Unit 2. Elements of C

2.1 C Standards(ANSI C and C99),
2.2 C Character Set (letters, digits, special characters and white spaces),
2.3 C Tokens (keywords, identifiers, operators, constants, and special symbols),
2.4 Escape sequence,
2.5 Delimiters,
2.6 Variables,
2.7 Data types (Basic, Derived, and User Defined),
2.8 Structure of a C program,
2.9 Executing a C program,
2.10 Constants/ Literals,
2.11 Expressions, Statements and Comments

Elements of C

C Standards(ANSI C and C99)

- Everything before standardization is generally called "K&R C". This was "the C language" from 1972-1989.
- **The first C standard was released 1989 nationally in USA, by ANSI. This release is called C89 or ANSI-C. From 1989-1990 this was "the C language".**
- **The year after, the American standard was accepted internationally and published by ISO (ISO 9899:1990). This release is called C90. From 1990-1999, C90 was "the C language".**
- A minor update was released in 1995, sometimes referred to as "C95". The main change was introduction of wide character support.
- **In 1999, the C standard went through a major revision (ISO 9899:1999). This version of the standard is called C99. From 1999-2011, this was "the C language".**
- **In 2011, the C standard was changed again (ISO 9899:2011). This version is called C11. The update had a lot of focus on multi-core, multi-processing and expression sequencing. From 2011-2017, this was "the C language".**
- In 2017, C11 was revised and various defect reports were solved. This standard is informally called C18 and was released as ISO 9899:2018. It contains no new features, just corrections. It is the current version of the C language.

Elements of C

C Character Set (Alphabet of C Language): The characters that constitute the C language. These character sets are:

1. **ENGLISH ALPHABETS(Letters)**

Uppercase letters	A-Z
Lowercase letters	a-z

2. **DIGITS** 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

3. **SPECIAL CHARACTERS**

~ ! @ # $ % ^ & * ( ) _ + = - / < | > ? \ . , ‘ “ : ; [ ] { }

4. **WHITE SPACES**

space bar , tab

Elements of C

C Tokens : Tokens are basic building blocks of a language. C language has following tokens.

1. ***Keywords***: are predefined, reserved words in C and each of which is associated with specific features. These words help us to use the functionality of C language. They have special meaning to the compilers.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	volatile
const	float	short	unsigned

-HGC

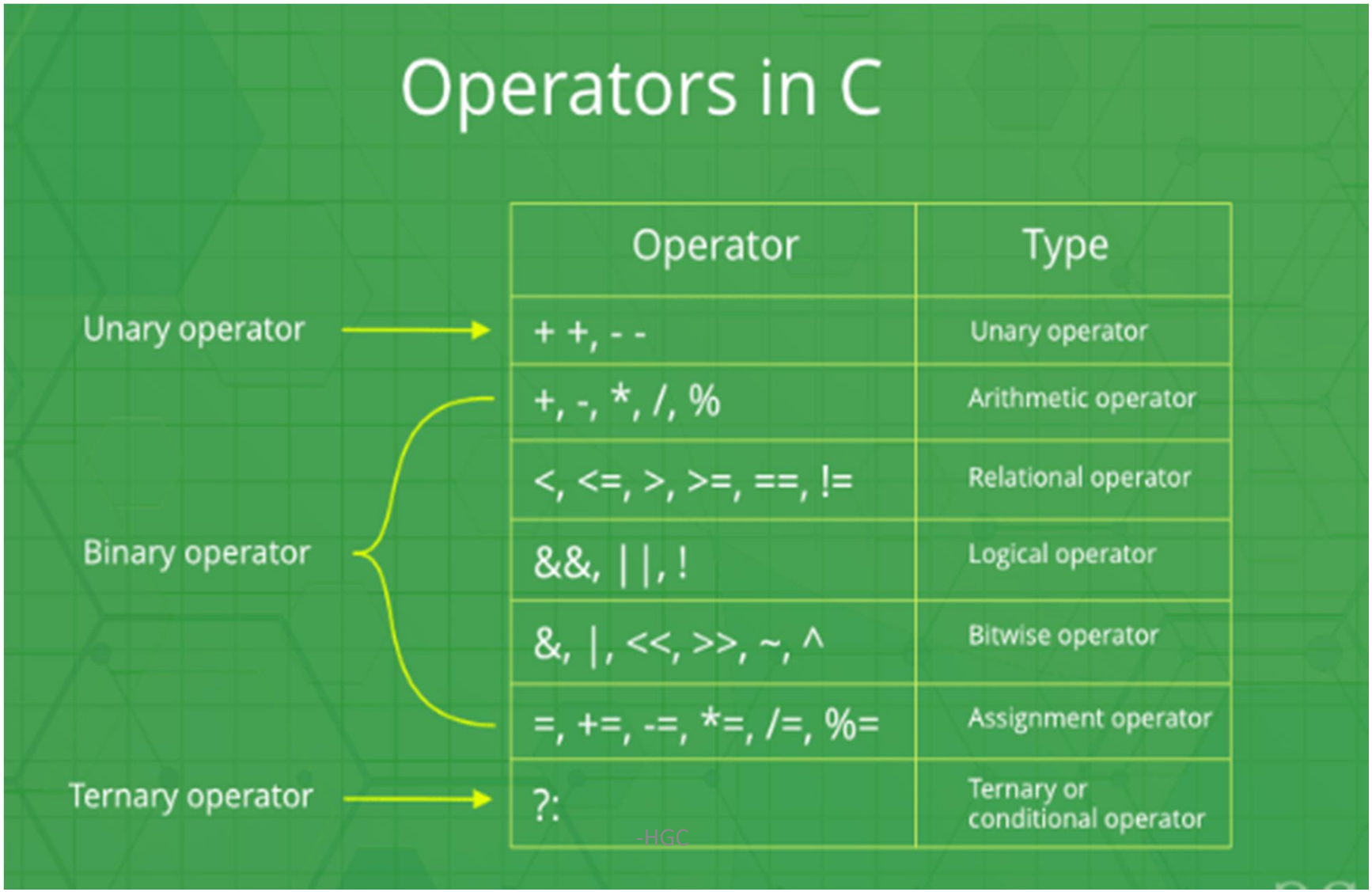
Elements of C (Tokens)

2. Identifiers :

- Program elements those are used for naming of variables, functions, array , user defined types etc are identifiers.
- These are user-defined names which consist of alphabets, number, underscore '_'.
- Identifier's name should not be same on the same scope.
- Keywords cannot not be used as identifiers.
- Below are the rules for naming C identifiers −
  - It must begin with letter or underscore.
  - Only letters, digits and underscore can be used, no other special characters, punctuations are allowed.
  - It must not contain white-space.
  - It should not be a keyword.
  - It should be up to 31 characters long.
    - E.g. Valid identifier: x, a1, _xy, a_b , a12x etc,
    - Invalid: int, 1a, $usd, a.b etc.

Elements of C (Tokens)

3. Operators: can be defined as symbols that help us to perform specific mathematical and logical computations on operands.



Elements of C (Tokens)

4. **Constants:**

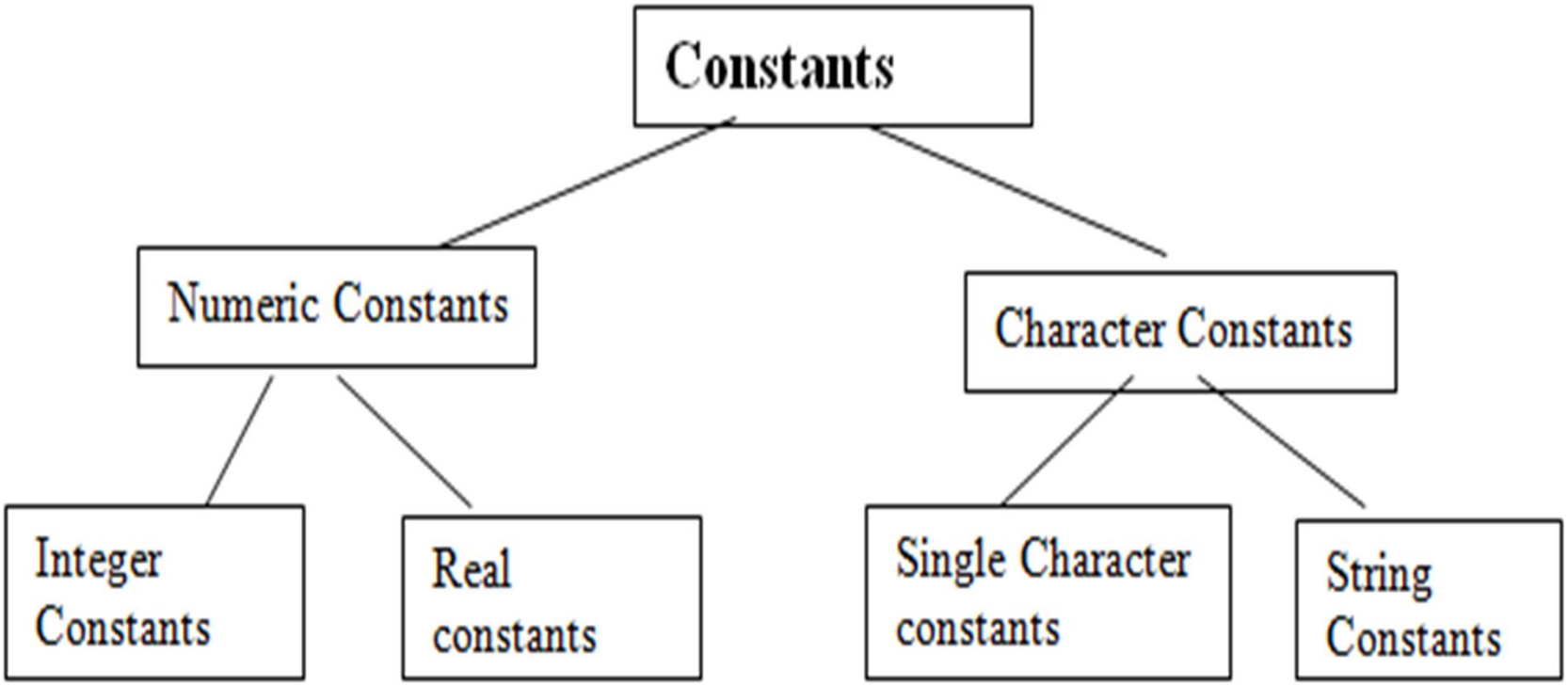
- A constant is a literal or data storage location used by our program where a value can be stored but can't be changed during program execution.
- Constants in C may be numeric or character constants based on their values
- A constants in C can be divided as literal or symbolic constants based on whether it is used directly literal value or it is used as the given name in the program
  - E.g.

    #define PI 22/7 , here PI is symbolic which has given value 22/7

    const int PI = 3.14; here PI is symbolic and 3.14 is literal

More about Constants

Constants in C can be categorized as below based on the values they used



More about Constants

- **Integer Constants** are sequences of digits. There are three types of integers namely decimal ,octal and hexadecimal .

The largest integer value that can be stored is machine-dependent. It is 32767 on 16 bit machine and 2,147,483,647 on 32-bit machine. But it is also possible to store larger integer constants on these machines by appending qualifiers such as U,L and UL to the constants.

e.g.

- 56789U or 56789u for unsigned integer
- 987612347UL or 987612347ul for unsigned long integer.
- 9876543L or 9876543l for long integer.
- **Real Constants** are the numbers with fractional parts and optional leading + or – symbol.

  e.g. 0.9876 , -9.786 +98.765 etc which are in decimal notation.

  0.65e4 , 1.5e+5 -1.2E-1 etc. in scientific notations.
- **Single Character constants** contains a single character enclosed within a pair of single quote marks. Example: ‘c’ ‘%’ ‘&’ ‘7’ etc.

More about Constants

- **A String Constant** is a sequence of characters enclosed in double quotes. The characters may be letters, numbers special characters and blank space.

Example: “Well come”

“a”

“890”

“*&^%$” etc.

- *In addition C supports some special backslash character constants that are used in output functions called escape sequences.*
- *For example: ‘\n’ for newline, ‘\t’ for tab, ‘\b’ for back space, ‘\”’ for double quote ‘\\’ for \ and so on.*

More about Constants

All the types of above constants can be classified in to two main categories.

1. Literal Constants **2. Symbolic Constants.**

- A literal constant is a value that is typed directly into the source code wherever it is needed. Here are two examples:

  int count = 20;

  float tax_rate = 0.28;

- The 20 and the 0.28 are literal constants. The preceding statements store these values in the variables count and tax_rate.
- Note that one of these constants contains a decimal point, whereas the other does not.
- The presence or absence of the decimal point distinguishes floating-point constants from integer constants.

HGP

More about Constants

- A constant starting with the digit 0 is interpreted as an octal integer (the base-8 number system).
- Octal constants can contain the digits 0 through 7 and a leading minus or plus sign.

  e.g. 065 034 etc.

- A constant starting with 0x or 0X is interpreted as a hexadecimal constant (the base-16 number system).
- Hexadecimal constants can contain the digits 0 through 9, the letters A through F, and a leading minus or plus sign.

  e.g. 0x9F 0X7C 0xAB etc.

More about Constants

- A literal constant written with a decimal point is a floating-point constant and is represented by the C compiler as a double-precision number. Floating-point constants can be written in standard decimal notation, as shown in these examples:
  - **123.456 0.019 100.0**

- Floating-point constants also can be written in scientific notation.
- In C, scientific notation is written as a decimal number followed immediately by an E or e and the exponent: Examples below-
- 1.23E2 is 1.23 times 10 to the 2nd power, or 123
- 4.08e6 4.08 times 10 to the 6th power, or 4,080,000
- 0.85e-4 0.85 times 10 to the -4th power, or 0.000085

Symbolic Constants

A symbolic constant is a constant that is represented by a name (symbol) in a program.

- **Whenever we need the constant's value in our program, we use its name.**
- **The actual value of the symbolic constant needs to be entered only once, when it is first defined.**

- **Symbolic constants have two significant advantages over literal constants, as the following example shows. Suppose that we're writing a program that performs a variety of geometrical calculations. The program frequently needs the value ,, (3.14159) for its calculation. For example, to calculate the circumference and area of a circle with a known radius, we could write**
  - **circumference = 3.14159 * (2 * radius);**
  - **area = 3.14159 * (radius)*(radius);**

- **If, however, we define a symbolic constant with the name PI and the value 3.14, we could write**
  - **circumference = PI * (2 * radius);**
  - **area = PI * (radius)*(radius);**
- **If we have to change constant literal, just has to change at once in its definitions.**

Elements of C (Tokens)

5.Delimiters(special symbols):

Delimiters are tokens which are symbols used as a separator for variables, statements etc. for writing program statements. For example comma(,)

semicolon(;)

Space

Elements of C

Scape Sequences: The C languages support a concept called Escape Sequence. When a character is preceded by a backslash (\), it is called an escape sequence and it has a special meaning to the compiler. For example, \n in the following statement is a valid character and it is called a new line character

E.Seq	Description
\t	Inserts a tab in the text at this point.
\b	Inserts a backspace in the text at this point.
\a	Bell Character(Produce a beep sound)
\n	Inserts a newline in the text at this point.
\r	Inserts a carriage return in the text at this point.
\f	Inserts a form feed in the text at this point.
\’	Inserts a single quote character in the text at this point.
\”	Inserts a double quote character in the text at this point.
\\	Inserts a backslash character in the text at this point.
\?	Inserts a question mark characters in the text at this point.

HGC

Elements of C

Data Types:

A data type is the type of the value associated with a variable or constant. C language supports following data types;

1. Primary(or fundamental) data types
2. Derived data types
3. User-defined data types

Primary Data types and associated variables :

There are five basic data types in C associated with variables:

- **int** - integer: a whole number.
- **float** - floating point value: i.e. a number with a fractional part.
- **double** - a double-precision floating point value.
- **char** - a single character.
- **void** – holds no value and used to specify the type of function or what it returns.

Elements of C

Derived Data Types:

- C supports three derived data types:
  1. **Arrays:** Arrays are sequences of data items having homogeneous values. They have adjacent memory locations to store values.
  2. **Pointers:** These are powerful features which are used to access the memory and deal with their address.
  3. **References:** Function pointers allow referencing functions with a particular signatures.

Elements of C

User Defined Data Types.

- C allows the feature called *type definition* which allows programmers to define their identifier that would represent an existing data type. There are three such types:

Structure	It is a package of variables of different types under a single name. This is done to handle data efficiently. "struct" keyword is used to define a structure.
Union	These allow storing various data types in the same memory location. Programmers can define a union with different members, but only a single member can contain a value at a given time. Keyword union is used.
Enum	Enumeration is a special data type that consists of integral constants, and each of them is assigned with a specific name. "enum" keyword is used to define the enumerated data type.

-HGC

Elements of C

Variable:

- A variable is a named data storage location in the computer's memory.
- By using a variable's name in our program, we are, in effect, referring to the data stored there.
- A variable can have different values at different instant of time.
- Based of the value stored in variable, each variable is defined with a data type associated with it.
- The syntax of declaration of variable in C is:
  - <data type><space><variable name>;
    - e.g. **int** x; Here **int** is data type and **x** is variable(identifier)
  - Similarly, float y; char ch; etc.

    int count, num, my_num; Multiple variable of same type.

Data types and their Size C

Following table shows the data types and the byte required and Renges of values for different variable.

Variable Type	Keyword	Size(Bytes)	Range
Character	**char**	1	-128 to 127
Integer	**int**	2	-32768 to 32767
Short integer	**short**	2	-32768 to 32767
Long integer	**long**	4	**-2,147,483,648 to 2,147,438,647**
Unsigned character	**unsigned char**	1	0 to 255
Unsigned integer	**unsigned int**	2	0 to 65535
Unsigned short integer	**unsigned short**	2	0 to 65535
Unsigned long integer	**unsigned long**	4	0 to 4,294,967,295
Single-precision floating-point	**float**	4	3.4E-38 to 3.4E+38
Double-precision floating-point	**double**	8	**1.7 E-308 to 1.7E+308**

Elements of C

Statements:

- A statement is a complete direction instructing the computer to carry out some task.
- In C, statements are usually written one per line, although some statements can span multiple lines.
- C statements always **end with a semicolon** (except for preprocessor directives such as ***#define and #include.***
- For example: ***x = 2 + 3;*** is an assignment statement. It instructs the computer to add 2 and 3 and to assign the result to the variable x.
- ***printf("C Programming");*** ***-*** *is a output statement that prints the text string written in between " " as C Programming*

Elements of C

Statements and White Space :

- The C compiler isn't sensitive to white space.
- When the compiler reads a statement in the source code, it ignores white space.
- Thus, the statement **x=2+3;** is equivalent to this statement: **x = 2 + 3;**
- It is also equivalent to this:

```
x
=
2
+
3;
```

Note: But Within literal string constants, tabs and spaces aren't ignored; they are considered part of the string. **For example in "C Programming Language" the spaces also part of string not ignored.**

Elements of C

- Literal string constants are strings that are enclosed within quotes and interpreted literally by the compiler, space for space.
- Although it's extremely bad form, the following is legal:

```
printf(
"Hello, world!"
);
```

- Below, however, is not legal:

```
printf("Hello,
world!");
```

- To break a literal string constant line, we must use the backslash character (\) just before the break.
- Thus, the following is legal:

```
printf("Hello,\
world!");
```

Elements of C

Null Statements

- If we place a semicolon by itself on a line, we create a *null statement* i.e. a statement that doesn't perform any action. This is perfectly legal in C.

  e.g. the null statement is: ;

Compound Statements

- A compound statement, also called a block, is a group of two or more C statements enclosed in braces. Here's an example of a block:

```
{
   printf("Hello, ");
   printf("world!");
}
```

- In C, a block can be used anywhere a single statement can be used. Note that the enclosing braces can be positioned in different ways. The following is equivalent to the preceding example:

```
{printf("Hello, ");
printf("world!");}
```

- **But it is recommended to use the braces {} at proper positions so that it's easy to find the beginning and end of a block.**
- **Don't spread a single statement across multiple lines if there's no need to do so. Limit statement to one line if possible.**

Elements of C

Expressions

- In C, an expression is anything that evaluates to a numeric value. C expressions come in all levels of complexity.

Simple Expressions

- The simplest C expression consists of a single item: a simple variable, literal constant, or symbolic constant. Here are four expressions:

Expression	***Description***
PI	A symbolic constant (defined in the program)
20	A literal constant
rate	A variable
-1.25	Another literal constant

- A literal constant evaluates to its own value.
- A symbolic constant evaluates to the value it was given when we created it using the #define directive or in const statement
- **A variable evaluates to the current value assigned to it by the program.**

Elements of C

Complex Expressions

- Complex expressions consist of simpler expressions connected by operators. For example: 2 + 8 is an expression consisting of the sub-expressions 2 and 8 and the addition operator +. The expression 2 + 8 evaluates 10.
- We can also write C expressions of great complexity:

  1.25 / 8 + 5 * rate + rate * rate / cost

- Consider the following statement:

  x = a + 10;

- This statement evaluates the expression a + 10 and assigns the result to variable x.
- In addition, the entire statement x = a + 10 is itself an expression that evaluates to the value of the variable on the left side of the equal sign.
- Thus, we can write statements such as the following, which assigns the value of the expression a + 10 to both variables, x and y as:

  y = x = a + 10; More Complex expressino

- We can also write statements such as this:

  x = 6 + (y = 4 + 5);

- All of these are complex expression.

Elements of C

Comments

- Comments are those parts of the source code which are ignored by compiler during compilation.
- In C, comments are anything written between the characters /* and */

  e.g. ***/* This is my First C Program */***
- If multiple lines comments are needed, we can use multiple comments each per one line or multi-line single comment using the same starting and ending characters

  /*this is my first c program */

  /*this program prints a message hello */
- Or equivalently these comments can be written as :

/* this is my first c program

this program prings a message hello

****/***

The First C Program

```
/* A C program to pring Hello world */
#include<stdio.h>  /* Std input output header file */
#include<conio.h>  / * console I/O header file */
main()    / * main function */
{
        printf("Hello world");
        getch();
        return 0;
}
```

- *After comiplation and execution, the output of program:*

  Hello world

End of Unit 2 !

Thank You !!