NOT (!)**: Returns true if the operand is false.

#### →Assignment Operators

These operators are used to assign values to variables:

- **Assignment (=)**: Assigns the value of the right operand to the left operand.
- **Add and assign (+=)**: Adds the right operand to the left operand and assigns the result to the left operand.
- **Subtract and assign (-=)**: Subtracts the right operand from the left operand and assigns the result to the left operand.
- **Multiply and assign (\*=)**: Multiplies the left operand by the right operand and assigns the result to the left operand.
- **Divide and assign (/=)**: Divides the left operand by the right operand and assigns the result to the left operand.
- **Modulus and assign (%=)**: Finds the remainder after division of left operand by the right operand and assigns the result to the left operand.

#### Increment/Decrement Operators

These operators are used to increase or decrease the value of a variable by one:

- **Increment (++)**: Increases the value of the operand by one.

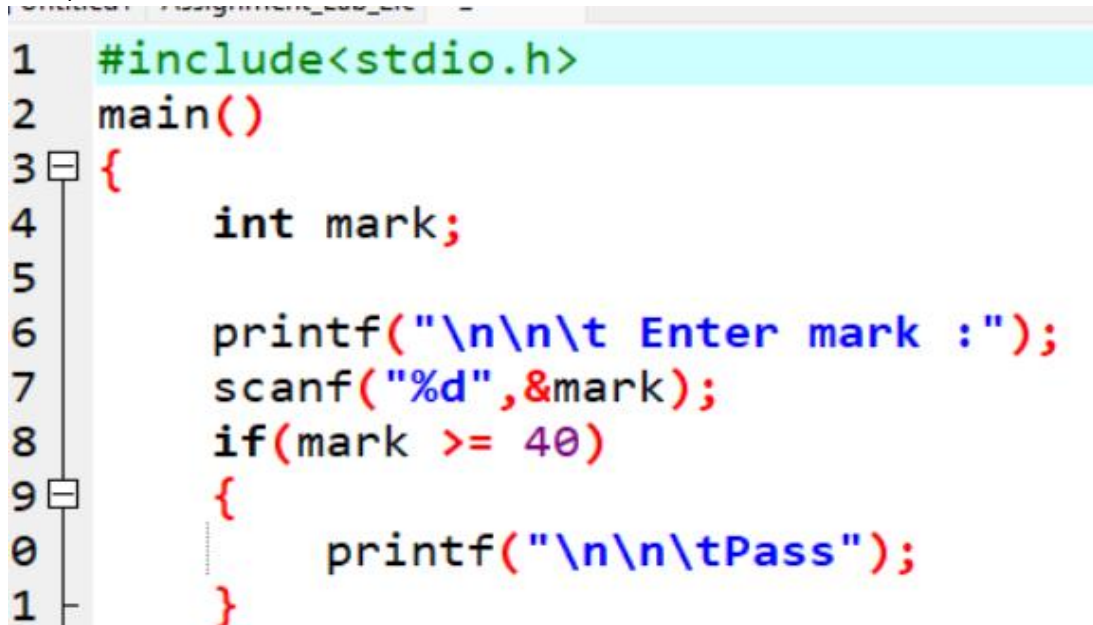
- **Decrement (--):** Decreases the value of the operand by one.

## 5. Control Flow Statements in C

❖ **THEORY EXERCISE:** Explain decision-making statements in C (if, else, nested if-else, switch). Provide examples of each.

→ If Statement:

- Syntax:  
if (condition) {  
    Block of code to be executed if the condition is true  
}
- Example:



```

1  #include<stdio.h>
2  main()
3  {
4      int mark;
5
6      printf("\n\n\t Enter mark :");
7      scanf("%d",&mark);
8      if(mark >= 40)
9      {
10         printf("\n\n\tPass");
11     }

```

→ if-else Statement:

- Syntax:  
if (condition)  
{  
    Block of code to be executed if the condition is true  
}  
else  
{  
    Block of code to be executed if the condition is false  
}
- Example:

```

1  #include<stdio.h>
2  main()
3  {
4      int mark;
5
6      printf("\n\n\t Enter mark :");
7      scanf("%d",&mark);
8      if(mark >= 40)
9      {
10         printf("\n\n\tPass");
11     }
12     else
13     {
14         printf("\n\n\tFail");
15     }
16 }

```

→if-else if-else:

- Syntax:
 

```

if (condition1) {
    Block of code if condition1 is true
}
else if (condition2)
{
    Block of code if condition2 is true
}
Else
{
    Block of code if none of the above conditions are true
}

```
- Example:

```

#include<stdio.h>
main()
{
    int a,b;
    char choice;
    printf("enter the number1 :");
    scanf("%d",&a);
    printf("enter the number2 :");
    scanf("%d",&b);
    printf("\n-----");
    printf("\n press + for addition");
    printf("\n press - for subtraction");
    printf("\n press * for product");
    printf("\n press / for division");
    printf("\n-----");
    printf("\n select option");
    scanf(" %c",&choice);

    if(choice=='+')
        printf("addition:%d",a+b);
    else if(choice=='-')
        printf("subtraction:%d",a-b);
    else if(choice=='*')
        printf("product:%d",a*b);
    else if(choice=='/')
        printf("division:%d",a/b);
    else
        printf("\n valid option choice");
}

```

→Nested if-else Statements:

- Syntax:
 

```

if (condition1) {
    if (condition2) {
        Block of code if condition1 and condition2 are true
    } else {
        Block of code if condition1 is true but condition2 is false
    }
} else {
    Block of code if condition1 is false
}

```
- Example:

```

#include<stdio.h>
main()
{
    int a,b,c;
    printf("enter any three number :");
    scanf("%d %d %d",&a,&b,&c);
    if(a>b)
    {
        if(a>c)
        {
            printf("\n\n\t %d is max",a);
        }
    }
}

```

→Switch Statement:

- Syntax:  
 switch (expression) {  
   case value1:  
     Code to execute if expression == value1  
     break;  
   case value2:  
     Code to execute if expression == value2  
     break;  
   Add more cases as needed  
   default:  
     Code to execute if no case matches  
   }
- Example:

```

#include<stdio.h>
main()
{
    int a,b;
    char choice;
    printf("enter the number1 :");
    scanf("%d",&a);
    printf("enter the number2 :");
    scanf("%d",&b);
    printf("\n-----");
    printf("\n press + for addition");
    printf("\n press - for subtraction");
    printf("\n press * for product");
    printf("\n press / for division");
    printf("\n-----");
    printf("\n select option");
    scanf(" %c",&choice);

    switch(choice)
    {
        case '+' : printf("\n\n\t addition :%d",a+b);
                   break;
        case '-' : printf("\n\n\t subtraction :%d",a-b);
                   break;
        case '*' : printf("\n\n\t product :%d",a*b);
                   break;
        case '/' : printf("\n\n\t division :%d",a/b);
                   break;
        default : printf("\n\n\t invalide choice");
                  break;
    }
}

```

## 6. Looping in C.

- ❖ **THEORY EXERCISE:** Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.

→ While Loop:



- Syntax:  
while condition:  
    Code to execute
- Description: A while loop executes a block of code repeatedly as long as the specified condition evaluates to `True`.
- When to Use: Scenarios: Ideal for situations like input validation or when iterating over dynamic data (e.g., reading from a file until it's empty).

→ For Loop:

- Syntax:  
for variable in iterable:  
    Code to execute
- Description: A for loop iterates over a sequence (like a list, string, range, or any iterable) and executes the code block once for each element.
- When to Use : Fixed number of iterations: Use when you know the number of iterations ahead of time, such as iterating over a list or a range of numbers.

→ Do-While Loop:

- Syntax:  
do {  
    Code to execute  
} while (condition);
- Description : A **do-while loop** guarantees that the code block will execute at least once before the condition is checked.
- When to Use Guaranteed first execution: Use when you want the loop body to run at least once regardless of the condition, and you need to check the condition after the first execution.

## 7. Loop Control Statements

❖ **THEORY EXERCISE:** Explain the use of `break`, `continue`, and `goto` statements in C. Provide examples of each.

- `break` Statement: The `break` statement is used to exit from a loop or switch-case statement prematurely. When a `break` is encountered, the program control is transferred immediately to the statement following the loop or switch block.
- Example:

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 10; i++) {
        if (i == 5) {
            break; // Exit the Loop when i equals 5
        }
        printf("%d ", i); // Output: 1 2 3 4
    }
}
```

- **Continue Statement:** The `continue` statement is used to skip the current iteration of a loop and proceed with the next iteration. When `continue` is executed, the remaining code in the loop for that iteration is skipped, and control is transferred to the loop's condition-checking statement.

- Example:

```
#include <stdio.h>

int main() {
    for (int i = 1; i <= 5; i++) {
        if (i == 3) {
            continue; // Skip the iteration when i equals 3
        }
        printf("%d ", i); // Output: 1 2 4 5
    }
}
```

- **goto statement:** The `goto` statement is used to transfer control to another part of the program. It is often considered a dangerous or poor practice because it can make code harder to follow and maintain. However, it can be useful for certain situations, such as error handling or breaking out of deeply nested loops.

- Example:

```
#include <stdio.h>
main()
{
    int i = 0;
start:
    if (i < 5)
    {
        printf("%d ", i); // Output: 0 1 2 3 4
        i++;
        goto start; // Jump back to the Label "start"
    }
}
```

## 8. Functions in C

❖ **THEORY EXERCISE:** What are functions in C? Explain function declaration, definition, and how to call a function. Provide examples.

→ **Functions in C:** A function in C is a block of code that performs a specific task. Functions help to break a program into smaller, manageable pieces, promote code reusability, and improve readability and maintainability.

- Example:

```
E_1_1.c  [*] Untitled1
1  #include<stdio.h>
2  void kevin();
3  main()
4  {
5      kevin();
6  }
7  void kevin()
```

## 9. Arrays in C

### ❖ THEORY EXERCISE: Explain the concept of arrays in C.

Differentiate between one-dimensional and multi-dimensional arrays with examples.

→ An array is a collection of elements that are of the same data type.

- One-Dimensional
- Example:

```
#include<stdio.h>
main()// one dim
{
    int arr[10]={10,20,30,40,50,60,70,80,90,100};
    int i;
    for(i=0;i<10;i++)
    {
        printf("\n\n\t Element : %d",arr[i]);
    }
}
```

- Multi- Dimensional
- Example:

```
#include<stdio.h>
main()
{
    int m[3][3][3],r,a,c;
    for(a=0;a<3;a++)
    {
        for(r=0;r<3;r++)
        {
            for(c=0;c<3;c++)
            {
                printf(" Enter element mat[%d][%d][%d] : ",a,r,c);
                scanf("%d",&m[a][r][c]);
            }
            printf("\n");
        }
        for(a=0;a<3;a++)
        {
            for(r=0;r<3;r++)
            {
                for(c=0;c<3;c++)
                {
                    printf(" %d",m[a][r][c]);
                }
                printf("\n");
            }
            printf("\n");
        }
    }
}
```

## 10. Pointers in C

❖ **THEORY EXERCISE:** Explain what pointers are in C and how they are declared and initialized. Why are pointers important in C?

→ A pointer is a variable that stores the memory address of another variable.

- Example:

```
#include<stdio.h>
main()
{
    int a=5;
    int *z=&a;
    printf("%p",&a);
    printf("\n%d",a);
}
```

## 11. Strings in C

❖ **THEORY EXERCISE:** Explain string handling functions like strlen(), strcpy(), strcat(), strcmp(), and strchr(). Provide examples of when these functions are useful.

- strlen(): It returns the length of string as a integer.
- Example:

```
1 #include<stdio.h>
2 #include<string.h>
3 main()
4 {
5     char str[20] = "Hello,World";
6     int length = strlen(str);
7     printf("\n\n\t Length of the string : %d",length);
8 }
```

- strcpy(): It copies one string into another.
- Example:

```
1 #include<stdio.h>
2 #include<string.h>
3 main()
4 {
5     char str[20] = "Hello,World",z[20];
6     strcpy(z,str);
7     printf("\n\n\t %s",str);
8     printf("\n\n\t %s",z);
9 }
```

- strcat(): It join two strings into one string.
- Example:

```
1 #include<stdio.h>
2 #include<string.h>
3 main()
4 {
5     char str1[20] = "Hello",str2[20]="World";
6     strcat(str1,str2);
7     printf("\n\n\t Add : %s",str1);
8 }
```

## 12. Structures in C :

❖ **THEORY EXERCISE:** Explain the concept of structures in C.

Describe how to declare, initialize, and access structure members.

- **Concept of Structures in C:** A **structure** in C is a user-defined data type that allows grouping of different types of variables (called members or fields) under a single name. Each member of a structure can have a different data type (e.g., integers, floats, arrays, or even other structures).
- **Declaring a Structure:** To define a structure, you use the **struct** keyword followed by the structure name and its members inside curly braces.

Here is the syntax to declare a structure:

```
struct structure_name {  
  
    data_type member1;  
  
    data_type member2;  
  
    // More members  
  
};
```

- **Initializing a Structure:** There are two ways to initialize a structure:

**At the time of declaration (Static Initialization):** You can initialize the structure members when you declare a structure variable:

```
struct Student student1 = {"John Doe", 20, 85.5};
```

**After Declaration (Dynamic Initialization):** You can declare the structure first and then assign values to its members individually:

```
struct Student student1;  
strcpy(student1.name, "John Doe"); // Using strcpy for string assignment  
student1.age = 20;  
student1.marks = 85.5;
```

- **Accessing Structure Members:** Structure members can be accessed using the **dot operator** (.) for individual structure variables. If the structure variable is a pointer, you use the **arrow operator** (->) to access the members.

## 13. File Handling in C :

❖ **THEORY EXERCISE:** Explain the importance of file handling in C.

Discuss how to perform file operations like opening, closing, reading, and writing files.

→ Importance of File Handling in C :

- Persistent Data Storage:
- Data Sharing:
- Efficient Data Management:
- Flexible Data Operations:

#### → Basic File Operations in C :

- Opening a File: To perform any operation on a file, you first need to open the file using the `fopen()` function. This function requires two arguments 1=The name of the file. 2=The mode in which you want to open the file.
- Syntax:
- `FILE *fopen(const char *filename, const char *mode);`
- Reading from a File: Once the file is opened in read mode, you can read from it using functions like `fgetc()`, `fgets()`, or `fread()`.
- Writing to a File: You can write data to a file using `fputc()`, `fputs()`, or `fwrite()`.
- Closing a File: After performing the required operations (read or write), it's essential to close the file using `fclose()` to release resources and ensure data integrity