 

**Advanced Placement Computer Science**

**Unit 8: Recursion**

Lesson: The Coolness that is recursion

*Last Updated: 1/15/2013*

“Men have become the tools of their tools” Henry David Thoreau

A recursive method is a method that calls itself

Example

public int factorial(int x)

{

if (x == 0

return 1; //base or stopping case

else

return x \* factorial(x-1); //recursive case

}

But how does this work?  
The method has a base case and a recursive case.

The work is done in the recursive case as we “redefine the problem in simpler terms”

To evaluate f(4), we must make the calls

f(4) = 4 \*

f(3) = 3 \*

f(2) = 2 \*

f(1) = 1 \*

f(0) = 1

The system stack then unwinds as each return value is handed back to the previous call

f(1) = 1 \* 1

f(2) = 2 \* 1

f(3)= 3 \* 2

f(4) = 4 \* 6 = 24 final answer

Anything that can be done recursively can be done iteratively

Factorial Example

public int factorial(int x)

{

int sum = 1; //initialize accumulator, since f(0) = 1, I choose 1

for (int i = x; i > 0; i--) {

sum = sum \* i;

}

//ie sum = 1 \*4, then sum = 4 \* 3, then sum = 12 \* 2, then sum = 24 \* 1

}

If any problem that recursion addresses can be handled with loops, then why use recursion?

Answer: recursion is simpler to code for certain problems.

Like trying to find the sum of all the bytes in a directory with all its sub directories

It could be the case that there are a couple of subdirectories or hundreds of subdirectories.

Either way, a recursive coding of this problem is fairly simple.

General Form of a Recursive Function:

if (one or more base cases)

then concrete answer

if (recursive case)

then make a recursive call (or calls)

In general, you need to keep getting closer to your goal when recursing

Where recursion excels:

Recursion excels when attempting to unwind some sort of problem that can easily be broken down into smaller parts.

1. finding all files or sum of files in a sub-directory
2. looking for possible pieces to flip during the game Othello
3. blob-filling a graphic space
4. small examples where the number of recursive calls can be limited

Where recursion stinks:

Recursion is slow and can run out of stack space on the computer very easily if the number of calls is too high

Use iteration when it is simpler

You try:

Write a recursive function to sum the digits of a number

Let’s think about a number, like 13457

How do we get the digits?

Answer: One way is to convert to a String and use the substring or charAt methods.

That is the first method that comes to my mind.

Alternatively, we could chop one digit off the back end of the number each phase.

A convenient way to do this is the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_operator. Yes, that’s it the modulus operator, followed by dividing by 10.

Modulus gives the remainder after dividing by 10 which is the LAST digit.

Dividing by 10 gives the quotient which will essentially drop the LAST digit.

Eventually for a positive number, you’ll get t0 0. I guess if the number is negative, we’ll throw an exception

public int sumDigits(int num) {

//error checking

if (num < 0) throw new IllegalArgumentException(“NO Negatives”);

if (num <10) {

return num; //this must be the sum

}

else {

int lastDigit = num % 10;

return lastDigit + sumDigits(num / 10);

}

}

Fibonacci Recursive Function (more than one base case)

The Fibonacci sequence is the numbers 1, 1, 2, 3, 5, 8, where after the first 2 you simply add the previous 2. Let’s try and find the nth Fibonacci number in the sequence.

public int findNthFib(int n) {

if (n == 1 || n == 2)

return 1;

else return findNthFib(n-1) + findNthFib(n-2);

}

It looks so simple but try tracing through for the 6th or 7th Fibonacci number, it works!!!