1. **Arithmetic instruction**

These instructions are used to perform some arithmetic calculations within the program . it uses arithmetic operators like +, -, \*, /, %, =, ++, --, += etc . the variables can have any simple to complex arithmetic calculation using these keywords.

Operator can’t perform any operation without any operands/data.

**Eg-:**

**3 + 4 \* 5** here 3,4and 5 are operands and + and \* are operators.

According to math 23 is right According to c language 35 is right b/c in c language

b/c in math we apply BODMAS we apply operator precedence rule for perform any.

rule for perform any operation. Operation.

**Operators**

An operator is a symbol use to perform operation in given programming language.

Types of operator in c language

C programming language offers various types of operators having different functioning capabilities.

* 1. Unary operator High
  2. Arithmetic Operators
  3. Bitwise Operators
  4. Relational Operators
  5. Logical Operators
  6. Assignment Operators
  7. Conditional Operator
  8. Compound Assignment operator Low
  9. Special Operators

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator Name** | **Operator** | **Description** | **Associative Rule** |
|  | () [] . ->  X++ | Parentheses (grouping) Brackets (array subscript) Member selection via object name Member selection via pointer  Unary pre-increment/pre-decrement | left-to-right |
| Unary | ++x  --x +  - !   (*type*) \* & sizeof | Unary pre-increment/pre-decrement Unary plus/minus Unary logical negation/bitwise complement Unary cast (change *type*) Dereference Address Determine size in bytes | right-to-left |
| Arithmetic | \*  /  % | Multiplication/division/modulus | left-to-right |
|  | +  - | Addition/subtraction | left-to-right |
| Bitwise | <<  >> | Bitwise shift left, Bitwise shift right | left-to-right |
|  | & | Bitwise AND |  |
|  | ^ | Bitwise exclusive OR | left-to-right |
|  | | | Bitwise inclusive OR | left-to-right |
|  | ~ | Bitwise Not | left-to-right |
| Relational | < < = | Relational less than/less than or equal to | left-to-right |
|  | * >= | Relational greater than/greater than or equal to | left-to-right |
|  | == != | Relational is equal to/is not equal to | left-to-right |
| Conditional | ? : | Ternary conditional | right-to-left |
| Assignment | = | Assignment | right-to-left |
| Compound Assignment | +=  -= | Addition/subtraction assignment |  |
|  | \*=  /= | Multiplication/division assignment |  |
|  | %=  &= | Modulus/bitwise AND assignment |  |
|  | ^=  |= | Bitwise exclusive/inclusive OR assignment |  |
|  | <<=  >>= | Bitwise shift left/right assignment |  |
|  | , | Comma (separate expressions) | left-to-right |
|  | ^=  |= |  |  |
|  | <<=  >>= |  |  |
|  | , | Comma (separate expressions) | left-to-right |

* 1. **Unary operator**

Operator require operands to perform its operation. Unary operators are those, which takes one operand to perform its task.

+ , - , ++, - -, sizeof()

Unary + and –

These operators should not be misinterpreted as addition and subtraction operators. These are unary + and –, used to make sign positive or negative. For example -3, +4, -345 etc.

Unary increment Operator

Pre-increment (++x) Job of pre-increment and post-increment operators are same but .

Post-increment(x++) there is a difference in priority. Pre-increment has higher priority

than post-increment. In fact, post-increment has the least priority

among all the operator.

Ex –

#include<stdio.h>

void main()

{

int x;

printf("Enter a value for x = ");

scanf("%d",&x);

x++; *//x=x+1; post-increment*

printf("\n number is = %d",x);

++x; *//x=x+1; pre-increment*

printf("\n number is = %d",x);

{

int a,b;

printf("\n Enter a value for b = ");

scanf("%d",&b);

a = b++; */\*post-increment has the least priority among all the operators.*

*so any value of b is given by user first assign in a then increase*

*the value of b.\*/*

    printf("\n number is = %d",a);

    printf("\n number is = %d",b);

    a = ++b; */\*Pre-increment has higher priority than post-increment.*

*so any value of b is given by user first increase the value*

*of b then assign in a.\*/*

    printf("\n number is = %d",a);

    printf("\n number is = %d",b);

}

}

**Unary Sizeof()**

sizeof() operator is used to evaluate the size of data type, variable or constant.

* Sizeof(DataType).
* Szeof(variable).
* Sizeof(constant).

Example:

#include<stdio.h>

void main()

{

    int x,y,z,l,k;

*// size of data-type.*

printf("Enter the name of Data-type is ");

scanf("%d",&x);

printf("\n Size of first given data type is %d",sizeof(x));

printf(" bytes");\*/

*// size of variable.*

printf(" \n print size of variable");

float f;

double d1;

char c;

x=sizeof(f);

y=sizeof(d1);

z=sizeof(c);

l=sizeof(k);

printf("\n%d\n%d\n%d\n%d",x,y,z,l);

printf(" bytes");\*\*/

*// size of constent.*

printf("\n print size of constant");

x=sizeof(45);

y=sizeof(3.56);*//real constant by default content in size of double b\c real conastant can be store easily.*

z=sizeof('A');*//here character constant contain 4 bytes on the basis of ACCII CODE(A=64.and 64 is type of integer).*

printf("\n%d\n%d\n%d",x,y,z);

}

* 1. **Arithmetic Operators/Binary operator.**

\*, \, %, +, -

Arithmetic operators is used to performs the operation with two operands.

Operators \*, / and % are having same priority but higher priority than + and -. Operator + and – are having equal priority.

Behavior of operators(+, -,\*)  
3+4 is 7  
3-4 is -1 +, - and \* operators behaves as expected  
3\*4 is 12

Behavior of division operators(/)

3/4 = 0 Operation between two integer will give integer result only.

mathematically result should be 0.75 but in c language  
 decimal point and subsequent digits are ignored and only integral

part is considered. So the answer is 0.

.  
 More examples:  
4/3 is 1  
12/5 is 2  
15/5 is 3  
-15/2 is -7

Now observer following expressions

3.0/4 is 0.75  
3/4.0 is 0.75 When at least one of the operand is real,   
3.0/4.0 is 0.75 the result will be real.

Behavior of Modulus operator (%)  
Modulus operator gives remainder as a result.

Following are few examples

5%2 is 1  
17%5 is 2  
10%2 is 0  
21%10 is 1  
3%4 is 3

Remember: In C language operands of modulus operator can’t be a real value. So 3.5%2 is an error.

Example:

#include<stdio.h>

void main()

{

*// behaveior of +, - and \* operators.*

    int x,y,z,m,n;

    printf("Enter fist value of x = ");

    scanf("%d",&x);

    printf("Enter second value of y = ");

    scanf("%d",&y);

    printf("\n sum of x and y is %d",x+y);

    printf("\n sub of x and y is %d",x-y);

    printf("\n multiplay of x and y is %d",x\*y);

*// behaveior of / operators.*

*/\*Operation between two integer will give integer result only*

*When at least one of the operand is real, the result will be real.\*/*

    printf("\n division of x and y is %d",x/y);

*// behaveior of % operators.*

*/\*Modulus operator gives remainder as a result.*

*In C language operands of modulus operator can’t be a real value. So 3.5%2 is*

*an error.*

*if small digit modulus(%) by large digit then ans come in small digit.\*/*

    printf("\n modulus of x and y is %d",x%y);

*/\*if we want to obtain last digit, last 2digit, last 3dig etc of given number*

*so we can given dig %10= last dig, given num%100=last 2dig.\*/*

    m=x%10;

    n=y%100;

    printf("\n%d\n%d",m,n);

*//if numerator is negative then ans is negative. no matter if both are negative or denominator is negative.*

y=5%2; *//ans=1*

z=5%(-2);*//ans=1*

m=-5%2; *//ans=-1*

n=-5%(-2);*//ans=-1*

printf("\n%d\n%d\n%d\n%d",y,z,m,n);

}

Association Rule ( L TO R)

If operators priority are equal then we need to use association rule.

**4 \* 3\ 2 % 5**

( L TO R)

Example

If 3+4-2= 3+4=7, then 7-2=5.

If 5-2+2= 5-2=3, then 3+2=5.

If4\*3\2%5=4\*3=12 , then 12\2=6, then 6%5=1.

If 8\2%3\*2=8\2=4, then 4%3=1, then 1\*3=3.

#include<stdio.h>

void main()

{

    int a,b,x,c,y,z,p;

    printf("Enter fist value of a = ");

    scanf("%d",&a);

    printf("Enter second value of b = ");

    scanf("%d",&b);

    printf("Enter second value of c = ");

    scanf("%d",&c);

*// printf("Enter second value of b = ");*

*//scanf("%d",&b);3+4-2*

    x = a+b-c; *// 3+4=7 then 7-2=5*

    y = a-b+c; *//3-4= -1 then -1+2=1*

    z = a\*b+c/b; *//3\*4=12 then 2/4=0 then 12+0 =12;*

    p = a\*b%c/b; *//3\*4=12 then 12%2=0 then 0/4=0;*

    printf(" result of this expression a+b\*a/c =%d",x);

    printf("\n result of this expression a+b\*a/c =%d",y);

     printf("\n result of this expression a+b\*a/c =%d",z);

      printf("\n result of this expression a+b\*a/c =%d",p);

}

*// if input the value of a=3 ,b=4, c=2*

* 1. **Bitwise Operators.**

Operators that performs on bits (0 or 1) are known as bitwise operators and also known as unary operator b/c these operator works with one operands. There are six bitwise operators

|  |  |
| --- | --- |
| & | AND |
| | | OR |
| ^ | XOR |
| ~ | NOT Bitwise not(~) operator is also known as unary operator b/c  These operator works with one operands. |
| >> | Right Shift |
| << | Left Shift |

**Behavior of operators**

Bitwise AND (&) Operators.

0&0 is 0

0&1 is 0 1&0 is 0   
1&1 is 1

Example - Bitwise AND applies on 5 and 12. We need to convert them in binary.

int x;

x=5 & 12;

5 = 00000000 00000000 00000000 00000101 12 = 00000000 00000000 00000000 00001100 4 = 00000000 00000000 00000000 00000100

Example

#include<stdio.h>

void main()

{

    int x,y,z,m,p,r;

    printf("Enter fist value of x = ");

    scanf("%d",&x);

    printf("Enter second value of y = ");

    scanf("%d",&y);

    printf("Enter second value of p = ");

    scanf("%d",&p);

     printf("Enter second value of r = ");

    scanf("%d",&r);

*//if x=5 and y=12. then value of x&y=4*

*//if x=23 and y=56. then value of x&y=16*

    z = x & y;

    m = p & r;

    printf("\n value of x and y using AND operator  %d",z);

    printf("\n value of x and y using AND operator  %d",m);

}

Bitwise OR ( | ) Operators.   
0|0 is 0   
0|1 is 1   
1|0 is 1   
1|1 is 1

Example - Bitwise OR applies on 5 and 12. We need to convert them in binary

int x;

x=5 & 12;

5 = 00000000 00000000 00000000 00000101 12 = 00000000 00000000 00000000 00001100 13 = 00000000 00000000 00000000 00001101

Example

#include<stdio.h>

void main()

{

    int x,y,z,m,p,r;

    printf("Enter fist value of x = ");

    scanf("%d",&x);

    printf("Enter second value of y = ");

    scanf("%d",&y);

    printf("Enter second value of p = ");

    scanf("%d",&p);

     printf("Enter second value of r = ");

    scanf("%d",&r);

*//if x=5 and y=12. then value of x|y=13*

*//if x=23 and y=56. then value of x|y=63*

    z = x | y;

    m = p | r;

    printf("\n value of x and y using OR operator  %d",z);

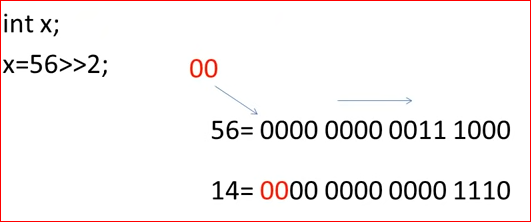
    printf("\n value of x and y using OR operator  %d",m);

}

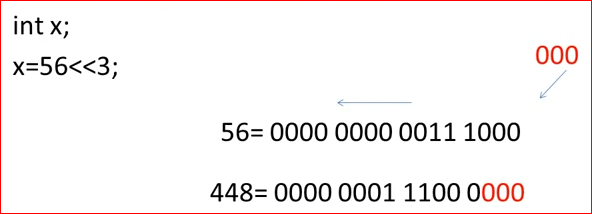
Bitwise XOR (^) Operators.   
0^0 is 0  
0^1 is 1  
1^0 is 1  
1^1 is 0

**Bitwise Right Shift(>>) and Left Shift(<<) Operators**

Right Shift(>>) :- Means add the zeros to the left side

****

Left Shift(<<) :- Means add the zeros to the right side



Example

#include<stdio.h>

void main()

{

    int x,y,z,m,p,r;

    printf("Enter fist value of x = ");

    scanf("%d",&x);

    printf("Enter second value of y = ");

    scanf("%d",&y);

    printf("Enter second value of p = ");

    scanf("%d",&p);

     printf("Enter second value of r = ");

    scanf("%d",&r);

*//if x=5 and y=12. then value of x^y=9*

*//if x=23 and y=56. then value of x^y=47*

    z = x ^ y;

    m = p ^ r;

    printf("\n value of x and y using XOR operator  %d",z);

    printf("\n value of x and y using XOR operator  %d",m);

}

Bitwise NOT (~) Operators.   
~0 is 1  
~1 is 0

Note

* If you use Code block , Visual stdio etc these IDE contain 4 bytes in memory to store any integer value.
* Any negative number store in memory in the form of 2’s compliment.

Int x=5; given binary in 1’s compliment if we add 1 in 1’s compliment then it covert in 2’s compliment

5=000000000000000000000000000000101+1=0000000000000000000000000000110=6

~5=11111111111111111111111111111010 = -6

Most Significant bit(MSB)- If MSB is zero then number is positive.

If MSB is one then number is negative .

**How to store any negative number in memory**

Let b1 is binary of 20.

-20|first find binary number of 20->b1

2’s compliment of b1 whatever comes in b1 after taking

-20- 2’s compliment of b1 i.e called binary of -20 .

Let b1 is binary of x.

> x ->b1

2’s 2’s compliment of b1 whatever comes in b1 after taking

-x->b2 2’s compliment of b1 i.e called binary of -x .again 2’s

compliment of b1 then again it convert in b1.

Most Significant bit(MSB)- If MSB is zero then number is positive.

If MSB is zero then number is negative .

**Example**

**X=~12; ans= -13**

#include<stdio.h>

void main()

{

    int x,y,p,r;

    printf("Enter value of x = ");

    scanf("%d",&x);

    printf("Enter value of r = ");

    scanf("%d",&r);

*//if x=12 then value of y = ~x = -13.*

*//if x=5 then value of y = ~x = -6.*

    y = ~x;

    p = ~r;

    printf("\n value of y using NOT(~) operator  %d",y);

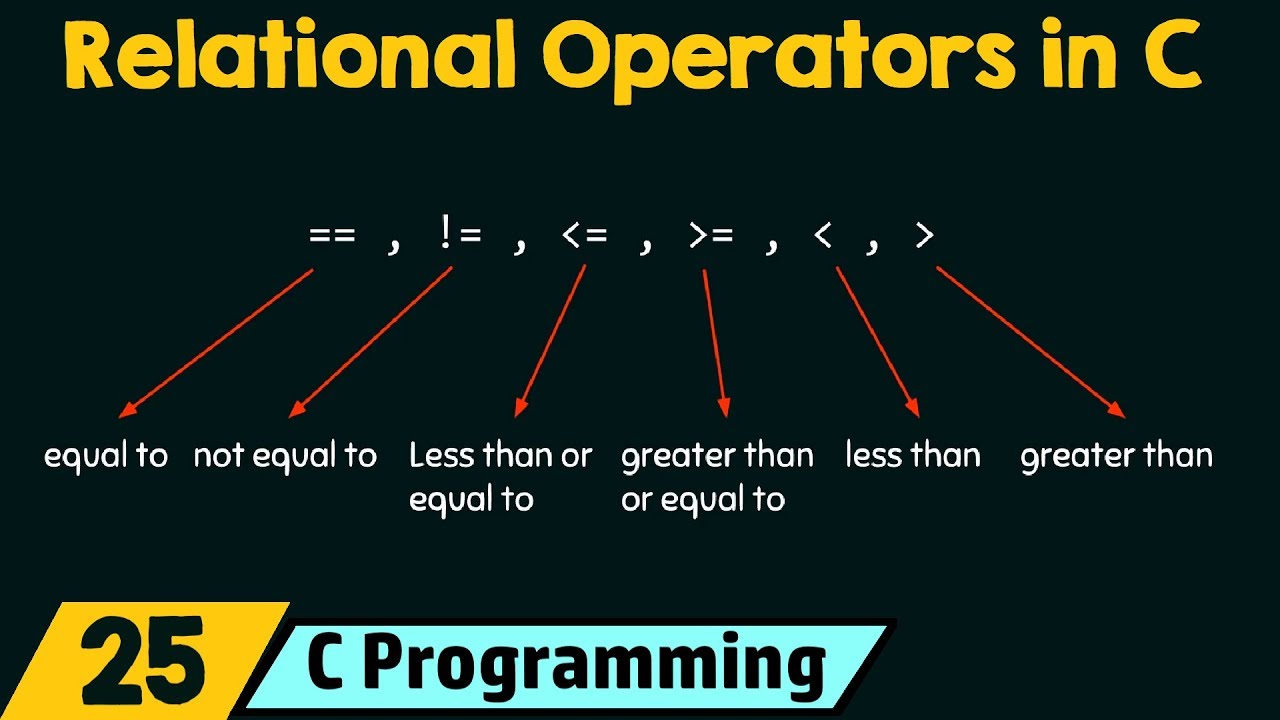
    printf("\n value of y using NOT(~) operator  %d",p);

}

## Relational Operators

Relational Operators are used to states the truth value of the expression. Result of these

operators are either 1 (for true) or 0 (for false). There are six such operators in C language.



Note 📝

* Four operator (<, >, <=, >=) have higher priority then the two operator(==, !=).
* Relational operator always yields result either 0 and 1.
* Every Non-zero value is true and zero is false.
* True is 1 and false is 0.

Example

#include<stdio.h>

void main()

{

*/\*< (less than), > (greater than), <= (less than or equal to), >= (greater than*

*or equal to)==(equal to), != (not equal to)\*/*

    int x,y,z,m,p,r,i,t,q,l,a,n;

    printf("Enter fist value of x = ");

    scanf("%d",&x);

    printf("Enter second value of l = ");

    scanf("%d",&l);

    printf("Enter second value of p = ");

    scanf("%d",&p);

     printf("Enter second value of r = ");

    scanf("%d",&r);

    z = x < l;

    m = p > r;

    q = x <= l;

    i = p >= r;

    t = x == l;

    y = p != r;

*//check four operator (<, >, <=, >=) have higher priority then the two operator(==, !=).*

    a = x<p>r!=l;

    n = l==p>x<4;

    printf("\nRelation between x and y using < operator  %d",z);

    printf("\nRelation between  x and y using > operator %d",m);

    printf("\nRelation between  x and y using <= operator %d",q);

    printf("\nRelation between  x and y using >= operator %d",i);

    printf("\nRelation between  x and y using == operator %d",t);

    printf("\nRelation between  x and y using != operator  %d",y);

    printf("\nRelation between  x and y using == operator %d",a);

    printf("\nRelation between  x and y using != operator  %d",n);

}

//some basic example.

 int s,t,u;

   s=5>6>3; //5>6 results 0. so the expression becomes x=0>3 which is false, thus the result is 0.

   t=5>4>3; //5>4 results 1. so the expression becomes x=1>3 which is false, thus the result is 0.

    u=4>3>0; //4>3 results 1. so the expression becomes x=1>0 which is true, thus the result is 1.

## Logical Operators

C provides three logical operators when we test more than one condition to make decisions. These are: && (meaning logical AND), || (meaning logical OR) and ! (meaning logical NOT).

There are three logical operators in C language:

|  |  |
| --- | --- |
| !TRUE | FALSE |
| !FALSE | TRUE |

Logical NOT(!)  
Logical NOT operator is also a unary operator, as it requires only one operand. Operand is treated as either true or false. NOT operator inverts the truth value, that is, it make false if the operand is true and makes true if the operand is false.

|  |  |
| --- | --- |
| ! | Logical NOT |
| && | Logical AND |
| || | Logical OR |

**Example**

#include<stdio.h>

void main()

{

    int p,r,a;

    printf("Enter the value of p ");

    scanf("%d",&p);

    printf("Enter the value of r ");

    scanf("%d",&r);

*//Use of Logical NOT(!) operator.*

*// Every non zero(1,2,3...) value is always true.*

    a = !p>r;

    printf("Value of this expression !p>r= %d",a);

}

Logical AND(&&)  
Logical AND operator is used to combine two expression, thus it is a binary operator.  
The behavior of logical AND is describes as:

|  |  |  |  |
| --- | --- | --- | --- |
| Expression 1 | && | Expression 2 | Result |
| True | && | True | True |
| True | && | False | False |
| False | && | Don’t Care | False |

Example

#include<stdio.h>

void main()

{

    int p,r,a;

    printf("Enter the value of p ");

    scanf("%d",&p);

    printf("Enter the value of r ");

    scanf("%d",&r);

*//Use of Logical AND(&& operator.*

*// Every non zero(1,2,3...) value is always true.*

*// if both values are true then true.*

    a = p && r;

    printf("Value of this expression !p>r= %d",a);

}

Logical OR(||)  
Logical OR operator is also used to combine two expressions. This is a binary operator. The behavior of logical OR operator is:

|  |  |  |  |
| --- | --- | --- | --- |
| Expression 1 | || | Expression 2 | Result |
| False | || | False | False |
| False | || | True | True |
| True | || | Don't Care | True |

**Example**

#include<stdio.h>

void main()

{

    int p,r,a;

    printf("Enter the value of p ");

    scanf("%d",&p);

    printf("Enter the value of r ");

    scanf("%d",&r);

*//Use of Logical AND(&& operator.*

*// Every non zero(1,2,3...) value is always true.*

*// if both values are false then false.*

    a = p || r;

    printf("Value of this expression !p>r= %d",a);

}

**Compound Assignment Operator**

Assignment operator is the most used operator in the expression. It is sometimes misinterpreted as equal to operators by beginners. Assignment operator is used to assign value to the variable.

Assignment operator (=)  
It is a binary operator where left operand must be a variable.

x=4; //In this expression value 4 is assigned to the variable x.

Following are invalid expressions:

4=x;  
3=4;  
a+3=5;

Left operand must be a variable.

Compound Assignment Operators(+=, -=, \*=, /=, %=)

|  |  |
| --- | --- |
| Operator | Description |
| = | Assign //x=4 In this expression value 4 is assigned to the variable x. |
| += | Increments then assign |
| -= | Decrements then assign |
| \*= | Multiplies then assign |
| /= | Divides then assign |
| %= | Modulus then assign |
| <<= | Left shift and assign |
| >>= | Right shift and assign |
| &= | Bitwise AND assign |
| ^= | Bitwise exclusive OR and assign |
| |= | Bitwise inclusive OR and assign |

Now we need to understand how to work compound assignment operator.**(+=, -=, \*=,%=)**

Int x=4;

X+ =5; //x=x+5;

X- =5; //x=x-5;

X\* =5; //x=x\*5;

X/= 5; //x=x/5;

X%= 5; //x=x%5;

But When both operation will be execute .so less time consume for execute this operation x+=5; compare to this operation x=x+5; .

Int X=4;

X=x\*5+3; is different as x\*=5+3;

4\*5=20. 5+3=8.

20+3=23. 4\*8=32;

Conditional Operator(ternary Operator)

C offers a ternary operator which is the conditional operator (? : in combination) to construct conditional expressions

Ternary operator.- It perform the operation using with three operands .

Conditional operator = Expression1? Expression2 : Expression3 ;

Method1= x>y ? printf(“a”) : printf(“b”) ;

B it means given condition is true then print ‘A’ or if condition is false then print ‘B’ .

Method2=

if(x>y)

Printf(“A”);

else

Printf(“B”);

example

Int x, a, b;

Printf(“ print the value according to condition”);

Scanf(“%d,%d”,&a,&b);

If (a>b)

X=a; x= a>b? a : b;

else

X=b;

     //check priority betwen conditional and assignment operator.

    int a=6,b=7,t;

     a>b?t=a:(t=b);//if you write a>b?t=a:t=b this type not a correct statement.

     printf(" \n print value according to condition=%d",t);

In this expression first execute condition operator and then

assignment operator .so given expression is wrong b/c

b can’t assign in constant value.

a>b?t=a:t =b; //a>b?t=a:t this exp given some constant value.

conditional operator assignment operator

example

#include<stdio.h>

void main()

{

    int x,y,z,p,q;

    printf("Enter fist value of x = ");

    scanf("%d",&x);

    printf("Enter fist value of y = ");

    scanf("%d",&y);

    z=x>y?x:y;

*//both the expression print nereast value of colom(:)*

    p=x>y?2,3,4:7,8,9;

    q=x<y?2,3,4:7,8,9;

    printf("\n Gretest number is  %d",z);

    printf("\n Gretest number is  %d",z=x>y?x:y);

    printf("\n print the value  %d",p);

     printf("\n print the value  %d",q);

*// printf("\n value of x and y using AND operator  %d",m);*

}