Lecture 1

Objectives :

- Learn About how Computers work
- Intoduction to Programming languages
- Learn About C programming
- Structure of C programs

How computer Work :

We all have computers, and we use them for a variety of purposes, including watching videos, playing games, performing mathematical calculations, communicating with friends, and many other applications. But the fundamental question is: how do these devices actually work?

The answer lies in their electrical nature. Computers are essentially electrical devices that perform all calculations using electrical signals. The central processing unit (CPU), often referred to as the "brain" of the computer, executes these calculations. To temporarily store data while the computer is in operation, it relies on memory, specifically Random Access Memory (RAM).

Crucially, all information within a computer is represented by electrical signals. This includes data stored in memory and the instructions that the CPU executes. These electrical signals can have only two distinct states: presence, typically represented by the digit "1," and absence, represented by "0."

With this binary representation in mind, we can understand that CPU instructions are essentially sequences of 1s and 0s. This sequence of binary digits is known as machine code, which is the most fundamental level of programming language that the CPU can directly understand

Introduction to Programming languages :

Programming languages are tools that scientists developed to facilitate communication with computers. Instead of writing instructions directly in binary code, which can become incredibly cumbersome for large programs, we can utilize programming languages with their more user-friendly syntax. This simplifies the coding process, making it easier to read, understand, and debug code.

However, computers cannot directly understand the syntax of these high-level programming languages. To bridge this gap, we use a program called a compiler. The compiler first checks the code for any errors and then translates it into machine code, a low-level language consisting of binary instructions that the computer can execute

C programming :

Introduction :

C is one of the most powerful and widely used programming languages. Numerous significant projects have been built using C, including operating systems like Windows and Linux, tools like Git, database management systems, embedded systems for microcontrollers, and many other critical applications.

A key strength of C lies in its ability to directly interact with hardware and manipulate it at a low level. However, a significant limitation of compiled C code is its lack of portability. The resulting machine code is typically specific to the particular processor architecture for which it was compiled. This means that machine code generated for one processor architecture (e.g., Intel x86) will generally not run correctly on another (e.g., ARM).

Text editer Vs Ide :

A text editor represents software where we can write and stylize our text. We can create our C programs using a text editor. However, to run them, we need to install a compiler and compile those programs into machine code before execution.

An IDE (Integrated Development Environment) is a text editor with a compiler integrated within it. This means that with an IDE, we don't need to install a separate compiler. We simply write our programs, click "Compile & Run," and the IDE will compile and execute the program for us. One of the most widely used C IDEs is **Code::Blocks**.

Remarque :

C program files have the extension ".c". If you're using a text editor to write your C code, remember to save the file with the ".c" extension. This tells the compiler that it's a C source code file.

Structure of C program :

C programs typically consist of three main sections:

- **Include Section:** This section includes header files (libraries) that provide access to external functions and definitions. These libraries contain pre-written code for common operations, such as input/output, mathematical functions, and string manipulation.
- **Function and Constant Declarations:**
    - **Function Declarations:** This section declares functions used within the program. A function declaration specifies the function's name, return type, and parameters.

  - **Constant Declarations:** This section defines constants, which are fixed values that do not change during program execution. Using constants improves code readability and maintainability.
- **Main Function:** The main() function is the entry point of every C program,it contains the instructions that the computer will execute when the program is run.

```
#include <stdio.h>


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

Code Explanation :

The code above is a simple C program designed to display the message "Hello, Reader" on the screen. Let's break down how it works:

- **Include Section:**
  The code begins with `#include <stdio.h>` ,this line includes the stdio.h library, which stands for "standard input/output.",this library provides essential functions for interacting with the user, such as `printf()` for printing output to the screen and `scanf()` for reading input from the user.
- **Function and Constant Declarations:**
  In this section, we typically declare any functions or constants that will be used within the program,However, in this specific example, there are no additional functions or constants declared, so this section is empty.
- **Main Function:**
  The `main()` function is the starting point of the program, `int main()` indicates that the `main()` function returns an integer value, the empty parentheses () signify that the `main()` function does not accept any arguments, the code within the curly braces {} is executed when the program runs, `printf("Hello, Reader\n");` uses the `printf()` function to display the message "Hello, Reader" on the screen, finally `return 0;` indicates that the program has executed successfully without any errors.

Variables and Data types :

We can visualize variables as containers or boxes used to store data within a computer program. Variables are used to hold and manipulate data values, such as numbers, text, or other information, throughout the execution of the program

Data types :

In C, variables can hold values that belong to one of the following data types:

- **char** Represents a single character (e.g., 'a', '!', '$'). It typically occupies 1 byte of memory and uses ASCII encoding to represent characters.
- **int** Represents integer numbers (whole numbers without decimal points). For example, 5, -10, 0. Arithmetic operations on integers often result in integer values (e.g., 4 / 3 will typically result in 1).
- **bool:** Represents a logical state. It can be either **true** (often represented internally as 1) or **false** (often represented internally as 0).
- **void:** Represents the absence of a return value. A function declared with a `void` return type does not return any value.
- **float** Represents floating-point numbers (numbers with decimal points), such as 3.14, -2.5, 1.0.
- **long** Similar to `int`, but typically used to represent larger integer values. It allocates more memory than a standard `int`, allowing for a wider range of numbers to be stored.
- **double** Represents floating-point numbers with higher precision than `float`. It uses more memory than `float` to store more decimal places accurately.
- **array** Arrays can be of fixed size or dynamically sized (using pointers, which will be discussed later). Individual elements within an array are accessed using an index, starting from 0. For example, `arr[0]` refers to the first element of the array `arr`.

```
#include <stdio.h>

 int main(){
     int a = 4;                          // this is int
     int b;                              // this is int
     float pi = 3.14;                    // this is float
     double m = 3.497488454;             // this is double
     char a = 'A'                        // this is char
     int arr[] = {4 ,5 , 7};             // int array
     char message[] = "hello";           // char array
     char name[] = {'A', 'l', 'i'};      // char array
     printf("Hello Reader");
     return 0;
 }
```

Remarque :

- To declare a variable, you must first specify its data type, then give it a meaningful name, and finally assign a value to it (optional).
- In C, each statement must be terminated with a semicolon (;).
- To create a character, we use single quotes (' '). To create an array of characters (a string), we can use double quotes (" ").

Constants :

A constant is a variable that has a fixed value and will not change while running our program. An example of a constant is **pi**, which has a fixed known value. There are two ways to declare constants:

- Using the `const` keyword before the type of the variable when creating it.
- Using the `#define` directive at the top of the program, after the include section. When creating constants, it is a good practice to use uppercase letters so that, while working, it’s easy to identify that the variable is a constant.

```
#include <stdio.h>

#define PI 3.14159

int main() {
        const float gravity = 9.81;
        // ... rest of your code ...
        return 0;
}
```

Comments :

Comments are lines of code that the compiler will ignore. They are used to add explanatory notes and improve the readability of your code. There are two ways to create comments in C:

- **Single-line comments:** Begin with // and continue until the end of the current line. All text following // on the same line is ignored by the compiler.
- **Multi-line comments:** Begin with /* and end with */. Any text between these two symbols, including multiple lines, will be ignored by the compiler.

Arthmetic Operators :

C provides basic arithmetic operators for manipulating numbers. These include:

- **Addition (+):** Adds two operands. When used with char types, it adds their ASCII values and returns the resulting character.
- **Subtraction (-):** Subtracts the second operand from the first. Similar to addition, it operates on ASCII values when used with char types.
- **Multiplication (*):** Multiplies two operands. This operator cannot be directly used with char types.
- **Division (/):**
    - For float or double types, it performs regular division and returns the exact result.

    - For int or long types, it performs integer division, which discards any fractional part of the result. It does not round the result.
- **Modulo (%):** Returns the remainder of the division between two operands. It is only applicable for int and long types.
- **Increment (++):** Increments the value of a variable by 1.
- **Decrement (--):** Decrements the value of a variable by 1.

```
#include <stdio.h>

 int main(){
     int a = 4;
     int b = 5;
     int c = a + b;
     int d = a - b;
     int e= a * b;
     int f = a / b;
     int g = a % b;
     printf("c = %i", c);
     printf("d = %i", d);
     printf("e = %i", e);
     printf("f = %i", f);
     printf("g = %i", g);
     return 0;
 }
```

Remarque :

We can create additional compound assignment operators by combining arithmetic operators with the assignment operator (=). These operators provide a concise way to modify the value of a variable.
Here are some examples:

- **+=:** Adds the value on the right-hand side to the variable on the left-hand side.
  For example, a += 4 is equivalent to a = a + 4.
- **-=:** Subtracts the value on the right-hand side from the variable on the left-hand side.
  For example, b -= 2 is equivalent to b = b - 2.
- ***=:** Multiplies the variable on the left-hand side by the value on the right-hand side.
  For example, c = *3 is equivalent to c = c* 3.
- **/=:** Divides the variable on the left-hand side by the value on the right-hand side.
  For example, d /= 5 is equivalent to d = d / 5.
- **%=:** Calculates the modulo (remainder) of the variable on the left-hand side with the value on the right-hand side and assigns the result to the variable.

For example, e %= 2 is equivalent to e = e % 2.

Working with Input and Output :

Output and Printing in C :

We print messages to the screen using the `printf()` function. Within the parentheses, we write the message we want to display.
To include the values of variables within the message, we use placeholder characters:

- **%d** for integers
- **%f** for floating-point numbers
- **%c** for characters
- **%s** for strings

These placeholder characters are then replaced with the actual values of the corresponding variables, which are passed as additional arguments to the `printf()` function.

```
#include <stdio.h>


 int main(){
     int age = 30;
     char name[] = "Alice";
     printf("My name is %s and I am %d years old.\n", name, age);
     return 0;
 }
```

Getting user Input in C :

Getting user input makes your programs more dynamic. Instead of hardcoding values within the program, you can allow the user to provide input, making the program more flexible and reusable.
The `scanf()` function is used to read input from the user. Within the `scanf()` function, you use placeholder characters to specify the data type of the input you expect.
Here are some common placeholder characters:

- **%d:** Reads an integer value from the user.
- **%f:** Reads a floating-point number from the user.
- **%c:** Reads a single character from the user.
- **%s:** Reads a string of characters (a sequence of characters) from the user.

```
#include <stdio.h>


 int main(){
     int age;
     char name[10];
     printf("Enter your name");
     scanf("%s", &name);
     printf("Enter your age");
     scanf("%d", &age);
     printf("My name is %s and I am %d years old.\n", name, age);
     return 0;
 }
```

Tasks :

Task 1 :

Write a program that reads the radius of a circle from the user and then displays its surface area.

Task 2 :

Write a C program that reads the name of a student, then reads three grades (as floating-point numbers) for the student, stores them in an array, and finally displays the student's name, their individual grades, and the average of their grades.