Suffix rules in Makefile

```
%.o: %.cpp %.h
    g++ -c –o $@ $<
```

- a rule that applies to all files ending in the .o suffix. The rule says that the .o file depends upon the .cpp version of the file and the .h files.
- $@ says to put the output of the compilation in the file named on the left side of the :, the $< is the first item in the dependencies list.

```
OBJ = main.o yourClass.o
a.out: $(OBJ)
    g++ -o $@ $^
```

- the special macros $@ and $^, which are the left and right sides of the :, respectively.

Homework #1

Fibonacci sequence using recursive function calls

- Write a program to compute the Fibonacci sequence 1, 1, 2, 3, 5, 8, 13, 21, ... using the recursive function $f(1) = f(2) = 1$, $f(n) = f(n-1) + f(n-2)$.
- You will need one *main()* method and one recursive method called *f()* to calculate the Fibonacci function. After the *main()* method introduces itself, it will perform the following steps in a loop:

HW #1 (2)

1. Prompt the user for a parameter *n* from 1 to 99.
2. If *n* is zero or negative, then the program ends.
3. If *n* is positive, then:
   - Call the recursive Fibonacci function.
   - Print the parameter *n* and the function result *f(n)*.
   - Go back to step #1.

HW #1 (3)

- The Fibonacci function will have one *int* parameter for *n*, and will return an *int* result for the function value *f(n)*.

- This function must be recursive. To compute *f(n)* when *n*>2, the function must call itself twice: once for the value of *f(n-1)* and once for the value of *f(n-2)*.

HW #1 (4)

- Your task is to:
  1. Implement the program using C++.
  2. Download Java J2SE from
     http://java.sun.com/j2se/
     and rewrite the program in Java.

- You are required to submit a single "**makefile**" as well.

HW #1 (5)

- Learn how to set up your **debugging** environment and get experience in, for example, setting breakpoints, stepping through the execution, and evaluating a variable or an expression.

HW #1 (6)

Fibonacci sequence using loops and an array

- Sometimes the Fibonacci program is very slow when calculating *f(n)* for large values of *n*. The reason that it is slow is because each recursive function calls two more recursive functions, which both call two more functions, resulting in approximately $O(2^n)$ function calls before the result is computed.
- There is another way to compute the Fibonacci sequence. You can compute the Fibonacci sequence by saving previous results in an array. This is much faster because its speed is $O(n)$ instead of $O(2^n)$.

HW #1 (7)

- Your *main()* method now can do the calculations inside a for loop.
- The array will be an array of long integers. The array should have 100 elements. The index of the first element in a Java array is zero (0). The Fibonacci function is defined starting at *f(1)*. To avoid confusion about function notation and index variables, ignore the first element. Then *f(n)* will be in array element *n*:

HW #1 (8)

- results[0] = 0; // dummy value (not used)
  results[1] = 1; // initial value for f(1)
  results[2] = 1; // initial value for f(2)

- Use a for loop to calculate results[3] to results[*n*].

```
// Here we are asked to read two numbers in command line and
// display the sum.
// One sample run test: a.out 3 4
// This program shows you the basics of building a C++ program.
//
#include <iostream>
// Using statements which allow you to use the streams/keywords
// in the std:: namespace without needing the std:: prefix
using std::cout;
using std::endl;

// Solution implemented entirely in main()
int main (int argc, char *argv[]) {
  if (argc != 3) {
   cout << "You have forgot to specify two numbers." << endl; exit(1); }
  cout << "The sum is : " <<  atoi (argv[1]) + atoi (argv[2]);
  return 0;
}
```

- The following is designed to familiarize you with the mechanics of creating, editing, compiling, and running a text-mode Java application.
- You do not have to hand it in, but you should write and run it.
- The source code in the following pages simply prompts for and accepts two numbers from the user, adds them, and displays the result.
- The file name, *Add.java* is case-sensitive and must match the class name in the program.

```
// Program to add two numbers... note that input is accepted as a
// String and then an attempt is made to convert it to a integer for
// calculations. Non-numeric input is detected by the Exception
// mechanism and a default value is assigned to the value.
//
// One sample run test: java Add 3 + 4

import java.io.*;

public class Add {
  public static void main(String args[]) {
    if(args.length==0) { System.out.println("No arguments are passed"); }
    else {
      int a =0; b = 0;
```

```
      // try to convert amt String to integer for calculation
      try { a = Integer.parseInt(args[0]); }
      catch (NumberFormatException e) {
        System.out.println("Bad numeric input; 1st num set to 100");
        a = 100; }
      String p=args[1];
      try { b = Integer.parseInt(args[2]); }
      catch (NumberFormatException e) {
        System.out.println("Bad numeric input; 2nd num set to 100");
        b = 100; }
      switch (p) { // OR switch (p.charAt(0))
        case “+”:  // OR case ‘+’:
          System.out.println("Addition of “ + a + " and “ + b + " : “ + (a+b));
          break;
        default: System.out.println("Please Enter '+‘ & '-' operator only.");
      } // end switch
    } // end else
  } // end main
} // end of class Add
```