Scope

Lecture 9

Outline

1. Scope
2. Constants

Variable Scope

- All variables have a set *scope*
  - Parts of the code where that variable can be used
- Variables declared in a function are *local variables* for that function
  - Can not be used outside of that function
- Function argument variables are treated as local variables in that function

Example

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

int factorial(int n);

int main()
{
     int input;
     cout << "Enter a number: ";
     cin >> input;
     cout << input << "! = " << factorial( input ) << endl;
     return 0;
}

int factorial(int n)
{
     int total = 1;
     while ( n > 0 )
     {
          total = total * n;
          n--;
     }
     return total;
}
```

input is local to the main function

total is local to the factorial function

n is local to the factorial function

Same Variable Names

- Variables in different scopes can have the same name (and be different types)
- They are different variables!
- Two variables with the same name but in different scopes are not related in any way
- For now, try not to reuse variables names in different functions to help avoid confusion

Example

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

double my_func();

int main()
{
     double my_num = 2.7;
     double res;
     cout << "In main, my_num=" << my_num << endl;
     res = my_func();
     cout << "In main, my_num=" << my_num << endl;
     cout << "In main, res=" << res << endl;
     return 0;
}

double my_func()
{
     double my_num = 5.2;
     cout << "In my_func, my_num=" << my_num << endl;
     return my_num;
}
```

my_num is local to the main function

my_num is local to the my_func function

Global Scope

- Variables and constants can be placed in the *global* scope by declaring them outside of all functions
- Most often useful for constants that are used in multiple functions
- Avoid using global variables when possible
  - In this course, you should **never** use global variables (only global constants)

Constants

- It's usually a good idea to name constants in your program if they have some special meaning
- By convention, variables names with ALL CAPITAL LETTERS are constants
- C++ includes **`const`** "variables" to strictly enforce the idea of a constant (value can not be changed after initialization)
  - Example: `const int CENTS_PER_DOLLAR = 100;`
  - Generic form: `const type NAME = value;`

Example

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

const double DOLLARS_PER_EURO = 1.27;

double dollars_to_euros(double dollars);
double euros_to_dollars(double euros);

int main()
{
     cout << "5 dollars is " << dollars_to_euros( 5.0 ) << " euros" << endl;
     cout << "5 euros is " << euros_to_dollars( 5.0 ) << " dollars" << endl;
     return 0;
}

double dollars_to_euros(double dollars)
{
     return ( dollars / DOLLARS_PER_EURO );
}

double euros_to_dollars(double euros)
{
     return ( euros * DOLLARS_PER_EURO );
}
```

Exercise

- Write a program that calculates the area and circumference of a circle given its radius. Specifically:
  - Use a global constant for the value of π (`3.14159`)
  - Write a function that calculates the area (πr^2)
  - Write a function that calculates the circumference ($2\pi r$)

Example

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

const double PI = 3.14159;

double circle_area(double radius);
double circle_circumference(double radius);

int main()
{
	double r;
	cout << "Enter the radius: ";
	cin >> r;
	cout << "The area is " << circle_area( r ) << "." << endl;
	cout << "The circumference is " << circle_circumference( r ) << "." << endl;
	return 0;
}

double circle_area(double radius)
{
	return ( PI * radius * radius );
}

double circle_circumference(double radius)
{
	return ( 2 * PI * radius );
}
```

Global Variable Gotcha

- If you have a global variable and a local variable in a function with the same name, the local variable "hides" the global one
- The two variables are declared in different scopes, so they are completely different variables
- The global variable will not be accessible within the same scope as the local variable that has the same name

Example

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

int my_var = 10;

void my_func();

int main()
{
    int my_var = 5;
    my_func();
    cout << my_var << endl;
    return 0;
}

void my_func()
{
    cout << my_var << endl;
}
```

my_var is global and should be accessible in all functions

my_var is redeclared within the main function here, so any uses of my_var in main will use the local variable, not the global one

No local my_var has been declared, so uses the global variable

Other Scope Rules

- Any variables declared within a code block (everything between a set of braces `{}`), are local to that block
- Variables declared inside of an `if-else` block, `while` loop, or `for` loop can only be used inside of that block or loop
- Similar rules apply for "hiding" variables of the same name from an outer scope as with global variables

Example (1)

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

int main()
{
    int i;
    for ( i=0; i<10; i++ )
    {
        int j;
        j = i*9;
        cout << j << endl;
    }
    return 0;
}
```

i can be used anywhere in the function

j can only be used within the loop body

Example (2)

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

int main()
{
    for ( int i=0; i<10; i++ )
    {
        cout << i << endl;
    }
    cout << i << endl;
    return 0;
}
```

ERROR!!
use of undeclared identifier 'i'

Wrap Up

- All variables and constants have a certain scope (global, local, block)
- Variables can only be used within the same scope or any sub-scopes
- Be very careful about reusing variable names
- Global constants are useful, but global variables should only be used in certain cases (and not in this class!)