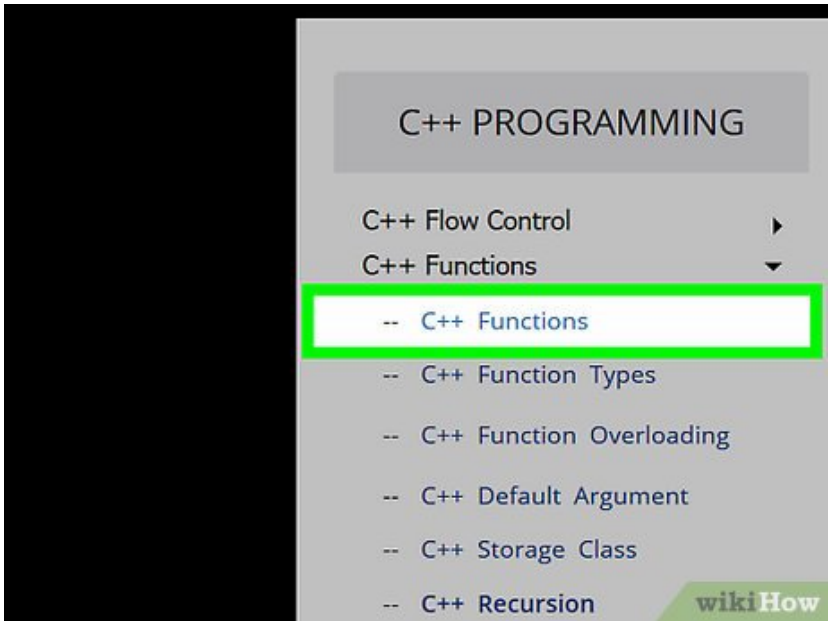
How to Create a Recursive Function in C++

Recursion is a major topic in computer science. It is a method where a problem is reduced or simplified to smaller versions of itself until a solution is achieved. This tutorial will demonstrate how to write a simple recursive function in...

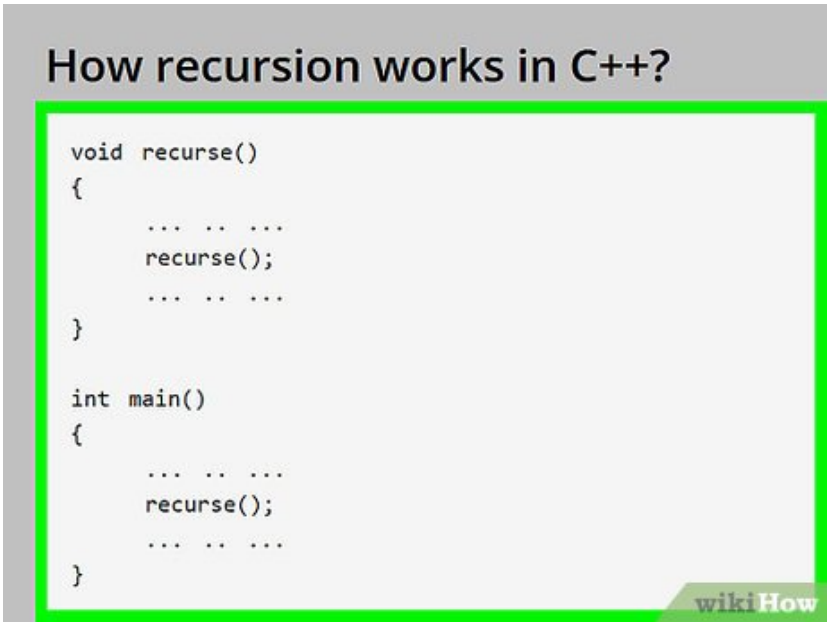
Part 1 of 2:

Understanding When to Use a Recursive Function

1. 

Know when you might need to use recursion. Imagine a scenario where you have a given mathematical problem and you need to simplify it to achieve a solution. For example, you have a number which needs to be reduced (subtracted) until it reaches a particular number. This is a very simple case of course but we can apply recursion to do this.

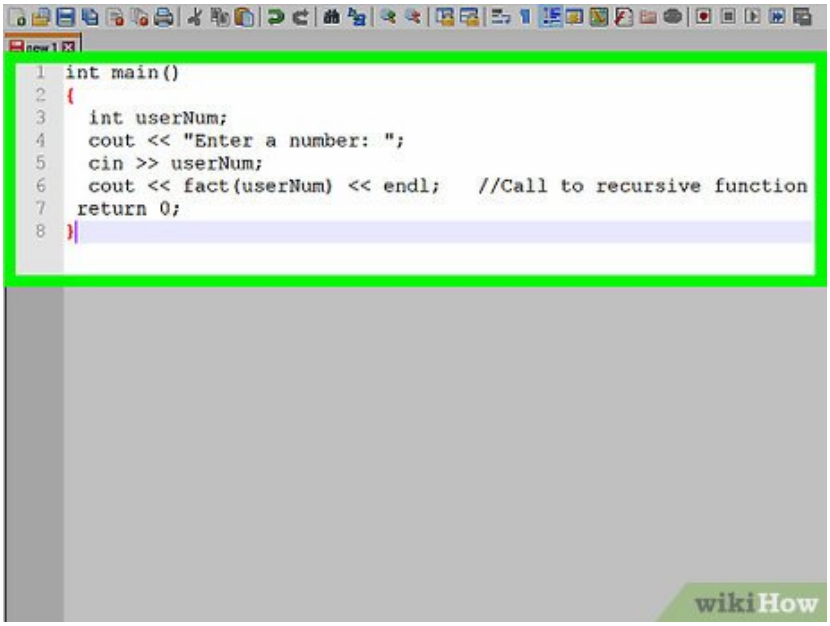
1. Another case might be finding different Fibonacci numbers. Fibonacci numbers are sums of the previous two numbers of the Fibonacci sequence. We can invent algorithms to find different numbers in the Fibonacci sequence recursively. Can you think of some other examples where you might find use in recursion?

2. 

Understand how recursion is used in C++. In recursion this process of simplification is attained when a function calls itself. In the programming language C++, you merely create a function wherein a statement exists that initiates a call to the function again. Just think of the reflection you see when a mirror is placed behind the mirror you're facing.

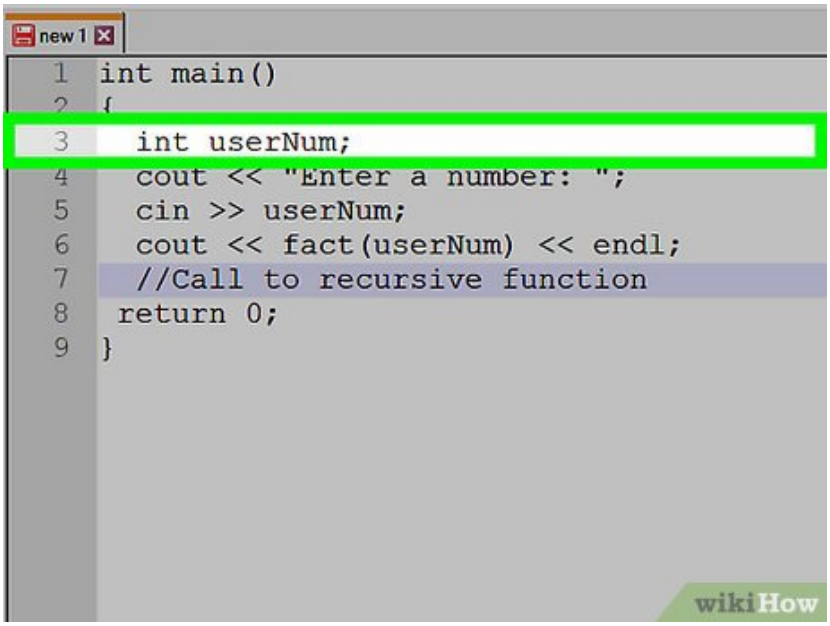
Part 2 of 2:

Creating a Recursive Function

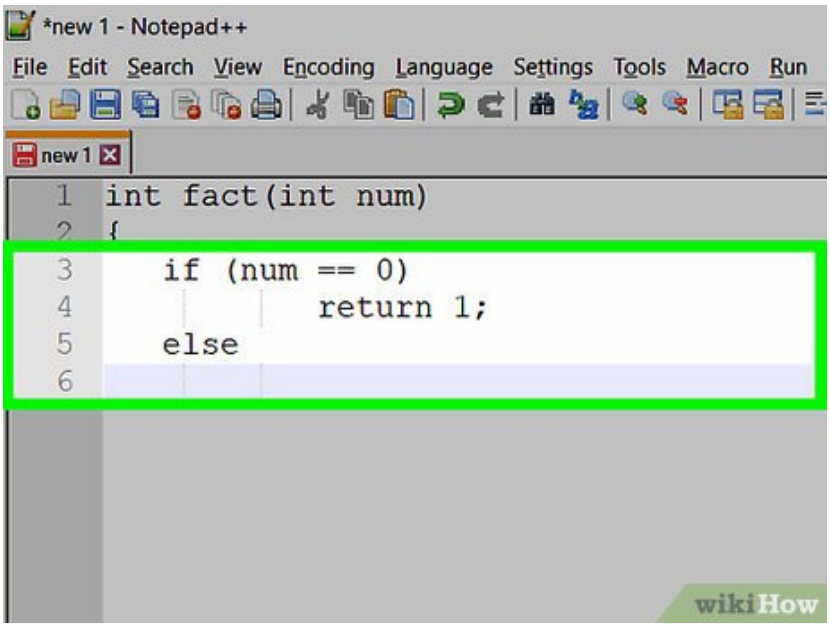
1. 

Write a basic program. A simple program that uses only a main function with an uninitialized `int` variable. In particular, this program will ask the user to input an integer number and store it in the uninitialized variable. Here's an example of such a program:

```
| int main() { int userNum; cout  "Enter a number: "; cin >> userNum;
cout  fact(userNum)  endl;   //Call to recursive function  return 0; }
```

2. 

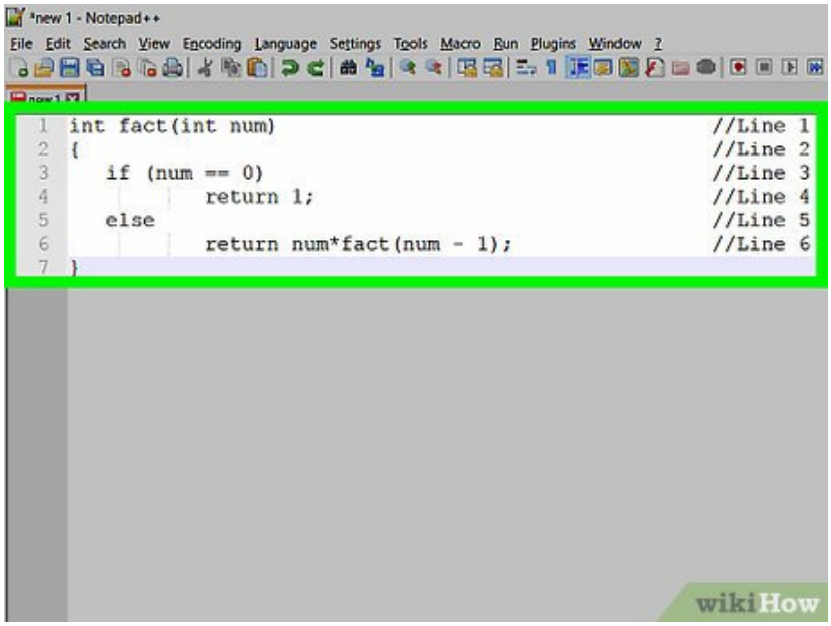
Create an `int` function that has one formal parameter. This function will take as input the user-entered variable from the main function in the basic program.

3. 

Write code in the function body that uses an `if/else` structure. Here is where things will get tricky as you implement the recursive algorithm in the function. You'll use recursion here to help you find the factorial of a number. The factorial of a number is the product of all positive integers less than or equal to it. For example, the factorial of 4 is: 4 x 3 x 2 x 1 = 24.

1. The `if` statement will return the final solution to the problem (and is usually but *not* always 1 or 0) and terminate the call to the function. This is called the **base case**.

2. The `else` structure can be used to call the function itself using the parameter thus creating the recursion. Note, however, that the parameter should be modified in this recall so it begins approaching the value of the base case in the if structure. Otherwise this will create an infinite recursion.

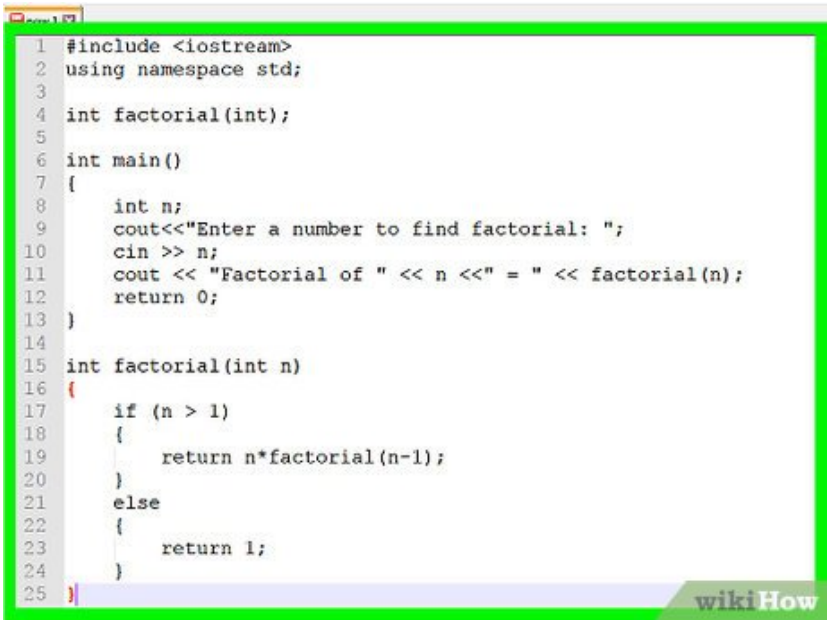
4. 

Copy an example. Here is an example recursive function that expands on the function call in the basic program code:

```
| int fact(int num)                                          //Line 1 {
                                                            //Line 2 if (num ==
0)                                          //Line 3 return 1;
                                      //Line 4 else
                                                  //Line 5 return num*fact(
num - 1);          //Line 6 }
```

1. As mentioned above, this particular function allows you to find the factorial of an integer. Note the base case in Lines 3 and 4 returns 1 if the user value of `num` is 0. The **general case** in Line 6 calls the function again recursively but it reduces the parameter `num` by 1 to approach the base case.

5.



```
#include <iostream>
using namespace std;

int factorial(int);

int main()
{
    int n;
    cout<<"Enter a number to find factorial: ";
    cin >> n;
    cout << "Factorial of " << n <<" = " << factorial(n);
    return 0;
}

int factorial(int n)
{
    if (n > 1)
    {
        return n*factorial(n-1);
    }
    else
    {
        return 1;
    }
}
```


Do more example problems with recursion. Programming concepts, like math, require practice in order for one to grasp the concept fully. The only way to fully understand recursion is by practicing more problems that implement it. As mentioned before, finding different Fibonacci numbers is a great place to start. Try constructing a recursive algorithm for this.