UNIT: 2
Basics of 'C'

❖ General structure of 'C' program:

Documentation section
Header File
Constants and Global variables Declaration
main () function section
{
Statement(s);
}
User Defined functions and procedures with their body

Fig:structure of 'C' program

Documentation section:

- Documentation consists of a set of comment line(/* */) giving the name of the program.

Header File(Link Section):

- Link section provides instruction to the compiler to link function from the system library(Header Files like #include<stdio.h>).

Constant Declaration:

- Definition section defines all symbolic constants
  Ex: #define PI 3.14

Global declaration section:

- There are some variable that are used more than one function. Such Variables are called global variable and that is outside all the function.

main () function section:

- Every c program must have one main () function section.
- This section contains two parts. Declaration part and executable part.
- Declaration part declares all the variables used in the executable part.
- These two parts must be appearing between opening and closing braces.

User Defined functions and procedures:

- The subprogram section contains all the user defines functions that are called in the main function.

Features of C language:

1) C is portable.
'C' Program can be run on any hardware.

2) C supports variety of built-in functions.(library functions printf() and scanf())

3) C supports bitwise operators.
The C support bitwise operator like AND, OR, NOT etc. normally these operations do not supported by higher level languages.

4) C is modular language.
The program written in from of functions. A C program is group of one or more functions. Dividing the program into small functions makes it to easy develop and maintain the programs.

5) C is Robust language.

❖ Character set:

In the C programming language, the character set refers to a set of all the valid characters that we can use in the source program for forming words, expressions, and numbers.
There are four type of character set in C programming.

- Alphabets - Upper case A-Z , Lower case a-z
- Digit- 0-9
- Special Character - %,&,*,etc.
- White space

❖ 'C' tokens, keyword and identifiers

- **C Token :** The smallest individual unit in a C program is known as C token.

C has six types of token as given below:

- Keywords (float,while)
- Identifier(main,amount)
- Constants(-15.2, 10)
- String("abc", "hello")
- Special Symbols ([,] , *, ! etc)
- Operators (+,-, *,/)

- **Keyword:**
  Keywords are those words whose meaning is predefined.
  - The programmer cannot change the meaning of keywords.
  - There are 32 keywords available in c.
  - All the keywords must be written in lowercase.
  - Examples:

    int if float while

- **Identifiers:**
  - Identifier is the words which are defined by the programmer in a program.
  - Identifier refers to the name of variable, array, function etc.
  - It consists of letters, digits and underscore.
  - First character must be an alphabet or underscore.
  - Keywords cannot be use as identifier.
  - Identifier name are case sensitive.
  - The name of identifier should be meaningful.

❖ Data Types in 'C':

Primary (basic) data type with size and range:

Data type is used to specify which type of value to be stored in variable.

Type	Size(Bytes)	Size(Bits)	Range
char	1	8	-128 to +128
int	2	16	-32768 to +32767
float	4	32	3.4 e -38 to 3.4 e +38
double	8	64	1.7 e -308 to 1.7 e +308

❖ Variable

A **variable** is a name which is used to store a temporary value.

Rules of variables Naming:

- The variable name may consist of letters (A-Z, a-z), digits (0-9) and underscore symbol (_).
  Example: int no_1; int no1; int NO1;
- The variable name must begin with a letter. Some system permits underscore as a first character.
  Example: int 1NO; is not valid
- The length should not be more than eight characters.
- Variable name should not be a keyword.

Example: int case is not valid

- A white space is not allowed in the name of variable.
  Example: int NO 1; is not valid.
- Variable is case sensitive. Uppercase and lowercase are different.
  Example: int MAX; int max;
  Here both MAX and max are treated as different variable.

❖ constant variable:

- Constant means fixed value.
- If the value of a variable remain constant during the execution of a program, the we can declare the variable with **const** at the time of initialization.

Syntax: const DataType ConstantName

Example: const float PI = 3.14;

❖ Introduction of different types of operators :

Operators in C

1) Arithmetic operator
2) Relational operator
3) Logical operator
4) Assignment operator
5) Increment and decrement operator
6) Conditional (ternary)operator
7) Bitwise operator
8) Special operator

Arithmetic operator :

- +, -, *, /, % are arithmetic operators.
- +, -, *, / are used with any data type.
- % is used only with integer type.

Arithmetic Operators:

1) **Binary plus(+):**
   - ✓ It add two operands.
   - ✓ Example :a+b
2) **Binary minus(-):**
   - ✓ It subtracts two operands.
   - ✓ Example :a-b
3) **Multiplication(*):**

- ✓ It multipliers two operands
- ✓ Example :a*b

4) Division(/) :

- ✓ It divides one operand by another operand.
- ✓ Example: a/b

5) Modulo operator (%):

- ✓ It divides one operand by another operand and produce reminder of the division.
- ✓ Example :a%b

Now assume that a=5 and b=2 then

1) a+b =7
2) a-b =3
3) a*b =10
4) a/b =2
5) a%b =1

Relational operator:

Relational operators are used for comparisons.

i) Equal to(= =)
It compares two operands and return TRUE (1) if both operands are equal.

ii) Not equal to (! =)
It compares two operands and return TRUE (1) if both operands are not equal.

iii) Less than(<)
It compares two operands and return TRUE (1) if operand1 is less than operand2.

iv) Less than equal to(<=)
It compares two operands and return TRUE (1) if operand1 is less than or equal to operand2.

v) Greater than(>)
It compares two operands and return TRUE (1) if operand1 is greater than operand2.

vi) Greater than equal to(>=)
It compares two operands and return TRUE (1) if operand1 is greater than or equal to operand2.

Examples:

Suppose a=5, b=7

1) a= =b false
2) a!=b true
3) a<b true
4) a<=b true
5) a>b false
6) a>=b false

Assignment Operator:

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+=	Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand.	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand.	C %= A is equivalent to C =C%A

Logical operator:

- Logical operators are used when we want to combine more than one condition.

Logical AND	**&&**
Logical OR	**||**
Logical NOT	**!**

Logical AND(&&)

- The truth table logical AND (&&) operator is given below:

Condition 1	Condition 2	Answer
False	**False**	**False**
False	**True**	**False**

True	**False**	**False**
True	**True**	**True**

Logical OR(||)

- The truth table logical OR (||) operator is given below:

Condition 1	**Condition 2**	**Answer**
False	**False**	**False**
False	**True**	**True**
True	**False**	**True**
True	**True**	**True**

Logical NOT (!):

- The truth table logical NOT (!) operator is given below:

Condition	**Answer**
False	**True**
True	**False**

- **Examples :**
  Suppose a=5, b=3, c=6;

  1) (a>b) && (a>c) - false
  2) (a>b) || (a>c) - true
  3) !(a>b) - false

Increment and decrement operator:

Increment Operators are the unary operators used to increment or add 1 to the operand value.
The Increment operand is denoted by the double plus symbol (++).
It has two types, Pre Increment and Post Increment Operators**.**

Pre-increment Operator:
The pre-increment operator is used to increase the original value of the operand by 1 before assigning it to the expression.

X = ++A;

In the above example, the value of operand 'A' is increased by 1, and then a new value is assigned to the variable 'X'.

Post increment Operator:

The post-increment operator is used to increment the original value of the operand by 1 after assigning it to the expression.

X = A++;

In the above example, the value of operand 'A' is assigned to the variable 'X'. After that, the value of variable 'A' is incremented by 1.

Pre- decrement Operator:

The pre- decrement operator is used to decreases the value of the operand by 1 before assigning it to the expression.

X = --A;

In the above example, the value of operand 'A' is decreased by 1, and then a new value is assigned to the variable 'X'.

Post decrement Operator:

The post- decrement operator is used to decrement the original value of the operand by 1 after assigning it to the expression.

X = A++;

In the above example, the value of operand 'A' is assigned to the variable 'X'. After that, the value of variable 'A' is decrement by 1.

Bitwise operator:

- The bitwise operators are used to manipulate data at bit level.
- These operators are used **for testing bits** or **shifting them left or right.**
    1) Bitwise AND(&):
    2) Bitwise OR(|):
    3) Bitwise Exclusive OR(^):
    4) Shift left(<<):
    5) Shift right(>>):
    6) One 's complement(~):

Bit 1	Bit 2	Bit1 & Bit2	Bit1 | Bit2	Bit1 ^ Bit2
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Example: Bitwise AND(&)

```
x=5
y=7
z = x & y
```

```
x  - -> 0000 0101
y  - -> 0000 0111
-------------------
z  - -> 0000 0101
```

Example: Bitwise or(|)

x=5, y=7
z = x & y

```
x  - ->  0000 0101
y  - -> 0000  0111
-------------------
z  - ->  0000  0111
```

Example: Bitwise Exclusive OR(^)

x=5, y=7
z = x & y

```
x  - ->  0000 0101
y  - -> 0000  0111
-------------------
z  - ->  0000  0010
```

Conditional operator (ternary operator):

- The conditional operator is also known as ternary operator because it has three operands.
- The general form of ternary operator is follow:
  - **Syntax:** e1 ? e2 : e3 ;
    - where e1 is logical expression(condition) , and evaluated first.
- If the condition is true then the statement is followed the? Is executed(e2 is executed) otherwise the statement followed the : is executed(e3 is executed).

Example:
max=(x>y) ? x : y;

First (x>y) is checked, if it is true then x is assigned to max , otherwise y is assigned to max.

❖ Operator precedence and their associativity:

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

Evaluation of Expressions:

- The expressions are evaluated by performing one operation at a time. The precedence and associativity of operators decide the order of the evaluation of individual operations.
- The expressions with parentheses are evaluated first. If two or more parentheses exist in an expression, the parentheses are evaluated from left to right. In the case of nested parentheses, the innermost parentheses are evaluated first, while the outermost one is evaluated last.
- If there are no parentheses, the order of evaluation of an expression is based on the operator precedence and associativity:

  Example:
  7 * (5 + 15) / (2 * 5) – 3
  = 7*20/10−3.
  = 140/10−3
  =14-3
  =11

❖ Type casting /Type conversion :

- The type conversion are automatic as part of expression evaluation.
- But some times the user needs explicit type conversion.
- Example, int / int is always int. But, if we need the answer in float , then type casting is used.

Syntax:

Example: (type _ name) expression ;

```
float average;
int sum=25, n=10;
average = (float) sum / n;
```

where sum / n results into 2.5 rather than 2, because the division is performed after converting sum in to float and the result of float / int is always float. So,

```
average =(float) sum / n;
    = (float) 25 / 10;
    =  25.0 / 10
    = 2.5
```

❖ Input and Output statements in 'C':

- When we say Input, it means to feed some data into a program. An input can be given in the form of a file or from the command line. C programming provides a set of built-in functions to read the given input and feed it to the program as per requirement.

- When we say Output, it means to display some data on screen, printer, or in any file. C programming provides a set of built-in functions to output the data on the computer screen as well as to save it in text or binary files.

Formatted input and output in 'C':

C provides standard functions scanf() and printf(), to perform formatted inputs and outputs.

scanf() function:

Syntax:

scanf(format_specifiers, &data1, &data2,……);

Example:

scanf("%d %c",&data1,&data2);

printf() function:

Syntax:

printf(format_specifiers, data1, data2,……);

Example:

printf("%d %c",data1,data2);