Lab 2 - Variables and If

Lab Topics

1. Computing values using variables
2. Reading in a value from the keyboard with `cin`
3. Making decisions use `if` statements
4. Play with the integrated debugger

Directions

- While completing these labs, you are encouraged to help your classmates and receive as much help as you like. Assignments, however, are individual work. **You may not work on assignments when working with others who are working on the lab.**

1. Experiment with a Variable

1. Create a new folder named `Lab2` with a file named `lab2.cpp`
   - *Hint: Don't put spaces in either your paths (folder names) or file names because it can be tricky to work with these from the command line.*
2. Have your code print out a message:
   ```
   SONG LIBRARY
   ************
   ```
3. Inside `main()`, declare a variable named `songs` of type `int` and initialize it to `0`:
   ```
   int songs = 0;
   ```
4. Output the value of `songs` to the screen:
   ```
   cout << "Initial number of songs: " << songs << endl;
   ```
   Run your program. The output should look like:
   ```
   Initial number of songs: 0
   ```
5. Add a statement which assigns the value 10 to `songs` (simulating adding 10 songs):
   ```
   songs = 10;
   ```
6. Add a statement to outputs `songs` to the screen. The extra output should look like:
   ```
   Number of songs after first adding: 10
   ```
7. Finally add a new statement to change the value of `songs` to 654 and again output to the screen. The combined output should look similar to:
   ```
   SONG LIBRARY
   ************
   Initial number of songs: 0
   Number of songs after first adding: 10
   Number of songs after today: 654
   ```
8. Go back and format your output so that it nicely lines up the data.
   - Make the code always use 3 columns for each number, and be right-aligned.
     Hint: Use `setw(3)` in your `cout` statements. You'll need the following as well:

```
#include <iomanip>
```

Hint: Add spaces to the end of the strings you are outputting (such as "Initial number of songs   ") to line up the ends of the strings.

Hint: Example `cout` with `setw()` and lined up initial strings:

```
cout << "Hello world: " << setw(3) << 42 << endl;
cout << "Hi:          " << setw(3) << 142 << endl;
```

- Desired output:

```
SONG LIBRARY
************
Initial number of songs:              0
Number of songs after first adding:  10
Number of songs after today:        654
```

2. **Understanding**
   Copy these questions, and paste them at the end of your `lab2.cpp` file; comment out the questions, and answers them (also in the comment).
   - What is the difference between defining a variable and using a variable?
   - When looking at a C++ program, how can you tell where a variable is defined?
   - What happens if you try to redeclare a variable?
     You can test this by changing your `songs = 10;` statement to `int songs = 10;`

2. Age in Days

Add the following to your `lab2.cpp`, leaving the previous code in the file.

1. Design and write a program which reads from the user's age from the keyboard (in years). Then, write out to the screen a message of the form
   `"On your last birthday you had lived ___ days!"`
   - There are 365 days per year; ignore leap years.
   - Start by writing the pseudo code (steps) in comments; then fill it in with real code.
   - *Hint: Use `cin` to read from keyboard.*

2. Modify your program to also display the number of complete decades (10's of years) which the user has lived (for example, 49 years is 4 decades).
   - *Hint: Divide the number of years (and `int`) by 10 years per decade (an int).*
     Caution: If you make either number a double, then it will give you 4.9.

3. Modify your program to also display how many years it will be until the user is one decade older.
   - *Hint: First calculate how many years it has been since the start of the last decade. What operator can give you this? If the user is 49 years old, then it's been 9 years since the start of their last decade.*

4. Modify your program to also display how many centuries old the user is exactly.
   - Here we want a number like 0.21 for 21 years old.
   - *Hint: If you divide 21 by 100 what do you get? If you divide 21 by 100.0, what do you get? Try having a constant YEARS_PER_CENTURY be a double instead of an int. Can you explain what is happening?*

5. Modify your program to also round the number of centuries to the nearest whole number and print the result.
   - Given a floating point number (such as stored in a `double`), you can *round* it up using `ceil()`, round it *down* using `floor()`, and round it to the *nearest whole number* using `round()`.
   - These functions are in the `cmath` file, so add the following to the top of your file: `#include <cmath>`
   - The round() function takes a `double` value and returns an `int`.
   - Here is a little program using these three:

```
// Round a number up, down, and to the nearest whole number
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
  double myNumber = 3.14159;
  cout << "Rounded up is " << ceil(myNumber) << endl;
  cout << "Rounded down is " << floor(myNumber) << endl;
  cout << "Rounded to the nearest " << round(myNumber) << endl;
  return 0;
}
```

6. Sample interaction of just this section (may look different if you choose):

```
Please enter your age (in years): 1
On your last birthday you had lived 365 days!
This is equivalent to 0 decades.
In 9 years, you'll be one decade older.
And you are now 0.01 centuries old!
This rounds to 0 centuries.
```

7. Another sample output:

```
Please enter your age (in years): 58
On your last birthday you had lived 21170 days!
This is equivalent to 5 decades.
In 2 years, you'll be one decade older.
And you are now 0.58 centuries old!
This rounds to 1 centuries.
```

8. Comment your program. You must:
    - Comment at the top of the file which briefly describes the program.
    - Comment every *section* of your code. You should have about one comment per 3-4 lines of code.

9. **Understanding**
   Write the answer to the following in your lab .cpp file
    - What are five inputs that make it easy (obvious) to test the number of days calculations? For example, can you at a glance know how many days are in 17 years? No? Well, what number of years could you check easily without using a calculator?

3. Minor or Senior

Add the following to your `lab2.cpp`, leaving the previous code in the file.

1. Using the age (in years) which the user entered above, print out “You are a minor in BC” if the age is less than 19
   - Make 19 a well named constant.
   - *Hint: Use an if statement to make this decision.*

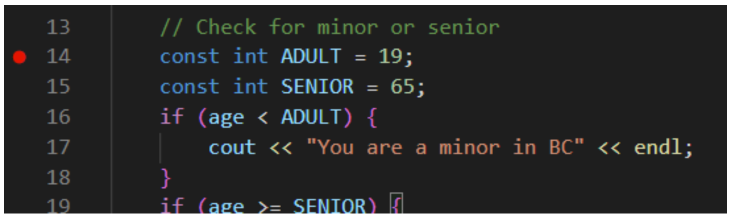
2. If the age is >= 65 then do all of the following:
   - Print out “You are a senior in BC”. Again use a well-named constant.
   - Print out how many years the user has been a senior. For example, if they are 67 print out “You have been a senior for 2 years.”
   - Prompt the user to enter a word how it feels to be a senior and print a positive message.
   - *Hint: You are doing multiple things in the “then” part of the if statement, so you must put all these things into a block.*

3. **Understanding**
   Write the answer to the following in your lab .cpp file
   - It is suggested to use the curly braces { } for the if statements.
     a. Did you use curly braces for both of your `if` statements?
     b. Imagine someone did not use curly braces for the first `if` statement. What would happen if they added a 2nd print statement in the “then” part without using curly braces?
   - Can you think of an invalid age which is still an integer? What does the program do when you enter this invalid age? What could you do to have your program not do anything if the user enters such an invalid age? (Optional: Make it do that!)

4. Integrated Debugger

Debugging a program can be hard. VS Code (and many integrated development environments (IDEs)) have built in debuggers. Let’s play with it and see how it helps!

1. First, watch this 5m video showing the steps; then try it yourself!
2. Open `lab2.cpp` and set a break point before your first “if” statement by clicking to line number to the left of the line of code.
   - You’ll then see a red circle beside the line.



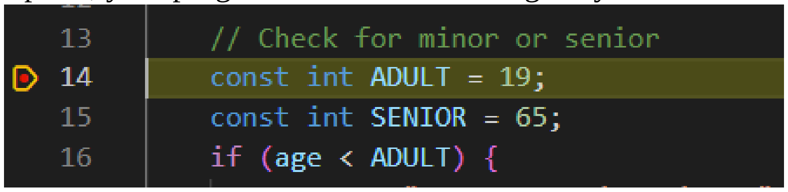
```
// Check for minor or senior
const int ADULT = 19;
const int SENIOR = 65;
if (age < ADULT) {
    cout << "You are a minor in BC" << endl;
}
if (age >= SENIOR) {
```

3. Start the program in the debugger:
   Run → Start Debugging

   If asked about what debug configuration you want, select:
   - g++ - Build and debug active file
   - C++ (GDB/LLDB)

   If this does not work, try deleting the file `.vscode/launch.json` from your project and trying again.

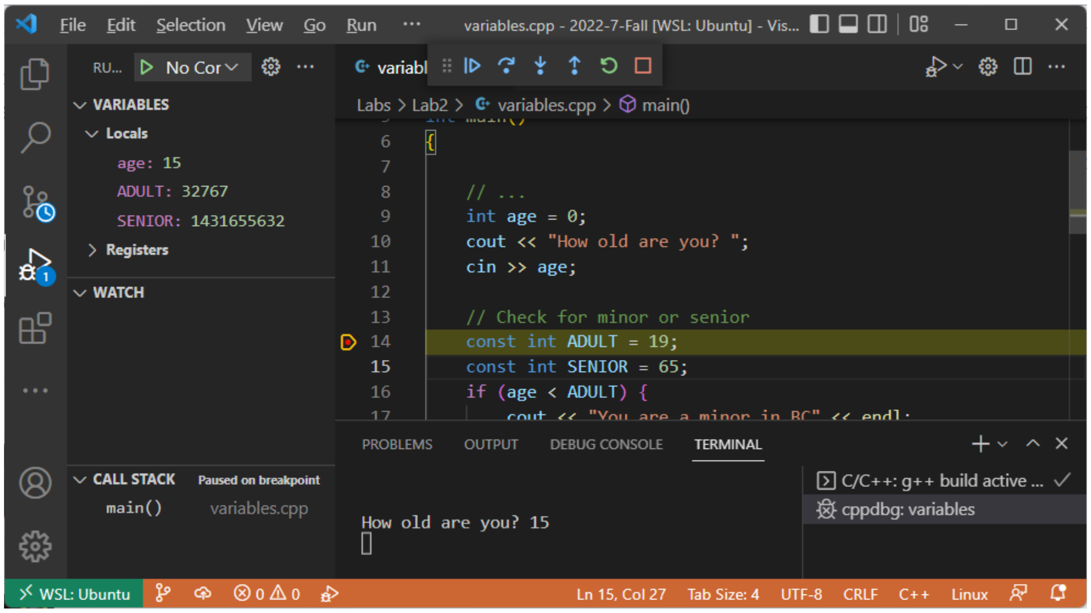
4. You'll now see a yellow bar at line 9 of your code. This shows you the line of the program which will be the next line to execute. If you have any `cin` statements before the breakpoint, your program will first be waiting for you to enter some value.



```
13    // Check for minor or senior
14    const int ADULT = 19;
15    const int SENIOR = 65;
16    if (age < ADULT) {
```

5. If VS Code did not do so automatically, switch to the run view on the left by clicking the icon:

   You should now see



File Edit Selection View Go Run ···
variables.cpp - 2022-7-Fall [WSL: Ubuntu] - Vis...
RU... No Cor
Labs > Lab2 > variables.cpp > main()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
C/C++: g++ build active ...
cppdbg: variables
WSL: Ubuntu
Ln 15, Col 27 Tab Size: 4 UTF-8 CRLF C++ Linux

VARIABLES
Locals
age: 15
ADULT: 32767
SENIOR: 1431655632
Registers
WATCH
CALL STACK Paused on breakpoint
main() variables.cpp

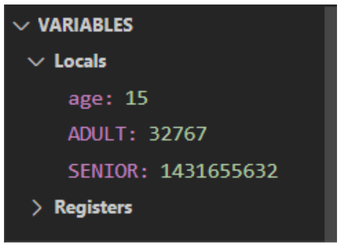
```
6     {
7
8         // ...
9         int age = 0;
10        cout << "How old are you? ";
11        cin >> age;
12
13        // Check for minor or senior
14        const int ADULT = 19;
15        const int SENIOR = 65;
16        if (age < ADULT) {
```

```
How old are you? 15
```

6. You can “step” (or “single step”) your program to execute one line of code at a time. We’ll talk about step-over vs step-into after seeing functions.

7. Experiment stepping:
   - Step over the current line by pressing its hotkey (look it up on the menu!).
   - Note that the yellow line changes to the next line.
      - Step-over a few times to watch what lines are executed.

8. On the left, notice the Variables section which includes Locals. Expand this if it’s not already expanded. It shows the value of some local variables.



```
age: 15
ADULT: 32767
SENIOR: 1431655632
```

9. Resume the program:
   - **Continue** (look up hotkey on menu) let the program run until it either finishes or hits a break point.
10. **Stop** debugging by either clicking the red square at the top, or Run → Stop Debugging.
    - This will end your current debugging run.
    - NOTE: You will have to stop debugging in order to rebuild your code after a change because debugging is running the program: it prevents the compiler from changing the executable file.

11. Re-run the debugger. It will encounter your breakpoint.
12. Mouse over any variable in scope to see its current value in a tool-tip.
13. Correct bugs as you find them.
    When you change the code, you'll need to restart the debugging session for the change to take effect.

Note on using VS Code in Windows *without* Dev Containers, Remote via SSH, or WSL:

The VS Code debugger under Windows, when used *without* these tools, often does not work well with `cin` or `cout` statements. This is why I recommend using one of these ways to write C++ in Windows.

Understanding questions to answer in your lab file:

1. How do you set a breakpoint on a line of code?
2. How do you single-step a program, one line at a time, to watch it execute?
3. How can you watch the values in a variable as it executes (without having to call `cout` all the time to print them)?