Chapter 6: Functions

Hanoi University of Science and Technology

December 8, 2025

Slide contents are mainly based on the following documents:

- Tony Gaddis, Starting out with C++, Six Edition, Pearson, 2009
- Tony Gaddis, Power Points Slides:
  `https://www.umsl.edu/~antognolij/gaddis.html`

Local and Global Variables

- Variables defined inside a function are local to that function.
  - Hidden from the statements in other functions
  - Other functions cannot access them.
- Other functions may have separate, distinct variables with the same name.

Local and Global Variables

Program 6-16

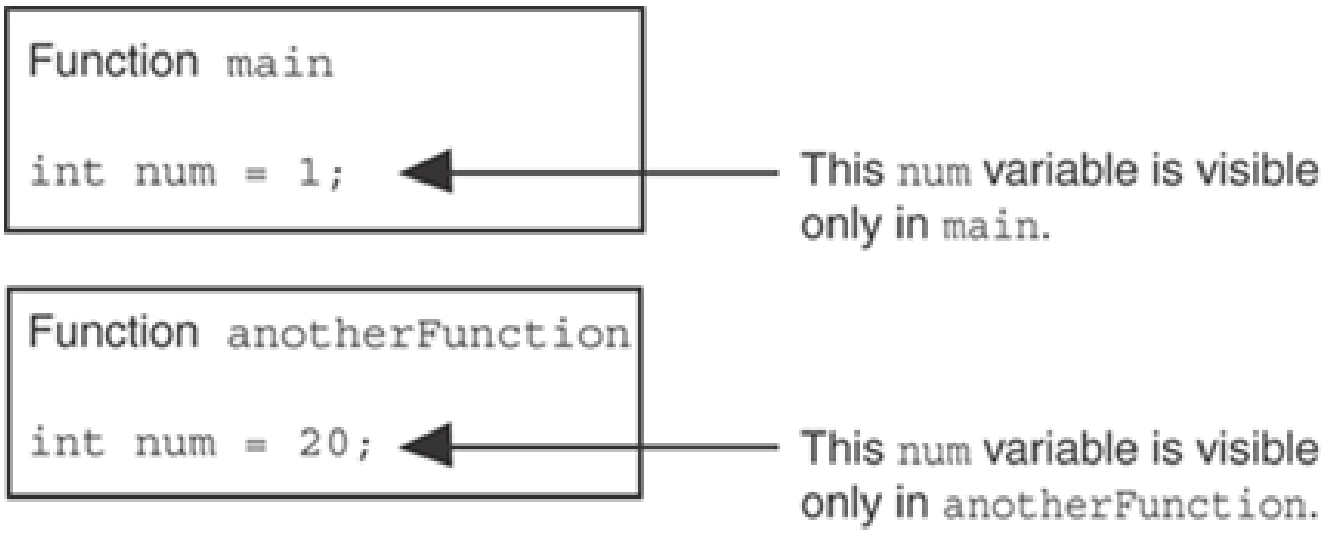
```
1  // This program shows that variables defined in a function
2  // are hidden from other functions.
3  #include <iostream>
4  using namespace std;
5
6  void anotherFunction(); // Function prototype
7
8  int main()
9  {
10    int num = 1;    // Local variable
11
12    cout << "In main, num is " << num << endl;
13    anotherFunction();
14    cout << "Back in main, num is " << num << endl;
15    return 0;
16 }
17
18 //*********************************************************
19 // Definition of anotherFunction                          *
20 // It has a local variable, num, whose initial value      *
21 // is displayed.                                          *
22 //*********************************************************
23
24 void anotherFunction()
25 {
26    int num = 20;  // Local variable
27
28    cout << "In anotherFunction, num is " << num << endl;
29 }
```

Program Output

```
In main, num is 1
In anotherFunction, num is 20
Back in main, num is 1
```

Local and Global Variables

- When the program is executing in `main`:
  - the `num` variable defined in `main` is visible
- When `anotherFunction` is called:
  - only variables defined inside it are visible
  - `num` variable in `main` is hidden.



Local Variable Lifetime

- A function's local variables exist only while the function is executing.
- When the function begins:
  - Its local variables and its parameter variables are created in memory.
- When the function ends:
  - The local variables and parameter variables are destroyed.

Global Variables

- A global variable is any variable defined outside all the functions in a program.
- The scope of a global variable is the portion of the program from the variable definition to the end.
- A global variable can be accessed by **all functions** that are defined **after** the global variable is defined.

Global Variables

Program 6-17

```
1  // This program shows that a global variable is visible
2  // to all the functions that appear in a program after
3  // the variable's declaration.
4  #include <iostream>
5  using namespace std;
6
7  void anotherFunction(); // Function prototype
8  int num = 2;            // Global variable
9
10 int main()
11 {
12     cout << "In main, num is " << num << endl;
13     anotherFunction();
14     cout << "Back in main, num is " << num << endl;
15     return 0;
16 }
17
18 //*****************************************************
19 // Definition of anotherFunction                      *
20 // This function changes the value of the             *
21 // global variable num.                               *
22 //*****************************************************
23
24 void anotherFunction()
25 {
26     cout << "In anotherFunction, num is " << num << endl;
27     num = 50;
28     cout << "But, it is now changed to " << num << endl;
29 }
```

Program Output

```
In main, num is 2
In anotherFunction, num is 2
But, it is now changed to 50
Back in main, num is 50
```

Global Variables

- Global variables:
  - Difficult to debug.

```
int count = 0; // global variable
void function1() { count++; }
void function2() { count += 5; }
```

  - Make functions difficult to reuse in other programs

Initializing Local and Global Variables

- Local variables are not automatically initialized. They must be initialized by programmer.
- Global variables (not constants) are automatically initialized to 0 (numeric) or NULL (character) when the variable is defined.

Static Local Variables

- Local variables only exist while the function is executing. When the function terminates, the contents of local variables are lost.
- `static` local variables retain their contents between function calls.
- `static` local variables are defined and initialized only the first time the function is executed. 0 is the default initialization value.

Static Local Variables

Program 6-21

```
1  // This program shows that local variables do not retain
2  // their values between function calls.
3  #include <iostream>
4  using namespace std;
5
6  // Function prototype
7  void showLocal();
8
9  int main()
10 {
11     showLocal();
12     showLocal();
13     return 0;
14 }
15
16 //*********************************************************
17 // Definition of function showLocal.                      *
18 // The initial value of localNum, which is 5, is displayed. *
19 // The value of localNum is then changed to 99 before the  *
20 // function returns.                                      *
21 //*********************************************************
22
23 void showLocal()
24 {
25     int localNum = 5; // Local variable
26
27     cout << "localNum is " << localNum << endl;
28     localNum = 99;
29 }
```

Program Output

```
localNum is 5
localNum is 5
```

A Different Approach, Using a Static Variable

Program 6-22

```
1  // This program uses a static local variable.
2  #include <iostream>
3  using namespace std;
4
5  void showStatic(); // Function prototype
6
7  int main()
8  {
9       // Call the showStatic function five times.
10      for (int count = 0; count < 5; count++)
11         showStatic();
12      return 0;
13 }
14
15 //*************************************************************
16 // Definition of function showStatic.                         *
17 // statNum is a static local variable. Its value is displayed *
18 // and then incremented just before the function returns.     *
19 //*************************************************************
20
21 void showStatic()
22 {
23      static int statNum;
24
25      cout << "statNum is " << statNum << endl;
26      statNum++;
27 }
```

Program Output

```
statNum is 0
statNum is 1
statNum is 2
statNum is 3
statNum is 4
```

A Different Approach, Using a Static Variable

Program 6-23

```
1  // This program shows that a static local variable is only
2  // initialized once.
3  #include <iostream>
4  using namespace std;
5
6  void showStatic(); // Function prototype
7
8  int main()
9  {
10     // Call the showStatic function five times.
11     for (int count = 0; count < 5; count++)
12        showStatic();
13     return 0;
14 }
15
16 //*************************************************************
17 // Definition of function showStatic.                         *
18 // statNum is a static local variable. Its value is displayed *
19 // and then incremented just before the function returns.     *
20 //*************************************************************
21
22 void showStatic()
23 {
24     static int statNum = 5;
25
26     cout << "statNum is " << statNum << endl;
27     statNum++;
28 }
```

Program Output

```
statNum is 5
statNum is 6
statNum is 7
statNum is 8
statNum is 9
```

Default Arguments

- A **default** argument is an argument that is passed automatically to a parameter if the argument is missing on the function call.
- Must be a constant declared in prototype:

```
void evenOrOdd(int = 0);
```

- Can be declared in header if no prototype
- Multi-parameter functions may have default arguments for some or all of them:

```
int getSum(int, int=0, int=0);
```

Default Arguments

Program 6-24

```
1  // This program demonstrates default function arguments.
2  #include <iostream>
3  using namespace std;
4
5  // Function prototype with default arguments
6  void displayStars(int = 10, int = 1);
7
8  int main()
9  {
10     displayStars();      // Use default values for cols and rows.
11     cout << endl;
12     displayStars(5);     // Use default value for rows.
13     cout << endl;
14     displayStars(7, 3);  // Use 7 for cols and 3 for rows.
15     return 0;
16 }
17
18 //*********************************************************
19 // Definition of function displayStars.                   *
20 // The default argument for cols is 10 and for rows is 1.*
21 // This function displays a square made of asterisks.     *
22 //*********************************************************
23
24 void displayStars(int cols, int rows)
25 {
26     // Nested loop. The outer loop controls the rows
27     // and the inner loop controls the columns.
28     for (int down = 0; down < rows; down++)
29     {
30         for (int across = 0; across < cols; across++)
31             cout << "*";
32         cout << endl;
33     }
34 }
```

Program Output

Default Arguments

- If not all parameters to a function have default values, the defaultless ones are declared first in the parameter list:

```
int getSum(int, int=0, int=0); // OK
int getSum(int, int=0, int); // NO
```

- When an argument is omitted from a function call, all arguments after it must also be omitted:

```
sum = getSum(num1, num2); // OK
sum = getSum(num1, , num3); // NO
```

Using Reference Variables as Parameters

- A mechanism that allows a function to work with the original argument from the function call, not a copy of the argument
- Allows the function to modify values stored in the calling environment
- Provides a way for the function to 'return' more than one value

Using Reference Variables as Parameters

- A reference variable is an alias for another variable
- Defined with an ampersand (&):

```
void getDimensions(int&, int&);
```

- Changes to a reference variable are made to the variable it refers to

Using Reference Variables as Parameters

Program 6-25

```
1  // This program uses a reference variable as a function
2  // parameter.
3  #include <iostream>
4  using namespace std;
5
6  // Function prototype. The parameter is a reference variable.
7  void doubleNum(int &);
8
9  int main()
10 {
11     int value = 4;
12
13     cout << "In main, value is " << value << endl;
14     cout << "Now calling doubleNum..." << endl;
15     doubleNum(value);
16     cout << "Now back in main. value is " << value << endl;
17     return 0;
18 }
19
20 //*************************************************************
21 // Definition of doubleNum.                                   *
22 // The parameter refVar is a reference variable. The value*
23 // in refVar is doubled.                                      *
24 //*************************************************************
25
26 void doubleNum (int &refVar)
27 {
28     refVar *= 2;
29 }
```

Program Output

```
In main, value is 4
Now calling doubleNum...
Now back in main. value is 8
```

Reference Variable Notes

- Each reference parameter must contain &
- Space between type and & is unimportant
- Must use & in both prototype and header
- Argument passed to reference parameter must be a variable – cannot be an expression or constant
- Use when appropriate – don't use when argument should not be changed by function, or if function needs to return only 1 value

Overloading Functions

- Overloaded functions have the same name but different parameter lists
- Can be used to create functions that perform the same task but take different parameter types or different number of parameters
- Compiler will determine which version of function to call by argument and parameter lists

Function Overloading Examples

- Using these overloaded functions:

```
void getDimensions(int); // 1
void getDimensions(int, int); // 2
void getDimensions(int, double); // 3
void getDimensions(double, double);// 4
```

- The compiler will use them as follows:

```
int length, width;
double base, height;
getDimensions(length); // 1
getDimensions(length, width); // 2
getDimensions(length, height); // 3
getDimensions(height, base); // 4
```

The `exit()` Function

- Terminates the execution of a program
- Can be called from any function
- Can pass an `int` value to operating system to indicate status of program termination
- Usually used for abnormal termination of program
- Requires `cstdlib` header file

The `exit()` Function

- Example: `exit(0);`
- The `cstdlib` header defines two constants that are commonly passed, to indicate success or failure:

```
exit(EXIT_SUCCESS);
exit(EXIT_FAILURE);
```

Exercises

- Challenging problems: 9, 10, 11, 22, 23.
- The least common multiple (LCM) of two integers a and b is the smallest positive integer that is divisible by both a and b. LCM has a close relationship with GCD through the following formula: $lcm(a, b) = (|a * b|)/gcd(a, b)$. Write a function to compute the LCM of any two numbers.
- Write the function to calculate the sum:

$$S = \frac{1}{2} + \frac{1}{2+4} + \ldots + \frac{1}{2+4+\ldots+(2^*n)}$$

- Write a function to calculate the value of the following expression:

$$S = \underbrace{\sqrt{x+\sqrt{x+\sqrt{x+\ldots+\sqrt{x+\sqrt{x}}}}}}_{n}$$