Scope, Lifetime, and More on Functions

- Identifiers can be declared either inside a block (local variables) or outside a block (function names).
- C++ has rules governing how a function may access parameters declared outside its own block.
- A value-returning function returns a single result to the expression from which it was called.
- We'll look at writing user-defined value-returning functions.

Scope of Identifiers

- Listing all of the places from which an identifier can be accessed legally is describing that identifier's scope.
- C++ defines several categories of scope for any identifier
  - Class scope. Postpone this until chapter 11.
  - Local scope. The scope of an identifier declared inside a block extends from the point of declaration to the end of that block.
  - Global scope. The scope of an identifier declared outside all functions and classes extends from the point of declaration to the end of the entire file containing the program code.

Global Variable Example

```
int gamma;        // Global variable

int main()
{
    gamma = 3;
      ⋮
}

void SomeFunc()
{
    gamma = 5;
      ⋮
}
```


Local Definitions Take Precedence Over Global Ones

```
void SomeFunc(float);

const int a = 17;        // A global constant
int b, c;                // Two global variables

int main()
{
    b = 4;               // Assignment to global b
    c = 6;               // Assignment to global c

    SomeFunc(42.8);

   return 0;
}
```


Local Definitions Take Precedence Over Global Ones

```
void SomeFunc(float c)   // Prevents access to global c

{
    float b;           // Prevents access to global b

    b = 2.3;             // Assignment to local b

    cout << "a = " << a;  // Output global a(17)
    cout << "b = " << b;  // Output local b(2.3)
    cout << "c = " << c;  // Output local c(42.8)
}
```


Scope Rules Explored

```
void Block1(int, char&);
void Block2();

int a1;               // One global variable
char a2;              // Another global variable
int main()
{
   ⋮
}
void Block1(int a1,   // Prevents access to global a1
            char& b2) // Has same scope as c1 and d2
{
   int c1;          // A variable local to Block1
   int d2;          // Another variable local to Block1
     ⋮
}
```

Scope Rules Explored (continued)

```
void Block2()
{
   int a1;             // Prevents access to global a1
   int b2;             // Local to Block2; no conflict
                       // with b2 in Block1
   while (…)
   {                  // Block3
      int c1;         // Local to Block3; no conflict
                       // with c1 in Block1
      int b2;          // Prevents nonlocal access to b2
                       // in Block2; no conflict with
                       // b2 in Block1
        ⋮
   }
}
```


Accessing Identifiers From Other Program Files

- Individually
  `extern int someInt;`
- Collectively, from a namespace
  `std::abs(beta);` (qualified name)
  `using std::abs;` (using declaration)
  `using namespace std;` (using directive)

Lifetime of A Variable

- Lifetime is the period of time during program execution when an identifier actually has memory allocated to it.
- Two types of variables
  - Static variables – storage remains allocated for the duration of the entire program
  - Automatic variables – storage is allocated at block entry and deallocated at block exit
- By default, variables declared within a block are automatics, this can be changed using the reserved word `static`.

Side Effects Illustrated

```
void CountChars();
int count = 0;     // Supposed to count input lines
char ch;           // Hold one input character
int main()
{
   cin.get(ch);
   while(cin)
   {
      count++;
      CountChars();
      cin.get(ch);
   }
   cout << count << " lines of input processed." << endl;
   return 0;
}
```


Side Effects Illustrated (continued)

```
void CountChars()

// Counts the number of characters on one input line
// and prints the count Note: main() has already
// read the first character on a line
{
   count = 0;
   while (ch != '\n')
   {
      count++;
      cin.get(ch);
   }
   cout << count << "characters on this line." << endl;
}
```


Another Scope Example

```
void one()
{
   int a = 10;
   cout << " a = " << a << endl;
}

void two ()
{
   int a = 0;
   one( );
   cout << " a = " << a << endl;
}

int main ()                    a = 20
{
   int a = 20;
   two( );
}
```

UAH CPE 112

Another Scope Example

```
void one()
{
   int a = 10;
   cout << " a = " << a << endl;
}

void two ()
{
   int a = 0;
   one( );
   cout << " a = " << a << endl;
}

int main ()
{
   int a = 20;
   two( );
}
```

a = 20

UAH CPE 112

Another Scope Example

```
void one()
{
   int a = 10;
   cout << " a = " << a << endl;
}

void two ()
{
   int a = 0;
   one( );
   cout << " a = " << a << endl;
}

int main ()
{
   int a = 20;
   two( );
}
```

a = 0

UAH CPE 112

Another Scope Example

```
void one()
{
   int a = 10;
   cout << " a = " << a << endl;
}

void two ()
{
   int a = 0;
   one( );
   cout << " a = " << a << endl;
}

int main ()
{
   int a = 20;
   two( );
}
```

a = 0

UAH CPE 112

Another Scope Example

```
void one()
{
   int a = 10;
   cout << " a = " << a << endl;
}

void two ()
{
   int a = 0;
   one( );
   cout << " a = " << a << endl;
}

int main ()
{
   int a = 20;
   two( );
}
```

a = 10

UAH CPE 112

Another Scope Example

```
void one()
{
   int a = 10;
   cout << " a = " << a << endl;
}

void two ()
{
   int a = 0;
   one( );
   cout << " a = " << a << endl;
}

int main ()
{
   int a = 20;
   two( );
}
```

a = 10

UAH CPE 112

Another Scope Example

```
void one()
{
   int a = 10;
   cout << " a = " << a << endl;
}

void two ()
{
   int a = 0;
   one( );
   cout << " a = " << a << endl;
}

int main ()
{
   int a = 20;
   two( );
}
```

a = 0

Another Scope Example

```
void one(void)
{
   int a = 10;
   cout << " a = " << a << endl;
}

void two (void)
{
   int a = 0;
   one( );
   cout << " a = " << a << endl;
}

int main (void)                    a = 20
{
   int a = 20;
   two( );
}
```