University of Massachusetts Dartmouth
Department of Electrical and Computer Engineering

ECE 160 Name: `payroll.cpp`
Lab 6 Payroll (Debug) Due: see `http://ece160.org`

However thorough your design and development of your programs are, it is rare that your programs will work exactly as specified the first time you run them. Frequently an engineer's worth is measured by how quickly they can debug a program (or circuit/system). MS Visual Studio provides an extremely useful debugging environment in which to trouble shoot programs. This lab provides an overview of how to use the debugger, by taking you through a series of steps which use the debugger.

1. Create a project (call it `payroll`) and enter the program on the next page. Try to keep the line numbers consistent between the handout and your program. Note, if line numbers do not show up when you are entering the program, click TOOLS/Options. Expand "Text Editor". NOTE – EXPAND means click on the ► to the left of "Text Editor" or whatever. Expand C/C++; click General; click the "Line numbers" check box; Click OK.

2. Save the project by doing one of the following:
   A. To save via menu interface: click File/Save All
   B. To save via toolbar, click 
   C. To save via a keyboard shortcut press Ctrl+Shift+S

3. Try compiling and running the program (without any debugging). Once again, there are several ways to do this:
   A. To run via the menu interface, click Debug/Start without debugging
   B. To run via a keyboard shortcut, press Ctrl+F5
   C. There is NO toolbar shortcut (The  key does NOT do Start without debugging)

   When a program is run (without any debugging), the program runs to completion, and then displays a "Press any key to continue..." message. This allows a user to look at the output from the program.

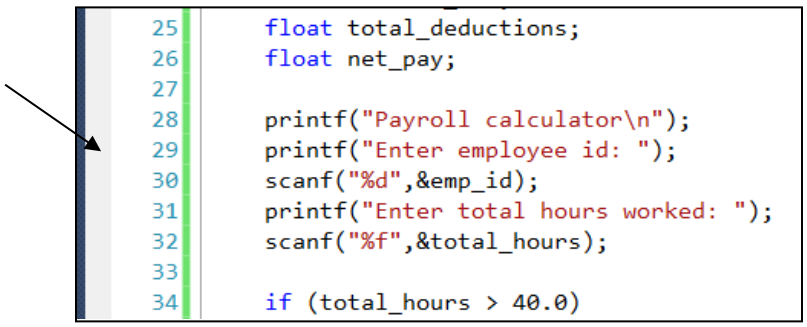
4. If you have entered the program correctly, five warning errors of the form `warning C4244: '=' : conversion from 'double' to 'float', possible loss of data,` will be shown, but no other errors. You do not need to worry about fixing these errors. If there are other errors, you need to compare the program on the next page with what you typed, and make corrections as needed. Generally, by clicking on an error message, the Integrated Development Environment (IDE) will position the cursor on (or near) the line where the error is. For example, if you leave a semi-colon off of a line, the IDE will report an error on the line following the line where the semi-colon should be.

   If you need to expand the size of the command line window (to avoid wrapped text), right-click  in the title bar of the window, click properties, click the layout tab, change window size width to 132. Click OK. Click "Save properties for future windows with same title. Click OK.

As you enter this program, try to maintain a correspondence between the line numbers on this page, and the line number within your program file (look to the left of your program as you are entering it. You should also add comments to the code. Your grade for this project is based solely on how well you comment the code.

```
 1 // ECE 160 - Payroll example for debugging
 2 // I, <name>, certify this is my own work and I have not collaborated with anyone
 3
 4 #define _CRT_SECURE_NO_WARNINGS
 5 #include <stdio.h>
 6
 7 #define STATE_RATE 0.12
 8 #define FEDERAL_RATE 0.22
 9 #define MEDICAL_AMOUNT 212.00
10 #define OVER_TIME_MULT 1.5
11
12 void main()
13 {
14     int emp_id;
15     float total_hours;
16     float reg_rate;
17     float over_rate;
18     float reg_hours;
19     float over_hours;
20     float reg_pay;
21     float over_pay;
22     float gross_pay;
23     float state_tax;
24     float federal_tax;
25     float total_deductions;
26     float net_pay;
27
28     printf("Payroll calculator\n");
29     printf("Enter employee id: ");
30     scanf("%d",&emp_id);
31     printf("Enter total hours worked: ");
32     scanf("%f",&total_hours);
33
34     if (total_hours > 40.0)
35     {
36         reg_hours = 40.0;
37         over_hours = total_hours - 40.0;
38     }
39     else
40     {
41         reg_hours = total_hours;
42         over_hours = 0.0;
43     }
44
45     printf("Enter pay rate: ");
46     scanf("%f",&reg_rate);
47     over_rate = reg_rate * OVER_TIME_MULT;
48
49     reg_pay = reg_hours * reg_rate;
50     over_pay = over_hours * over_rate;
51
52     gross_pay = reg_pay + over_pay;
53
54     state_tax = gross_pay * STATE_RATE;
55     federal_tax = gross_pay * FEDERAL_RATE;
56
57     total_deductions = state_tax + federal_tax + MEDICAL_AMOUNT;
58
59     net_pay = gross_pay - total_deductions;
60
61     printf(" Emp Total   Regular Overtim Regular Overtim Regular Overtim Gross   Federal State   Medical Total   Net\n");
62     printf("  ID Hours   Hours   Hours   Rate    Rate    Pay     Pay     Pay     Tax     Tax     Insuran Deduct  Pay\n");
63     printf("%04d %5.1f   %5.1f   %5.1f %7.2f %7.2f %8.2f%8.2f%8.2f %7.2f %7.2f %7.2f%8.2f%8.2f\n",
64         emp_id,total_hours,reg_hours,over_hours,reg_rate,over_rate,reg_pay,over_pay,
65         gross_pay,federal_tax,state_tax,MEDICAL_AMOUNT,total_deductions,net_pay);
66 }
```

5. Now it is time to get to some serious debugging. Note there is a gray column just to the left of the line numbers:



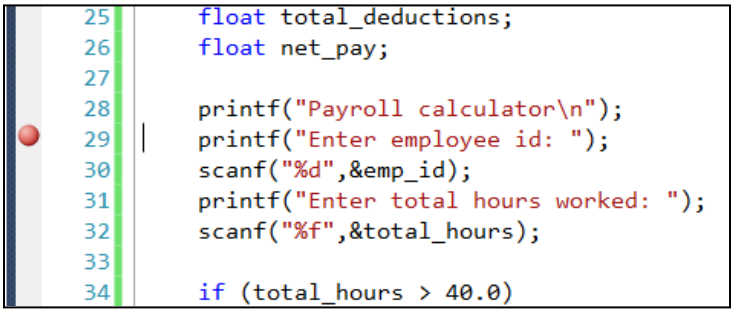
```
25      float total_deductions;
26      float net_pay;
27
28      printf("Payroll calculator\n");
29      printf("Enter employee id: ");
30      scanf("%d",&emp_id);
31      printf("Enter total hours worked: ");
32      scanf("%f",&total_hours);
33
34      if (total_hours > 40.0)
```

Clicking in this gray column sets a "breakpoint" on the given line. Click in the gray area to the left of line 29 which should be:

```
printf("Enter employee id: ");
```

An alternative way to set a breakpoint is to right click on the statement where you wish to set a breakpoint at, and select "Breakpoint>/Insert breakpoint"

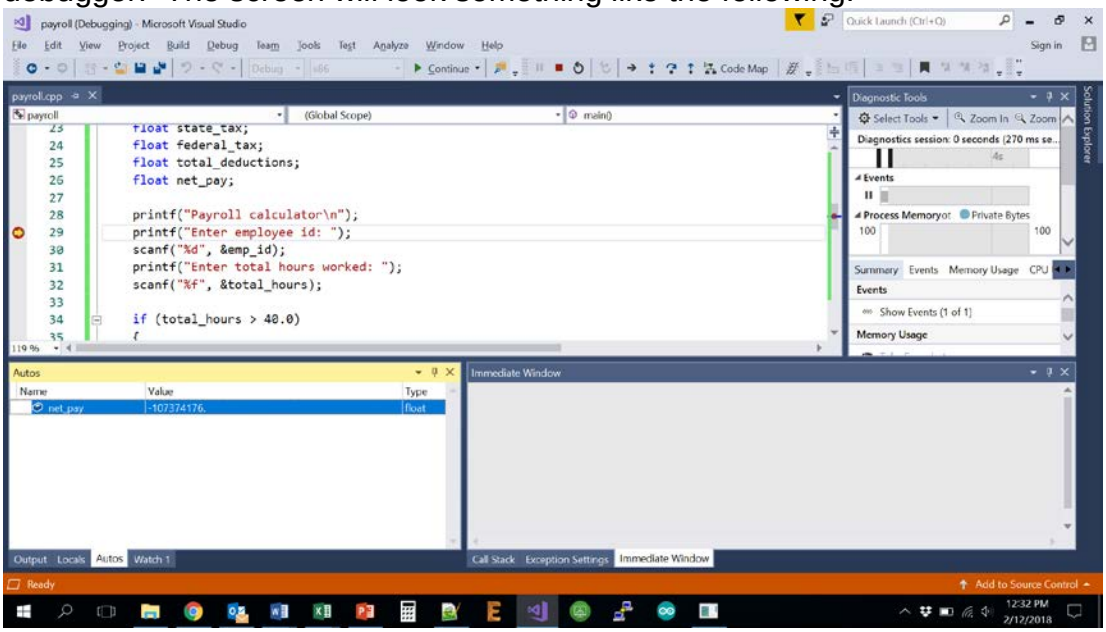
Once you click in the gray (or right click, etc), there should be a red dot in the gray area, and on the same line as the statement where you want to set the breakpoint. For example



```
25      float total_deductions;
26      float net_pay;
27
28      printf("Payroll calculator\n");
29      printf("Enter employee id: ");
30      scanf("%d",&emp_id);
31      printf("Enter total hours worked: ");
32      scanf("%f",&total_hours);
33
34      if (total_hours > 40.0)
```

6. Next, start the program WITH debugging. There are three ways to do this
   A. To run via the menu interface, click Debug/Start debugging
   B. To run via a keyboard shortcut, press F5
   C. To run via the toolbar click 

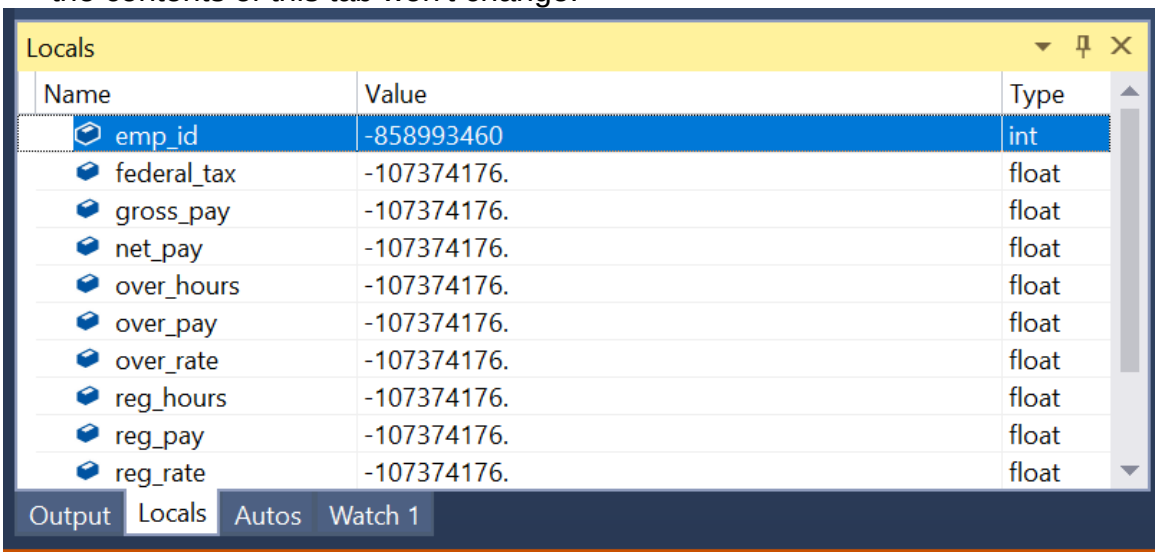
7. The program will execute until the line with the breakpoint set is ABOUT to be executed. The program will then stop running and control is transferred to the debugger. The screen will look something like the following:



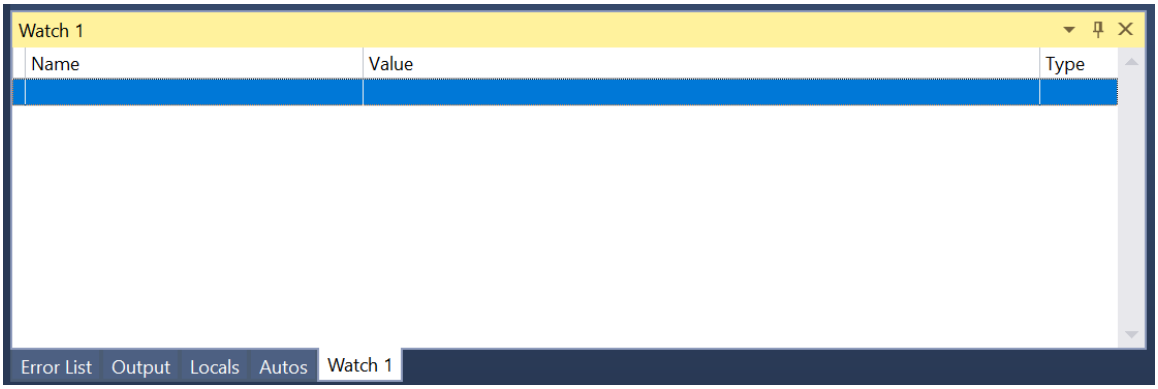
There are several areas of interest on the screen (window panes may be in different locations). Next to line 29, you can see the red dot (indicating a break point), and also a yellow arrow, which indicates the line ABOUT TO BE executed. A lower pane has several tabs along the bottom that may be selected to view various information. If you do not see tabs "Autos", "Locals", and "Watch 1" as shown above, see #9.

A. Click on the "Autos" tab to see a box that contains a list of the variables that the debugger automatically chooses, based on where the breakpoint is set. In this case each variable has the value -1.0737418e+008. In hex this number corresponds to CCCCCCCC, which is the default value the system assigns to all variables when in debug mode.

B. The "Locals" box (click on the "Locals" tab) contains all of the variables in the current function. As we only have one function (main) in this program, the contents of this tab won't change.



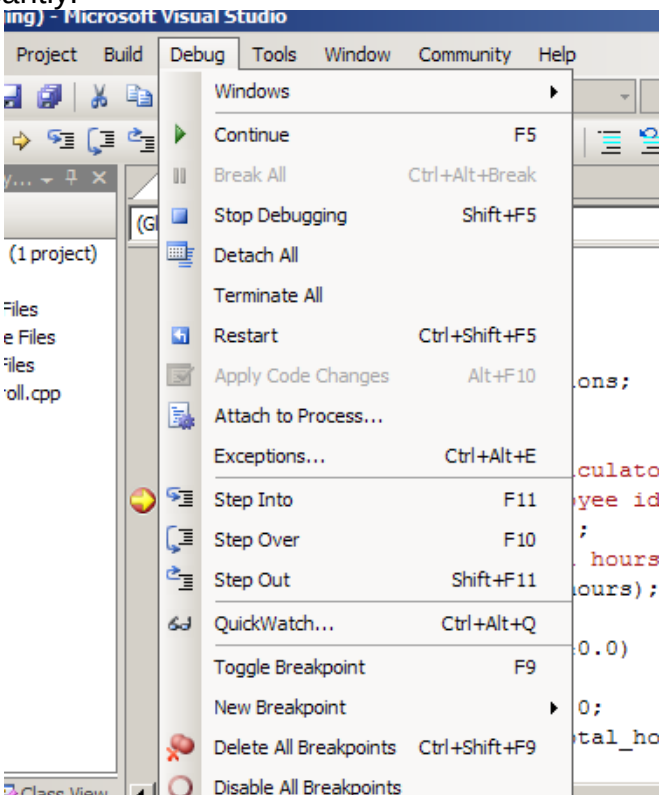
C. The Watch 1 tab allows you to specify variable of interest. You can add variables by clicking on the left column, and typing a variable name. Initially, there are no variables in the watch list.



D. Other tabs may be present. One possibility is the Threads tab which shows what processes are running as a result of this program. Another possibility is the Modules tab which shows which files contain code that is being used by this process. Unless you are writing multi-threaded applications (the same program running multiple times on the same machine and somehow interacting), these tabs are not needed. These two tabs may not be shown by default...it is not a problem. Possibly other tabs may be shown which may be ignored.

8. In general, you will want to step through the program one line at a time. There are a number of ways to do this.

   A. By the menus. Once a breakpoint is reached, the Debug menu changes significantly:



   There are several choices – we will look at each of them:

   i. Continue – continue executing the program. The program will continue to execute until it either completes or another breakpoint is found.
   ii. Stop Debugging – Stop executing the program
   iii. Restart – restart the program
   iv. Step Into – execute the current statement. If the statement is a call to a function (either a system function like printf/scanf, or a user defined function) the debugger will stop at the first line in the function.
   v. Step Over – execute the current statement. If the statement is a call to a function (either a system function like printf/scanf, or a user defined function), the entire function will be executed and the debugger will pause prior to executing the next statement.
   vi. Step Out – Run the program until you exit from the current function. This is generally used when you wanted to "Step Over", but instead picked "Step Into"

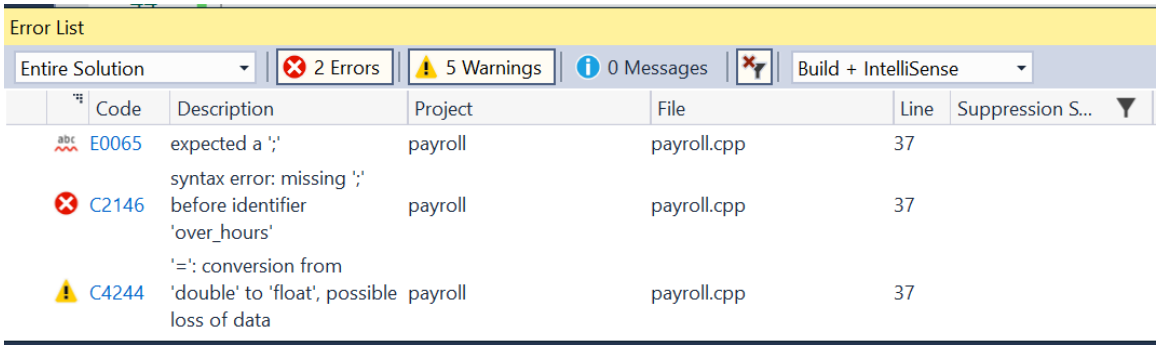
   B. There are also both keyboard shortcuts and toolbar buttons that may be used instead of the menu options. They are all documented on the dropdown menu. For example, the "Step Over" command may be picked from the menu, or by selecting  from the toolbar, or by pressing F10 the keyboard short.

9. MS Visual Studio is incredibly customizable. Tabs and windows can be moved around every-which-way. This can often be the source of confusion for someone new to visual studio.

   If a tab does not appear... For example, say the "Locals" tab does not appear. Click Debug / Windows / Locals. A "Locals" window will appear on screen. Click and drag the title bar of the "Locals" window down just to the right of the "Watch 1" tab. The "Locals" window will "dock". You may move the "Locals" tab to the left if you wish.

10. You may set as many breakpoints as you wish. Whenever a breakpoint is encountered, control is transferred to the debugger, and you can examine/change variables. You may continue execution (until the next breakpoint) by pressing F5, clicking  on the toolbar, or picking Debug/Continue from the menu.

11. Another window you may wish to use is the "Error List" window. Normally, errors appear in the "Output" window. The "Error List" window shows both fatal and warning errors in a tabular format. By default, the "Error List" window is not enabled. To enable, click View / Error List. The window should look something like the one below. A sample error list (if you leave the ; off the end of line 36 would look as follows:



   You may selectively display fatal errors, warnings, both, or neither. You may toggle what is displayed by clicking 

or 

12. This lab has been a brief introduction to the debugger. You are encouraged to explore the use of the debugger…set breakpoints, examine various variable values, try changing the value of variables, etc. Becoming familiar with this debugger will save you many hours as the semester progresses. Learning this debugger will also be very useful when you take ECE 161 and/or ECE 264, as the same environment is used (at least at present).

13. Deliverables – submit the file payroll.cpp as you would normally submit a file. Your grade will be completely based on the documentation. You should comment the code fully.