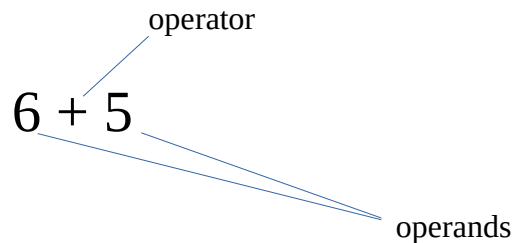


Operands & Operators

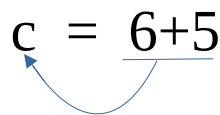
Operands & Operators

Operands are literals or variables which take participate in the operation. Operators define what to do with operands then produce the result.



Operation

Operation is also important for the program because, without operation, the result couldn't be generated. Every statement of assignment operator (=) defines an operation. Operation contains operands & operators.



6+5 stored in c variable.

There are various categories of operator in c language.

| Operator group | Example | Description |
|----------------------|-----------------------|---|
| Arithmetic Operators | +, -, *, etc. | Helpful in mathematical operations. |
| Relational Operators | >, <, !=, ==, <=, >=. | Used to compare the operands to each other. |
| Logical Operators | &&, , !. | helpful to make decision among multiple conditions. |
| Bitwise Operators | &, , ^, ~, <<, >> | Used to perform bit-level operations. |
| Assignment Operators | =, +=, -=, *=, etc. | assign the result into operand after the operation of operands. |
| Unary Operators | +, -, ++, --. | auto-increment/decrement & sign mention. |
| Ternary Operator | ()?: | works like if else block. |
| Special Operators | (,), &, * etc. | commonly useful in a program. |

Arithmetic Operators

There are various types of operators.

These operators are used to perform mathematical operations. They are also known as binary operators because they took at least 2 operands to perform an operation.

| Operation | Operator /sign | Example | Description |
|----------------|----------------|--------------|--|
| Addition | + | $c = a + b$ | Use for the addition of the numbers. addition of a & b stored into c. |
| Subtraction | - | $c = a - b$ | Use fo subtraction. Subtract b from a and store result into c. |
| Multiplication | * | $c = a * b$ | used to multiply 2 numbers. Multiplication of a & b stored into c. |
| division | / | $c = a / b$ | Division operator, divide a by b then store result into c. |
| modulo | % | $c = a \% b$ | Modulo operator, which returns the remainder after division. The Operands must be in integer. Result stored into c. |

Program 1: Write a program to find reminder using modulo operator.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c;
    clrscr();
    printf("Enter 2 numbers : ");
    scanf("%d,%d",&a,&b); // enter 2 numbers separate by comma(,)
    c = a%b;
    printf("                result : %d %% %d = %d\n",a,b,c);
                                // %% used to print single '%'
}
```

getch();

}

OUTPUT:

```
Enter 2 numbers : 7,4
result : 7 % 4 = 3
```

Program 2: Write a program to find simple interest of given P,R,N.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    double SI,R,P,N;
    clrscr();
    printf("Enter P,R,N : ");
    scanf("%lf,%lf,%lf",&P,&R,&N);

    SI = (P*R*N)/100;           // formula to find simple interest

    printf("Simple interest : %.3lf",SI);
    getch();
}
```

}

OUTPUT :

```
Enter P,R,N : 100000,0.8,3
Simple interest : 2400.000
```

Relational Operators

The relational operator checks the relation between literals or variables. If Relation is true then return **1 as TRUE otherwise 0 as FALSE**. So 1 means TRUE & 0 means FALSE. Relational operators are used to comparing things & make the decision by if-else statements. We will check the if-else statement here in brief & later in detail.

| Operator Name | Operator sign | Example | Description |
|---------------------------|---------------|---------|---|
| Greater than | > | a>b | Returns 1 if a is greater than b, otherwise 0 even both are the same. |
| Less than | < | a<b | Returns 1 if a is less than b, otherwise 0 even both are the same. |
| Greater than or equals to | >= | a>=b | Returns 1 if a is greater than b or both are same, otherwise 0. |
| Less than or equals to | <= | a<=b | Returns 1 if a is less than b or both same, otherwise 0. |
| Not equals to | != | a!=b | Returns 1 if both are not the same, otherwise 0. |
| Equals to | == | a==b | Returns 1 if both are the same, otherwise 0. |

Program 3: Write a program to perform relational operators.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
    clrscr();
    printf("Enter 2 numbers : ");
    scanf("%d,%d",&a,&b);

    printf("%d > %d : %d\n",a,b,a>b); // greater than
    printf("%d < %d : %d\n",a,b,a<b); // less than
    printf("%d >= %d : %d\n",a,b,a>=b); // greater than or equals to
    printf("%d <= %d : %d\n",a,b,a<=b); // less than or equals to
    printf("%d != %d : %d\n",a,b,a!=b); // not equals to
    printf("%d == %d : %d\n",a,b,a==b); // equals to

    getch();
}
```

OUTPUT :

```
Enter 2 numbers : 7,4
? > 4 : 1
? < 4 : 0
? >= 4 : 1
? <= 4 : 0
? != 4 : 1
? == 4 : 0
```

If - else statement :

The if-else statement works on condition. In the c language true value is represented by 1 & false represent by 0. let's understand if-else by its syntax.

Syntax:

```
if(condition)
{
    // true block
}
else
{
    //false block
}
```

if the condition is TRUE (1) then a true block will be executed otherwise false block will be executed.

Program 4: Write a program to find maximum number from 2 numbers.

```
#include<stdio.h>
#include<conio.h>
void main(){
    int a,b;
    clrscr();
    printf("Enter 2 numbers : ");
    scanf("%d,%d",&a,&b);
    if(a>b) //'a>b' is condition , use of greater than
    {
        //true block
        printf("a is max : %d\n",a);
    }
    else
    {
        //false block
        printf("b is max : %d\n",b);
    }
    getch();
}
```

OUTPUT:

```
Enter 2 numbers : 10,20
b is max : 20
```

Logical Operators

the logical operator used to check more than one condition at a time. They are just like a logic gate of computer science.

1) Logical AND(&&)

Logical AND is used to make a decision when all conditions must be true. It returns 1 when all condition or relation is True otherwise 0. Here let's see that by the table.

| A | B | Logical AND(&&) |
|---|---|-------------------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Program 5: write a program to check whether a student is pass or fail by given marks.

```
#include<stdio.h>
#include<conio.h>
void main(){
    float marks;
    clrscr();
    printf("Enter marks : ");
    scanf("%f",&marks);
    /*
        marks should be greater than or equals to 33 &
        less than or equals 100 to pass the phase.
    */
    if(marks>=33 && marks<=100)
    {
        printf("Pass %c\n",3);
    }
    else
    {
        printf("Better luck Next TIME !! %c\n",2);
    }
    getch();
}
```

OUTPUT 1:

```
Enter marks : 31
Better luck Next TIME !! 8
```

OUTPUT 2:

```
Enter marks : 87
Pass ♥
```

2) Logical OR (||)

Logical OR is used to check at least one condition should be true among multiple conditions. It returns 0 if all conditions are false otherwise 1.

| A | B | Logical OR() |
|---|---|------------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |

| | | |
|---|---|---|
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Program 6: write a program to check whether an entered character is alphabet or not.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char ch;
    clrscr();
    printf("Enter a character : ");
    scanf("%c",&ch);
    /*
       an alphabet could be in uppercase or lowercase.
    */
    if((ch>='a' && ch<='z') || (ch>='A' && ch<='Z'))
    {
        printf("%c is alphabet .\n",ch);
    }
    else
    {
        printf("%c is not alphabet.\n");
    }
    getch();
}
```

OUTPUT 1:

```
Enter a character : A
A is alphabet.
```

3) Logical NOT (!)

Logical NOT is used to inverse the logic of the condition. In simple words, it will convert false results to true & true results to false.

| A | Logical NOT(!) |
|---|------------------|
| 0 | 1 |
| 1 | 0 |

Program 7: write a program to display a message when an entered OPT is wrong.

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

```

    const float OTP = 871199; //float because, out of int's range.
    float user_OTP;
    clrscr();

    printf("Enter the OTP : ");
    scanf("%f",&user_OTP);

    if(!(user_OTP == OTP))
    {
        printf("Invalid OTP");
        getch();          // wait until the user press any key.
        exit(0);          //to exit the program.
    }

    printf("Welcome .... %.0f\n",user_OTP);
    getch();
}

```

OUTPUT 1:

```

Enter the OTP : 871199
Welcome .... 871199

```

OUTPUT 2:

```

Enter the OTP : 872577
Invalid OTP_

```

Bitwise Operators

Bitwise operator performs the operation at the bit level, which means first all values converted into binary form (0 and 1) & then operation takes place. Bitwise operator is used to performing a logical operation like the logic circuit's operation. make blocks of a 4-bit while converting the number into binary like 4 bit, 8 bit, 12 bit, etc.

1) Bitwise AND (&)

Bitwise AND(&) operator check every corresponding bit of numbers. If both bits are 1 then returns 1, else returns 0. This operator exactly does like logical AND does in a circuit.

| A | B | Bitwise AND(&) |
|---|---|------------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Program 8: write a program to demonstrate the use of bitwise AND(&).

```

#include<stdio.h>
#include<conio.h>
void main(){
    int a,b,c;
    clrscr();
    printf("Enter 2 numbers : ");

```

```
scanf("%d,%d",&a,&b);
c = a & b; // bitwise AND( & )
printf("%d & %d = %d \n",a,b,c);
getch();
}
```

OUTPUT:

```
Enter 2 numbers : 5,4
5 & 4 = 4
```

Explanation:

let a = 5, b = 4.

let's convert them into binary

$$\begin{array}{rcl}
 a \rightarrow 5_{10} & \rightarrow & (0\ 1\ 0\ 1)_2 \\
 b \rightarrow 4_{10} & \rightarrow & (0\ 1\ 0\ 0)_2 \\
 & \& & \downarrow \downarrow \downarrow \downarrow \\
 & & (0\ 1\ 0\ 0)_2 \rightarrow 4_{10} \rightarrow c
 \end{array}$$

see, AND(&) operator check bit by bit & perform AND(&) operation with a corresponding bit. This is how we get 4 in answer.

2) Bitwise OR (|)

Bitwise OR (|) check every corresponding bit of numbers. If both bit are 0 then returns 0, else returns 1. This operator exactly does like logical OR does in a circuit.

| A | B | Bitwise OR() |
|---|---|-----------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Program 9: write a program to demonstrate the use of bitwise OR(|).

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c;
    clrscr();
    printf("Enter 2 numbers : ");
    scanf("%d,%d",&a,&b);
    c = a | b; // bitwise OR( | )
    printf("%d | %d = %d \n",a,b,c);
    getch();
}
```

OUTPUT:


```
Enter 2 numbers : 5,4
5 | 4 = 5
```

Explanation:

let a = 5, b = 4.

let's convert them into binary

$$\begin{array}{rcl}
 a \rightarrow 5_{10} & \rightarrow & (0\ 1\ 0\ 1)_2 \\
 b \rightarrow 4_{10} & \rightarrow & (0\ 1\ 0\ 0)_2 \\
 & & \downarrow \downarrow \downarrow \downarrow \\
 & & (0\ 1\ 0\ 1)_2 \rightarrow 5_{10} \rightarrow c
 \end{array}$$

3) Bitwise Ex-OR (^)

Bitwise Ex-OR operator checks every corresponding bit of both numbers & if both bits are the same then it returns 0 else 1.

| A | B | Bitwise Ex-OR(^) |
|---|---|--------------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Program 10: write a program to demonstrate the use of bitwise Ex-OR operator.

```
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
    int a,b,c;
    clrscr();
    printf("Enter 2 numbers : ");
    scanf("%d,%d",&a,&b);
    c = a ^ b; // bitwise Ex-OR( ^ )
    printf("%d ^ %d = %d \n",a,b,c);
    getch();
}
```

OUTPUT:

```
Enter 2 numbers : 5,4
5 ^ 4 = 1
```

Explanation:

let a = 5, b = 4.

let's convert them into binary

$$\begin{array}{rcl}
 a \rightarrow 5_{10} & \rightarrow & (0\ 1\ 0\ 1)_2 \\
 b \rightarrow 4_{10} & \rightarrow & (0\ 1\ 0\ 0)_2 \\
 & & \wedge \downarrow \downarrow \downarrow \downarrow \\
 & & (0\ 0\ 0\ 1)_2 \rightarrow 1_{10} \rightarrow c
 \end{array}$$

4) Bitwise negation operator (~)

This bitwise operator is known as 1's complement of the number. It is a unary operator which took only one variable or literal, unlike other bitwise operators.

Program 11: write a program to demonstrate the use of bitwise negation operator.

```
#include<stdio.h>
#include<conio.h>
void main(){
    int a,b,c;
    clrscr();
    printf("Enter a number: ");
    scanf("%d",&a,&b);
    c = ~a; // bitwise negation( ~ )
    printf(" ~ %d = %d \n",a,c);
    getch();
}
```

OUTPUT:

```
Enter a number: 6
~ 6 = -7
```

Explanation:

let a = 6.

it's easy to solve this by equation.

$$\begin{aligned} c &= -(a+1) \\ &= -(6+1) \\ &= -(7) \\ &= -7 \end{aligned}$$

5) bitwise left shift (<<)

Bitwise left shift (<<) operator is used to shift bits of the number to the left side by the given number. When you shift bits to the left side, the number will get greater than before.

Program 12: write a program to demonstrate the use of bitwise left shift operator.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c;
    clrscr();
    printf("Enter 2 numbers : ");
    scanf("%d",&a,&b);

    c = a << b; // bitwise left-shift( << )

    printf("%d << %d = %d \n",a,b,c);
```

```
getch();
}
```

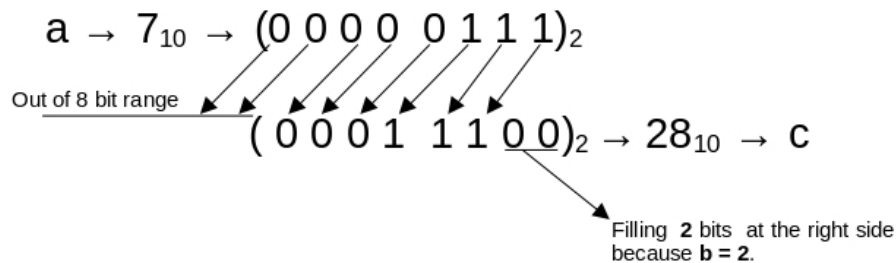
OUTPUT:

```
Enter 2 numbers : 7,2
7 << 2 = 28
```

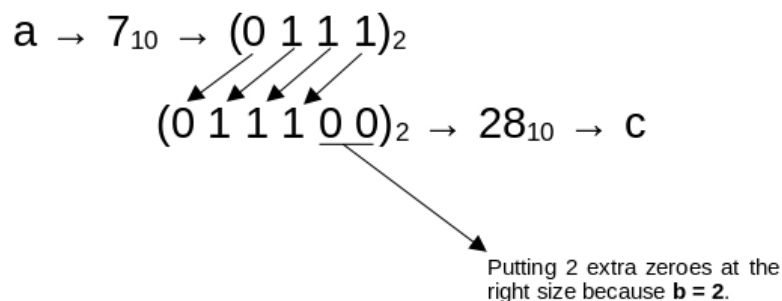
Explanation:

let $a = 7$, $b = 2$.

Method 1: Traditional way



Method 2 : easy way



However, we can get the answer by the following equation.

$$\begin{aligned}
 c &= a * 2^b \\
 &= 7 * 2^2 \\
 &= 7 * 4 \\
 &= 28
 \end{aligned}$$

6) Right shift

Bitwise right shift ($>>$) operator is used to shift bits of the number to the right side by the given number. When you shift bits to the right side, the number will get smaller than before.

Program 13: write a program to demonstrate the use of bitwise right shift operator.

```
#include<stdio.h>
#include<conio.h>
void main()
```

```

{
    int a,b,c;
clrscr();
    printf("Enter 2 numbers : ");
    scanf ("%d,%d",&a,&b);

    c = a >> b; // bitwise right-shift( >> )

    printf("%d >> %d = %d \n",a,b,c);
getch();
}

```

OUTPUT:

```

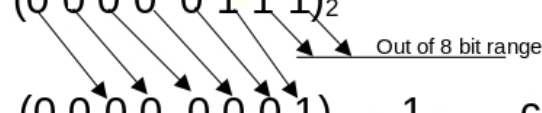
Enter 2 numbers : 7,2
7 >> 2 = 1

```

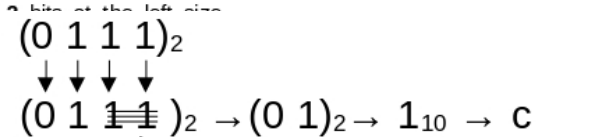
Explanation:

let $a = 7$, $b = 2$.

Method 1: Traditional way

$a \rightarrow 7_{10} \rightarrow (00000111)_2$

 $(00000001)_2 \rightarrow 1_{10} \rightarrow c$

Method 2 : easy way

$a \rightarrow 7_{10} \rightarrow (0111)_2$

 $(01)_{2} \rightarrow 1_{10} \rightarrow c$
 Canceling 2 bits at the right side because $b = 2$.

like, we solved left shift operation by an equation we can also solve right shift with the following equation.

$$\begin{aligned}
 c &= a/2^b \\
 &= 7/2^2 \\
 &= 7/4 \\
 &= 1.75 \\
 &= 1 \quad \text{Due to integer data type}
 \end{aligned}$$

Program 14: write a program to perform all bitwise operators.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c;

```

```
clrscr();
printf("Enter 2 numbers : ");
scanf("%d,%d",&a,&b);

c = a & b; //bitwise AND( & )
printf("%d & %d = %d \n",a,b,c);

c = a | b; //bitwise OR( | )
printf("%d | %d = %d \n",a,b,c);

c = a ^ b; //bitwise Ex-OR( ^ )
printf("%d ^ %d = %d \n",a,b,c);

c = ~a; //bitwise negation
printf(" ~ %d = %d \n",a,c);

c = a << b; //bitwise left-shift
printf("%d << %d = %d \n",a,b,c);

c = a >> b; //bitwise right-shift
printf("%d >> %d = %d \n",a,b,c);
getch();
}
```

OUTPUT:

```
Enter 2 numbers : 7,2
7 & 2 = 2
7 | 2 = 7
7 ^ 2 = 5
~ 7 = -8
7 << 2 = 28
7 >> 2 = 1
```

Assignment Operators

The assignment operator is used to assign a literal or a result to the one of its operand. In other words one of its operand will update by an operation. Usually, The assignment operator contains 2 operators one of them is the actual operator (like +, -, /, etc) & another one is a normal assignment (=). The actual operator defines which operation would take place in the expression.

| Assignment operator | example | meaning | description |
|---------------------|---------|---------|---|
| = | a=10 | a=10 | simple assignment which is used to assign a literal without any external operation. |
| += | a+=b | a=a+b | assign the sum of a and b into a. |
| -= | a-=b | a=a-b | assign the subtraction of a and b into a. |
| *= | a*=b | a=a*b | assign the multiplication of a and b into a. |
| /= | a/=b | a=a/b | assign the division of a and b into a. |
| %= | a%=b | a=a%b | assign the remainder of a/b into a. |

| | | | |
|-----|-------|--------|-----------------------------------|
| &= | a&=b | a=a&b | assign the result of a&b into a. |
| = | a =b | a=a b | assign the result of a b into a. |
| ^= | a^=b | a=a^b | assign the result of a^b into a. |
| <<= | a<<=b | a=a<<b | assign the result of a<<b into a. |
| >>= | a>>=b | a=a>>b | assign the result of a>>b into a. |

Program 15: write a program to illustrates the various assignment operators.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b;
clrscr();
    printf("Enter 2 numbers : ");
    scanf ("%d,%d",&a,&b);

    a+=b; //a = a+b;
    printf("value of a after += : %d\n",a);

    a*=b; //a = a*b;
    printf("value of a after *= : %d\n",a);

    a<<=b; //a = a<<b;
    printf("value of a after <<= : %d\n",a);

    a>>=b; //a = a&b;
    printf("value of a after >>= : %d\n",a);

    a%=10; //a = a%10;
    printf("value of a after %%= : %d\n",a);

    getch( );
}
```

OUTPUT:

```
Enter 2 numbers : 9,3
value of a after += : 12
value of a after *= : 36
value of a after <<= : 288
value of a after >>= : 36
value of a after %=: 6
```

Unary operators

The unary operator has only one operand to perform the operation, after the operation result would be stored in the same operand.

1) unary plus (+) / unary minus (-)

The unary plus (+) is used to represent the number is positive. However, you need only a minus sign for the negative number indication for the positive number you don't need a sign but you can use plus (+) sign for that.

Program 16: write a program to convert the negative number to positive number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int num;
    clrscr();
    printf("Enter the number : ");
    scanf("%d",&num);

    if(num<0) // if number is less than zero then it must be -ve
    {
        num*=-1; // num = num * -1;
        /*
           to convert -ve to +ve, multiply the number with -1
        */
    }
    getch();
}
```

OUTPUT:

```
Enter the number : -34
Number : 34
```

2) Increment (++)

An increment operator has only one operand & it adds 1 to the operand. wisely it is used for auto-increment the operand. There are 2 types of increment.

1) Post-Increment (X++)

This operator access the value of variable then increment the operand.

2) Pre-Increment (++X)

This operator increment the operand then access the value.

Program 17: write a program to demonstrates the use of increment operator.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i=10;
    clrscr();
    printf("during post-increment : %d\n",i++);
```

```

printf(" after post-increment : %d\n",i);

printf("\n-----\n\n");
i = 10; //reset the value.
printf(" during pre-increment : %d\n",++i);
printf(" after pre-increment : %d\n",i);
getch();
}

```

OUTPUT:

```

during post-increment : 10
after post-increment : 11
-----

during pre-increment : 11
after pre-increment : 11

```

3) Decrement (--)

An decrement operator has only one operand & it subtracts 1 to the operand. It works auto-decrement.

1) post-decrement (X--)

This operator access the value then subtracts 1 from the operand.

2) pre-decrement (--X)

This operator subtracts 1 from the operand then access the value.

Program 18: write a program to demonstrates the use of increment operator.

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int i=10;
clrscr();
    printf("during post-decrement : %d\n",i--);
    printf(" after post-decrement : %d\n",i);
    printf("\n-----\n\n");
    i = 10; //reset the value.
    printf(" during pre-decrement : %d\n",--i);
    printf(" after pre-decrement : %d\n",i);
getch();
}

```

OUTPUT:

```

during post-decrement : 10
after post-decrement : 9
-----

during pre-decrement : 9
after pre-decrement : 9

```


Ternary Operator

Which is also known as conditional (?:) operator. The Conditional operator tool 3 operands that's why it is known as the ternary operator.

Syntax:

(condition)?expression1:expression2;

example:

(a>b)?printf("a is max"):printf("b is max");

if the **condition** is true then **expression1** would be executed otherwise, **expression2** will be executed.

Program 19: Write a program to find the minimum number from 2 numbers using the ternary operator.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,min;
    clrscr();
    printf("Enter 2 numbers : ");
    scanf("%d,%d",&a,&b);
    min = (a<b)?a:b;          // min = a, if a is less than b else min = b.
    printf("The minimum : %d\n",min);

    getch();
}
```

OUTPUT:

```
Enter 2 numbers : 5,2
The minimum : 2
```

special operators

These operator contain other additional operators which are used for commonly use in program.

| Operator | Meaning | Example | Description |
|----------|------------------|----------------|---|
| , | comma | a, b, c; | To separate the tokens. |
| & | ampersand | scanf("%d",&a) | To access address of a variable. |
| * | pointer operator | int *p = &a; | To create a pointer & access the data from address. |
| sizeof | sizeof operator | sizeof(int); | Used to find size of a variable or a data type in byte. |

Program 20: Write program to illustrates the special operators.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a=10;
    int *p = &a; // &a(address of a ) assigned to p pointer.
clrscr();
    printf("a : Address %p\t data : %d\n",&a,a); //&a gives address
    printf("p : Address %p\t data : %d\n",p,*p); // *p access data

    printf("Size of int : %d\n",sizeof(int));
getch();
}
```

OUTPUT:

```
a : Address FFF4      data : 10
p : Address FFF4      data : 10
Size of int : 2
```

Precedence & associativity

Precedence :

Operator's precedence defines which operator have more importance in the operation, when an expression contains multiple operators. With help of the precedence of operator compile will know which operator had to solve first. May be multiple operators have same precedence.

Associativity:

Operator's associativity helpful when an expression contains multiple operators with same precedence. Associativity tells us the direct so we can solve the operation with help of direction.

| Precedence | Associativity |
|---|---------------|
| () , [], . , -> , X++, X-- | left-to-right |
| ++X, --X, +(unary plus), -(unary minus), !, ~, *(pointer), &(ampersand), sizeof | right-to-left |
| *, %, / (Arithmetic) | left-to-right |
| +, - (Arithmetic) | left-to-right |
| <<, >> (Bitwise) | left-to-right |
| <, <=, >, >= (Relational) | left-to-right |
| ==, != (Relational) | left-to-right |
| & (Bitwise AND) | left-to-right |
| ^ (Bitwise Ex-OR) | left-to-right |
| (Bitwise Ex-OR) | left-to-right |

| | |
|--|---------------|
| &&(Logical AND) | left-to-right |
| (Logical OR) | left-to-right |
| ?: (Ternary operator) | right-to-left |
| All assignment operators(like =, +=, *=) | right-to-left |
| , (comma) | left-to-right |

Evaluation of expression

Expression:

An expression contains multiple operators to perform a task/operation. There is precedence & associativity to solve the expression. let's understand this statement with an example.

Example: find the answer of **10*3/3<<2+7-4**.

```

answer = 10*3/3<<2+7-4      // multiplication
        = 30/3<<2+7-4      // division
        = 10<<2+7-4        // addition
        = 10<<9-4          // subtraction
        = 10<<5            // left-shift
        = 320              // answer

```