Scope of Variables and Global Variables

Larry Caretto
Computer Science 106
Computing in Engineering and Science

March 30, 2006

California State University Northridge

Outline

- Scope of a variable
  - Region of program where variable is recognized
  - Same variable name in different functions represents different variables
  - Can have limited scope (within braces) for variables in a single function
- Global variables, declared outside function, available to all functions
- Summary of functions

Scope of a Variable

- Scope of a variable is the part of program that can use the variable
- We see that we can have the same variable name in different functions
- These names, although the same, occupy two different memory locations in the computer and are not related
- Even within a single function we can limit the part of a function in which a variable is in scope (exists)

Background

- All variables must be declared (given a type) before they are used
- Variables can be declared given a value when declared or later in the code
- Usually assign a value before first use
- Scope refers only to declaring a variable, not to assigning it a value
  - This is just a reminder that we have to initialize variables as well as declare them

Basic Rule for Scope

- A variable defined in a set of braces only exists within those braces
- It can be used anywhere in the program below its initial declaration
  - This includes sets of braces that are opened below the initial declaration
- After close of brace where variable is declared, the variables "goes out of scope" it cannot be used

Example of Scope

```
double x, c = 4;
if ( c == 4 )
{
   x = 12;
   double y = 2;  // limited scope
}
cout << x << "  " << y;
// statement above will give syntax
// error; y is not defined here
```

Another example of Scope

```
double y = 0, c = 4;
if ( c == 4 )
{
   double y = 2;  // different var-
        // iable with limited scope
}
cout << "y = " << y;
// statement above will print y = 0
// from initial declaration of y
```


Last example Revisited

```
double y = 0, c = 4;
if ( c == 4 )
{
    y = 2;  // same as variable
            // declared above
}
cout << "y = " << y;
// statement above will print y = 2
// from setting in if block
```


Scope Exercise

- What is printed from following code?

```
int x = 3;
for ( int i = 0; i < 5; i++)
{
    x += i;
}
cout << "x = " << x;
```

- Output is x = 13

Another Scope Exercise

- What is printed from following code?

```
int x = 3;
for ( int i = 0; i < 5; i++)
{
    double x += i;
}
cout << "x = " << x;
```

- Output is x = 3

Where to Declare Variables

- Current programming practice declares variables as close to first time of use as possible
- May have to be declared earlier in the code to give appropriate scope
  - First use of variable may be inside a loop
  - We must declare it prior to the loop if we want to use variable after the loop ends

Another Example

- Code below will not work because yesNo goes out of scope after closing brace

```
do
{  // other program statements here
   cout << "Another run(Y/N)? ";
   char yesNo; // bad location
   cin >> yesNo;
}
while( yesNo == 'Y' || yesNo == 'y' );
```

Another Example Corrected

- Code below works because yesNo is declared before brace opening the loop

```
char yesNo; // correct location
do
{// other program statements here
   cout << "Another run(Y/N)? ";
   cin >> yesNo;
}
```


Global Variables

- Global variables have scope of more than one function
  - Declared outside function boundaries
  - Have scope of all functions from declaration to end of file
  - Usually declared at top of program to be present in all functions
  - Considered bad programming practice unless necessary for some reason
  - Use only when variable must be accessed by several functions or there are problems in passing the variable

Trace Global Variables

- What is program output?

```
int status = 0;   // global
int main() {
   cout << status << "   ";
   f1();   f2();
   cout << " " << status // more
}
void f1() { status = 1; }
void f2() {cout << status << endl;}
```

- Program output is 0 1 1

Project Two Global Variables

- In project two the main function calls a function which calls a third function
- We want to get data from main to the third function
- We do not want to rewrite the second function, but it does not allow us to pass the necessary information
- Use global variables to get the information from main to third function
- See example next chart

Global Variable Example

```
double L, alpha;  // global
int main() {
  double x = 2, y = 6;
  L = 10; alpha = 4e-6;   //
 set
  cout << f1(x,y);
}
double f1( double x, double y
 ) {
  return f2(x,y);
```


Duplicate Variable Names

- A function can declare a variable with the same name as a global variable
- In this case the global variable is not available to the function
- The local variable defined by the function is the same as any usual variable defined in a function

```
double x = 12, y = 32;  // global
int main() {
  double x = 3;  // main does not
                 // global x
```

Trace This Global Example

```
double x = 5, y = 12;  // global
int main() {
  double x = 3;
  double z = x + y;
  y = f2( z );
   // more code in main
}
double f2( double y ) {
   return x + y;
}
 double f4() {
   return 2 * y;
}
```

What does this print?

z = 3 + 12 = 15

cout >> f4();

prints 40

return 5 + 15 = 20

return 2 * 20 = 40

Function Summary

- Use functions to organize code
- Elements of a function
  - Header with type, name, and argument list
  - Body with code that function executes
  - Statement to return information through function name in calling program must be included in function body
  - Prototype at start of program which is header with a semicolon
- Function name calls function and returns value

Function Summary II

- When writing code, complete code for one function before starting a new function
- Can call any function from any other function
- Call function by placing name of function to be called in code
- Transfer control and data to function called

Function Summary III

- Pass information to function through argument list in function header
  - Correspondence by position (order) of arguments in header and position of arguments in calling function
  - Default of pass by value will not change arguments in calling function
  - Pass by reference (requires ampersand(&) in function header and prototype) changes arguments in calling function

Function Summary IV

- Scope of variables is part of program where a variable can be used
- Variables can only be used within braces where there are declared and only following the declaration
- Global variables, declared outside any function, can be used by any function following the declaration