COMPUTER SCIENCE : C++

DEFAULT ARGUMENTS IN C++ FUNCTIONS

The default arguments are used when you provide no arguments or only few arguments while calling a function. The default arguments are used during compilation of program. For example, lets say you have a user-defined function sum declared like this: int sum(int a=10, int b=20), now while calling this function you do not provide any arguments, simply called sum(); then in this case the result would be 30, compiler used the default values 10 and 20 declared in function signature. If you pass only one argument like this: sum(80) then the result would be 100, using the passed argument 80 as first value and 20 taken from the default argument.

Default arguments in C++

```
#include <iostream.h>
using namespace std;
int sum(int a, int b=10, int c=20);

int main(){
   /* In this case a value is passed as
    * 1 and b and c values are taken from
    * default arguments.
   cout<<sum(1)<<endl;

   /* In this case a value is passed as
    * 1 and b value as 2, value of c values is
    * taken from default arguments.
   cout<<sum(1, 2)<<endl;

   /* In this case all the three values are
    * passed during function call, hence no
    * default arguments have been used.
       cout<<sum(1, 2, 3)<<endl;
   return 0;
}
int sum(int a, int b, int c){
   int z;
   z = a+b+c;
   return z;
}
```

Output:31 23 6

Rules of default arguments

As you have seen in the above example that I have assigned the default values for only two arguments b and c during function declaration. It is up to you to assign default values to all arguments or only selected arguments but remember the following rule while assigning default values to only some of the arguments:

If you assign default value to an argument, the subsequent arguments must have default values assigned to them, else you will get compilation error.

For example: Lets see some valid and invalid cases.
Valid: Following function declarations are valid –

```
int sum(int a=10, int b=20, int c=30);
int sum(int a, int b=20, int c=30);
int sum(int a, int b, int c=30);
```

Invalid: Following function declarations are invalid –

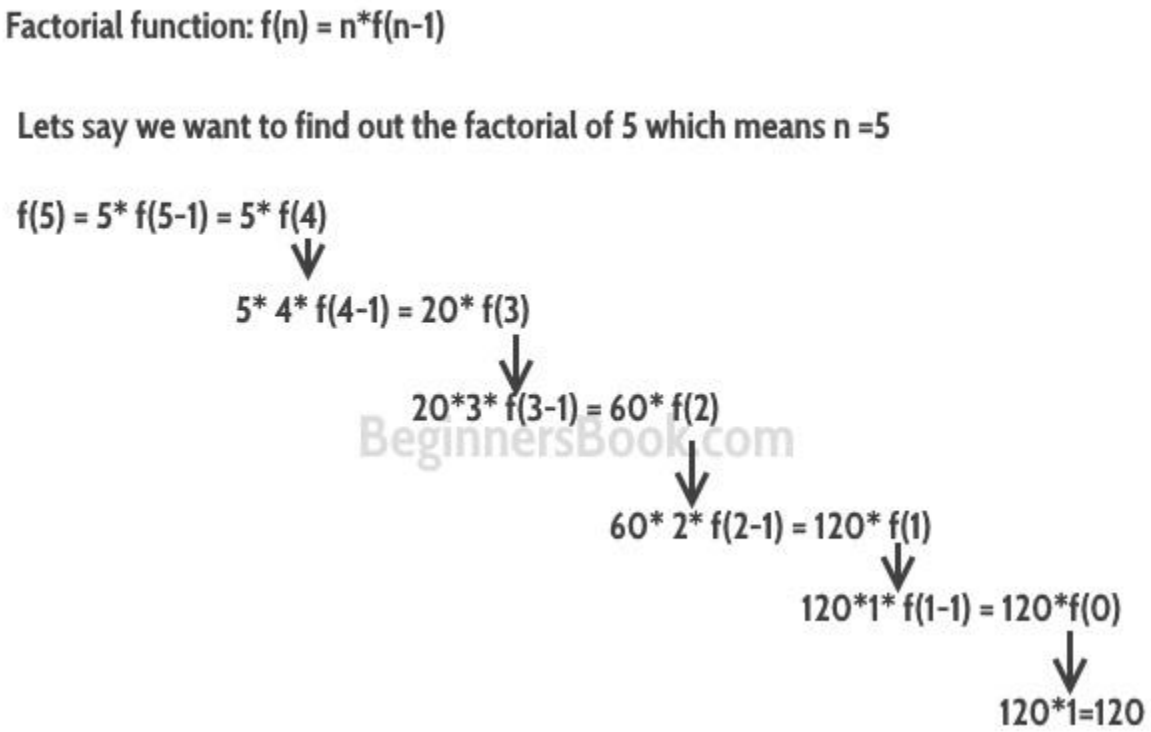
```
/* Since a has default value assigned, all the
 * arguments after a (in this case b and c) must have
 * default values assigned
 */
int sum(int a=10, int b, int c=30);

/* Since b has default value assigned, all the
 * arguments after b (in this case c) must have
 * default values assigned
 */
int sum(int a, int b=20, int c);

/* Since a has default value assigned, all the
 * arguments after a (in this case b and c) must have
 * default values assigned, b has default value but
 * c doesn't have, thats why this is also invalid
 */
int sum(int a=10, int b=20, int c);
```

C++ Recursion

The process in which a function calls itself is known as recursion and the corresponding function is called the **recursive function**. The popular example to understand the recursion is factorial function.

Factorial function: f(n) = n*f(n-1), base condition: if n<=1 then f(n) = 1. Don't worry we wil discuss what is base condition and why it is important.

In the following diagram. I have shown that how the factorial function is calling itself until the function reaches to the base condition.



.

C++ recursion: Factorial

```
#include <iostream>
using namespace std;
//Factorial function
int f(int n){
   /* This is called the base condition, it is
    * very important to specify the base condition
    * in recursion, otherwise your program will throw
    * stack overflow error.
    */
   if (n <= 1)
        return 1;
   else
       return n*f(n-1);
}
int main(){
   int num;
   cout<<"Enter a number: ";
   cin>>num;
   cout<<"Factorial of entered number: "<<f(num);
   return 0;
}
```

Output:

```
Enter a number: 5
Factorial of entered number: 120
```

Base condition

In the above program, you can see that I have provided a base condition in the recursive function. The condition is:

```
if (n <= 1)
        return 1;
```

The purpose of recursion is to divide the problem into smaller problems till the base condition is reached. For example in the above factorial program I am solving the factorial function f(n) by calling a smaller factorial function f(n-1), this happens repeatedly until the n value reaches base condition(f(1)=1). If you do not define the base condition in the recursive function then you will get stack overflow error.

Direct recursion vs indirect recursion

Direct recursion: When function calls itself, it is called direct recursion, the example we have seen above is a direct recursion example.

Indirect recursion: When function calls another function and that function calls the calling function, then this is called indirect recursion. For example: function A calls function B and Function B calls function A.

Indirect Recursion in C++

```
#include <iostream.h>
using namespace std;
int fa(int);
int fb(int);
int fa(int n){
   if(n<=1)
      return 1;
   else
      return n*fb(n-1);
}
int fb(int n){
   if(n<=1)
      return 1;
   else
      return n*fa(n-1);
}
int main(){
   int num=5;
   cout<<fa(num);
   return 0;
}
```

Output: 120