LABORATORY 2

Attacking your first problem

OBJECTIVE

This laboratory provides you with your first opportunity to decompose a problem into manageable pieces and solve it. It will also introduce the process of compiling a program that uses a library called an Application Programmer Interface or API. You will probably be doing a few things that are new to you. If you have any questions or problems, just ask your laboratory instructor for help.

KEY CONCEPTS

- Expression evaluation
- Simple input and output
- Hand checking code
- Expressing mathematical equations in `C++`
- Project files

GETTING STARTED

- Using the procedures in the introductory laboratory handout, create the working directory `\cpplab` on the appropriate disk drive and obtain a copy of self-extracting archive `lab02.exe`. The copy should be placed in the `cpplab` directory. Execute the copy to extract the files necessary for this laboratory.
- Many of the activities that are performed in the laboratory can be done in groups but you should work the exercises yourself.

2.1 SOLVING YOUR A, B, C'S

- Examine the program below. Next to the insertion statements write what you expect the output to be. If you have a partner, discuss your answers for objects `a`, `b`, `c`, and `d` with your partner. Come to an agreement on what the values should be. If you cannot agree, talk to a laboratory instructor.

```
int main() {
    // Object definitions and initializations
    int a = 3;
    int b = 12;
    int c = 6;
    int d = 1;

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

    // 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;
}
```

- Start the `C++` compiler as described in the opening laboratory. Open the file called `simpmath.cpp`. It should contain the preceding program along with several additional comments and output statements.
- Build and run the program to observe the output.
- Did you get the same answers for your manual calculations as you did from the computer program? If there are differences, try to figure out why. If you cannot determine the reason for the differences, ask a laboratory instructor for help.
- Allowing a user to supply input values is a better technique than hard-coding the values because it makes the program more general. Now you will modify `simpmath.cpp` to extract user input from the standard input stream. First, delete the hard-coded initialization of `a`, `b`, `c`, and `d`.
- Add a prompt that tells the users of the program what you want them to do. In this case, you want to prompt the user to supply a value for object `a`. Then add a statement to do the extraction. Your code might look like the following:

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

- It is important to prompt the user for each object separately. So add lines of code to the program that prompt the user to enter values for objects `b`, `c`, and `d`. Be sure to store the user's keyboard input in the appropriate object. Save the program.
- 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 cannot figure out, discuss it with your partner. If you cannot figure it out together, ask a laboratory instructor for help.
- After developing a program or modifying an existing one, a key question is, Does the program run correctly? One way is to *hand check* the program. Hand checking a program involves computing the results by hand for some input and making sure the results agree with what the computer outputs for the same input. You can hand check your modified program by using as inputs the values that were used to initialize the integer objects `a`, `b`, `c`, and `d` in the original program. Run your program and enter the values that were used to initialize the objects `a`, `b`, `c`, and `d` in the original program. Did you get the same results?
- ✎ Once the program is working, demonstrate it to a laboratory instructor. ✓
- Close the file.

2.2 OPERATION ORDER IS IMPORTANT

Now let's consider a slightly more challenging problem—writing the general solution to an algebraic problem. Suppose you have a simple problem that you wish to solve. You should take the following steps to write a program to solve the problem:

— Determine the inputs and outputs.

— Define objects for inputs and outputs.

— Compute the answer (in parts, if it is complicated).

— Output the answer.

This process seems relatively straightforward, so let's give it a try. For the next part of the lab, you are going to write a program that solves the following equations:

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

- $\dfrac{10b + 4a}{3c} + \left(\dfrac{\frac{cb}{a}}{4/d}\right)$
- $\dfrac{10b + 4a}{3c} + \left(\dfrac{\frac{cb}{a}}{4/d}\right) \times \dfrac{10b + 4a}{3c} + \left(\dfrac{\frac{cb}{a}}{4/d}\right)$

You and your partner should work independently on the next several steps.

- Determine the types of the objects. For these equations, you can use the type `float` because three of the equations contain a division operation. If you used integer objects, the result of the division operation would be truncated, which would produce erroneous results.
- Determine how many inputs and outputs there will be. Will you need any temporary space to store partial computations? Or do you want to try computing the larger problem with one huge equation?
- List the definitions of all the objects you will need for input, output, and temporary computations in the area provided below. We do the first declaration for you.

```
float Result1;
```

- Write the solution for each equation 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 the correct computation of each problem. We do the first one for you. Does your solution agree? If not, review your answers for it and the other equations.

```
Result1 = (2 * a * a) + (4 * a) - 29;
```

- Write the `C++` solution to equation #2 here:

- Write the `C++` solution to equation #3 here:

- Write the `C++` solution to equation #4 here:

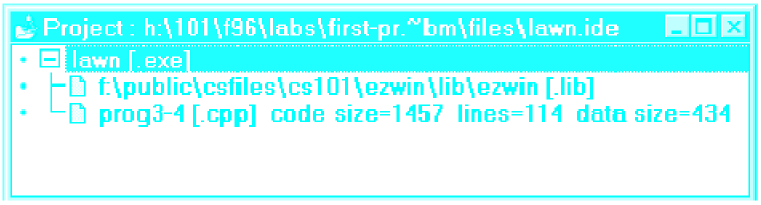
- Compare your answers with your partner's. Do they agree? Are they equivalent but different? Once you have determined that your answers are correct, open the file `compute.cpp`.
- Use this file as a basis for computing the above equations.

- Complete the program in that file. Save your work often. In particular, always save it before you do a compile and run. Once it is working, demonstrate it to a laboratory instructor. ✓

2.3 EZWINDOWS

- Close all the open files that you are using with the `C++` compiler. Pull down the Project menu and select Open project. Open the project `lawn.ide`. A *project file* is a file that contains information that the compiler uses to build an executable program or application.

  Most applications of any size contain source code modules (i.e., files containing `C++` source code). In addition, they use libraries of routines that have been written by professional programmers. The project file contains information that, among other things, tells the compiler where to find the necessary files to build the program and where to write the executable.
- Depending upon your access permissions for the `C++` compiler, when you open a project, you can get errors regarding files the compiler cannot create or write on the hard disk drive. Ignore these. A subwindow that can lists the files in the project should appear. If the files needed to create `lawn.exe` are not shown, click the plus sign (+) beside `lawn.exe`. You should see something similar to the following window.

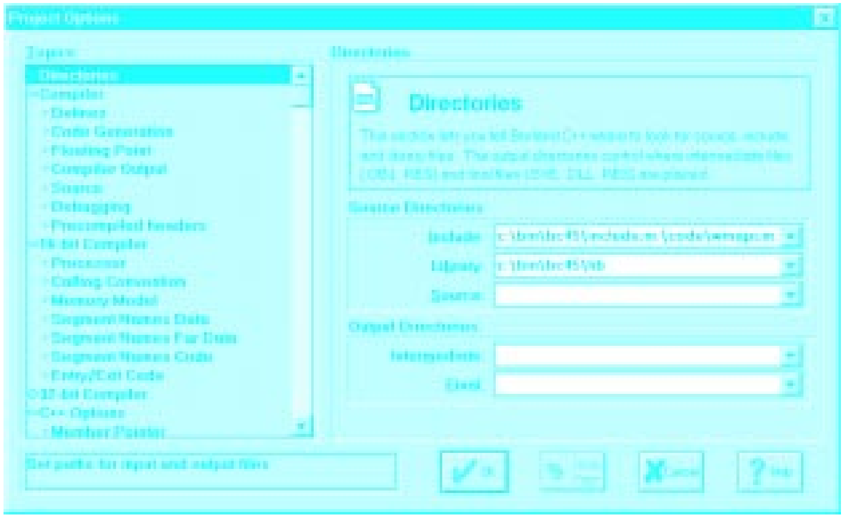


```
Project : h:\101\f96\labs\first-pr.~bm\files\lawn.ide
lawn [.exe]
  f:\public\csfiles\cs101\ezwin\lib\ezwin [.lib]
  prog3-4 [.cpp]  code size=1457  lines=114  data size=434
```

The necessary modules for this project are the EzWindows library and the source file `prog3-5.cpp`. The EzWindows library contains the code that supports creating and manipulating graphical objects such as rectangles, ellipses, and triangles. The library is described in detail in the Appendix.

- In addition to the modules that are part of the application, the project file specifies how to build the application, where to find the include files, and where to write the executable. You can access this information through the Project Options window. To view some of this information, pull down the

Options menu and select Project. A window like the following should appear:



From this menu, the programmer can tell the compiler where to look for certain files. The Include path specifies where the compiler should look for the include files. The Library path specifies where the compiler should look for library files. These paths should already be set to the appropriate directories.

Of interest to us are the Output directories. These specify where the compiler should write files. For our laboratories activities, these should contain the path where you placed the laboratory files (for example, `c:\cpplab`). After making sure these fields contain the correct path, close the Project Options window by clicking on the OK button.

- To build and run an application that has a project file is simple. First click on the lightning bolt button in the control bar area of the window. When the program runs, respond to the prompts and observe the output the program creates. Discuss the output with your laboratory instructor. ✓
- Close the windows created by the application and the message window. Bring the Project window to the foreground. Point your mouse at `prog3-5.cpp` in the Project window and double-click. A new window lists the contents of the file `prog3-5.cpp`. Examine the file and get a feel for what the program is doing. One of the first things to notice is that there is no function named `main()`. Because this program will be doing graphics using the EzWindows API, the program conceptually begins execution in a function called `ApiMain()`.
- Scroll in the source window and find the following lines:

```
// Open a window and display the lawn
SimpleWindow Display("Lawn and House Plot",
 DisplayLength, DisplayHeight);
```

```
Display.Open();
```

These lines create the window that contains the diagram of the lawn and house and make it appear on the screen. Find the lines in the program where the values of `DisplayLength` and `DisplayHeight` are set. Change these values to `10` and `12` respectively. Run the program again and observe the difference in the size of the window created. Now change the code that creates the window so that the string is

```
"Weedwacker's Lawn and Garden Service"
```

Run the program again and observe what happens.

- Examine the following lines from the program:

```
RectangleShape Lawn(Display, DisplayWidth / 2.0,
 DisplayHeight / 2.0, Green,
 LawnLength * ScaleFactor,
 LawnWidth * ScaleFactor);
Lawn.Draw();
// Display the house
RectangleShape House(Display, DisplayWidth / 2.0,
 DisplayHeight / 2.0, Yellow,
 HouseLength * ScaleFactor,
 HouseWidth * ScaleFactor);
House.Draw();
```

These lines define and draw two objects `Lawn` and `House`. They represent the lawn and the house. The type of these objects is `RectangleShape`. A `RectangleShape` is an object that is defined in the EzWindows API. The code

```
House.Draw();
```

sends a message to the object `House` telling it to draw itself on the screen. Comment out this line and run the program. Show the display to your laboratory instructor. ✓

- Exit from the `C++` compiler.
- You do not need to save the modified files.

2.4 FINISHING UP

- Copy any files you wish to keep to your drive.
- Delete the directory `\cpplab`.
- Hand in your check-off sheet.