PIC 10A 2B

TA: Bumsu Kim

UCLA

Today…

- Scope of Variables
  - Local and Global Variables, and References
- Exercises Problems
  - isPalindrome, reverse
- HW4 Hints

UCLA

Scope of Variables

- C++ allows you to “shadow” a variable name
  - Each block defines its own scope region, and you can define a new variable with the same name (outside the scope)
  - This is called **name hiding** or **shadowing**

```
int i = -1;          // (A)
cout << i << endl;   // (B)
for (int i = 0; i < 3; ++i) {      // (C)
       cout << i << endl;          // (D)
       {
              cout << "   " << i << endl; // (E)
              int i = 30;                 // (F)
              cout << "   " << i << endl; // (G)
       }
       cout << "      " << i << endl;    // (H)
       cout << endl;
}
cout << "         " << i << endl; // (I)
```

UCLA

Scope of Variables

```
int i = -1;            // (A)
cout << i << endl;     // (B)
for (int i = 0; i < 3; ++i) {      // (C)
        cout << i << endl;         // (D)
        {
                cout << "    " << i << endl; // (E)
                int i = 30;                  // (F)
                cout << "    " << i << endl; // (G)
        }
        cout << "        " << i << endl;     // (H)
        cout << endl;
}
cout << "            " << i << endl;  // (I)
```

- Output:

```
-1                  // (A)
0                   // (C)
   0                // (C)
   30               // (F)
      0             // (C)

1
   1
   30
      1

2
   2
   30
      2

         -1
```

Scope of Variables

- When out of scope, a variable is “destructed” and not accessible anymore

```
{
        int j = 1;
        cout << "j = " << j << endl;
}
cout << "j = " << j << endl;
```

Compile Error: j is undefined

- Every variable defined in some block is called a “local” variable
- If a variable is defined outside of all the blocks, it’s called a “global” variable
  - and accessible everywhere

```
#include <iostream>
using namespace std;
int global_int_var = 12; // global var
int main() {
        cout << global_int_var << endl;
        return 0;
}
```

UCLA

Avoid Global Variables!

- Global variables seem very useful because of its ease of access, however, it is a double-edged sword
- When your program gets larger, it becomes more unpredictable

```
int g_mode;          // declare global variable (will be zero-initialized by default)
void doSomething() {
    g_mode = 2;      // set the global g_mode variable to 2
}
int main() {
    g_mode = 1;      // note: this sets the global g_mode variable to 1.  It does not declare a local g_mode variable!

    doSomething();

    // Programmer still expects g_mode to be 1
    // But doSomething changed it to 2!

    if (g_mode == 1) {
        std::cout << "No threat detected.\n";
    } else {
        std::cout << "Launching nuclear missiles...\n";
    }
    return 0;
}
```

UCLA

Avoid (non-const) Global Variables!

- Another important reason for avoiding global variables is that it makes debugging more difficult
- If you referenced the global `g_mode` variable 400 times in your code, you need to look through every use of it to understand how it's being used in different cases, what its valid values are, etc.

- There aren't many cases where global variables are useful
  - However, sometimes they are still useful
  - `std::cout` and `std::cin` are global variables
  - A *log* file, where you dump error or debug information, is another good example

UCLA

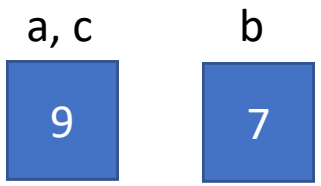
(R) References

- A **reference** to a variable is just "another name" of the other

```
T tvar;                    // variable of type T (can be int, double, string, etc)
T& ref_tvar = tvar;        // a reference to a type T variable
```

- They refer to the same "address" in the memory

```
int a = 5;
int b = a;
int& c = a;
cout << a << ' ' << b << ' ' << c << endl;
b = 7;
cout << a << ' ' << b << ' ' << c << endl;
c = 9;
cout << a << ' ' << b << ' ' << c << endl;
```




```
5 5 5
5 7 5
9 7 9
```

Exercise – isPalindrome

- Write a predicate function that checks if the input string is a palindrome or not
  - A palindrome is a word, number, phrase, or other sequence of symbols that reads the same backwards as forwards, such as the words madam or racecar.

- To Do:
  - Do we need additional libraries to include?
  - What should be the signature of the function?
  - Definition of the function

Standard (Doxygen) Documentation Style

```
/**
<A short one line description>

<Longer description>
<May span multiple lines or paragraphs as needed>

@param  Description of function's input parameter
@param  ...
@return Description of the return value
*/
```

UCLA

Exercise – reverse

- Write a procedure that reverses the input string, by writing a helper function "swap" that swaps two characters

- Documentations for those functions:

```
/**
* swaps two characters
*
* @param two  characters of type ref_to_char
*/
```

```
/**
* Reverses the input string
*
* @param str  Input string to be reversed
*/
```

UCLA

HW 4 Hints

- Problem 1: Review the exercise from Week 3: “Prime factorization”
  - In fact this covers the first problem of the HW.
  - Please make it clear in the submission that you referred to my code (if you did)

- Problem 2: Review the exercise from Week 4: “Random_Walk_Simulator”
  - It is a simplified, 1-D version of HW4 P2
  - Now, instead of having 50% chance to go up/down, you need 25% chance to go up/down/left/right
  - In addition to the boundary check, you also need to check if it’s back at the origin
  - The rest is more or less the same!

- Let me know if you want me to go over those previous exercises again

UCLA

Your Feedback is welcome

- Don’t hesitate to give a feedback on the discussion
- Use the link on my Github repo, or the link below:
  - https://forms.gle/erZj1iSgHNrHQuXk6

My Github repo on the web looks like:

code	Week2 Tu
LICENSE	Initial commit
README.md	Update README.md

README.md

PIC10A

PIC10A discussion 2B, UCLA for Fall 2022

Google form link for feedbacks: https://forms.gle/erZj1iSgHNrHQuXk6

Click this link

UCLA