Scope, Unit Testing, and Debugging

Thomas Merkh
tmerkh55@gmail.com

Section 2, Week 6

May 12, 2020
Overview

1. Scope

2. Unit Testing

3. Debugging with Visual Studio

4. Next Time
Last time we discussed optional arguments for functions. As a refresher:

- Parameters may be passed either by value or by reference
- A reference to a variable is essentially a variable that stores another variable’s memory address
- Variables passed by reference may be altered from within a function
Last time we discussed optional arguments for functions. As a refresher:

- Parameters may be passed either by value or by reference
- A reference to a variable is essentially a variable that stores another variable’s memory address
- Variables passed by reference may be altered from within a function

Today, we will learn about scopes, debuggers, and revisit ideas from modular programming.
1. Scope

2. Unit Testing

3. Debugging with Visual Studio

4. Next Time
*Scope* in programming refers to the environment or region of a program in which a given variable is declared (and used). We have already dealt with scope several times before, but here we will solidify the idea.

```cpp
#include <iostream>
using namespace std;

int main ()
{
  int k = 5;
  for (int i = 0; i < 5; i++)
  {
    k = i;
  }
  cout << ' ' << k << endl;
  cout << ' ' << i << endl;
  return 0;
}
```

What is wrong with this code?
Scope in programming refers to the environment or region of a program in which a given variable is declared (and used). We have already dealt with scope several times before, but here we will solidify the idea. Consider the code:

```cpp
#include <iostream>
using namespace std;

int main() {
    int k = 5;
    for (int i = 0; i < 5; i++){
        k = i;
    }
    cout << 'k has value: ' << k << endl;
    cout << 'i has value: ' << i << endl;
    return 0;
}
```

What is wrong with this code?
Answer: The error message “’i’: undeclared identifier” will be thrown. Why doesn’t $i$ have the value 5?
Answer: The error message “‘i’: undeclared identifier” will be thrown. Why doesn’t i have the value 5?

The problem has to do with the scope of variable i. Specifically, i went out of scope when the for loop concluded. When a variable goes out of scope, it becomes inaccessible, or in this case, the variable is deleted and the memory is freed.

- The { } tokens enclose scopes, which can be nested within one another
- Variables declared outside of a scope will be accessible in the inner scope(s), but the reverse does not hold
When calling a function, the function can only access the parameters that are passed into it, any variables defined within the function’s code block, or variables defined as global\(^1\). This has several consequences that we have encountered:

- A variable’s name can be reused inside of a function and not overwrite variables elsewhere with the same name
- Variables declared inside of a function are (usually) not accessible outside of the function after it concludes
- Functions can still manipulate variables defined in alternative scopes (recall passing by reference)

\(^1\)Avoid global variables whenever possible.
What do you think the output of this program will be?

```cpp
#include <iostream>
using namespace std;

int func(int k, int p){
    int m = 6;
    int n = k*p*m;
    return k+p+m;
}

int main() {
    int k = 5;
    int m = 9;
    int n = 1;
    int p = func(n,m);
    cout << 'm has value: ' << m << endl;
    cout << 'n has value: ' << n << endl;
    cout << 'p has value: ' << p << endl;
    return 0;
}
```
Scope

Unit Testing

Debugging with Visual Studio

Next Time
Unit testing is a method of testing software during development. This is intimately related to the idea of Modular Programming, or writing a code using many simple, concise functions which are easy to check. Unit testing is:

- Testing the smallest stand-alone piece of a program, i.e. a function, a complicated loop, an I/O section, etc.
- The lowest level of software testing, followed by integration testing, and then system testing
- Most useful when performed after every sizable change to the code is implemented

Unit testing saves time, resources, frustration, and if you are doing this professionally, money.
The Message: Do Unit Testing

If you aren’t sold on this Modular Programming and Unit Testing thing already, then consider how unchecked errors compound in difficulty to fix.

- If there is a single change, then the new error must be due to that
- If there are three changes, then there are now $2^3 = 8$ possibilities for which changes work
- Furthermore, by fixing one change, this could induce errors in the previous functioning changes

Avoid this compounding frustration. Test your programs.
Scope

Unit Testing

Debugging with Visual Studio

Next Time
A debugger is a component of many IDEs which can assist the user in uncovering the source of coding errors. As your projects grow in size, knowing efficient methods for finding coding errors becomes evermore important.

- **Benefit:** Efficient error finding and resolution.
- **Drawback:** There is another learning curve on top of learning an IDE + language.
- **Benefit:** Debugging is a marketable skill. Not every computer programmer is familiar with debugging.
The most fundamental aspect of a debugger is the *breakpoint*.

- Breakpoints are locations at which your program will pause while debugging.
- Breakpoints can be set by left-clicking in the margin next to the desired line of code.
- Breakpoints appear as red dots in (most) IDEs.

We will see an example of this in the slides that follow.
Once breakpoints are set, hitting F5 on your keyboard runs the code up until the first (next) breakpoint. Hit F5 to continue to the next breakpoint, until the program concludes. Other commands which are useful are:

- F11 - steps through the code line by line, continuing on to the next line each time F11 is hit
- F10 - steps through the code line by line, but hops over function calls (the function call still executes)
- Setting a watch - while debugging, a *watch* can be set by scrolling over a variable and “add watch”. This allows the user to watch the value of a variable and track changes over time
Figure: A image of the Visual Studio IDE with a single breakpoint set. This is displayed as a red dot in the left margin of the code.
Figure: From left to right, the white arrows are indicating: (1) The current stop-point/breakpoint. (2) The "play" button, which allows you to run the single line of code. (3) The debug menu. (4) The continue button (F5 equivalent).
It's worth the 30 minutes it takes to familiarize yourselves with these couple commands. Most important debugging activities can be located under the `debug` menu. There are some icon-equivalents to the F# commands discussed on previous slides, like how the “Continue” button is really just the F5 command.

1 Scope

2 Unit Testing

3 Debugging with Visual Studio

4 Next Time
At last... Things to do between now and then:

- Classes: public, private, designation, constructors, member functions