Module 2 – Introduction to Programming

-Dhruv Makwana

Overview of C Programming

- **Overview of C Programming**

C programming was created by Dennis Ritchie in 1972 at Bell Labs. It was developed from an older language called B. C became famous because it was used to write the UNIX operating system.

C is a powerful and simple language that gives more control to the programmer. Many other languages like C++, Java, and Python came from C.

- **Importance of C Programming**

C is important because it is fast, flexible, and close to hardware. It is used to build operating systems, embedded systems, and device drivers. It also helps to understand how computers work deeply

- **Why C is Still Used Today**

Even today, C is used in many areas like Linux, Windows, mobile devices, and IoT systems. It is still trusted for performance and speed.

Learning C makes your programming base strong and helps in learning other languages easily**.**

Setting Up Environment

- **Installing a C Compiler (e.g., GCC)**

To install a C compiler, first, download and install a compiler like GCCOn Linux, you can install GCC using the command sudo apt-get install gcc. On Windows, you can use MinGW to install GCC. After installation, ensure the compiler is added to the system's PATH variable to compile C programs from the command line.

- **Install GCC Compiler**

1. Go to: https://www.mingw-w64.org/
2. Download the installer.
3. Choose these settings during installation:
   - Architecture: x86_64
   - Threads: posix
   - Exception: see
4. After install:
   Path Set:C:\MinGW\bin
   This Pc-> right click ->properties->advanced system settings
   ->environment variables->click on path -> new -> paste your path
   ->click on OK
5. Open Command Prompt and check by typing: gcc --version
   - If version shows – it's installed correctly.

- **VS Code Setup**

1. Download from: https://code.visualstudio.com/
2. Install the C/C++ Extension:
   - Open VS Code → Extensions (Ctrl+Shift+X)
   - Search: 'C/C++' by Microsoft → Install it
3. Make sure MinGW is installed (as above)

Basic Structure of a C Program

C program is made up of different parts. Each part has its own role.

1) **Header File**

   Its use for include libraries.
   EX. #include<stdio.h>

2) **Main() function**

   Every program start from here
   EX. Int main()
   {
   }

3) **Comment**

   Comment is use for explain code
   Compiler ignore this lines
   There are Two Types of comment:-
   1)Single Line comment (//Single line comment)
   2)Multi Line Comment (/* Multi Line comment*/)

4) **Data Types**

   Its use to Define Type of Variable.
   EX. 1) int -> For whole numbers
   2) float -> For decimal numbers
   3) char -> For single characters

5) **Variables**
   Used to store values. Must be declared before use.
   EX. Int *(data type)* age*(variable)* = 20*(value)*;

Operators in C

1) Arithmetic Operators

Used to perform basic mathematical operations.

Operator	Meaning	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
%	Modulus (remainder)	a % b

2) Relational Operators

Used to compare two values. Result is either true (1) or false (0).

Operator	Meaning	Example
==	**Equal to**	**a == b**
!=	**Not equal to**	**a != b**
>	**Greater than**	**a > b**
<	**Less than**	**a < b**
>=	**Greater or equal**	**a >= b**
<=	**Less or equal**	**a <= b**

3) Logical operators

Used to combine conditions. Result is true (1) or false (0).

Operator	Meaning	Example
&&	Logical AND	a > 0 && b > 0
!	Logical NOT	! (a > b)

4) Assignment Operators

Used to assign values to variables.

Operator	Meaning	Example
=	Assign	a = 10
+=	Add and assign	a += 5 (a = a + 5)
-=	Subtract and assign	a -= 3
*=	Multiply and assign	a *= 2
/=	Divide and assign	a /= 2
%=	Modulo and assign	a %= 2

5) Increment/Decrement Operators

Used to increase or decrease value by 1.

Operator	Meaning	Example
++	Increment	a++ or ++a
--	Decrement	a-- or --a

6) Bitwise Operators

Used to perform operations on binary digits (bits).

Operator	Meaning	Example
&	Bitwise AND	a & b
`	`	Bitwise OR
^	Bitwise XOR	a ^ b
~	Bitwise NOT	~a
<<	Left shift	a << 1
>>	Right shift	a >> 1

7) Conditional (Ternary) Operator

Shorthand for if-else condition.

Syntax:
condition ? expression1 : expression2

Example:
```
int a = 10, b = 20;
int max = (a > b) ? a : b;
```

It means:
If a > b is true, then max = a, otherwise max = b.

Control Flow Statements in C

1) if statement

Checks a condition.

Code:

```
if (a > 0)
 {
  printf("Positive");
 }
```

2) if-else statement

If true → one block, else → another block.

Code:

```
if (a % 2 == 0) {
  printf("Even");
} else {
  printf("Odd");
}
```

3) nested if-else

Used for multiple conditions.

Code:

```
if (marks > 90) {
  printf("Grade A");
} else if (marks > 75) {
  printf("Grade B");
} else {
  printf("Grade C");
}
```

4) switch statement

Checks value from multiple options.

Code:

```
switch(day)
{
   case 1: printf("Monday");
   break;
   case 2: printf("Tuesday");
   break;
   default:
   printf("Invalid");
}
```

Looping in C

1) While loop

Condition is checked first, then code runs.
Used when number of iterations is not known in advance.

Syntax:

```
while (condition)
 {
   // code
 }
```

Example:

```
int i = 1;

        while (i <= 5)
        {
           printf("%d ", i);
           i++;  }
```

2) for loop

Used when the number of times to repeat is known.
Has all control (start, condition, update) in one line.

Syntax:

```
for (int i = 1; i <= 5; i++)

 {

   // code
 }
```

Example:

```
    for (int i = 1; i <= 5; i++)
   {
      printf("%d ", i);
   }
```

3) do-while loop

Code runs at least once, even if condition is false.
Useful when code must run first, then check condition.

Syntax:

```
Do
 {
   // code
 }
while (condition);
```

Example:

```
int i = 1;
do
 {
   printf("%d ", i);
   i++;
  }
while (i <= 5);
```

Loop Control Statements

1) break statement

uses to exit a loop or switch immediately.
Mostly used in for, while, or switch.

Example:

```
for (int i = 1; i <= 5; i++)
 {
    if (i == 3)
         {
                break; // stops loop
         }
    printf("%d ", i);
 }
```

Output: 1 2

2) continue

Used to skip the current iteration and go to the next one.
Example:

```
for (int i = 1; i <= 5; i++)
 {
    if (i == 3)
        {
              continue; // skips 3
         }
    printf("%d ", i);
}
```

Output: 1 2 4 5

3) goto

Used to jump to a labeled statement.
Example:

```
int num = 1;
if (num == 1) {
    goto skip;
}
printf("This won't print");
skip:printf("Jumped here!");
```

Output: Jumped here!

Functions in C

What are Functions in C?

A function is a block of code that performs a specific task.
It helps to reuse code and make programs organized.

1) Function Declaration (Prototype)

Tells the compiler function name, return type, and parameters.

Ex:

```
int add(int a, int b);  // declaration
```

2) Function Definition

The actual code of the function (what it does).

Ex:

```
int add(int a, int b) {   // definition
    return a + b;
}
```

3) Function Call

Used to run the function.

Ex:

```
int result = add(5, 3);  // call
```

Full Example:-

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

int cal()
{
    int a = 6;
    int b=  5;
    int c=a+b;

    return c;
}

int main()
{

      printf("%d",cal());



    return 0;
}
```

ARRAYS

Array is collocation of element with same data type. Array always start with 0.

Example:

```
int marks[5] = {90, 85, 76, 88, 95};
```

There are two type of array

1) **One-Dimensional**

Example:

```
int num[3] = {10, 20, 30};
printf("%d", num[1]);
                                                    // Output: 20
```

2) **Two-dimensional**

Example:

```
int table[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
printf("%d", table[1][2]);
                                                    // Output: 6
```

Pointers

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

Example:

```
int a = 10;
int *p;      // pointer declaration
p = &a;      // storing address of 'a' in pointer 'p'
```

&a -> means "address of a"
*p -> means "value at address stored in p"

- **Pointer Declaration and Initialization**

Declaration:

```
data_type *pointer_name;
```

Initialization:

```
int num = 5;
int *ptr = &num;
```

EX:

```
#include <stdio.h>

int main() {
    int num = 10;
    int *ptr;       // declaring a pointer

    ptr = &num;     // storing address of num in ptr

    printf("Value of num: %d\n", num);
    printf("Address of num: %p\n", &num);
    printf("Value stored in ptr: %p\n", ptr);
    printf("Value at address ptr is pointing to: %d\n", *ptr);
   return 0;

                }
```

String

String:-
A string in C is a collection of characters stored in a character array

1. **strlen()** :- Count length of a string
   Use: To find how many characters are in a string.

   Example:

   strlen("Hello");   // gives 5

2. **strcpy()** :- Copy one string into another

   Use: To duplicate a string.

   Example:

   strcpy(dest, "World");   // dest becomes "World"

3. **strcat() :-** Join two strings

   Use: To make one string by joining two.

   Example:

   strcat("Good ", "Morning");   // makes "Good Morning"

4. **strcmp() :-** Compare two strings

   Use: To check if two strings are same or different.

   Example:

   strcmp("apple", "apple");   // returns 0 (same)

5. **strchr() :-** Find a character in a string

   Use: To search a letter inside a string.

   Example: strchr("Ved Jani", 'J');   // finds 'J' inside string

Structures

In C language, a structure is a way to group different types of data under one name.

For example, if we want to store a student's name, age, and marks, we can put them together inside a structure.

1. Declaring a Structure

We use the struct keyword.

Ex:

```
struct Student {
    char name[20];
    int age;
    float marks;
};
```

2. Initializing a Structure

We can give values when we create a structure variable

Ex:

```
struct Student s1 = {"Ved", 20, 85.5};
```

Or we can assign later:

```
struct Student s2;
s2.age = 21;
s2.marks = 90.0;
strcpy(s2.name, "Jani");
```

3. Accessing Structure Members

We use the dot (.) operator to access members.

Ex:

```
printf("Name: %s\n", s1.name);
printf("Age: %d\n", s1.age);
printf("Marks: %.2f\n", s1.marks);
```

File Handling in C

Importance of File Handling

In C, file handling is very important because it allows us to:

- Store data permanently (not lost when the program ends).
- Read and write data from/to files.
- Manage large data easily.
- Share information between programs using files.

Without file handling, data would only stay in memory (RAM) and be lost when the program stops.

Modes of File:

"r" → read

"w" → write (new file or overwrite)

"a" → append (add at end)

Common File Operations in C

1.Opening a File

To open a file, use the fopen() function

Ex:

```
FILE *file = fopen("data.txt", "r");
```

→Returns a file pointer if successful, or NULL if it fails (for example, if the file doesn’t exist in read mode).

2. Closing a File

Always close the file with fclose() when finished:

Ex:

```
fclose(file);
```

→This ensures that all data is saved properly and resources are freed.

3. Writing to a File

To write data, use fprintf(), fputs(), or fwrite() for different types of output.

Ex:

```
FILE *file = fopen("output.txt", "w"); // "w" means write mode
fprintf(file, "Hello, world!\n");
fclose(file);
```

→fprintf() works like printf() but writes to a file.

→fputs() writes strings, and fwrite() is often used for binary data.

4. Reading from a File

To read data, use fscanf(), fgets(), or fread().

Ex:

```
FILE *file = fopen("input.txt", "r");
char line;
fgets(line, 100, file);
fclose(file);
```

→fscanf() is like scanf(), but reads from a file.

→fgets() reads a whole line, fread() reads binary data.