Unit 2

Basic of C

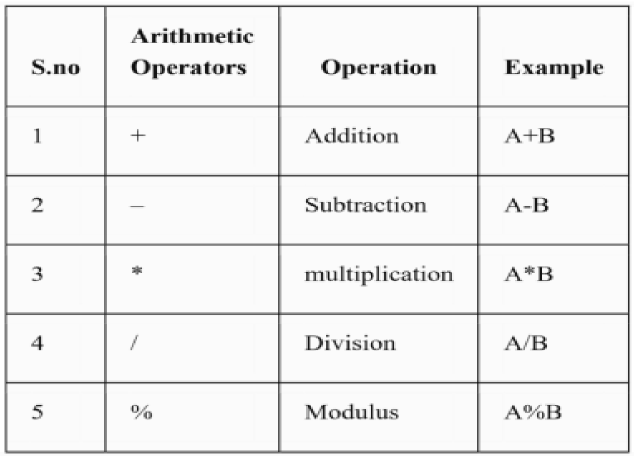
Operator: An operator is a Symbol that operates. An operator acts on some variables called operands to get the desired result. An operator is a symbol that tells the computer to perform mathematical or logical operations on operands. Operators are used in programs to manipulate data. C operators can be classified into several categories:

Types of Operators:

1) Arithmetic Operators
2) Relational Operators
3) Logical Operators
4) Assignment Operators
5) Unary Operators (increment/decrement)
6) Conditional Operators/ Ternary Operators
7) Bitwise Operators

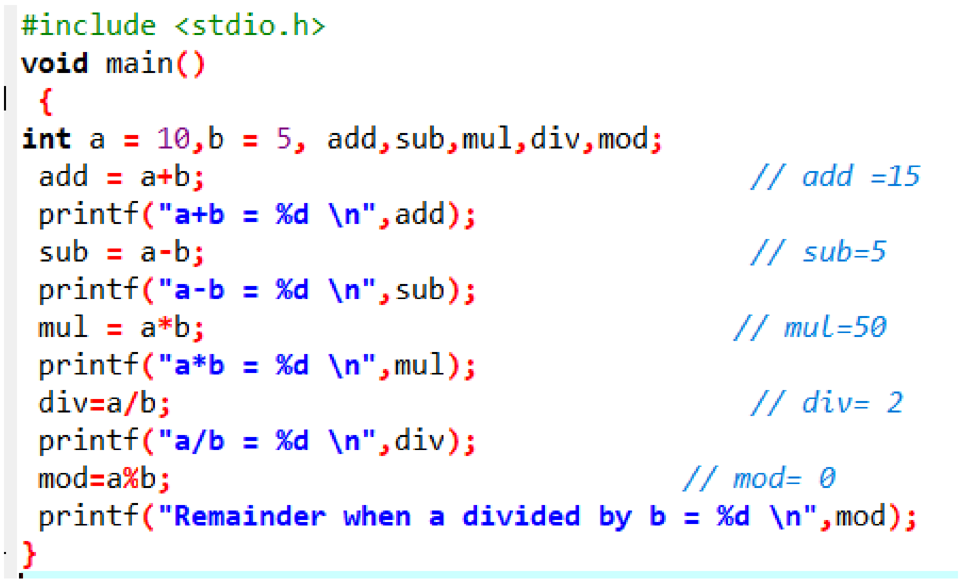
Arithmetic Operators:

An arithmetic operator performs mathematical operations such as addition (+), subtraction (-), multiplication (*), division (/) and modulus division (%) on numerical values (constants and variables).



S.no	Arithmetic Operators	Operation	Example
1	+	Addition	A+B
2	–	Subtraction	A-B
3	*	multiplication	A*B
4	/	Division	A/B
5	%	Modulus	A%B

//C Program to demonstrate the working of arithmetic operators



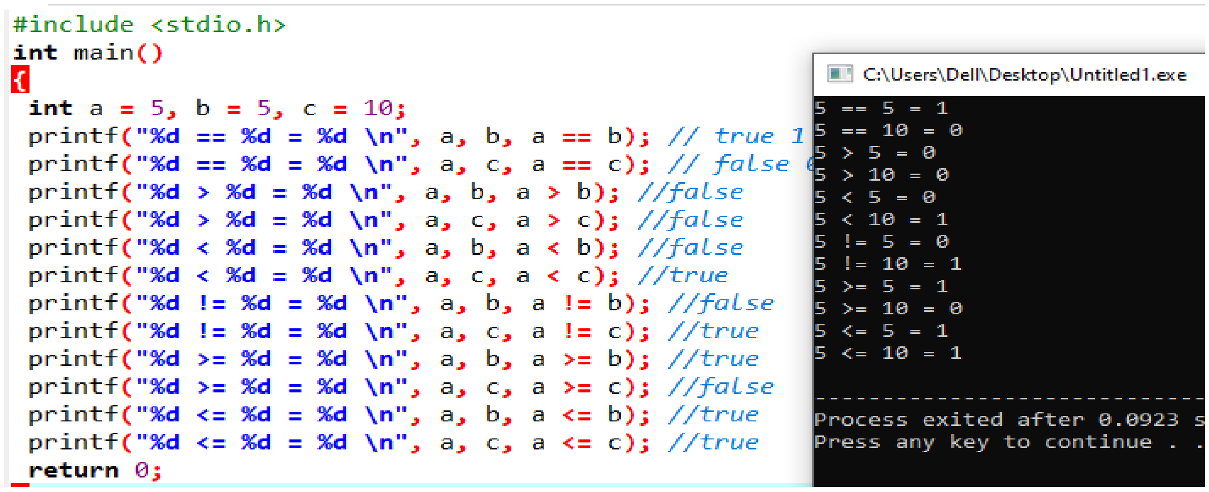
```
#include <stdio.h>
void main()
 {
int a = 10,b = 5, add,sub,mul,div,mod;
 add = a+b;                                   // add =15
 printf("a+b = %d \n",add);
 sub = a-b;                                   // sub=5
 printf("a-b = %d \n",sub);
 mul = a*b;                                  // mul=50
 printf("a*b = %d \n",mul);
 div=a/b;                                     // div= 2
 printf("a/b = %d \n",div);
 mod=a%b;                                 // mod= 0
 printf("Remainder when a divided by b = %d \n",mod);
}
```

Relational Operators:

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0. Operands may be variables, constants or expressions. Relational operators are used in decision-making and loops.

Operator	Meaning	Example	Return value
<	is less than	2<9	1
<=	is less than or equal to	2 <= 2	1
>	is greater than	2 > 9	0
>=	is greater than or equal to	3 >= 2	1
==	is equal to	2 == 3	0
!=	is not equal to	2!=2	0

// C Program to demonstrate the working of relational operators



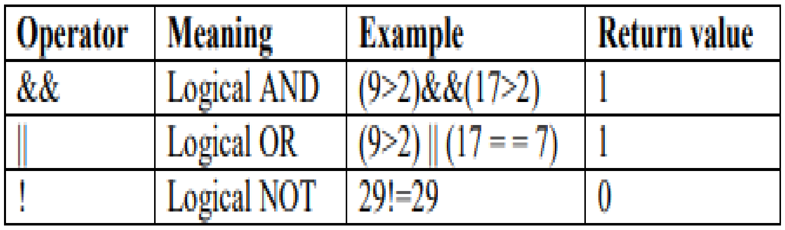
```
#include <stdio.h>
int main()
{
 int a = 5, b = 5, c = 10;
 printf("%d == %d = %d \n", a, b, a == b); // true 1
 printf("%d == %d = %d \n", a, c, a == c); // false
 printf("%d > %d = %d \n", a, b, a > b); //false
 printf("%d > %d = %d \n", a, c, a > c); //false
 printf("%d < %d = %d \n", a, b, a < b); //false
 printf("%d < %d = %d \n", a, c, a < c); //true
 printf("%d != %d = %d \n", a, b, a != b); //false
 printf("%d != %d = %d \n", a, c, a != c); //true
 printf("%d >= %d = %d \n", a, b, a >= b); //true
 printf("%d >= %d = %d \n", a, c, a >= c); //false
 printf("%d <= %d = %d \n", a, b, a <= b); //true
 printf("%d <= %d = %d \n", a, c, a <= c); //true
 return 0;
```

C:\Users\Dell\Desktop\Untitled1.exe

```
5 == 5 = 1
5 == 10 = 0
5 > 5 = 0
5 > 10 = 0
5 < 5 = 0
5 < 10 = 1
5 != 5 = 0
5 != 10 = 1
5 >= 5 = 1
5 >= 10 = 0
5 <= 5 = 1
5 <= 10 = 1

--------------------------------
Process exited after 0.0923 s
Press any key to continue . .
```

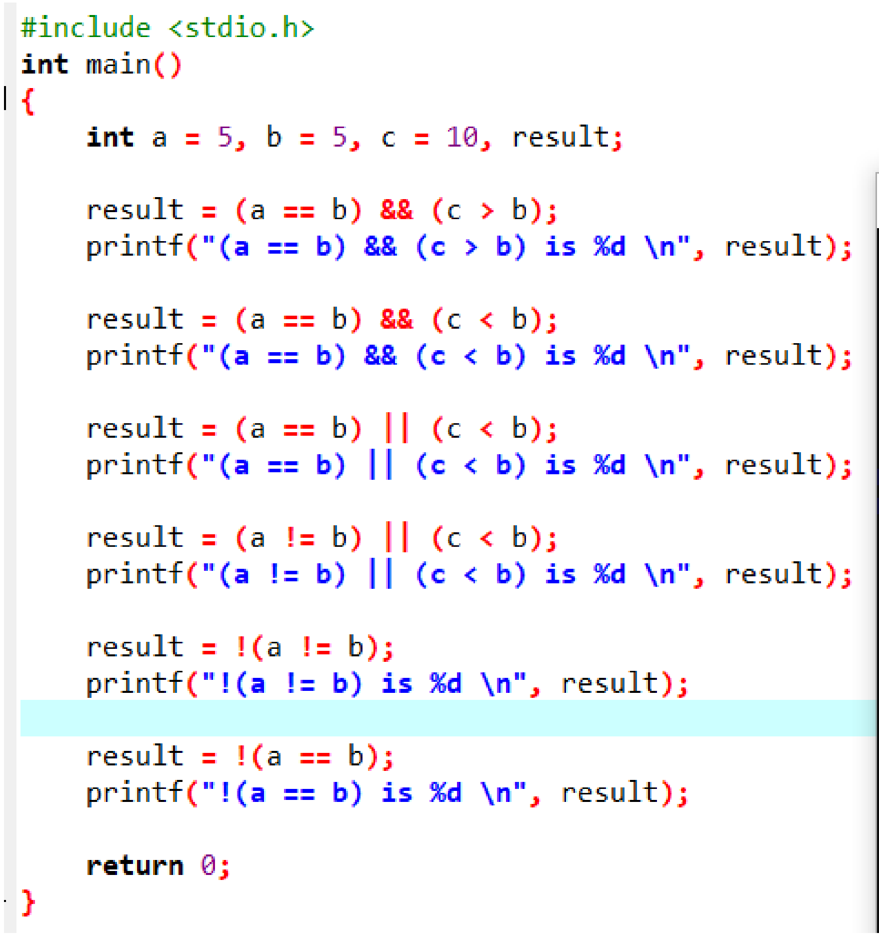
Logical operators :

Expressions that use logical operators are evaluated to be either true (1) or false (0). There are generally three logical operators used in C, Logical AND (&&), Logical OR (||) and Logical NOT (!).

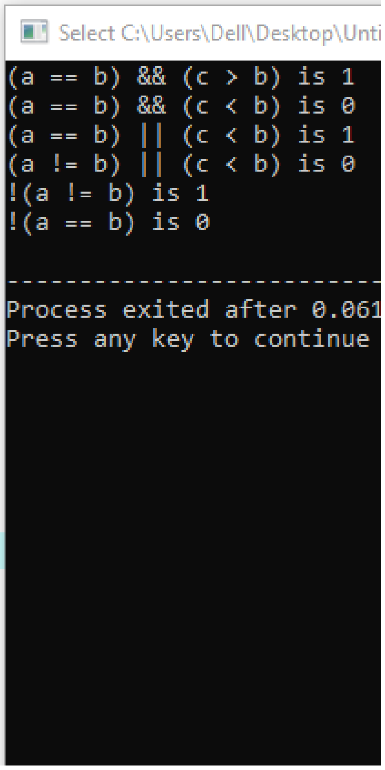


Operator	Meaning	Example	Return value
&&	Logical AND	(9>2)&&(17>2)	1
||	Logical OR	(9>2) || (17 = = 7)	1
!	Logical NOT	29!=29	0

// C Program to demonstrate the working of logical operators



```c
#include <stdio.h>
int main()
{
    int a = 5, b = 5, c = 10, result;

    result = (a == b) && (c > b);
    printf("(a == b) && (c > b) is %d \n", result);

    result = (a == b) && (c < b);
    printf("(a == b) && (c < b) is %d \n", result);

    result = (a == b) || (c < b);
    printf("(a == b) || (c < b) is %d \n", result);

    result = (a != b) || (c < b);
    printf("(a != b) || (c < b) is %d \n", result);

    result = !(a != b);
    printf("!(a != b) is %d \n", result);

    result = !(a == b);
    printf("!(a == b) is %d \n", result);

    return 0;
}
```



Select C:\Users\Dell\Desktop\Unti

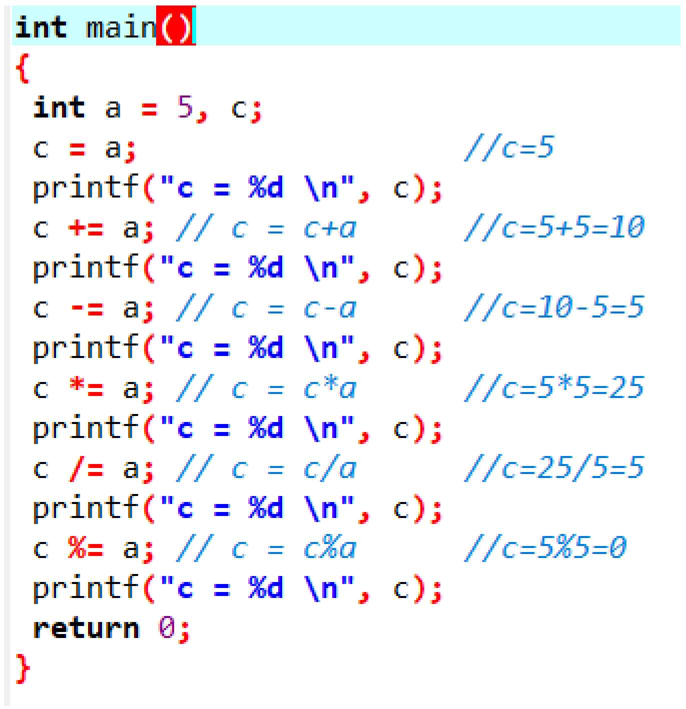
```
(a == b) && (c > b) is 1
(a == b) && (c < b) is 0
(a == b) || (c < b) is 1
(a != b) || (c < b) is 0
!(a != b) is 1
!(a == b) is 0

--------------------------
Process exited after 0.061
Press any key to continue
```

Assignment Operator:

An assignment operator assigns a value to a variable and is represented by "=" (equals to) symbol.
Syntax: Variable name = expression.
The **Compound assignment operators** (+=, *=, -=, /=. %=) are also used.

Operator	Example	Meaning
+=	x += y	x=x+y
- =	x - = y	x=x-y
* =	x * = y	x=x*y
/ =	x / = y	x=x/y
% =	x % = y	X=x%y

// C Program to demonstrate the working of assignment operators



```
int main()
{
 int a = 5, c;
 c = a;                    //c=5
 printf("c = %d \n", c);
 c += a; // c = c+a        //c=5+5=10
 printf("c = %d \n", c);
 c -= a; // c = c-a        //c=10-5=5
 printf("c = %d \n", c);
 c *= a; // c = c*a        //c=5*5=25
 printf("c = %d \n", c);
 c /= a; // c = c/a        //c=25/5=5
 printf("c = %d \n", c);
 c %= a; // c = c%a        //c=5%5=0
 printf("c = %d \n", c);
 return 0;
}
```

Unary Operators:

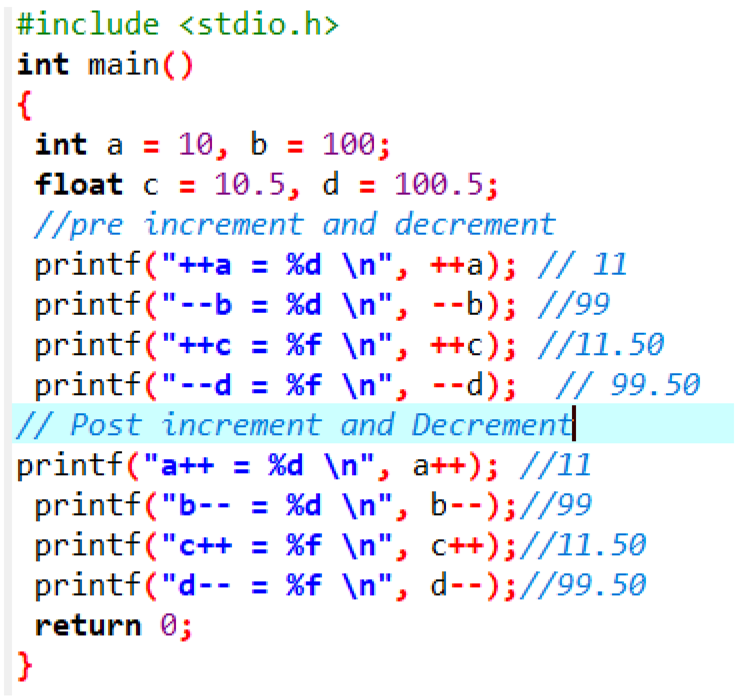
A operator acts up on a single operand to produce a new value is called a unary operator.

a. Increment/Decrement Operator:

The increment operator (++) increases its operand by 1 and the decrement (--) decreases its operand by 1. They are the unary operators (i.e. they operate on a single operand). It can be written as:
x++ or ++x (post and pre-increment)
x—or –x (post and pre decrement)

Operator Meaning
++x Pre increment
- -x Pre decrement
x++ Post increment
x-- Post decrement
Where

- ++x : Pre increment, first increment, and then do the operation.
- - -x : Pre decrement, first decrements, and then do the operation.
- x++ : Post increment, first do the operation and then increment.
- x- - : Post decrement, first do the operation and then decrement.



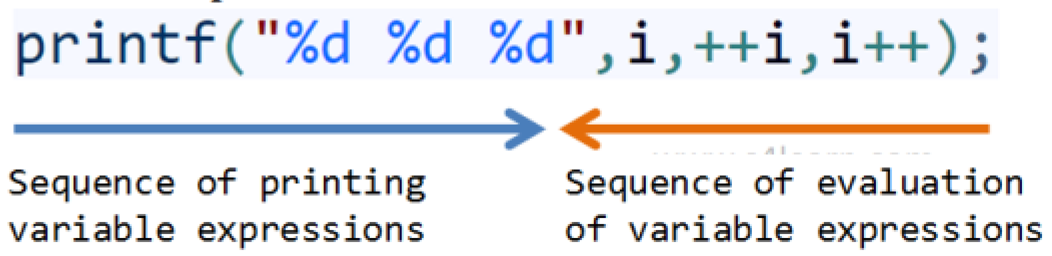
```
#include <stdio.h>
int main()
{
  int a = 10, b = 100;
  float c = 10.5, d = 100.5;
  //pre increment and decrement
  printf("++a = %d \n", ++a); // 11
  printf("--b = %d \n", --b); //99
  printf("++c = %f \n", ++c); //11.50
  printf("--d = %f \n", --d);  // 99.50
// Post increment and Decrement
printf("a++ = %d \n", a++); //11
  printf("b-- = %d \n", b--);//99
  printf("c++ = %f \n", c++);//11.50
  printf("d-- = %f \n", d--);//99.50
  return 0;
}
```

Multiple increment operators inside printf

```
#include<stdio.h>
void main() {
  int i = 1;
  printf("%d %d %d", i, ++i, i++);
}
```

Output : 3 3 1

Pictorial representation



```
printf("%d %d %d",i,++i,i++);
```

Sequence of printing variable expressions
Sequence of evaluation of variable expressions

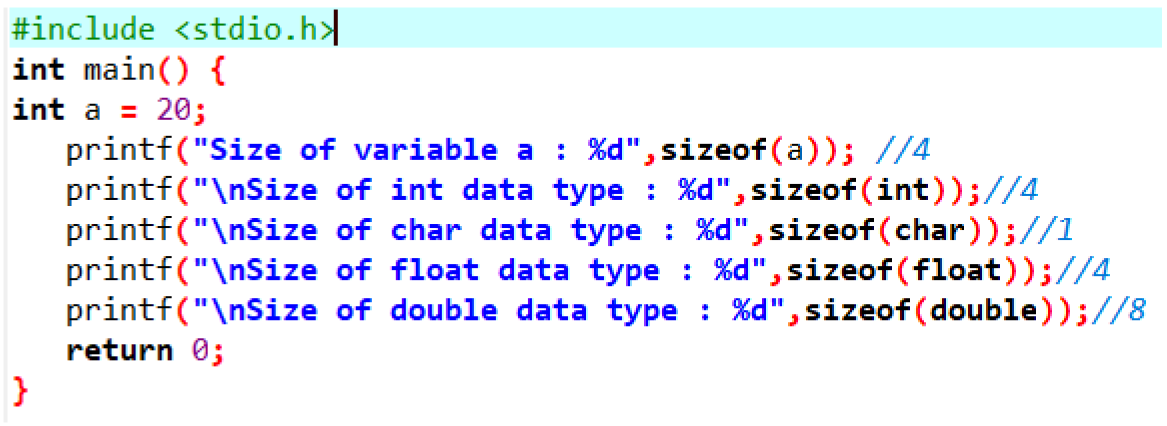
Explanation of program

1. Whenever more than one format specifiers (i.e %d) are directly or indirectly related with same variable (i,i++,++i) then we need to evaluate each individual expression from right to left.
2. As shown in the above image evaluation sequence of expressions written inside printf will be – i++,++i,i
3. After execution we need to replace the output of the expression at the appropriate place

No	Step	Explanation
1	Evaluate i++	At the time of execution we will be using older value of i = 1
2	Evaluate ++i	At the time of execution we will be increment value already modified after step 1 i.e i = 3
2	Evaluate i	At the time of execution we will be using value of i modified in step 2

b. **Sizeof Operator:**

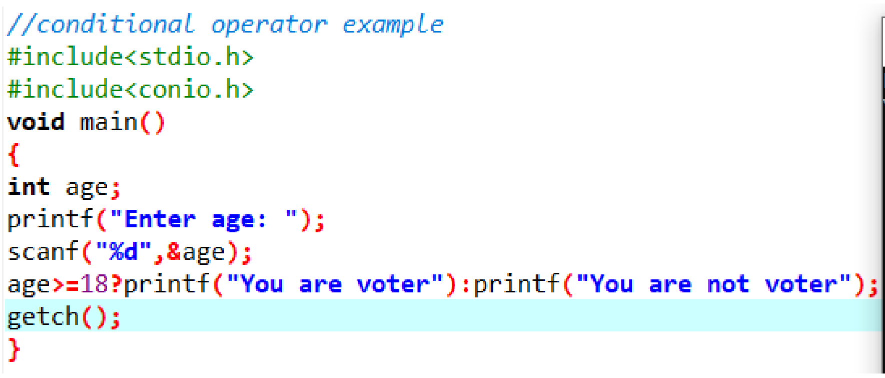
The **sizeof()** operator is commonly used in C. It determines the size of the expression or the data type specified in the number of char-sized storage units. When **sizeof**() is used with the data types, it simply returns the amount of memory allocated to that data type.



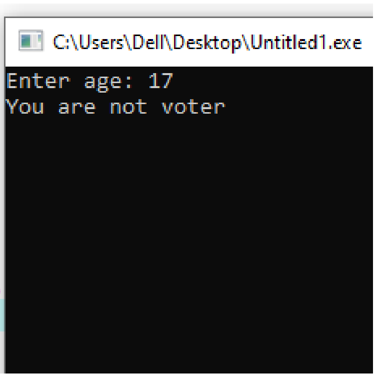
```
#include <stdio.h>
int main() {
int a = 20;
   printf("Size of variable a : %d",sizeof(a)); //4
   printf("\nSize of int data type : %d",sizeof(int));//4
   printf("\nSize of char data type : %d",sizeof(char));//1
   printf("\nSize of float data type : %d",sizeof(float));//4
   printf("\nSize of double data type : %d",sizeof(double));//8
   return 0;
}
```

Conditional Operator/ Ternary operator:

conditional operator checks the condition and executes the statement depending of the condition.
A conditional operator is a ternary operator, that is, it works on 3 operands.
Conditional operator consist of two symbols.
1 : question mark (?).
2 : colon (:).
Syntax : condition ? exp1 : exp2;



```c
//conditional operator example
#include<stdio.h>
#include<conio.h>
void main()
{
int age;
printf("Enter age: ");
scanf("%d",&age);
age>=18?printf("You are voter"):printf("You are not voter");
getch();
}
```



```
Enter age: 17
You are not voter
```

Bitwise operators

These operators which are used for the manipulation of data at bit level are known as bitwise operators. These are used for testing the bits or shifting them right or left.

Operator	Meaning
&	Binary (bitwise) AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Bitwise left shift
>>	Bitwise right shift
~	Bitwise Ones Complement

a. **Bitwise AND operator**
   a = 1100; // (12 in decimal)
   b = 1010; // (10 in decimal)
   c = a & b; // (8 in decimal)
   c = 1000;

b. **Bitwise OR operator**
   a = 1100; // (12 in decimal)
   b = 1010; // (10 in decimal)
   c = a | b; // (14 in decimal)
   c = 1110;

c. **Exclusive OR (XOR) operator**
   c = a ^ b; // (6 in decimal)
   c = 0110;
   (Note: if both bits are same then zero otherwise one).

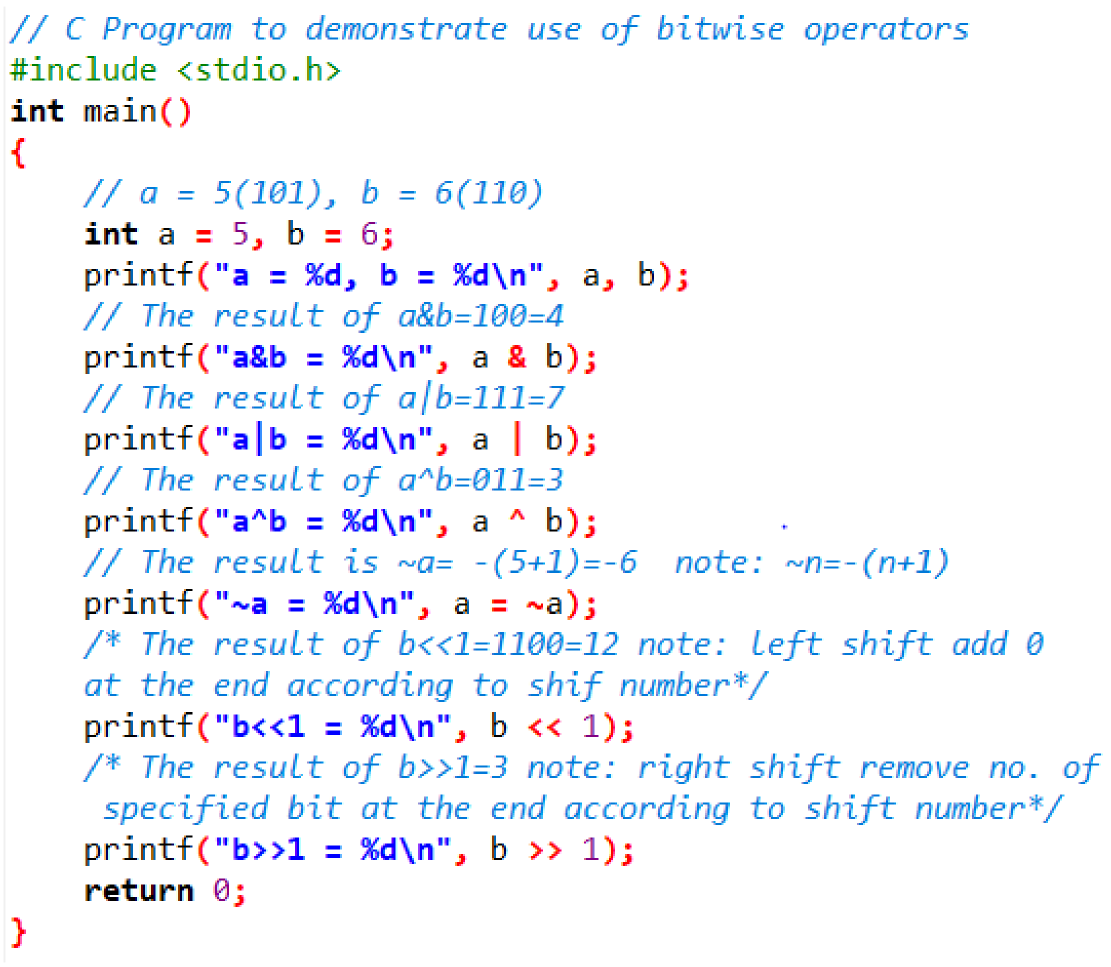
d. **Bitwise Complement (~) operator**
   ~a= ~12= -13
   (Note: bitwise Complement of N is - (N+1))

e. **Left Shift operator:**
   a<<1
   = 12<<1
   Therefore a<<1 = $(11000)_2 = (24)_{10}$

Note: In left shift operater, first convert the given number into binary form and append the specified no of Zero's at the end. Then the resulting binary is convert in decimal form which is final answer.

f. **Right Shift operator:**
   a>>1
   =12>>1
   Therefore a>>1 = $(110)_2 = (6)_{10}$

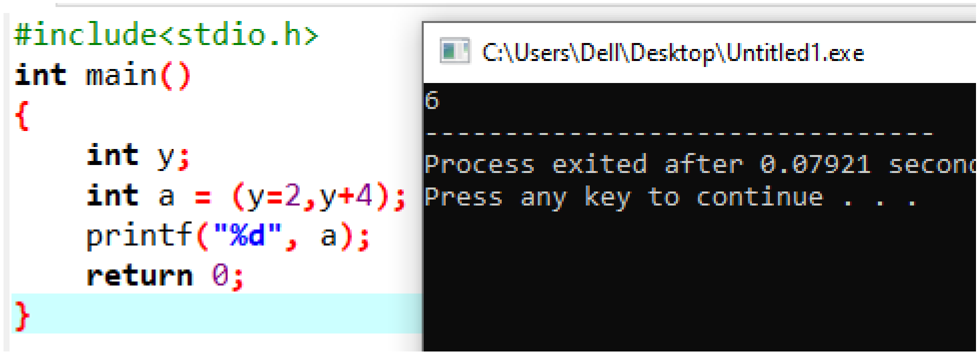
Note: In right shift operater, first convert the number given binary form and then remove the no. of specified bits from right side. Then the resulting binary number is Convert in decimal from which is final answer.



```c
// C Program to demonstrate use of bitwise operators
#include <stdio.h>
int main()
{
    // a = 5(101), b = 6(110)
    int a = 5, b = 6;
    printf("a = %d, b = %d\n", a, b);
    // The result of a&b=100=4
    printf("a&b = %d\n", a & b);
    // The result of a|b=111=7
    printf("a|b = %d\n", a | b);
    // The result of a^b=011=3
    printf("a^b = %d\n", a ^ b);
    // The result is ~a= -(5+1)=-6  note: ~n=-(n+1)
    printf("~a = %d\n", a = ~a);
    /* The result of b<<1=1100=12 note: left shift add 0
    at the end according to shif number*/
    printf("b<<1 = %d\n", b << 1);
    /* The result of b>>1=3 note: right shift remove no. of
     specified bit at the end according to shift number*/
    printf("b>>1 = %d\n", b >> 1);
    return 0;
}
```

Comma Operator:

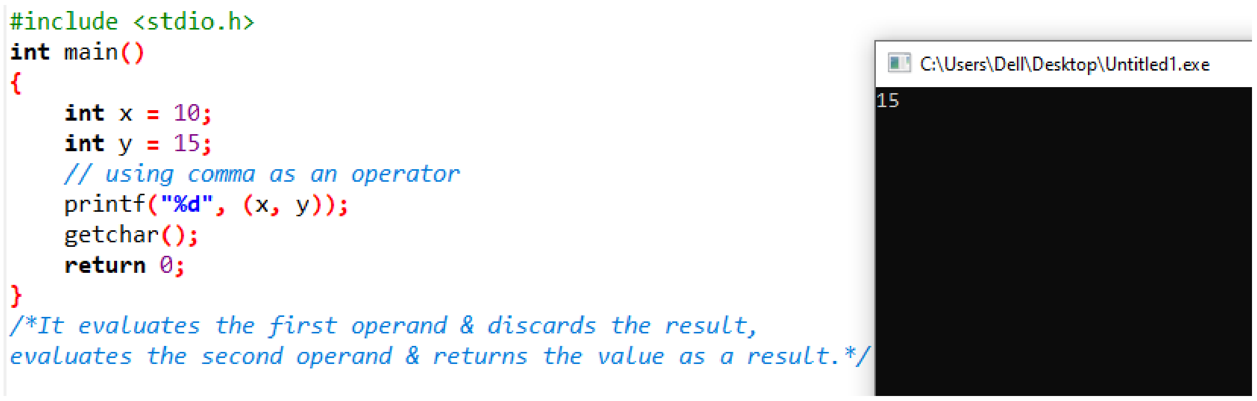
The comma in the form of an operator is used to assign multiple numbers of values to any variable in a program.



```c
#include<stdio.h>
int main()
{
    int y;
    int a = (y=2,y+4);
    printf("%d", a);
    return 0;
}
```

C:\Users\Dell\Desktop\Untitled1.exe

```
6
--------------------------------
Process exited after 0.07921 second
Press any key to continue . . .
```



```
#include <stdio.h>
int main()
{
    int x = 10;
    int y = 15;
    // using comma as an operator
    printf("%d", (x, y));
    getchar();
    return 0;
}
/*It evaluates the first operand & discards the result,
evaluates the second operand & returns the value as a result.*/
```

C:\Users\Dell\Desktop\Untitled1.exe

```
15
```

EXPRESSIONS:

• An expression is a sequence of operands and operators that reduces to a single value.
• Expression can be simple or complex.
• An operator is a syntactical token that requires an action be taken.
• An operand is an object on which an operation is performed.

A simple expression contains only one operator. E.g: 2 + 3 is a simple expression whose value is 5. A complex expression contains more than one operator. E.g: 2 + 3 * 2.
To evaluate a complex expression we reduce it to a series of simple expressions. In first we will evaluate the simple expression 3 * 2 (6)and then the expression 2 + 6, giving a result of 8.
The order in which the operators in a complex expression are evaluated is determined by a set of priorities known as precedence, the higher the precedence, the earlier the expression containing the operator is evaluated.
The following table shows the precedence and Associativity of operators:

Rules of Associativity

Operator	Associativity
( ) [ ] ->	Left to right
! ~ ++ -- - (type) * & sizeof()	Right to left
* / %	Left to right
+ -	Left to right
<< >>	Left to right
< <= > >=	Left to right
== !=	Left to right
&	Left to right
|	Left to right
^	Left to right
&&	Left to right
||	Left to right
?:	Right to left
= += -= /= %= &= |= ^=	Right to left
,	Left to right

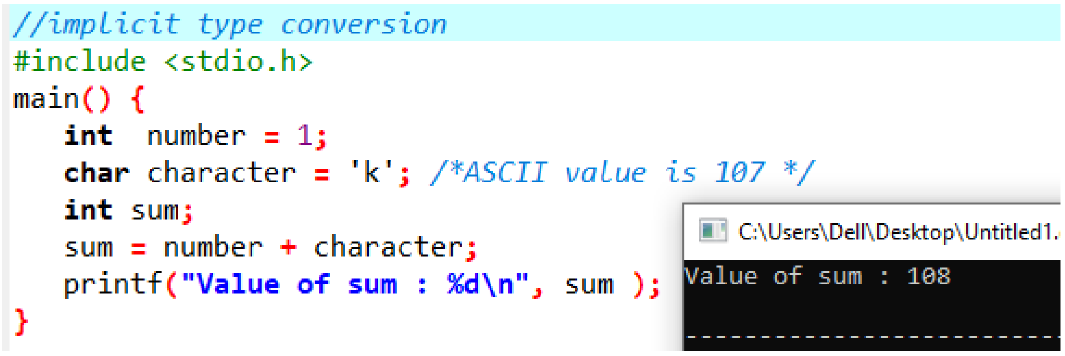
TYPE CONVERSION:

In an expression that involves two different data types, such as multiplying an integer and a floating-point number to perform these evaluations, one of the types must be converted.

We have two types of conversions:

1. Implicit Type Conversion
2. Explicit Type Conversion

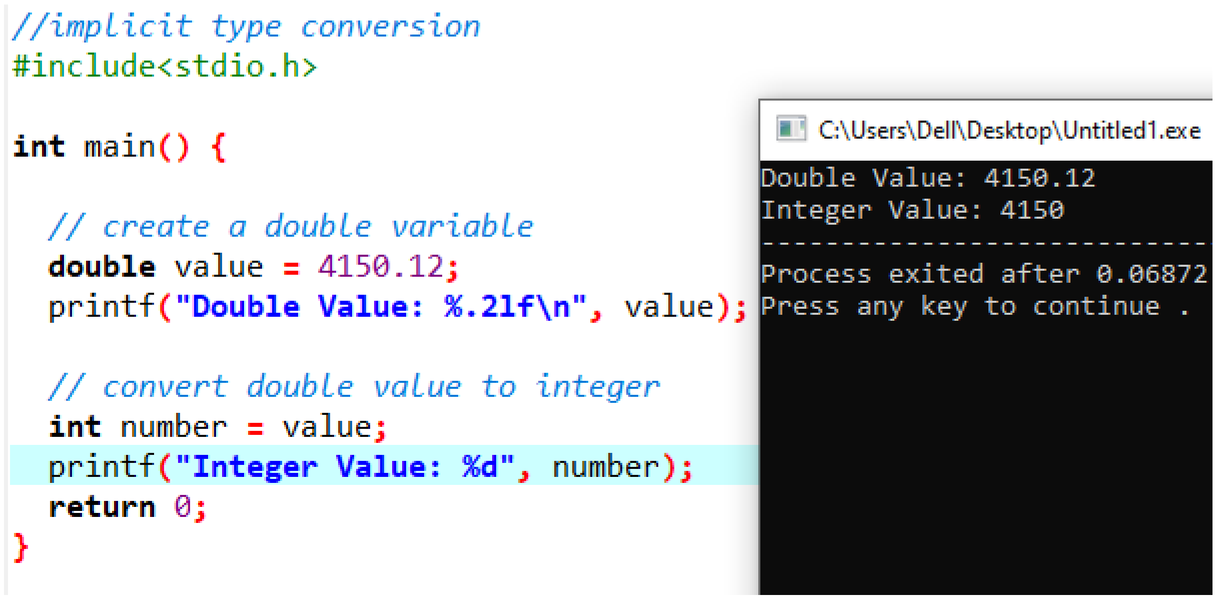
IMPLICIT TYPE CONVERSION: When the types of the two operands in a binary expression are different automatically converts one type to another. This is known as implicit type conversion.



```c
//implicit type conversion
#include <stdio.h>
main() {
    int  number = 1;
    char character = 'k'; /*ASCII value is 107 */
    int sum;
    sum = number + character;
    printf("Value of sum : %d\n", sum );
}
```

C:\Users\Dell\Desktop\Untitled1.

```
Value of sum : 108

-------------------------
```



```c
//implicit type conversion
#include<stdio.h>

int main() {

  // create a double variable
  double value = 4150.12;
  printf("Double Value: %.2lf\n", value);

  // convert double value to integer
  int number = value;
  printf("Integer Value: %d", number);
  return 0;
}
```

C:\Users\Dell\Desktop\Untitled1.exe

```
Double Value: 4150.12
Integer Value: 4150
----------------------------
Process exited after 0.06872
Press any key to continue .
```

EXPLICIT TYPE CONVERSION: Explicit type conversion uses the unary cast operator to convert data from one type to another. To cast data from one type to another, we specify the new type in parentheses before the value we want converted.

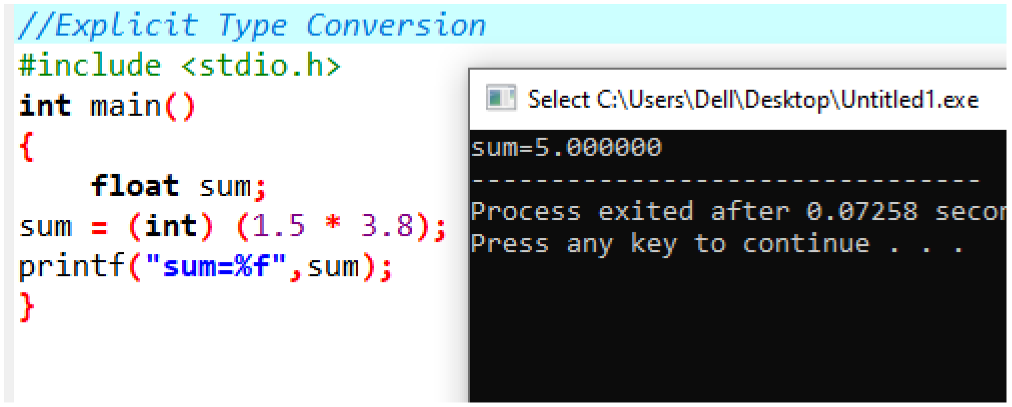
Syntax: (data type) variable/expression/value;

Example:

float sum;

Sum = (int) (1.5 * 3.8);

The typecast (int) tells the C compiler to interpret the result of (1.5 * 3.8) as the integer 5, instead of 5.7. Then, 5.0 will be stored in sum, because the variable sum is of type float.

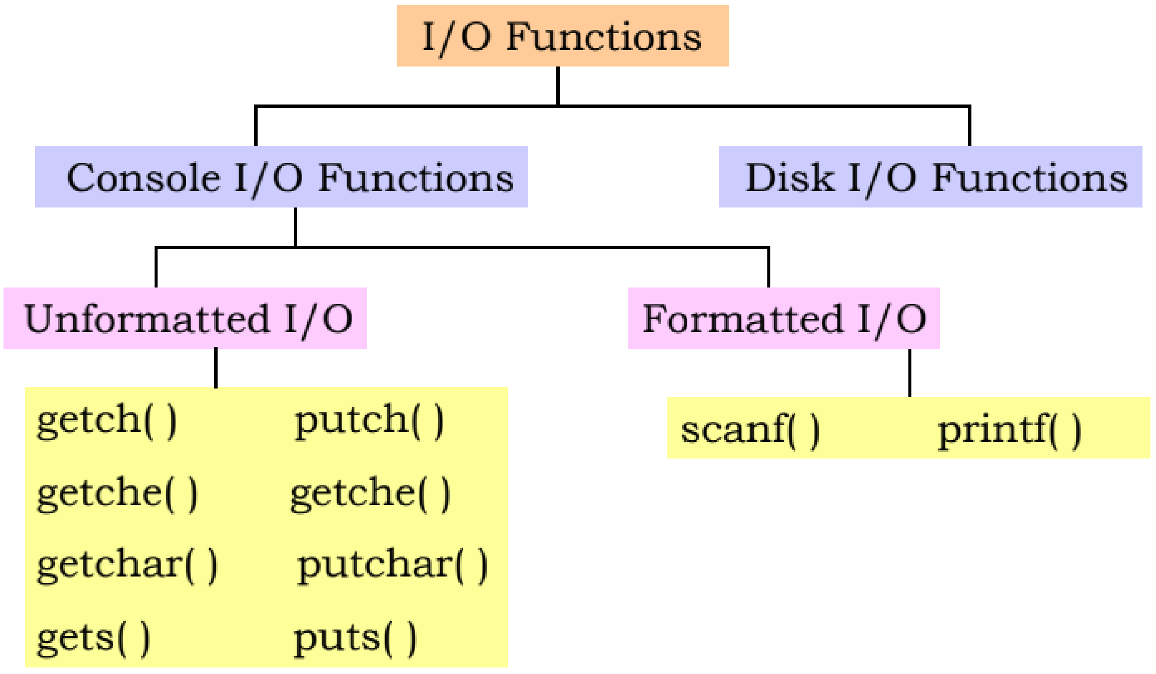


```c
//Explicit Type Conversion
#include <stdio.h>
int main()
{
    float sum;
sum = (int) (1.5 * 3.8);
printf("sum=%f",sum);
}
```

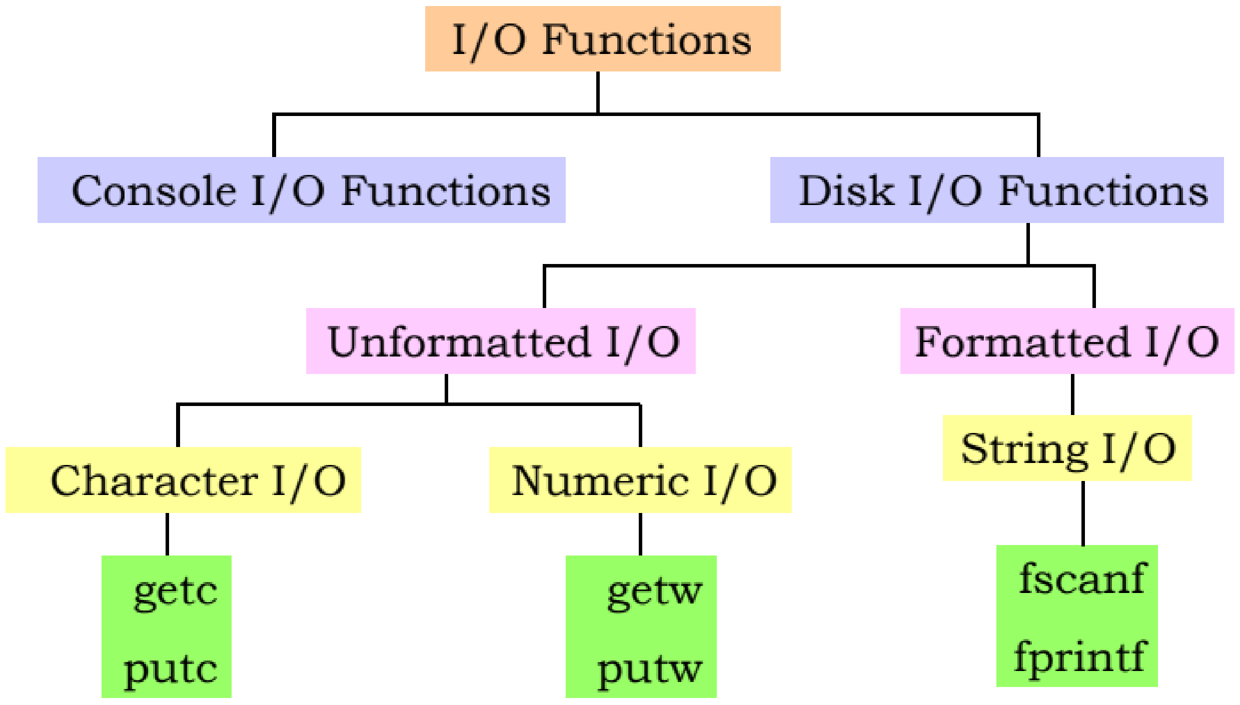
Select C:\Users\Dell\Desktop\Untitled1.exe

```
sum=5.000000
--------------------------------
Process exited after 0.07258 secon
Press any key to continue . . .
```

Console Input/Output Functions



Disk Input/Output Functions



Formatted Input Function

scanf( )

- int scanf(char *format, args....) -- reads from stdin and puts input in address c
  variables specified in argument list.
- Returns, number of charecters read.
- The address of variable or a pointer to one is required by scanf.
  Example: scanf(``%d",&i);
- We can just give the name of an array or string to scanf since this corresponds to the start address of the array/string.

Example:

```
char string[80];
scanf(``%s",string);
```

printf()

- The printf function is defined as follows:
  int printf(char *format, arg list ...) --
  prints to stdout the list of arguments according specified format string. Returns number of characters printed.
- The format string has 2 types of object:
- *ordinary characters* -- these are copied to output.
- *conversion specifications* -- denoted by % and listed in Table.

Program illustrating Formatted I/O Functions

```
#include <stdio.h>

int main()
{
  int day;
  int month;
  int year;
  char name[30];

  printf(“Enter your name:\n”>
  scanf(“%s”, name);

  /* skipping spaces */
  printf(“Hi %s. Enter birthdate as: dd mm yyyy\n”, name);
  scanf("%d %d %d", &day, &month, &year);

  /* alternative */
  printf(“Hi %s.  Enter birthdate as: dd-mm-yyyy\n”, name);
  scanf("%d-%d-%d", &day, &month, &year);

  return 0;
}
```

Note: no ampersand for strings

Conversion specifier

Literal characters

Unformatted Console I/O Functions

getch( ) and getche( )

- This function will read a single character the instant it is typed without waiting for the Enter key to be hit.
- These functions return the character that has been more recently typed.
- The ‘e’ in ***getche()*** function means it echoes(displays) the character that has been typed.
- The ***getch()*** just return the character that has been typed without echoing it on the screen.

Unformatted Console I/O Functions

getchar( ) and putchar( )

- The *getchar( )* accepts a single character the instant it has been typed.
- The *getchar( )* echoes the character that has been typed.
- It requires the Enter key to be typed following the character that has been typed.
- The functions ***putch()*** and ***putchar()*** print the character on the screen.

Program illustrating Console Input Functions

```
main()
{       char ch;
        printf("Hello\n");
        getch();
        printf("Type any character\n");
        ch=getche();
        printf("Type any character\n");
        ch=getchar();
}
```

Output:

```
Hello
Type any character A
Type any character B
```

Program illustrating Console Output Functions

```
main()
{       char ch='D';
        putch(ch);
        putchar(ch);
        putch('S');
        putchar('S');
}
```

Output:

```
DDSS
```