Recursion

Lecture 8

Recursion

- Recursive Function
  - A function that calls itself
  - The ability to invoke itself enables a recursive function to be repeated with different parameter values

Recursion

- Recursion can be an alternative solution to iteration
- In many instances, the use of recursion enables a very natural, simple solution to a problem that otherwise would be very difficult to solve

Recursion

- Recursion is an important and powerful tool in problem solving and programming

The Nature of Recursion

- Problems that lend themselves to a recursive solution have the following characteristics:
  - One or more simple cases of the problem have a straightforward solution
  - The other cases can be redefined in terms of problems that are closer to the simple case
  - By applying this redefinition process every time the recursive function is called, eventually the problem is reduced entirely to simple cases, which are relatively easy to solve

The Nature of Recursion

- General form of a recursive algorithm

```
if this is a simple case
    solve it
else
    redefine the problem using recursion
```

The Nature of Recursion

- A problem of size *n* can be split into
  - a sub-problem of size 1
    - Can be solved easily
  - a sub-problem of size *n* – 1
    - Can be split further into
      - a sub-problem of size 1
        - Can be solved easily
      - a sub-problem of size *n* – 2
        - Can be split further into …
- At the end, we solve easily *n* problem of size 1

The Nature of Recursion

- Example: Multiply 6 by 3, assuming we know how do add and we know that x * 1 = x
  - Split the problem:
    1. Multiply 6 by 2
    2. Add 6 to the result
  - Split 1 further:
    1. Multiply 6 by 2
        1. Multiply 6 by 1
        2. Add 6 to the result of problem 1.1
    2. Add 6 to the result

The Nature of Recursion

- Implementation

```
int multiply (int m, int n)
{
    int ans;

    if (n == 1)
            ans = m;
    else
            ans = m + multiply (m, n - 1);

    return (ans);
}
```

The Nature of Recursion

- To solve a problem recursively
  - First, trust the function to solve a simpler version of the problem
  - Then, build the solution to the whole problem on the result from the simpler version

Tracing a Recursive Function

- Hand tracing an algorithm's execution provides valuable insight into how that algorithm works
- To understand recursion and debug a function

Recursive Math Functions

- Many mathematical functions are defined recursively
  - Example: Factorial of n (n!) is
    - 0! = 1
    - n! = n x (n - 1)!, for n > 0

Recursive Math Functions

- Factorial - implementation is straightforward

```
int factorial (int n)
{
    int ans;

    if (n == 0)
      ans = 1;
    else
      ans = n * factorial (n - 1);

    return (ans);
}
```

Recursive Math Functions

- The Fibonacci numbers
  - Sequence of numbers that have many uses
- The Fibonacci sequence is
  - 0, 1, 1, 2, 3, 5, 8, …
  - The sequence is produced as follows
    - $\text{Fibonacci}_0 = 0$
    - $\text{Fibonacci}_1 = 1$
    - $\text{Fibonacci}_n = \text{Fibonacci}_{n-1} + \text{Fibonacci}_{n-2}$, for $n > 1$

Recursive Math Functions

- Fibonacci - implementation is straightforward

```
int fibonacci (int n)
{
    int ans;

    if (n == 0  ||  n ==1)
      ans = n;
    else
      ans = fibonacci (n - 1) + fibonacci (n - 2);

    return (ans);
}
```

Recursion

- Recursion is also useful for processing varying-length lists
  - Strings
  - Linked-lists
  - Etc.

Another Example

- Example: Function to count the number of times a particular character ch appears in a string str
  - Split the problem:
    1. Check the rest of the string
    2. Update the counter if the first character is x

  - Split 1 further:
    1. Check the rest of the string
        1. Check the rest of the string
        2. Update the counter if the second character is x
    2. Update the counter if the first character is x

Another Example

- Implementation

```
int
count  (char ch, char *str)
{
    if (*str == '\0')
            return  0;

    if (ch == *str)
            return ( 1 + count (ch, str + 1));
    else
            return (count (ch, str + 1));
}
```