## Function with default arguments

A function can be called without specifying all its arguments. This can be achieved only if the function declaration provides default values for those arguments that are not specified.

**Example:**

```
#include <iostream>
using namespace std;
void repchar(char='*', int=45); //declaration with
                                //default arguments
int main()
{
repchar();            //prints 45 asterisks
repchar('=');         //prints 45 equal signs
repchar('+', 30);     //prints 30 plus signs
return 0;
}

void  repchar(char ch, int n)
{
    for(int j=0; j<n; j++)
         cout << ch;
    cout << endl;
}
```

**Note:**

- If one argument is missing when the function is called, it is assumed to be the last argument.
- The missing arguments must be the trailing arguments—those at the end of the argument list.

## Recursive Function

A recursive function is a function that calls itself in order to perform a task of computation. There are two basic components of a recursive solution:

- Termination step, stating the solution when the process comes to an end.
- Inductive step, calling the function itself with a renewed (lesser) parameter value.

**Example:** Write a C++ program that computes the factorial of a positive integer number using the **recursive** function **factorial()**.

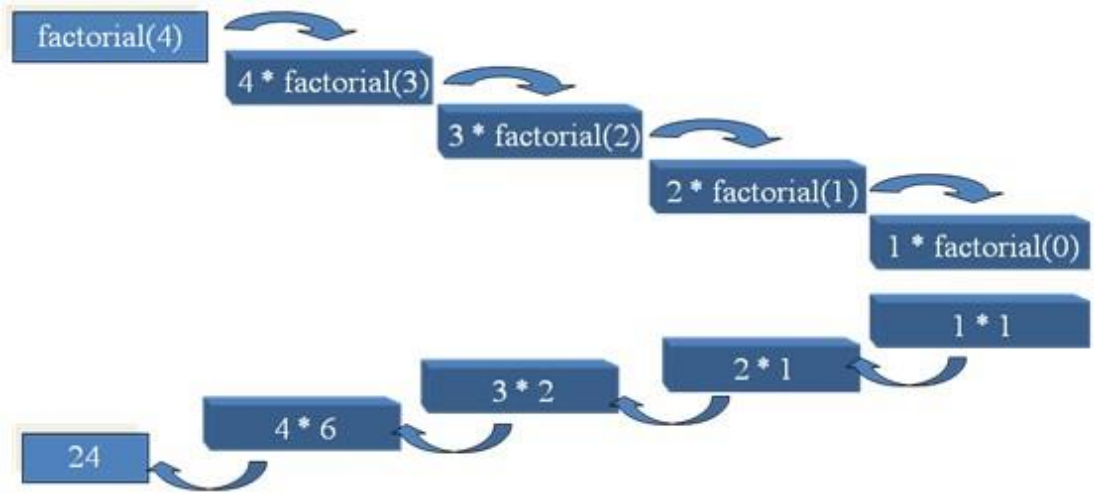
```
#include <iostream>
using namespace std;
long factorial(int);
int main()
{
  int number;
  cout<<"Enter a positive integer number: ";
  cin>>number;
  cout<<"The factorial is " << factorial(number);
return 0;
}

long factorial (int n)
{
  if (n == 0)
     return 1;
  else
    return (n * factorial(n - 1));
}
```

Termination Step

Inductive Step

Calling the function factorial with argument 4, i.e. factorial(4):



**Important Notes**

- Recursive functions must have a termination step and an inductive step.
- Recursive function calls in inductive step must signify a reduction of argument value.
- Be aware of infinite recursion problem.
- Recursion is implicit while iteration is explicit.
- Recursion is slower than iteration.

**Example:** Write a C++ program that counts the number of digits of an entered integer number using the **recursive** function **countingDigits()**.

```
#include<iostream>
using namespace std;
int countingDigits(long);
int main()
{
 long number;
 cout<<"Enter an integer number: ";
 cin>>number;
 cout<<"No. of digits is " <<countingDigits(number)
     <<endl;
return 0;
}

int countingDigits(long n)
{
   if (n / 10 == 0)
          return 1;
    else
          return( countingDigits(n/10) + 1 );
}
```

## Function Overloading

Overloading refers to the use of the same thing for different purposes. Function overloading means that we can use the same function name to create functions that perform a variety of different tasks. These overloaded functions have the same function name but with different argument lists (i.e. different number and/or different data types of arguments). An overloaded function appears to perform different activities depending on the kind of data sent (passed) to it. It performs one operation on one kind of data but another operation on a different kind.

**Example:** Write a C++ program that computes the area of square and the area of rectangle using the **overloaded** function **area()**.

```
#include<iostream>
using namespace std;
int area(int);
int area(int , int);
int main()
{
 int length , width;
 cout<<"Enter a length of square: ";
 cin>>length;
 cout<<"The area of square is "<<area(length)<<endl;
 cout<<"Enter a length and width of rectangle: ";
 cin>>length>>width;
 cout<<"The area of rectangle is "
     <<area(length,width)<<endl;
return 0;}

int area(int a)
{
 return (a * a);
}

int area(int a , int b)
{
 return (a * b);
}
```

**Example:** Write a C++ program that computes the volume of cube, cylinder, and rectangle using the **overloaded** function **volume()**.

```
#include<iostream>
using namespace std;
int volume(int);
double volume(double , int);
long volume(long, int, int);
int main()
{
 cout<< volume(10) <<endl
     << volume(2.5 , 8) <<endl
     << volume(100, 75, 15) << endl;
return 0;
}

int volume(int s)
{
 return (s * s * s);
}

double volume(double r , int h)
{
 return (3.14519 * r * r * h);
}

long volume(long l, int b, int h)
{
 return (l * b * h);
}
```

## Homework:

1. Write a C++ program that computes the power of an entered integer number using the **recursive** function **power()**.
2. Write a C++ program that adds three numbers of different numeric data types (e.g. integers, float, double) using the **overloaded** function **add()**.
3. Write a C++ program that sort an array (with size of 12) elements for two different numeric data types (integers, float) using the **overloaded** function **sort()**.