

Computer Programming: C

By: Ritu Devi

Arithmetic Operators

- **Arithmetic operator** are used for mathematical calculation. These operator are binary operator that work with integer, floating point number and every character.

| Operator | Purpose |
|----------|---|
| + | addition |
| - | subtraction |
| * | multiplication |
| / | division |
| % | gives the remainder in integer division |

- Integer and Float Conversions:

| Operation | Result | Operation | Result |
|-------------|--------|-------------|--------|
| $5 / 2$ | 2 | $2 / 5$ | 0 |
| $5.0 / 2$ | 2.5 | $2.0 / 5$ | 0.4 |
| $5 / 2.0$ | 2.5 | $2 / 5.0$ | 0.4 |
| $5.0 / 2.0$ | 2.5 | $2.0 / 5.0$ | 0.4 |

- Hierarchy of operators:

| Priority | Operators | Description |
|-----------------|-----------|--|
| 1 st | * / % | multiplication, division, modular division |
| 2 nd | + - | addition, subtraction |
| 3 rd | = | assignment |

E.g., $i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8$

Assignment Operator

- *Assignment operator* are used to assign the value or an expression or a value of a variable to another variable.
- For Example:
 - $x = 8$.
 - $s = x + y - 2$.
 - $x = y = z = 20$.
- $x = x + 2$ can also be written as $x += 2$. Here, ' $+=$ ' is called the *Compound Assignment operator*. Other forms are:
 - $*=$, $/=$, $\%=$ and $-=$.

Increment And Decrement Operators

- C has two useful operators increment ($++$) and decrement ($--$). These are unary operators because they operate on a single operand.
- Increment operator ($++$) increments the value of the variable by 1 and Decrement operator ($--$) decrements the value of the variable by 1.
 - $++x$ is equivalent to $x = x + 1$
 - $--x$ is equivalent to $x = x - 1$
- These operators are of two types
 - Prefix increment / decrement operator
 - Postfix increment / decrement operator

- **Prefix Increment / Decrement:**

- The statement $y = ++x$ is equivalent to these two statements
 - $x = x + 1;$
 - $y = x;$

- **Postfix Increment / Decrement:**

- The statement $y = x++;$ is equivalent to these two statements
 - $y = x;$
 - $x = x + 1;$

- These operators should be used only with variables; they can't be used with constants or expressions.

- For example the expressions $++5$ or $++(x+y+z)$ are invalid.

Relational Operator

- Relational operator are used to compare values of two expressions.

| Operator | Meaning |
|----------|--------------------------|
| < | less than |
| <= | less than or equal to |
| = | equal to |
| != | Not equal to |
| > | Greater than |
| >= | Greater than or equal to |

Example

```
main()
{
    int a=10,b=20,c=30,d,e;
    d=a>b;
    e=(b<=c);
    printf(“%d %d”,d,e);
    getch();
}
```

Output is: 0 1

Logical Operator

- C allows usage of three logical operators. The first two operators, **&&** and **||**, allow two or more conditions to be combined in an expression.

| Operator | Meaning |
|----------|---------|
| && | AND |
| | OR |
| ! | NOT |

- Logical AND compare two operands and return 1 if both condition are true else return 0 (false).
- Logical OR compare two operand and return 1 if any one condition true.
- Logical NOT if the condition is true result is false and if the condition is false result true.

Example

- Suppose three variables having values like, $a = 10$, $b = 5$, $c = 0$

| Expression | | Result | Value of expression |
|------------------------------|----------------|--------|---------------------|
| $(a == 10) \ \&\& \ (b > a)$ | true && false | false | 0 |
| $(b >= a) \ \&\& \ (b == 3)$ | false && false | false | 0 |
| $a \ \&\& \ b$ | true && true | true | 1 |
| $a \ \&\& \ c$ | true && false | false | 0 |

| Expression | | Result | Value of expression |
|---------------------------|----------------|--------|---------------------|
| $a \ \ b$ | true true | true | 1 |
| $a \ \ c$ | true false | true | 1 |
| $(a < 9) \ \ (b > 10)$ | false false | false | 0 |
| $(b != 7) \ \ c$ | true false | true | 1 |

- **Not (!) Operator** is a unary operator and it negates the value of the condition.

| Expression | | Result | Value of expression |
|------------|--------|--------|---------------------|
| !a | !true | false | 0 |
| !c | !false | true | 1 |
| !(b>c) | !true | false | 0 |
| !(a && c) | !false | true | 1 |

Conditional Operator

- The conditional operators `?` and `:` are sometimes called **ternary operators**. since they take three arguments. Their general form is,
 - *expression 1 ? expression 2 : expression 3*
 - It says that: “if expression 1 is true (that is, if its value is non-zero), then the value returned will be expression 2, otherwise the value returned will be expression 3”.
- For Example
 - `int x, y ;`
 - `scanf ("%d", &x) ;`
 - `y = (x > 5 ? 3 : 4) ;`
- *The limitation of the conditional operators is that after the `?` or after the `:` only one C statement can occur.*

The equivalent **if** statement will be,

```
if ( x > 5 )  
    y = 3 ;  
else  
    y = 4 ;
```

Comma Operator

- The expressions are separated by the comma operator. The comma operator helps to make the code more compact

For example

```
#include<stdio.h>
main( )
{
    int a , b , c, sum;
    sum =(a=8,b=7,c=9,a+b+c);
    printf("Sum %d\n",sum);
}
```

sizeof Operator

- **sizeof** is an unary operator. This operator gives the size of its operand in terms of bytes. The operand can be a variable, constant or any data type.
- Example

```
#include<stdio.h>
main( )
{
    int var;
    printf ("Size of int =%d", sizeof(int));
    printf("Size of float =%d", sizeof(float));
    printf ("Size of var = %d", sizeof(var));
    printf("Size of an integer constant = %d", sizeof(45));
}
```

Bitwise Operators

- C has the ability to support the manipulation of data at the bit level. Bitwise operators are used for operations on individual bits. Bitwise operators operate on integers only.

| Bitwise operator | Meaning |
|------------------|------------------|
| & | bitwise AND |
| | bitwise OR |
| ~ | one's complement |
| << | left shift |
| >> | right shift |
| ^ | bitwise XOR |

Bitwise operator

- **& (bitwise AND)** Takes two numbers as operands and does AND on every bit of two numbers. The result of AND is 1 only if both bits are 1.
- **| (bitwise OR)** Takes two numbers as operands and does OR on every bit of two numbers. The result of OR is 1 if any of the two bits is 1.
- **^ (bitwise XOR)** Takes two numbers as operands and does XOR on every bit of two numbers. The result of XOR is 1 if the two bits are different.
- **<< (left shift)** Takes two numbers, left shifts the bits of the first operand, the second operand decides the number of places to shift.
- **>> (right shift)** Takes two numbers, right shifts the bits of the first operand, the second operand decides the number of places to shift.
- **~ (bitwise NOT)** Takes one number and inverts all bits of it


```

#include <stdio.h>
int main()
{
    // a = 5(00000101), b = 9(00001001)
    unsigned char a = 5, b = 9;

    // The result is 00000001
    printf("a = %d, b = %d\n", a, b);
    printf("a&b = %d\n", a & b);

    // The result is 00001101
    printf("a|b = %d\n", a | b);

    // The result is 00001100
    printf("a^b = %d\n", a ^ b);

    // The result is 11111010
    printf("~a = %d\n", a = ~a);

    // The result is 00010010
    printf("b<<1 = %d\n", b << 1);

    // The result is 00000100
    printf("b>>1 = %d\n", b >> 1);

    return 0;
}

```

OUTPUT:

```

a = 5, b = 9
a&b = 1
a|b = 13
a^b = 12
~a = 250
b<<1 = 18
b>>1 = 4

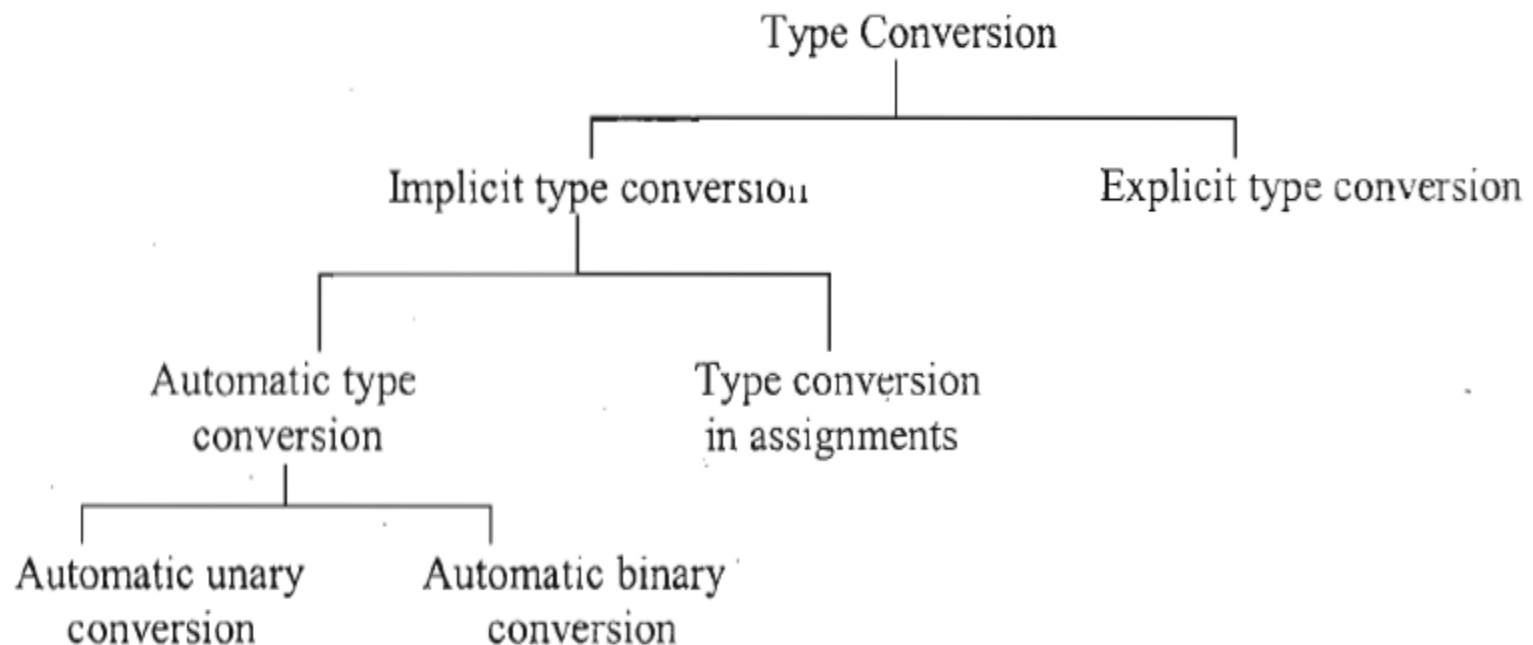
```

Some facts about bitwise operators

- The **left shift** and **right shift** operators should not be used for negative numbers
- The bitwise operators should not be used in place of logical operators. The result of logical operators (&&, || and !) is either 0 or 1, but bitwise operators return an integer value.
- The left-shift and right-shift operators are equivalent to multiplication and division by 2 respectively.

Type Conversion in C

- C provides the facility of mixing different types of variables and constants in an expression. In these types of operations data type of one operand is converted into data type of another operand. This is known as *type conversion*.



Implicit Type Conversions

- These conversions are done by the C compiler. The two types of implicit type conversions are *automatic type conversions* and *type conversion in assignment*.
- **Automatic Conversion**
 - **Automatic unary conversions:** All operands of type char and short will be converted to int before any operation. Some compilers convert all float operands to double before any operation.
 - **Automatic binary conversions:** Whenever there are two operands of different data types the operand with a lower rank will be converted to the type of higher rank operand. This is called promotion of data type.

- **Type Conversion In Assignment:**

If the types of the two operands in an assignment expression are different, then the type of the right hand side operand is converted to the type of left hand operand.

For Example

1. Some high order bits may be dropped when long is converted to int, or int is converted to short int or char.
2. Fractional part may be truncated during conversion of float type to int type.

Explicit Type Conversion Or Type Casting

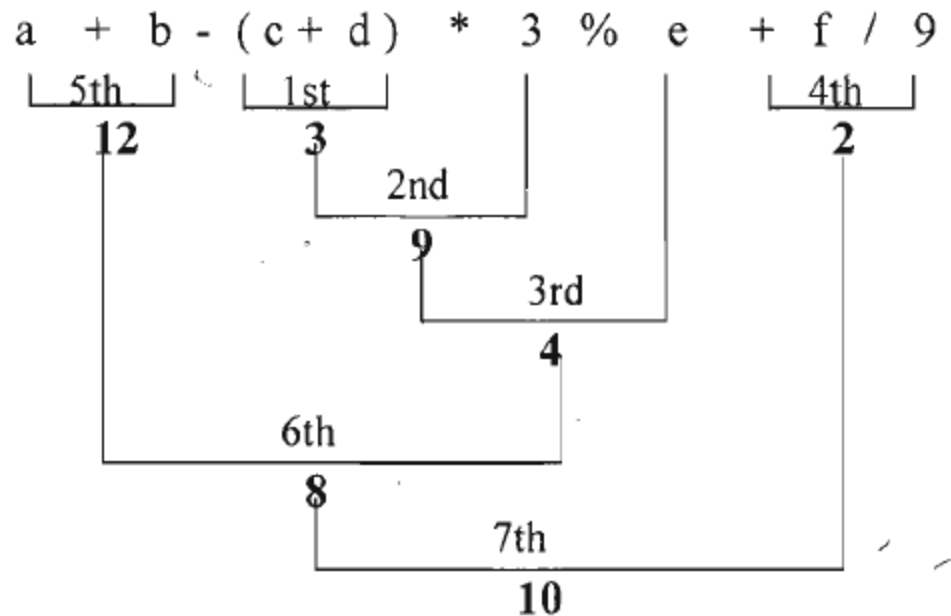
```
main()
{
    int x=5,y=2;
    float p,q;
    p=x/y;
    printf("p = %f\n",p);
    q=(float)x/y;
    printf("q = %f\n",q);
}
```

- In these types of cases we can specify our own conversions known as type casting or coercion. The cast operator is a unary operator that is used for conversion of an expression to a particular data type temporarily. The expression can be any constant or variable
- The syntax of cast operator is-
 - (datatype) expression
 - Here the datatype along with the parentheses is called the cast operator

Precedence And Associativity Of Operators

- For evaluation of expressions having more than one operator, there are certain precedence and associativity rules defined in C.
- Whenever an expression contains more than one operator, the operator with a higher precedence is evaluated first, Consider the following expression
 - $2 + 3 * 5$
- Now consider a situation when two operators with the same precedence occur in an expression For example- - ,
 - $5 + 16 / 2 * 4$
- To solve these types of problems, an associativity property is assigned to each operator.

- All the operators either associate from left to right or from right to left.
- If we want to change the order of precedence of any operation, we can use parentheses.



Exercises

• $a = 17, b = 5, c = 6, d = 3, e = 5$ [ans-5]
 $a \% 6 - b / 2 + (c * d - 5) / e$

• $a = 4, b = 5, c = 6, d = 3, e = 5, f = 10$ [ans-0]
 $a * b - c / d < e + f$

• $a = 8, b = 5, c = 8, d = 3, e = 65, f = 10, g = 2, h = 5, k = 2$ [ans-1]
 $a - b + c / d == e / f - g + h \% k$

• $a = 8, b = 3, c = 2, d = 3, e = 2, f = 11$ [ans-1]
 $a - b \ || \ (a - b * c) + d \ \&\& \ e - f \% 3$

| Operator | Description | Precedence level | Associativity |
|--|--|------------------|---------------|
| () [] → . | Function call Array subscript Arrow operator Dot operator | 1 | Left to Right |
| + - ++ -- ! ~ * & (datatype) sizeof | Unary plus Unary minus Increment Decrement Logical NOT One's complement Indirection Address Type cast Size in bytes | 2 | Right to Left |
| * / % | Multiplication Division Modulus | 3 | Left to Right |
| + - | Addition Subtraction | 4 | Left to Right |

C Instructions

There are basically three types of instructions in C:

- a) **Type Declaration Instruction:** used to declare the type of variables being used in the program .

For example

- `int i = 10, j = 25 ;`
- `float a = 1.5, b = a + 3.1 ;`

- b) **Arithmetic Instruction:** To perform arithmetic operations between constants and variables.

- c) **Control Instruction:** To control the sequence of execution of various statements in a C program.

- **Control Instructions:**

- Enable us to specify the order in which the various instructions in a program are to be executed by the computer.
- It determine the 'flow of control' in a program. There are four types of control instructions in C.
 1. Sequence Control Instruction.
 2. Selection or Decision Control Instruction.
 3. Repetition or Loop Control Instruction.
 4. Case Control Instruction.

Decision Control Instruction

A decision control instruction can be implemented in C using:

- The **if** statement
- The **if-else** statement
- The conditional operators

The *if* Statement

- The general form of **if** statement is:

if (expression)
statement ;

- For example

```
main()
{   int a;
    printf("enter value of a");
    scanf("%d",&a);
    if(a>25)
    {       printf("no.is greater than 25");
    }
    printf("\n bye");
    getch();
}
```

The if-else Statement

- we can execute one group of statements if the expression evaluates to true and another group of statements if the expression evaluates to false.

```
main()
{
    int n,c;
    printf("\n enter value of n");
    scanf("%d",&n);
    c=n%2;
    if(c==0)
        printf("no is even");
    else
        printf("no is odd");
    getch();
}
```

Nesting of if... else

```
if(condition 1)
{
    if(condition 2)
        statementA1;
    else
        statementA2;
}
else
{
    if(condition 3)
        statementB1;
    else
        statementB2 ;
}
```



```
/*Program to find largest number from three given numbers*/  
main()  
{  int a, b, c, large;  
    printf("Enter three numbers ") ;  
    scanf("%d%d%d",&a,&b,&c);  
    if (a>b)  
    {      if(a>c)  
            large=a;  
        else  
            large=c;  
    }  
    else  
    {      if (b>c)  
            large=b;  
        else  
            large=c;  
    }  
    printf ("Largest number is %d\n", large) ;  
} /*End of main() */
```

Else If Ladder

- There is one more way in which we can write program
- **Syntax Of Else If Leader:**

```
if(test_condition1)
{
    statement 1;
}
else if(test_condition2)
{
    statement 2;
}
else
{
}
```

```
void main ( )
{
    int num = 10 ;
    if ( num > 0 )
        printf ("\n Number is Positive");
    else if ( num < 0 )
        printf ("\n Number is Negative");
    else
        printf ("\n Number is Zero");
}
```

Loop Control Instruction

- **Loops** are used when we want to execute a part of the program or a block of statements several times.
- There are three loop statements in C
 - **for** statement .
 - **while** statement .
 - **do-while** statement.

The while loop

- The general form is

 initialise loop counter ;

 while (test loop counter using a condition)

 {

 Statements.

 increment loop counter ;

 }

For example:

```
main( )
{
    int i = 1 ;
    while ( i <= 10 )
    {
        printf ( "%d\n", i ) ;
        i = i + 1 ;
    }
}
```

For loop

- The General form is:

```
for( initialise counter ; test counter ; increment counter )  
{  
    Statements  
}
```

- Some examples are

- `for (i = 10 ; i ; i --)`
 `printf("%d", i);`
- `for (i < 4 ; j = 5 ; j = 0)`
 `printf("%d", i);`
- `for (i = 1; i <= 10 ; printf("%d", i++));`
- `for (scanf ("%d", &i) ; i <= 10 ; i++)`

Nesting of loop

- **while** and **for** loops can be nested. a for loop can occur within a while loop, or a while within a for.

```
/* Demonstration of nested loops */
main( )
{
    int  r, c, sum ;
    for ( r = 1 ; r <= 3 ; r++ ) /* outer loop */
    {
        for ( c = 1 ; c <= 2 ; c++ ) /* inner loop */
        {
            sum = r + c ;
            printf ( "r = %d c = %d sum = %d\n", r, c, sum ) ;
        }
    }
}
```

```
r = 1 c = 1 sum = 2
r = 1 c = 2 sum = 3
r = 2 c = 1 sum = 3
r = 2 c = 2 sum = 4
r = 3 c = 1 sum = 4
r = 3 c = 2 sum = 5
```

Do-while loop

- Here firstly the statements inside loop body are executed and then the condition is evaluated. General form is:

```
Do
{
    statements;
} while ( this condition is true );
```

- For example:

```
main( )
{
    Do
    {
        printf ( "Hello there \n" );
    } while ( 4 < 1 );
}
```

Case control instruction :Switch

- **Switch** statement is a multi-way decision making statement which selects one of the several alternative based on the value of integer variable or expression.
- The **integer expression** following the keyword **switch** is any C expression that will yield an integer value.
- The keyword **case** is followed by an integer or a character constant.
- Each constant in each **case** must be different from all the others.

```
switch ( integer expression )  
{  
    case constant 1 :  
        do this ;  
    case constant 2 :  
        do this ;  
    case constant 3 :  
        do this ;  
    default :  
        do this ;  
}
```


Switch-case Example

```
main(.)
{
    int choice;
    printf("Enter your choice : ");
    scanf("%d",&choice);
    switch(choice)
    {
        ✓ case 1:
            printf("First\n");
        ✓ case 2:
            printf("Second\n");
        case 3:
            ✓ printf("Third\n");
        default:
            ✓ printf("Wrong choice\n");
    }
}
```

Output:

```
Enter your choice : 2
Second
Third
Wrong choice
```

```
main ()
{
    int choice;
    printf("Enter your choice ");
    scanf("%d",&choice);
    switch(choice)
    { case 1:
        printf("First\n");
        break;
        case 2:
        printf("Second\n");
        break;
        case 3:
        printf("Third\n");
        break;
        default:
        printf("Wrong choice\n");
    }
```

Limitations of using switch

- A float expression cannot be tested using a **switch**
- cases can never have variable expressions
- Multiple cases cannot use same expressions

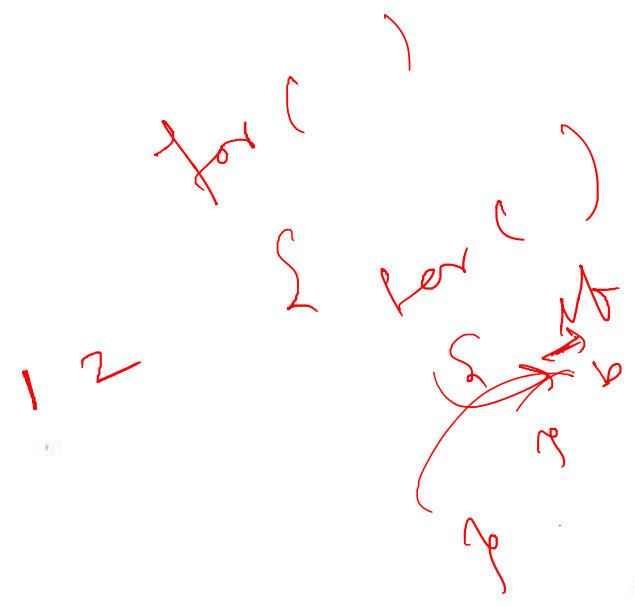
break vs continue vs goto statement

- A **break** statement takes the execution control out of the loop.
- A **continue** statement skips the execution of the statements after it and takes the control to the beginning of the loop.
- The **goto** is an unconditional control statement that transfers the flow of control to another part of the program. The goto statement can be used as:

```
goto label;  
.....  
.....  
label:  
    statement;  
.....  
.....
```

Example: break

```
int n;  
for(n=1;n<=5;n++)  
{  
    if(n==3)  
    {  
        printf("I understand the use of break\n");  
        break;  
    }  
    printf("Number = %d\n",n);  
}  
printf("Out of for loop\n");
```



Example: continue

```
main()  
{  
    int i=1,n,sum=0;  
    float avg;  
    printf("Enter 10 positive numbers : \n");  
    while(i<=10)  
    {  
        printf("Enter number %d : ",i);  
        scanf("%d", &n);  
        if(n<0)  
        {  
            printf("Enter only positive numbers\n");  
            continue;  
        }  
        sum+=n;  
        i++;  
    }  
    avg=sum/10.0;  
    printf("Sum = %d    Avg = %f\n",sum,avg);  
}
```

Example: goto

```
#include<stdio.h>
main( )
{
    int n;
    printf("Enter the number : ");
    scanf(" %d", &n);
    if(n%2==0)
        goto even;
    else
        goto odd;
even :
    printf("Number is even");
    goto end;
odd :
    printf("Number is odd");
    goto end;
end:
    printf("\n");
}
```

Exercises

1. Program to print the sum and product of digits of any number.
2. Program to find the factorial of any number.
3. Multiply two positive numbers without using * operator.
4. Program to generate Fibonacci series: 1,1,2,3,5,8,13,34,55,89, ..
5. Program to print armstrong numbers from 100 to 999(for e.g., $371 = 3^3 + 7^3 + 1^3 = 27 + 343 + 1 = 371$).
6. Program to find whether a number is prime or not.
7. Program to find the LCM and HCF of two numbers

Pyramids

| | | | | | | | |
|-------|-----------|-----------|----------------|------------|-----------|-----------|-----------|
| * | 1 | 1 | 1 | 2 | 1 | 5 | 5 |
| ** | 2 2 | 1 2 | 2 3 | 3 4 | 0 1 | 5 4 | 4 4 |
| *** | 3 3 3 | 1 2 3 | 4 5 6 | 4 5 6 | 1 0 1 | 5 4 3 | 3 3 3 |
| **** | 4 4 4 4 | 1 2 3 4 | 7 8 9 10 | 5 6 7 8 | 0 1 0 1 | 5 4 3 2 | 2 2 2 2 |
| ***** | 5 5 5 5 5 | 1 2 3 4 5 | 11 12 13 14 15 | 6 7 8 9 10 | 1 0 1 0 1 | 5 4 3 2 1 | 1 1 1 1 1 |
| (a) | (b) | (c) | (d) | (e) | (f) | (g) | (h) |

| | | | | | | |
|-----------|-----------|-----------|-----------|-----------|-------|-------|
| * * * * * | 5 5 5 5 5 | 1 2 3 4 5 | 5 4 3 2 1 | 1 1 1 1 1 | * | * |
| * * * * | 4 4 4 4 | 1 2 3 4 | 5 4 3 2 | 2 2 2 2 | ** | ** |
| * * * | 3 3 3 | 1 2 3 | 5 4 3 | 3 3 3 | *** | *** |
| * * | 2 2 | 1 2 | 5 4 | 2 2 | **** | **** |
| * | 1 | 1 | 5 | 1 | ***** | ***** |
| (i) | (j) | (k) | (l) | (m) | (n) | (o) |