

# Fundamentals of Computers and Computing

CSE 1101

(Introduction to Programming)

Abdullah Al Shiam

(B.Sc. & M.Sc. RU)

Lecturer

Computer Science and Engineering  
Sheikh Hasina University, Netrokona

# Computer programming

- ❑ Computer Programming is defined as the process of creating computer software using a programming Language. Computer programs are written by Human individuals(Programmers)
- ❑ A **computer program** is a step by step set of instructions that a computer has to work through in a logical sequence in order to carry out a particular task. The computer executes these instructions (obeys the instructions) when told to do so by the user.

# Programming Languages

- Programming languages are the vocabulary and set of grammatical rules for instructing a computer to perform specific tasks. There are many different types of programming languages each having a unique set of **keywords** (words that it understands) and a special syntax (grammar) for organising program instructions.

## Program to Display "Hello, World!"

```
#include <stdio.h>
int main() {
    // printf() displays the string inside quotation
    printf("Hello, World!");
    return 0;
}
```

## Input & Variables

- Input operations get data into the programs
- A user is *prompted* to enter data:  
`Write "Enter the price in pounds"`  
`Input PoundPrice`
- Computer programs store data in named sections of memory called *variables*. In the example above, the variable is named **PoundPrice**. The value of a variable can, and often does, change throughout a program.

- Variables are containers for storing data values.
- In C, there are different types of variables (defined with different keywords), for example:
  - **int** - stores integers (whole numbers), without decimals, such as 123 or -123
  - **float** - stores floating point numbers, with decimals, such as 19.99 or -19.99
  - **char** - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes



# Declaring (Creating) Variables

To create a variable, specify the **type** and assign it a **value**:

## Syntax

```
type variableName = value;
```

Where *type* is one of C types (such as `int`), and *variableName* is the name of the variable (such as `x` or `myName`). The **equal sign** is used to assign a value to the variable.

So, to create a variable that should **store a number**, look at the following example:

## Example

Create a variable called `myNum` of type `int` and assign the value `15` to it:

```
int myNum = 15;
```

## Programming Example

- Simple programming problem: *Convert a price from British pounds into Dollars.*
- Pseudocode

Input the price of the item, PoundPrice, in pounds

Compute the price of the item in dollars:

Set DollarPrice = 1.62 \* PoundPrice

Write DollarPrice



# Types of Data

- Numeric Data
  - Integer data, i.e., whole numbers, 10 25 -45 0
  - Floating point data – have a decimal point 23.0, -5.0
- Character data (alphanumeric)
  - All the characters you can type at the keyboard
  - Letters & numbers not used in calculations
- Boolean data
  - TRUE/FALSE

## Program to Add Two Integers

```
#include <stdio.h>
int main() {

    int number1, number2, sum;

    printf("Enter two integers: ");
    scanf("%d %d", &number1, &number2);

    // calculating sum
    sum = number1 + number2;

    printf("%d + %d = %d", number1, number2, sum);
    return 0;
}
```

# Conditionals

- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`
- Equal to: `a == b`
- Not Equal to: `a != b`

- if statement

Three basic formats,

```
if (expression){  
    statement ...
```

```
}
```

```
if (expression) {  
    statement ...
```

```
}else{
```

```
    statement ...
```

```
}
```

# The if Statement

Use the `if` statement to specify a block of C code to be executed if a condition is `true`.

## Syntax

```
if (condition) {  
    // block of code to be executed if the condition is true  
}
```

## Example

```
int x = 20;  
int y = 18;  
if (x > y) {  
    printf("x is greater than y");  
}
```

# The else Statement

Use the `else` statement to specify a block of code to be executed if the condition is `false`.

## 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

```
int time = 20;  
if (time < 18) {  
    printf("Good day.");  
} else {  
    printf("Good evening.");  
}  
// Outputs "Good evening."
```

## Program to Check Even or Odd

```
#include <stdio.h>
int main() {
    int num;
    printf("Enter an integer: ");
    scanf("%d", &num);


    // true if num is perfectly divisible by 2
    if(num % 2 == 0)
        printf("%d is even.", num);
    else
        printf("%d is odd.", num);

    return 0;
}
```



# If...Then...Elseif Statement

- ❑ If there are more than two alternative choices, using just **If...Then....Else** statement will not be enough. In order to provide more choices, we can use the **If...Then...Elseif** Statement. The general format for the if...then.. Else statement is
- ❑ **If** (**condition**) **Then**
- ❑     **expression**
- ❑ **ElseIf** **condition** **Then**
- ❑     **expression**
- ❑ **ElseIf** **condition** **Then**
- ❑     **expression**
- ❑ **Else**
- ❑     **expression**
- ❑ **End If**

- 
- When executing a block **If** (second syntax), *condition* is tested. If *condition* is **True**, the statements following **Then** are executed. If *condition* is **False**, each **Elseif** (if any) is evaluated in turn. When a **True** condition is found, the statements following the associated **Then** are executed. If none of the **Elseif** statements are **True** (or there are no **Elseif** clauses), the statements following **Else** are executed. After executing the statements following **Then** or **Else**, execution continues with the statement following **End If**.

- An example

```
if(score >= 80){  
    A++; }  
else if(score >= 70){  
    A; }  
else if(score >= 60){  
    B++; }  
else if (score>= 50){  
    B; }  
else if (score>= 40){  
    B; }  
else{  
    F;  
}
```

# Write the steps of writing a program in c

- Step 1: Create a file with extension dot c
- Step 2: Write header file at the top of the program
- Step: Declare main function
- Step 4: Declare variable if needed
- Step 5: Make output based on the problem
- Step 6: Compile the source code
- Step 7: Run the source code

What qualities does a good programmer need?

# Low level language vs high level language

## Difference Between High-Level and Low-Level Languages

Parameter	High-Level Language	Low-Level Language
Basic	These are programmer-friendly languages that are manageable, easy to understand, debug, and widely used in today's times.	These are machine-friendly languages that are very difficult to understand by human beings but easy to interpret by machines.
Ease of Execution	These are very easy to execute.	These are very difficult to execute.
Process of Translation	High-level languages require the use of a compiler or an interpreter for their translation into the machine code.	Low-level language requires an assembler for directly translating the instructions of the machine language.
Efficiency of Memory	These languages have a very low memory efficiency. It means that they consume more memory than any low-level language.	These languages have a very high memory efficiency. It means that they consume less energy as compared to any high-level language.



Portability	These are portable from any one device to another.	A user cannot port these from one device to another.
Comprehensibility	High-level languages are human-friendly. They are, thus, very easy to understand and learn by any programmer.	Low-level languages are machine-friendly. They are, thus, very difficult to understand and learn by any human.
Dependency on Machines	High-level languages do not depend on machines.	Low-level languages are machine-dependent and thus very difficult to understand by a normal user.
Debugging	It is very easy to debug these languages.	A programmer cannot easily debug these languages.
Maintenance	High-level languages have a simple and comprehensive maintenance technique.	It is quite complex to maintain any low-level language.
Usage	High-level languages are very common and widely used for programming in today's times.	Low-level languages are not very common nowadays for programming.
Speed of Execution	High-level languages take more time for execution as compared to low-level languages because these require a translation program.	The translation speed of low-level languages is very high.