Laboratory 2:
Attacking Your First (Simple) Problem

Overview: Up until this point you've spent some time entering and running existing C++ programs to learn the edit-compile-link-load-run process and to discover some of the features of the language. Today, you'll generate some significant segments of code on your own for the first time. The focus of the activity is the use of *expressions* (which perform almost all of the important computational tasks in C++ programs)

Objectives: Before you leave lab today, you should: know the types of the fundamental objects (`int`, `float`, etc.) and how to use them to *declare* or *create* objects; know the basic arithmetic *operations* on these objects; understand what an expression is; understand how expressions *compute* values and how rules of *precedence* affect those computations; understand how expressions *assign* values, and how they can be used to read values into a fundamental variable object and write values to the screen. You should know how to *prompt* for input values, and you should know the main purpose of the file **iostream.h**.

1. Setting up for the lab

Please perform the following activities as you wait for class to formally begin:

a. Boot up the machine and log on to the CS Department server, as described in Lab 1.

b. Put your lab diskette in the *A:* drive and make a subdirectory *LAB02* in the *LABS* directory.

c. Place all of the files necessary for Lab 2 into you new subdirectory. Remember that you can access these files from the web site for the class at:
http://s9000.furman.edu/chealy/cs11/lab02

2. Solving your "a,b,c's"

Basic expressions and precedence rules

As discussed in class and in the textbook, the evaluation of expressions in C++ is governed by specific *precedence rules*. Use your notes or textbook if you like to help you with the following steps.

a. Examine the program on the next page and write down alongside the `cout` statements what you expect the output to be. To do this, you need to keep track of what values are contained in the variables `a`, `b`, `c` and `d`.

```
int main ()
{
  // Variable declarations and initializations:
  int a = 3;
  int b = 20;
  int c = 6;
  int d = 4;

  // Now calculate the results.
  d *= a - 1;
  c += 2 * a;
  d = (d - b) / c;
  c = c * b % a;
  b /= 2 + a++;

  // Finally, display the results.
  cout << "a: " << a << endl;
  cout << "b: " << b << endl;
  cout << "c: " << c << endl;
  cout << "d: " << d << endl;

  // Exit indicating a lack of errors.
  return 0;
}
```

b. Start Borland C++ 4.5 as described in Lab 1. Open the file called *simpmath.cpp*. Click on the lightning bolt to run the program, and observe the output.

Did you get the same answers from the computer program as you did from your manual calculations? If there are differences, redo your manual calculations to see where the mistake was. If you can't determine the discrepancy, call me over.

Interactive input/output

It is often (but not always) better to let the user of a program input the values of the objects rather than hard-code them in the program. In this case, we want to *prompt* the user to input values for the variable objects *a*, and then *read* the value for *a*.

c. Add C++ statements to prompt and read for the value of *a*. Your code might look something like:

```
cout << "Enter a value for object a: ";
cin >> a;
```

Note: Don't forget to remove the code that currently gives *a* its initial value.

d. It is important to prompt the user for each variable separately. So add additional lines of code to the program that will prompt the user to enter values for the variable objects *b*, *c* and *d*. Save the program.

e. Compile and run your improved program. Be sure to save the program before each compilation and run. If you get an error message that you can't figure out, discuss it with your partner. If you can't figure it out together, call me over.

f. Manually check your computations to be sure you understand how the expressions are executed. √

The necessity of declarations

g. Remove the line

```
    int a;
```

from your program. Try to compile and run it. Try to make sense of what happened. Ask me if you can't. Put the line back in.

h. Change all your declarations to be of data type **float**.

i. Compile the program again. You should get a syntax error for the statement:

```
    c = c * b % a;
```

Why did this error occur?

j. Comment out the offending line

```
    // c = c * b % a;
```

Compile and execute the program. Did you notice any change in the output?

k. Manually check your computations to be sure you understand how the expressions are executed. √

3. Writing your own calculations

Now let's consider a slightly more challenging problem—writing the general solution to an algebraic problem. Suppose you have a problem that you wish to solve. What steps must you take to write a program which solves that problem?

— Decide what the inputs and outputs are.
— Declare variable objects for inputs and outputs.
— Compute the answer (in sub-parts, if it is complicated).
— Output the answer.

This seems like a relatively straightforward process to follow, so let's give it a try. For the next part of the lab, we are going to write a program that evaluates the following expressions:

- $2a^2 + 4a - 29$
- $\dfrac{4c + ac}{3b}$

- $\frac{4c+ac}{3b}+\left(\frac{cb/a}{4/d}\right)^2$

a. First you should decide what type you want for the inputs (a, b, c, d). Knowing the types of the inputs will often imply the types the outputs will be. How many inputs and outputs will there be? Is the result of the computation a different type from the objects in the computation? Will you need any temporary space (i.e. another object) to store partial computations? Think about this a minute and circle the segments of the equations above that may be placed in a temporary object.

b. Write down the declarations of all the variables you will need for input, output and temporary computations in the area provided below. Assume for this part that all the variables are of type int. Also assume that all results are to be of type int. The first declaration is done for you.

```
int a;
```

c. Next, write down the solution for each expression as it will need to appear in C++ code in order to be correct. Pay particular attention to the order of operations in each problem. Be sure to use parentheses as needed to enforce correct computation of each problem. Write the C++ solution to expression #1 here:

Write the C++ solution to expression #2 here:

Write the C++ solution to expression #3 here (save yourself some work and use the result from expression #2):

d. Open the new file *compute.cpp* and add the code you wrote above to this file so it will carry out your calculations. (When type in your data in response to your executing program, use the values a=3, b=20, c=6, and d=4.) Save your work often. Did it work right away? Manually check your answers to convince yourself that the expressions are working properly.

e. Print out a *hard copy* of your program. The print command is under the File menu option. You may keep this printout to reference when working on the remainder of the lab. √

4. Revise the Compute.cpp Program To Use Float

a. Edit the *compute.cpp* program so that all the variables are of type float.

b. Enter the following statements after the data have been input. (Use the same values for a, b, c, and d as in part 3.)

```
cout << setprecision(2);
cout << setiosflags( ios::fixed | ios::showpoint);
```

c. Print the input data as float values.

d. Compute and the values of the equations. Do you get the same values as in part 3?

f. Manually check your computations to be sure you understand how the expressions are executed. Save the program with the name *compute2.cpp*. √

g. Log out of the computer and turn in the hard copies of *compute.cpp* and *compute2.cpp* as well as your check-off sheet. Please be sure your name is on all of these documents.