UNIT 2 : Input/Output Statements and Operators:

2.1 Input/Output statements:

    2.1.1 Concepts of Header files (STDIO,CONIO)

        2.1.1.1 Concepts of pre-compiler directives.

        2.1.1.2 Use of #inlcude and #define

2.2 Input/Output Statements:

    2.2.1 Input statements : scanf(), getc(), getch(), gets(), getchar()

    2.2.2 Output Statements: printf(), putc(),puts(), putchar()

    2.2.3 Type specifiers (formatting strings) : %d, %ld, %f, %c, %s, %lf

2.3 Operators :

2.3.1 Arithmetic operators (+, -, *, /, %, ++, --,)

2.3.2 Logical Operators (&&, ||, !)

2.3.3 Relational Operators (>, <, ==, >=, <=, !=)

2.3.4 Bit-wise operators (&, |, ^ , <<, >>)

2.3.5 Assignment operators (=, +=, -=, *=, /=, %=)

2.3.6 Ternary Operator and use of sizeof() function.

2.4 Important Built-in functions:

2.4.1 Use of <string.h> : (strlen, strcmp, strcpy, strcat,strrev)

2.4.2 Use of <math.h> : (abs(), floor(), round(), ceil(), sqrt(), exp(), log(), sin(), cos(), tan(), pow() and trunc())

2.1 Input/Output statements:

- 2.1.1 Concepts of Header files (STDIO,CONIO)
  - 2.1.1.1 Concepts of pre-compiler directives.
  - 2.1.1.2 Use of #inlcude and #define

Concept of Header Files

- A header file is a file with extension **.h** which contains C function declarations and macro definitions to be shared between several source files.
- There are two types of header files: the files that the programmer writes(**User Defined Header File**) and the files that comes with your compiler(**Built In Header File**).

Include Syntax

- #include <file>
- #include "file"

stdio.h

- The header file stdio.h stands for **Standard Input Output**. It has the information related to input/output functions.
- printf(), scanf(), getc(), putc() etc are in stdio.h

conio.h

- The header file conio.h stands for **Console Input Output**.
- clrscr(), getch(), getche(), etc are in conio.h

Concepts of pre-compiler directives

Use of #inlcude

- The #**include** preprocessor directive is **used** to paste code of given file into current file. It is **used include** system-defined and user-defined header files. If included file is not found, compiler renders error.

Include Syntax

- #include <file>
- #include "file"

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
	printf("FYBCA");
	getch();
}
```

Output:

```
FYBCA
```

#define

- The **#define** directive allows the definition of macros within your source code.
- Macro definitions are not variables and cannot be changed by your program code like variables.
- You generally **use** this syntax when creating constants that represent numbers, strings or expressions

Syntax

- #define CNAME value
  - CNAME : The name of the constant. Most C programmers define their constant names in uppercase, but it is not a requirement of the C Language.
  - Value : The value of the constant.

Example:

```
#include<stdio.h>
#include<conio.h>
#define PI 3.14
void main()
{
    printf("%f",PI);
    getch();
}
```

Output:

```
3.140000
```

2.2 Input/Output Statements:

2.2.1 Input statements : scanf(), getc(), getch(), gets(), getchar()

2.2.2 Output Statements: printf(), putc(),puts(), putchar()

2.2.3 Type specifiers (formatting strings) : %d, %ld, %f, %c, %s, %lf

- **Input** means to provide the program with some data to be used in the program and Output means to display data on screen or write the data to a printer or a file.
- **C** programming language provides many built-in functions to read any given **input** and to display data on screen when there is a need to output the result.

2.2.1 Input Statements

- Formatted Input: scanf()
- Unformatted Input: getc(), getchar(), gets(), getch()

Formatted I/O Statements or scanf()

- When we accept the data using scanf() function, it is formatted statement.
- It refers to an input data that has been arranged in a particular format.
- Example:
- 123   3.14   VNSGU

Syntax:

- scanf("Control String", Arg1, Arg2, Arg3… Argn);
- Where
  - Control string is
    - format specifier or control string. It contains field specification which direct the interpretation of input data.
    - Control string specify the field format in which the data is to be entered.
    - Generally, Control String consist the conversion character % and data type character.
  - Arg1, Arg2, Arg3… Argn specifies the address location of the data.

Example:

```
void main()
{
    int a;
    float b;
    char c;
    clrscr();
    printf("Enter integer, float character:");
    scanf("%d%f%c",&a,&b,&c);
    printf("You have Entered : %d%f%c",a,b,c);
    getch();
}
```

- Inputting Integer values
- Inputting Float values
- Inputting Character values
- Inputting String values

Inputting Integer values

- To input Integer values %d format specifier is used.

Example:

```
void main()
{
    int a;
    clrscr();
    printf("Enter integer:");
    scanf("%d",&a);
    printf("You have Entered : %d",a);
    getch();
}
```

Inputting Float values

- To input Float values %f format specifier is used.

Example:

```
void main()
{
    float b;
    clrscr();
    printf("Enter float :");
    scanf("%f",&b);
    printf("You have Entered : %f",b);
    getch();
}
```

Inputting Character values

- To input Character values %c format specifier is used.

Example:

```
void main()
{
    char c;
    clrscr();
    printf("Enter character:");
    scanf("%c", &c);
    printf("You have Entered : %c",c);
    getch();
}
```

Inputting String values

- To input String values %s format specifier is used.

Example:

```
void main()
{
    char c[100];
    clrscr();
    printf("Enter character:");
    scanf("%s", c);
    printf("You have Entered : %s",c);
    getch();
}
```

- Unformatted Input:
  - getc()
  - getchar()
  - gets()
  - getch()

getc()

- It reads a single character from a given input stream and returns the corresponding integer value (typically ASCII value of read character) on success.
- Syntax:
  - int getc(FILE *stream);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char c;
    clrscr();
    c=getc(stdin);
    printf("You have entered %c",c);
    getch();
}
```

getchar()

- This function is used to take input of a single character.
- Syntax:
  - var name=getchar();

Example

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char c;
    clrscr();
    c=getchar();
    printf("You have entered %c",c);
    getch();
}
```

gets()

- This function is used to take input of a String:
- Syntax:
  - gets(String Variable);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char c[50];
    clrscr();
    gets(c);
    printf("You have entered %s",c);
    getch();
}
```

getch()

- **Getch** is used to hold the output sceen and wait until user gives any type of input(i.e. Until user press any key) so that they can read the character and due to this we able to see the output on the screen.
- Syntax:
  - int getch(void);

- getch() also reads a single character from the keyboard. But it does not use any buffer.

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    clrscr();
    printf("%c",getch());
    getch();
}
```

2.2.2 Output Statements

- Formatted Output: printf()
- Unformatted Output: putc(), puts(), putchar()

Formatted Output or printf()

- printf("Control String",Arg1,Arg2,Arg3,….Argn);
  - Where
  - Control String consist the one of the following:
    - Characters that will be printed on the screen as they appear
    - Format Specification that defines the output format for display its item
    - Escape Sequence characters like \n,\t,\b etc…
  - Arg1, Arg2, Arg3… Argn are the variables or constant which we want to print.

Example:

```
void main()
{
    int a;
    float b;
    char c;
    clrscr();
    printf("Enter integer, float character:");
    scanf("%d%f%c",&a,&b,&c);
    printf("You have Entered : %d\n%f\n%c",a,b,c);
    getch();
}
```

- Displaying Integer values
- Displaying Float values
- Displaying Character values
- Displaying String values

Displaying Integer values

- To Display Integer values %d format specifier is used.

Example:

```
void main()
{
    int a;
    clrscr();
    printf("Enter integer:");
    scanf("%d",&a);
    printf("You have Entered : %d",a);
    getch();
}
```

Displaying Float values

- To Display Float values %f format specifier is used.

Example:

```
void main()
{
    float b;
    clrscr();
    printf("Enter float :");
    scanf("%f",&b);
    printf("You have Entered : %f",b);
    getch();
}
```

Displaying Character values

- To Display Character values %c format specifier is used.

Example:

```
void main()
{
    char c;
    clrscr();
    printf("Enter character:");
    scanf("%c", &c);
    printf("You have Entered : %c",c);
    getch();
}
```

Displaying String values

- To Display String values %s format specifier is used.

Example:

```
void main()
{
    char c[100];
    clrscr();
    printf("Enter character:");
    scanf("%s", c);
    printf("You have Entered : %s",c);
    getch();
}
```

Escape Sequence:

- Escape Sequence is also known as Backslash Character(\).
- It is composed of two or more characters starting with backslash \.
- For example: \n represents new line.

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark
\0	Null

- Unformatted Output:
  - putc()
  - puts()
  - putchar()

putc()

- It writes a character (an unsigned char) specified by the argument **char** to the specified stream and advances the position indicator for the stream.
- Syntax:
  - int putc(int char, FILE *stream)

Example:

puts()

- This function is used to print a String:
- Syntax:
  - puts(String Variable);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char c[50];
    clrscr();
    gets(c);
    puts(c);
    getch();
}
```

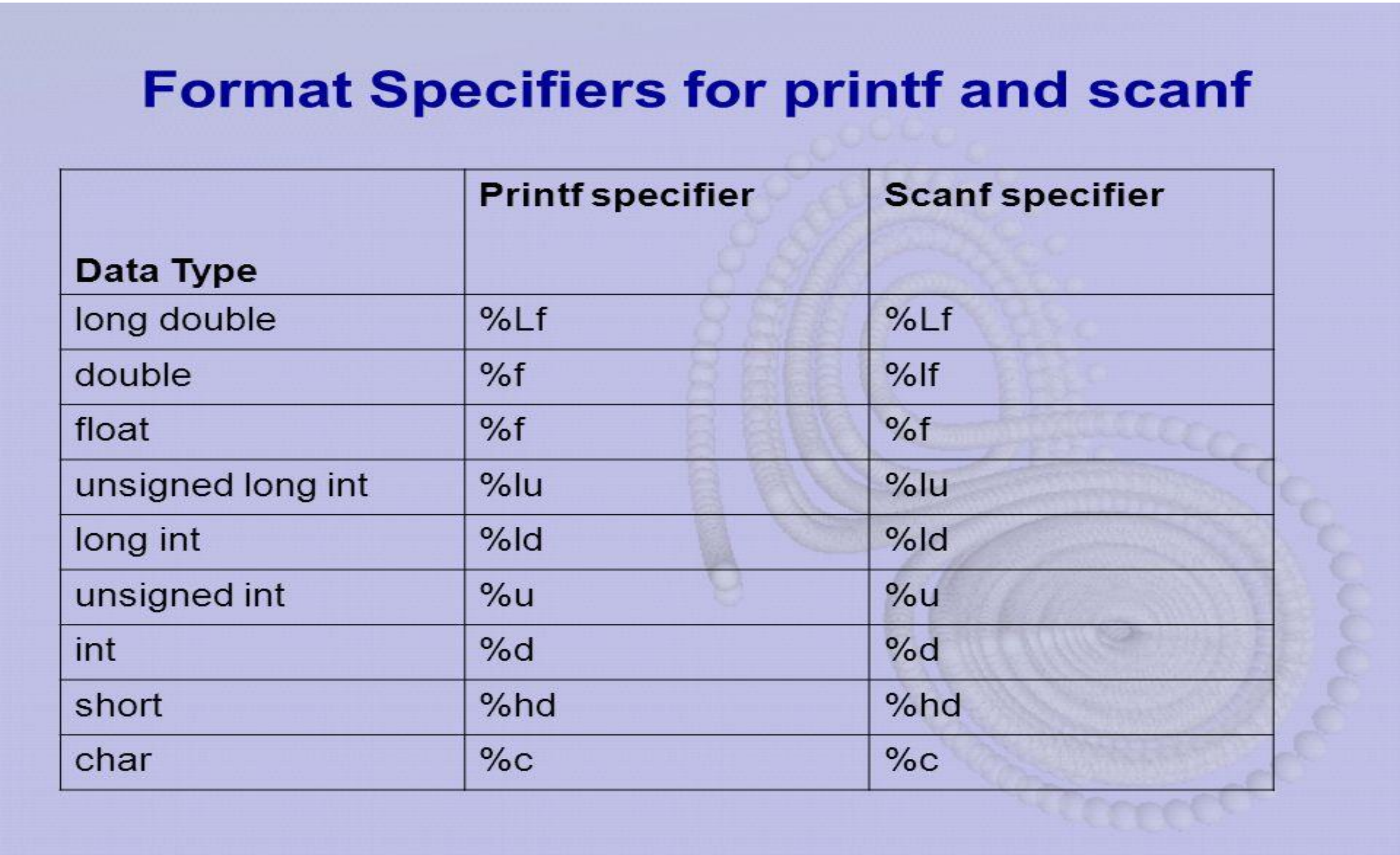
putchar()

- This function is used to take print a single character.
- Syntax:
  - putchar(char variable);

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char c;
    clrscr();
    c=getchar();
    putchar(c);
    getch();
}
```

2.2.3 Type specifiers (formatting strings)



Format Specifiers for printf and scanf

Data Type	Printf specifier	Scanf specifier
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
short	%hd	%hd
char	%c	%c

2.1 Operators

- Operators are symbols that indicates the type of operations, ‘C’ has to perform on data or value of variables.

OR

- An operator is a symbol that operates on a value or a variable.

- ‘C’ has a wide range of operators to perform various operations.

1 Arithmetic operators (+, -, *, /, %, ++, --)

2 Logical Operators (&&, ||, !)

3 Relational Operators (>, <, ==, >=, <=, !=)

4 Bit-wise operators (&, |, ^ , <<, >>)

5 Assignment operators (=, +=, -=, *=, /=, %=)

6 Ternary Operator and use of sizeof() function.

1. Arithmetic Operator

- An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values (constants and variables).

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)

Example:

```
// Working of arithmetic operators
#include <stdio.h>
#include <conio.h>
void main()
{
    int a = 9,b = 4, c;
    c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("a Modulo b = %d \n",c);
    getch();
}
```

Output:

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
a Modulo b=1
```

Unary Operator

Operator	Meaning of Operator
++	Increment
--	Decrement

- Increment operators are used to **increase** the value of the variable by **one** and decrement operators are used to **decrease** the value of the variable by **one** in C programs.
- **Syntax**:

  Increment operator:
  - ++var_name; (or) var_name++;

  Decrement operator:
  - – -var_name; (or) var_name – -;
- **Example**:

  Increment operator : ++ i ;   i ++ ;

  Decrement operator : – – i ;   i – – ;

Operator	Operator/Description
Pre increment operator (++i)	value of i is incremented before assigning it to the variable i
Post increment operator (i++)	value of i is incremented after assigning it to the variable i
Pre decrement operator (--i)	value of i is decremented before assigning it to the variable i
Post decrement operator (i--)	value of i is decremented after assigning it to variable i

Example:

```
// Working of unary operators
#include <stdio.h>
#include <conio.h>
void main()
{
        int a = 5;
        printf("a = %d \n",a);
        printf("a++ = %d \n",a++);
        printf("++a = %d \n",++a);
        printf("a-- = %d \n",a--);
        printf("--a = %d \n",--a);
        printf("a = %d \n",a);

        getch();
}
```

Output:

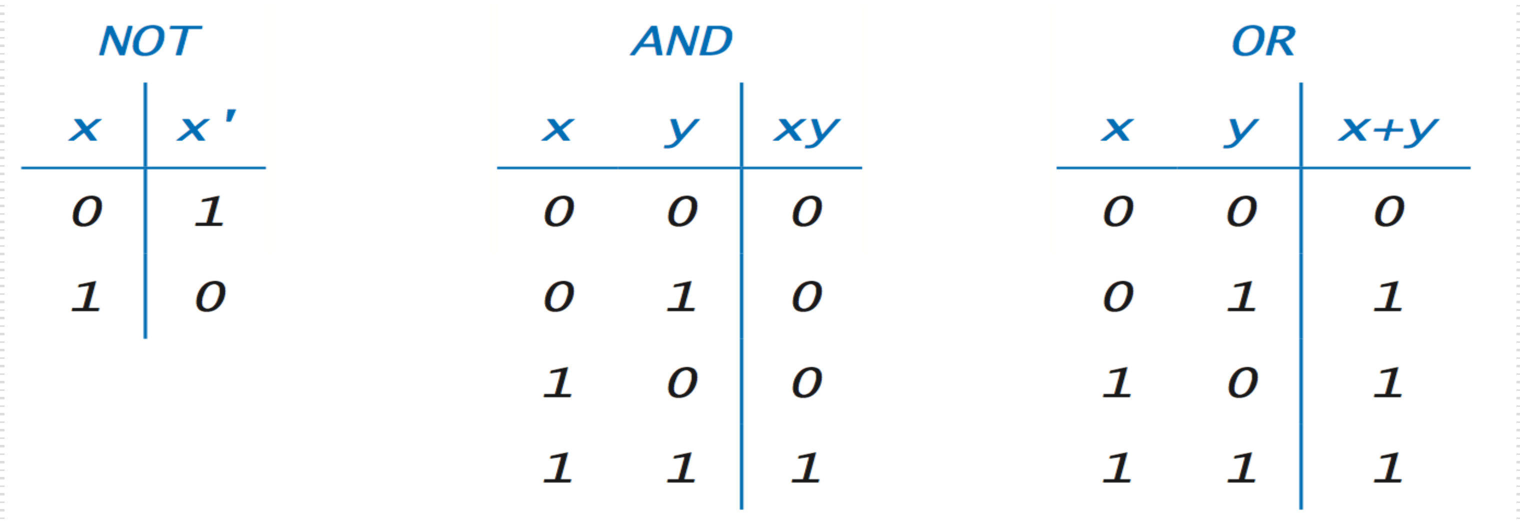
```
a = 5
a++ = 5
++a = 7
a-- = 7
--a = 5
a = 5
```

2. Logical Operator

- These operators are used to perform logical operations on the given expressions.

Operator	Description
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.
||	Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true.
!	Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false.

Truth Table



NOT

x	x'
0	1
1	0

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a = 5, b = 5, c = 10, result;
    result = (a == b) && (c > b);
    printf("%d \n", result);
    result = (a == b) && (c < b);
    printf("%d \n", result);
    result = (a == b) || (c < b);
    printf("%d \n", result);
    result = (a != b) || (c < b);
    printf("%d \n", result);
    result = !(a != b);
    printf("%d \n", result);
    result = !(a == b);
    printf("%d \n", result);
    getch();
}
```

3. Relational or Comparison Operator

- A **relational operator** checks the relationship between two operands.
- If the relation is true then the result of the relational expression is 1, if the relation is false then the result of the relational expression is 0.

Operator	Description
==	Checks if the values of two operands are equal or not. If yes, then the condition becomes true.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.
<	Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true.

Operator	Description	Example	Result
>	Greater than	1 > 2	0
>=	Greater than or equal to	3 >= 2	1
<	Smaller than	10 < 5	0
<=	Smaller than or equal to	6 <= 7	1
==	equal to	98==98	1
!=	not equal to	10 != 9	1

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int x = 12, y = 13;
    printf("x = %d\n", x);
    printf("y = %d\n\n", y);
    printf("x > y : %d\n", x > y);
    printf("x >= y : %d\n", x >= y);
    printf("x < y : %d\n", x < y);
    printf("x <= y : %d\n", x <= y);
    printf("x == y : %d\n", x == y);
    printf("x != y : %d\n", x != y);
    getch();
}
```

Output:

```
x = 12
y = 13

x > y : 0
x >= y : 0
x < y : 1
x <= y : 1
x == y : 0
x != y : 1
```

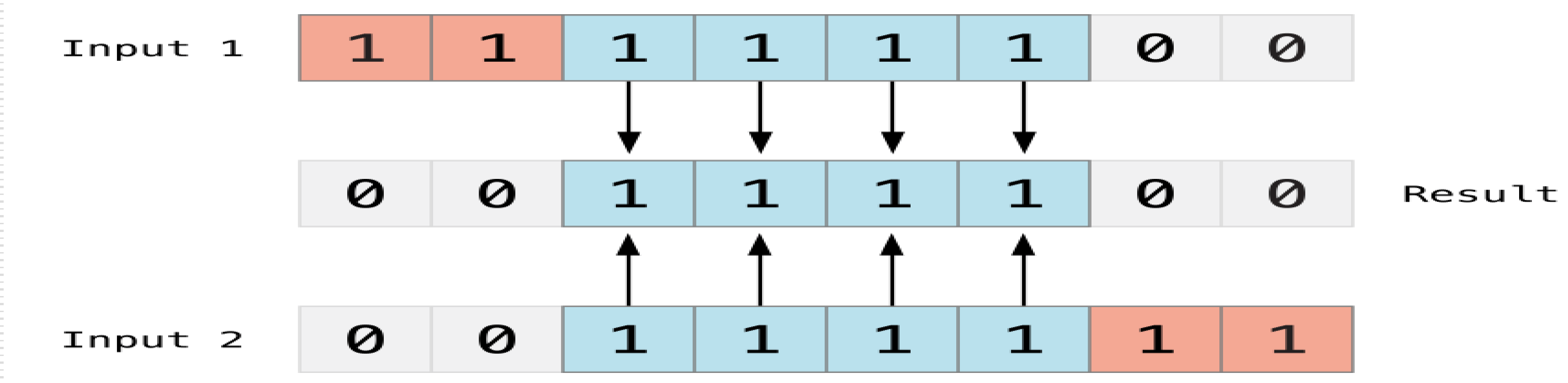
4. Bitwise Operators

- &, |, ^ , <<, >>

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right

Bitwise &

- The *bitwise AND operator* (&) combines the bits of two numbers. It returns a new number whose bits are set to 1 only if the bits were equal to 1 in *both* input numbers:



Example:

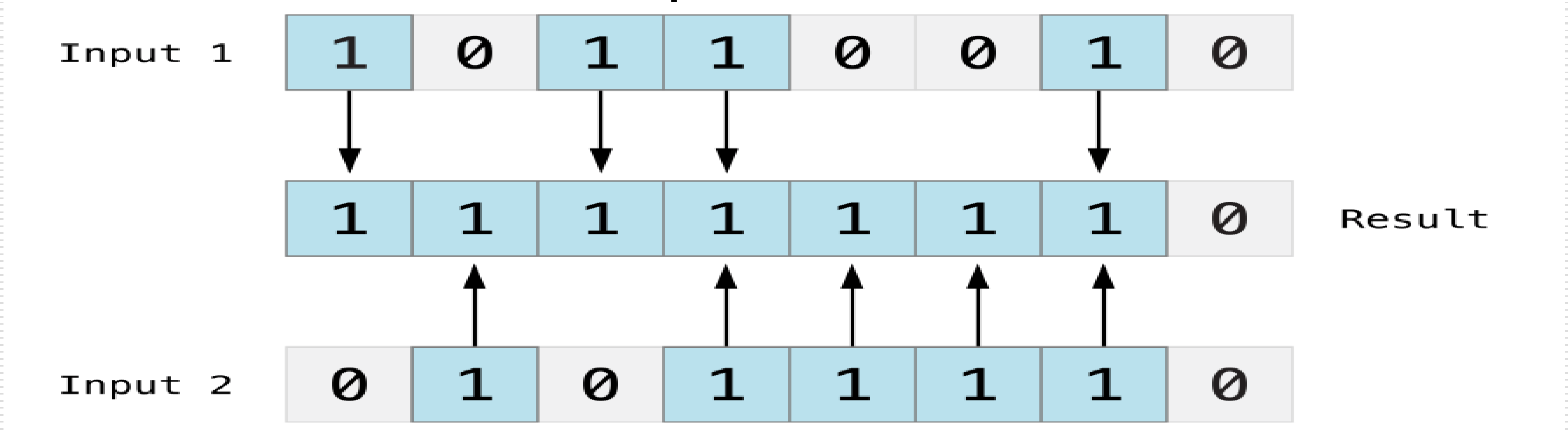
```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=7,b=5;
    clrscr();
    printf("%d",(a&b)); // Bitwise &
}
```

Output:

```
5
```

Bitwise |

- The *bitwise OR operator* (|) compares the bits of two numbers. The operator returns a new number whose bits are set to 1 if the bits are equal to 1 in *either* input number:



Example:

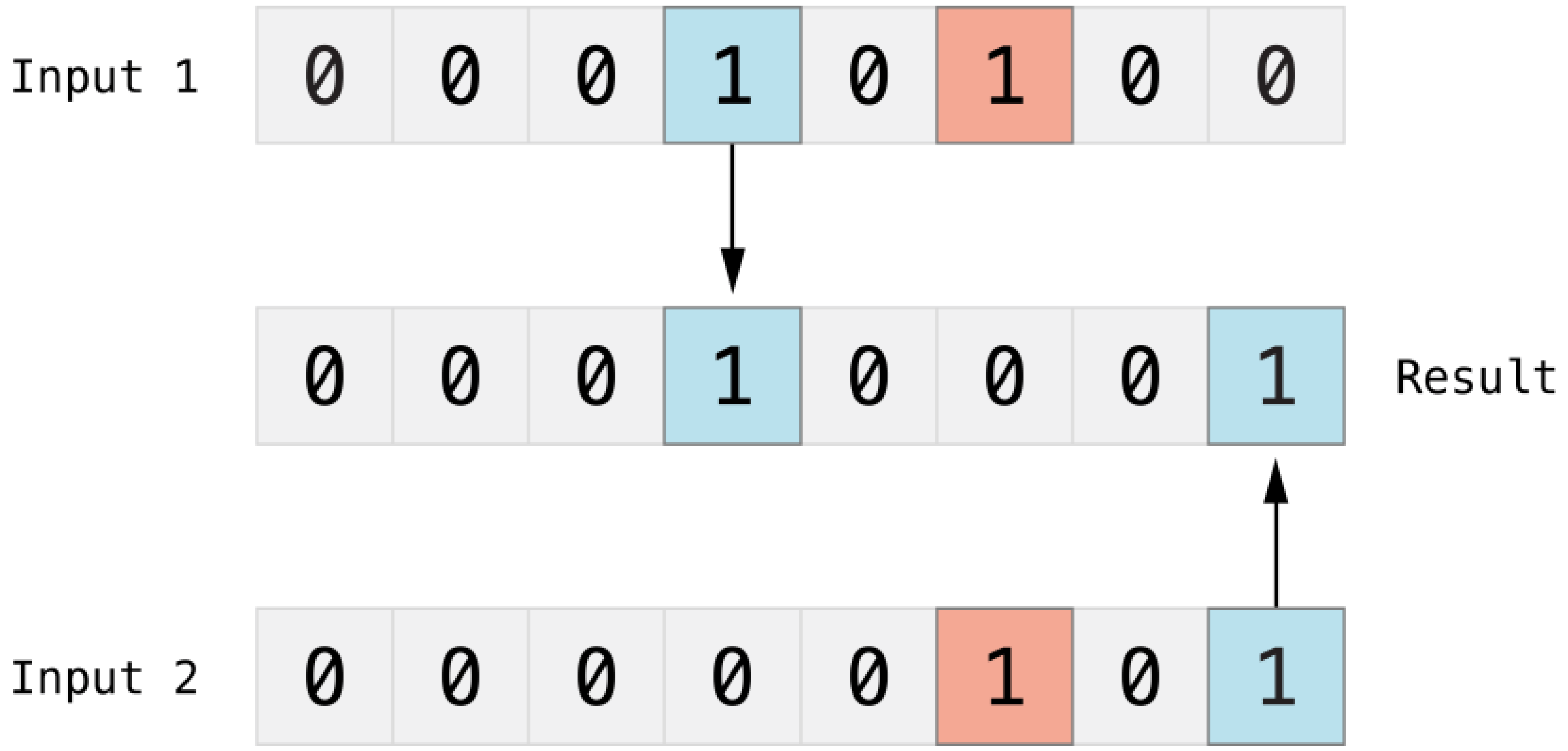
```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=7,b=5;
    clrscr();
    printf("%d",(a|b)); // Bitwise |
}
```

Output:

```
7
```

Bitwise ^ (XOR)

- The *bitwise XOR operator*, or “exclusive OR operator” (^), compares the bits of two numbers. The operator returns a new number whose bits are set to 1 where the input bits are different and are set to 0 where the input bits are the same:


Input 1
0 0 0 1 0 1 0 0
0 0 0 1 0 0 0 1
Result
Input 2
0 0 0 0 0 1 0 1

Example:

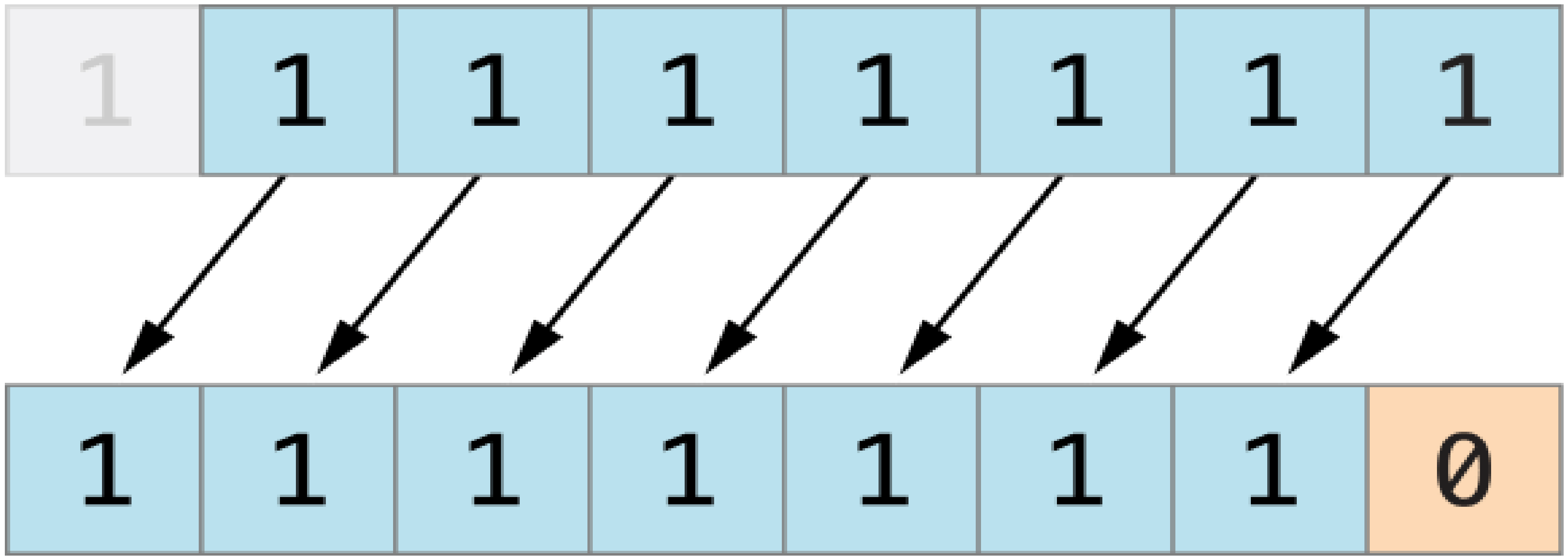
```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=7,b=5;
    clrscr();
    printf("%d",(a^b)); // Bitwise XOR
}
```

Output:

```
2
```

Bitwise << (Shift Left)

- **<< (left shift)** Takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.
- Or in other words left shifting an integer “x” with an integer “y” (x<<y)


1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 0

Example:

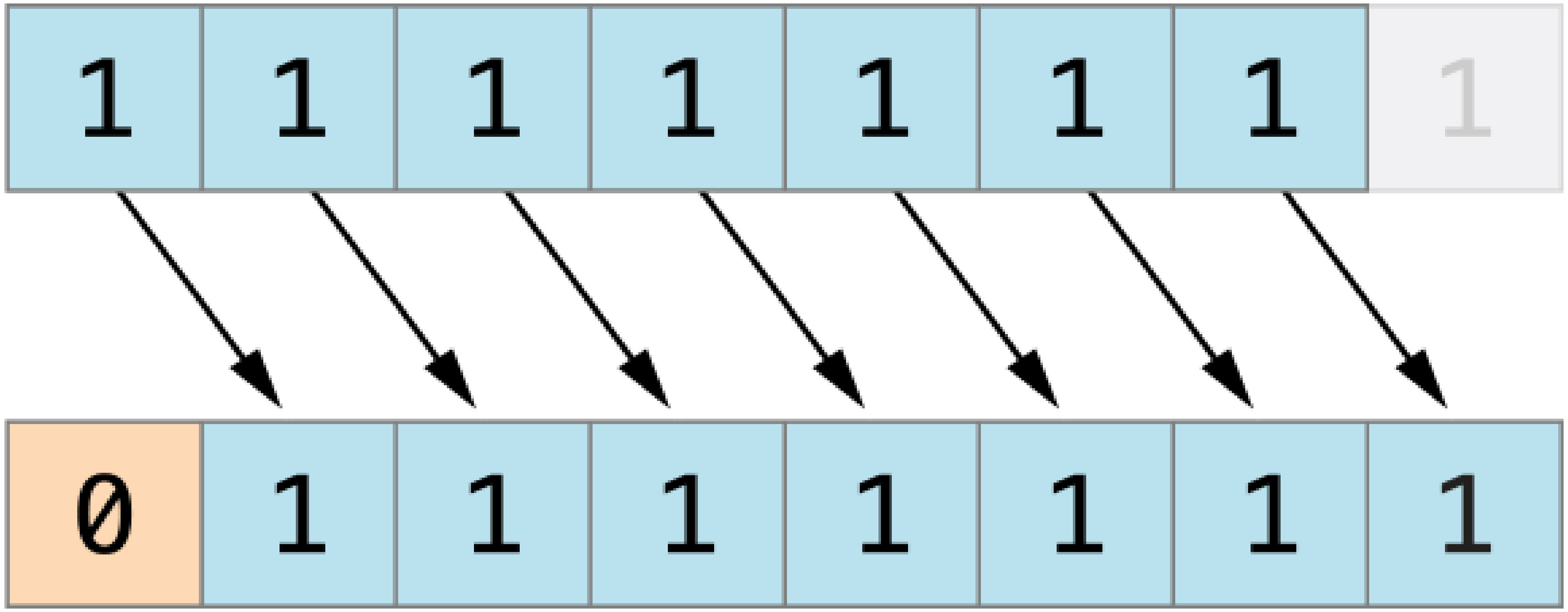
```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=7;
    clrscr();
    printf("%d",(a<<1)); // Bitwise Left Shift
}
```

Output:

```
14
```

Bitwise >> (Shift Right)

- **>> (right shift)** Takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.
- Or in other words right shifting an integer “x” with an integer “y” (x>>y)


1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=7;
    clrscr();
    printf("%d",(a>>1)); // Bitwise RightShift
}
```

Output:

```
3
```

5. 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

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a = 5, c;
    c = a;      // c is 5
    printf("c = %d\n", c);
    c += a;     // c is 10
    printf("c = %d\n", c);
    c -= a;     // c is 5
    printf("c = %d\n", c);
    c *= a;     // c is 25
    printf("c = %d\n", c);
    c /= a;     // c is 5
    printf("c = %d\n", c);
    c %= a;     // c = 0
    printf("c = %d\n", c);
    getch();
}
```

Output:

```
c = 5
c = 10
c = 5
c = 25
c = 5
c = 0
```

6. Conditional or Ternary or ?: Operator

- Conditional operators return one value if condition is true and returns another value is condition is false.

- Syntax:
  - Expr1 ? Expr2 : Expr3
- Expr1 will be condition, it will executes first,
- If Expr1 is true then Expr2 will be executed
- If Expr1 is false then Expr3 will be executed

$$\begin{tikzcd}[column sep=small]
\text{Expr1} \arrow[rrrr, bend left=50, "\text{FALSE}"] \arrow[rr, bend right=50, "\text{TRUE}"'] & ? & \text{Expr2} & : & \text{Expr3}
\end{tikzcd}$$

Example:

```
// Working of Conditional operator
#include <stdio.h>
#include <conio.h>
void main()
{
    int a = 5,b=7,c;
    c=a < b ? a : b;
         printf("%d",c);

         getch();
}
```

Output:

- 5

sizeof() Operator

- It returns the size of a variable. It can be applied to any data type, float type, pointer type variables.
- When sizeof() is used with the data types, it simply returns the amount of memory allocated to that data type.
- The output can be different on different machines like a 32-bit system can show different output while a 64-bit system can show different of same data types.

Example:

```
#include <stdio.h>
int main() {
int a = 16;
   printf("Size of variable a : %d\n",sizeof(a));
   printf("Size of int data type : %d\n",sizeof(int));
   printf("Size of char data type : %d\n",sizeof(char));
   printf("Size of float data type : %d\n",sizeof(float));
   printf("Size of double data type : %d\n",sizeof(double));
   return 0;
}
```

Output:

```
Size of variable a : 2
Size of int data type : 2
Size of char data type : 1
Size of float data type : 4
Size of double data type : 8
```

Expressions

- An expression is a combination of variables, constants and operators.
- every expression results in some value of a certain type that can be assigned to a variable.

- Arithmetic Expression
- Boolean OR Logical Expression

Arithmetic Expression

- An **arithmetic** expression contains only arithmetic operators and operands.
- An Arithmetic expression is a combination of variables, constants and arithmetic operators(+,-,*,/ ,%).

Example:

- 5+7
- 8/2
- a+a
- a-b

Boolean OR Logical Expression

- A Boolean expression is a logical statement that is either TRUE or FALSE.
- An Boolean expression is a combination of variables, constants and Logical operators(&&,||,!).

Example:

- a<b && a<c
- b>a || b>c
- !(a>b)

Algebraic Expression	**C Expression**
a x b – c	a * b – c
(m + n) (x + y)	(m + n) * (x + y)
(ab / c)	a * b / c
$3x^2$ +2x + 1	3*x*x+2*x+1
(x / y) + c	x / y + c
$2\pi r$	2*3.14*r

Evaluation & Assignment Of Expression

- Expressions are evaluated using an assignment statement of the form.
- **Variable = expression;**
- Variable is any valid C variable name. When the statement is encountered, the expression is evaluated first and then replaces the previous value of the variable on the left hand side. All variables used in the expression must be assigned values before evaluation is attempted.

Operator Priorities

- Operator precedence determines the **grouping** of **terms** in an expression and decides **how** an **expression** is **evaluated**. Certain operators have **higher precedence** than others; for example, the **multiplication** operator has a **higher** precedence than the **addition** operator.

- For example, x = 7 + 3 * 2;
- here, x is assigned 13, not 20 because operator * has a higher precedence than +,
- so it first gets multiplied with 3*2 and then adds into 7.

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

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a = 20;
    int b = 10;
    int c = 15;
    int d = 5;
    int e;

    e = a + b * c + d;
    printf("%d\n",  e );

    e = (a+b)+(c+d);
    printf("%d\n" ,  e );

    e = (a + b) * (c / d);
    printf("%d\n",  e );

    e = (a-b)+(c-d);
    printf("%d\n" ,  e );
    getch();
}
```

Output

- 175
- 50
- 90
- 20

Important Built-in functions:

2.4.1 Use of <string.h> : (strlen, strcmp, strcpy, strcat,strrev)

2.4.2 Use of <math.h> : (abs(), floor(), round(), ceil(), sqrt(), exp(), log(), sin(), cos(), tan(), pow() and trunc())

String Functions

- strlen()
- strcmp()
- strcpy()
- strcat()
- strrev()
- strlwr()
- strupr()

strlen()

- strlen() function calculates the length of string.
- **Syntax:**
  - strlen(string_name);
- here string_name is the string which we want length
- Function strlen() returns the value of type integer.

Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
	char a[50]="VNSGU";
	clrscr();
	printf("%d",strlen(a));
	printf("\n%d",strlen("20BCA"));
	getch();
}
```

Output :

```
5
5
```

strcmp()

- strcmp() compares two string and returns value 0, if the two strings are equal.
- Function strcmp() takes two arguments, i.e, name of two string to compare
- **Syntax**:
  - strcmp(string1,string2);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
      char a[50]="VNSGU";
      char b[50]="VNSGU";
      clrscr();
      printf("%d",strcmp(a,b));
      getch();
}
```

Output :

```
0
```

strcpy()

- strcpy() copies the content of one string to the content of another string.
- **Syntax:**
  - strcpy(destination,source);
- Here, source and destination are both the name of the string. This statement, copies the content of string source to the content of string destination.

Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
	char a[50]="VNSGU";
	char b[50]="";
	clrscr();
	strcmp(b,a);
	printf("%s",b);
	getch();
}
```

Output :

```
VNSGU
```

strcat()

- strcat() concatenates(joins) two strings.
- It takes two arguments, i.e, two strings and resultant string is stored in the first string specified in the argument.
- **Syntax:**
  - strcat(first_string,second_string);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
	char a[50]="VNSGU";
	char b[50]="SURAT";
	clrscr();
	strcmp(a,b);
	printf("%s",a);
	getch();
}
```

Output :

```
VNSGUSURAT
```

strrev()

- The strrev() function is used to reverse the given string.
- **Syntax:**
  - strrev(string);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
	char a[50]="VNSGU";
	clrscr();
	printf("%s",strrev(a));
	getch();
}
```

Output :

```
UGSNV
```

Mathematical Functions

- abs()
- floor()
- round()
- ceil()
- sqrt()
- exp()
- log()
- sin()
- cos()
- tan()
- pow()
- trunc()

abs()

- abs ( ) function in C returns the absolute value of an integer.
- The absolute value of a number is always positive. Only integer values are supported in C.
- <u>Syntax :</u>
  - int abs (int n);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
        int a=-77;
        clrscr();
        printf("%d",abs(a));
        printf("\n%d",abs(88));
        getch();
}
```

Output :

```
77
88
```

floor()

- floor( ) function in C returns the nearest integer value which is less than or equal to the floating point argument passed to this function.
- Syntax
  - double floor (double x);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
	float a=7.77;
	clrscr();
	printf("%f",floor(a));
	getch();
}
```

Output :

```
7.000000
```

round()

- round( ) function in C returns the nearest integer value of the float/double/long double argument passed to this function.
- If decimal value is from ”.1 to .5″, it returns integer value less than the argument. If decimal value is from “.6 to .9″, it returns the integer value greater than the argument.
- Syntax:
  - double round (double a);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
	float a=7.77;
	clrscr();
	printf("%f",round(a));
	getch();
}
```

Output :

```
8.000000
```

ceil()

- ceil( ) function in C returns nearest integer value which is greater than or equal to the argument passed to this function.
- <u>Syntax:</u>
  - double ceil (double x);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
	float a=7.77;
	clrscr();
	printf("%f",ceil(a));
	getch();
}
```

Output :

```
8.000000
```

sqrt()

- sqrt( ) function in C is used to find the square root of the given number.
- Syntax:
  - double sqrt (double x);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    clrscr();
    printf("%f",sqrt(64));
    getch();
}
```

Output :

```
8.000000
```

exp()

- The exp() function computes the exponential (Euler's number) raised to the given argument.
- Syntax:
  - double exp(double arg);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    clrscr();
    printf("%f",exp(1));
    getch();
}
```

Output :

```
2.718282
```

log()

- log( ) function is used to calculates natural logarithm.
- Syntax:
  - double log(double arg);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    clrscr();
    printf("%f",log(10));
    getch();
}
```

sin()

- sin() function in C are used to calculate sine values.
- Syntax:
  - double sin(double arg);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    clrscr();
    printf("%f",sin(10));
    getch();
}
```

cos()

- cos() function in C are used to calculate cosine values.
- Syntax:
  - double cos(double arg);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    clrscr();
    printf("%f",cos(10));
    getch();
}
```

tan()

- tan() function in C are used to calculate tangent values.
- Syntax:
  - double tan(double arg);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    clrscr();
    printf("%f",tan(10));
    getch();
}
```

pow()

- pow( ) function in C is used to find the power of the given number.
- Syntax:
  - double pow (double base, double exponent);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    clrscr();
    printf("%f",pow(5,2));
    getch();
}
```

Output :

```
25.000000
```

trunc()

- Truncates a double value after the decimal point and gives the integer part as the result. The return value and the arguments are of the type double.
- ***Syntax:***
  double trunc(double x);

Example:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    clrscr();
    printf("%f",trunc(5.55));
    getch();
}
```

Output :

```
5.000000
```