**CS2 (**Intro toData Structures)

Recursive Functions in C++

A **recursive method** is one which invokes **itself**. The idea of calling one function from another immediately suggests the possibility of a function calling itself. The function-call mechanism in C++ supports this possibility, which is known as recursion. Recursion is a powerful general-purpose programming technique, and is the key to numerous critically important computational applications, ranging from combinatorial search and sorting methods that provide basic support for information processing to the Fast Fourier Transform for signal processing. Example:

void display()

{

cout << "This is a recursive function.\n";

display();

}

This function outputs the string “This is a recursive function.\n”, and then calls itself. Each time it calls itself, the cycle is repeated. Can you see a problem with the function? There’s no way to stop the recursive calls. This function is like an infinite loop because there is no code to stop it from repeating.

The following is a modification of the **“display”** function. It passes an integer argument, that holds the number of times the function is to call itself.

void display(int num)

{

if (num > 0)

{

cout << "This is a recursive function.\n";

display(num − 1);

}

}

For example, let’s say a program calls the function.

#include <iostream>

using namespace std;

void display(int); //Function prototype

int main()

{

display(5);

return 0;

}

void display(int num)

{

if (num > 0)

{

cout << "This is a recursive function.\n";

display(num − 1);

}

}

**Program Output**

This is a recursive function.

This is a recursive function.

This is a recursive function.

This is a recursive function.

This is a recursive function.

// This program demonstrates a recursive function for counting the number of times a character // appears in a string.

#include <iostream>

#include <string>

using namespace std;

int numChars(char, string, int); // Function prototype

int main()

{

string str = "abcddddef";

// Display the number of times the letter 'd' appears in the string.

cout << "The letter d appears "<< numChars('d', str, 0) << " times.\n";

return 0;

}

int numChars(char search, string str, int subscript)

{

if (subscript >= str.length())

{

// Base case: The end of the string is reached.

return 0;

}

else if (str[subscript] = = search)

{

return 1 + numChars(search, str, subscript+1);

}

else

{

return numChars(search, str, subscript+1);

}

}

**Program Output**

The letter d appears 4 times.

Definitions of the Factorial Function:

0! = 1

1! = 1

5! = 1 x 2 x 3 x 4 x 5 = 120

n! = 1 x 2 x 3 … (n-1) x n

**Iterative Implementation**

Here is a typical iterative C++ implementation of the factorial function:

int factorial(int n)

{

int i, f=1;

for(i=1; i <= n; i++)

f \*= i;

return f;

}

**A Recursive Definition**

Recursive Implementation of the Factorial Function.

n!= (n-1)! n

int factorial(int n)

{

if(n = = 1)

return 1;

else

return n \* factorial(n-1);

}

**Recursive Version:**

int factorial(int n)

{

return (n = = 1) ? 1 : (n \* factorial(n-1));

}

This generates a linear recursive process, as the following example shows:

fac (5)

5 \* fac (4)

5 \* (4 \* fac (3))

5 \* (4 \* (3 \* fac (2)))

5 \* (4 \* (3 \* (2 \* fac (1))))

5 \* (4 \* (3 \* (2 \* 1)))

5 \* (4 \* (3 \* 2))

5 \* (4 \* 6)

5 \* 24

120

**Fibonacci Sequence**

In computer science recursion is when a function calls itself to return a value multiple times.

Recursion yields a solution by reducing the problem to smaller and smaller versions of itself.

// Fibonacci Demonstration to show recursion

#include<iostream>

using namespace std;

//function prototype

int fib(int n);

int main ()

{

// Here we print the first 10 Fibonacci numbers

for(int n=0; n<10; n++)

{

cout<<fib(n)<<"\n"; // C++ version to print

}

Cout<<endl;

return 0;

} // end main

// This function calculates the Fibonacci number recursively

int fib(int n)

{

if (n <= 1) return n; // returns 0 or 1

return fib(n-1) + fib(n-2);

} // end fib

Recursion can be hard to grasp sometimes. Just evaluate it on a piece of paper for a small number:

**fib(4)**

-> fib(3) + fib(2)

-> fib(2) + fib(1) + fib(1) + fib(0)

-> fib(1) + fib(0) + fib(1) + fib(1) + fib(0)

-> 1 + 0 + 1 + 1 + 0

-> 3

The Fibonacci Sequence is the series of numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

The next number is found by adding up the two numbers before it.

* The 2 is found by adding the two numbers before it (1+1)
* Similarly, the 3 is found by adding the two numbers before it (1+2),
* And the 5 is (2+3),
* and so on!

Example: the next number in the sequence above would be 21+34 = **55**

It is that simple!

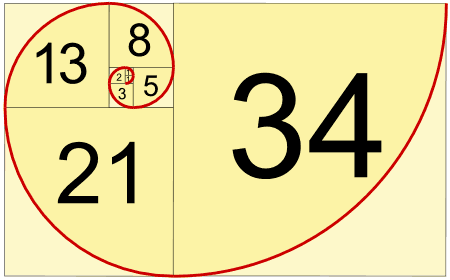
Here is a longer list:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, ...

*Can you figure out the next few numbers?*

**Makes A Spiral**

When you make squares with those widths, you get a nice spiral:



Do you see how the squares fit neatly together?   
For example 5 and 8 make 13, 8 and 13 make 21, and so on.

**The Rule**

The Fibonacci Sequence can be written as a "Rule".

First, the terms are numbered from 0 onwards like this:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ***n =*** | ***0*** | ***1*** | ***2*** | ***3*** | ***4*** | ***5*** | ***6*** | ***7*** | ***8*** | ***9*** | ***10*** | ***11*** | ***12*** | ***13*** | ***14*** | ***...*** |
| **xn =** | **0** | **1** | **1** | **2** | **3** | **5** | **8** | **13** | **21** | **34** | **55** | **89** | **144** | **233** | **377** | **...** |

So term number 6 is called x6 (which equals 8).

And here is a surprise. If you take any two successive *(one after the other)* Fibonacci Numbers, their ratio is very close to the **Golden Ratio** "**φ**" which is approximately **1.618034...**

**Golden Ratio**

|  |  |
| --- | --- |
|  | In fact, the bigger the pair of Fibonacci Numbers, the closer the approximation. Let us try a few: |

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** |  | **B / A** |
| 2 | 3 |  | 1.5 |
| 3 | 5 |  | 1.666666666... |
| 5 | 8 |  | 1.6 |
| 8 | 13 |  | 1.625 |
| ... | ... |  | ... |
| 144 | 233 |  | 1.618055556... |
| 233 | 377 |  | 1.618025751... |
| ... | ... |  | ... |

Note: this also works if you pick two **random** whole numbers to begin the sequence, such as 192 and 16 (you would get the sequence *192, 16, 208, 224, 432, 656, 1088, 1744, 2832, 4576, 7408, 11984, 19392, 31376, ...*):

|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **B** |  | **B / A** |
| **192** | **16** |  | 0.08333333... |
| 16 | 208 |  | 13 |
| 208 | 224 |  | 1.07692308... |
| 224 | 432 |  | 1.92857143... |
| ... | ... |  | ... |
| 7408 | 11984 |  | 1.61771058... |
| 11984 | 19392 |  | 1.61815754... |
| ... | ... |  | ... |

It takes longer to get good values, but it shows you that it is not just the Fibonacci Sequence that can do this!

Using The Golden Ratio to Calculate Fibonacci Numbers

And even more surprising is that we can **calculate any Fibonacci Number** using the Golden Ratio:

http://www.mathsisfun.com/numbers/images/fibonacci-formula-phi.png

The answer always comes out **as a whole number**, exactly equal to the addition of the previous two terms.

Example:

http://www.mathsisfun.com/numbers/images/fibonacci-formula-phi-6.png

When I used a calculator on this (only entering the Golden Ratio to 6 decimal places) I got the answer 8.00000033. A more accurate calculation would be closer to 8.

Try it for yourself!

**A Pattern**

Here is the Fibonacci sequence again:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *n =* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12* | *13* | *14* | *15* | *...* |
| xn = | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 | 34 | 55 | 89 | 144 | 233 | 377 | 610 | ... |

There is an interesting pattern:

* Look at the number **x3 = 2**. Every **3**rd number is a multiple of **2** (2, 8, 34, 144, 610, ...)
* Look at the number **x4 = 3**. Every **4**th number is a multiple of **3** (3, 21, 144, ...)
* Look at the number **x5 = 5**. Every **5**th number is a multiple of **5** (5, 55, 610, ...)

And so on (every **n**th number is a multiple of **xn**).

**Terms Below Zero**

The sequence can be extended backwards!

Like this:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *n =* | *...* | *-6* | *-5* | *-4* | *-3* | *-2* | *-1* | ***0*** | *1* | *2* | *3* | *4* | *5* | *6* | *...* |
| xn = | ... | -8 | 5 | -3 | 2 | -1 | 1 | 0 | 1 | 1 | 2 | 3 | 5 | 8 | ... |

*(Prove to yourself that adding the previous two terms together still works!)*

In fact the sequence below zero has the same numbers as the sequence above zero, except they follow a +-+- ... pattern. It can be written like this:

**x**−n = (−1)n+1 **x**n

Which says that term "-n" is equal to (−1)n+1times term "n", and the value (−1)n+1 neatly makes the correct 1,-1,1,-1,... pattern.

**History**

Fibonacci was not the first to know about the sequence, it was known in India hundreds of years before!

**About Fibonacci The Man**

His real name was Leonardo Pisano Bogollo, and he lived between 1170 and 1250 in Italy.

"Fibonacci" was his nickname, which roughly means "Son of Bonacci".

As well as being famous for the Fibonacci Sequence, he helped spread through Europe the use of [Hindu-Arabic Numerals](http://www.mathsisfun.com/place-value.html) (like our present number system 0,1,2,3,4,5,6,7,8,9) to replace [Roman Numerals](http://www.mathsisfun.com/roman-numerals.html) (I, II, III, IV, V, etc). That has saved us all a lot of trouble! Thank you Leonardo.