Tony Gaddis 5th Ed
Starting Out with C++

COMPUTER SCIENCE

CHAPTER 6
FUNCTIONS

Mohammad Al-Husseini

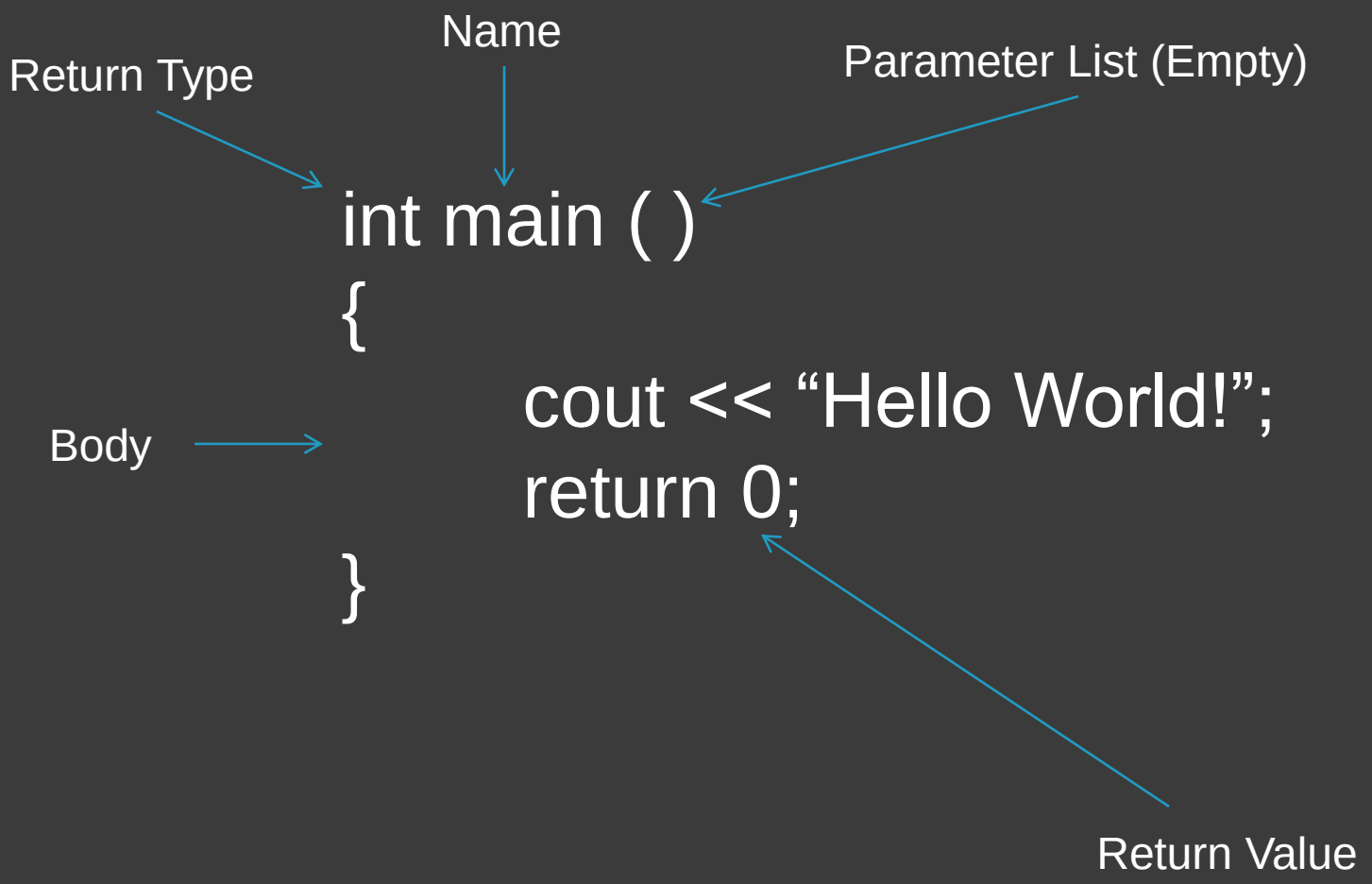
Modular Programming

- Functions can make our code:
  - Shorter
  - Better Organized
  - Easier to Read
  - Easier to Debug
  - More Reusable
  - More Efficient
  - And just generally better
- You've already used many functions
  - main, pow, strcmp, and more
  - But you will learn to create your own

Defining a Function

- You have to first define your function, before you can use it
- Function Definitions contain the following:
  - Return Type – The type of value that will be returned from the function (int, char, etc)
    - Void is a new data type (exclusive to functions) that indicates no value returned
  - Name – Programmer Defined and Meaningful
  - Parameter List – Data that will be sent into the function along with their data types
  - Body – Enclosed within braces, this is the code that the function will run when it is called

A Familiar Function



Calling a Function

- Calling means to execute/run the function
- When you define a function, you create the function header
  - void helloWorld ()
- When you call the function, you use its name and an argument list
  - helloWorld();
- Parameters are values that a function takes in
- Arguments are values that are given to a function
- When a function finishes executing, it returns to the point at which it was called in the code

Calling a Function

```
1  // This program has two functions: main and displayMessage
2  #include <iostream>
3  using namespace std;
4
5  //*****************************************
6  // Definition of function displayMessage  *
7  // This function displays a greeting.     *
8  //*****************************************
9
10 void displayMessage()
11 {
12    cout << "Hello from the function displayMessage.\n";
13 }
14
15 //*****************************************
16 // Function main                          *
17 //*****************************************
18
19 int main()
20 {
21    cout << "Hello from main.\n";
22    displayMessage();
23    cout << "Back in function main again.\n";
24    return 0;
25 }
```

Call Them Everywhere

```
1  // The function displayMessage is repeatedly called from a loop.
2  #include <iostream>
3  using namespace std;
4
5  //*****************************************
6  // Definition of function displayMessage  *
7  // This function displays a greeting.     *
8  //*****************************************
9
10 void displayMessage()
11 {
12    cout << "Hello from the function displayMessage.\n";
13 }
14
15 //*****************************************
16 // Function main                          *
17 //*****************************************
18
19 int main()
20 {
21    cout << "Hello from main.\n";
22    for (int count = 0; count < 5; count++)
23       displayMessage(); // Call displayMessage
24    cout << "Back in function main again.\n";
25    return 0;
26 }
```

Multiple Functions

```
1  // This program has three functions: main, first, and second.
2  #include <iostream>
3  using namespace std;
4
5  //*****************************************
6  // Definition of function first           *
7  // This function displays a message.      *
8  //*****************************************
9
10 void first()
11 {
12    cout << "I am now inside the function first.\n";
13 }
14
15 //*****************************************
16 // Definition of function second          *
17 // This function displays a message.      *
18 //*****************************************
19
20 void second()
21 {
22    cout << "I am now inside the function second.\n";
23 }
24
25 //*****************************************
26 // Function main                          *
27 //*****************************************
28
29 int main()
30 {
31    cout << "I am starting in function main.\n";
32    first();    // Call function first
33    second();   // Call function second
34    cout << "Back in function main again.\n";
35    return 0;
36 }
```

Call Them Everywhere

```
1  // This program has three functions: main, deep, and deeper
2  #include <iostream>
3  using namespace std;
4
5  //****************************************
6  // Definition of function deeper         *
7  // This function displays a message.     *
8  //****************************************
9
10 void deeper()
11 {
12    cout << "I am now inside the function deeper.\n";
13 }
14
15 //****************************************
16 // Definition of function deep           *
17 // This function displays a message.     *
18 //****************************************
19
20 void deep()
21 {
22    cout << "I am now inside the function deep.\n";
23    deeper();   // Call function deeper
24    cout << "Now I am back in deep.\n";
25 }
26
27 //****************************************
28 // Function main                         *
29 //****************************************
30
31 int main()
32 {
33    cout << "I am starting in function main.\n";
34    deep();     // Call function deep
35    cout << "Back in function main again.\n";
36    return 0;
37 }
```

Checkpoint

1. Is the following a function header or call?
   A. calcTotal();
   B. void showResults()
   C. helloWorld()
2. Assume the user enters 10 into the program on the next page. What will the program display?

Checkpoint

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

void func1()
{
        cout << “Able was I” << endl;
}

void func2()
{
        cout << “I saw Elba” << endl;
}

int main()
{
        int input;
        cout << “Enter a number: “;
        cin >> input;

        if (input < 10)
        {
                func1();
                func2();
        }
        else
        {
                func2();
                func1();
        }

        return 0;
}
```

Function Prototypes

- Also known as Function Declarations
- A function must be defined before it is called
- This requires you to put the entire function above function main
  - But this isn't a great solution and many programmers prefer to see main first
- Instead of defining the function first, you can place a prototype first instead
- This is just the header of the function
  - void displayMessage();
- The function must be defined later in the code

Function Prototypes

```
1  // This program has three functions: main, First, and Second.
2  #include <iostream>
3  using namespace std;
4
5  // Function Prototypes
6  void first();
7  void second();
8
9  int main()
10 {
11    cout << "I am starting in function main.\n";
12    first();    // Call function first
13    second();   // Call function second
14    cout << "Back in function main again.\n";
15    return 0;
16 }
17
18 //*************************************
19 // Definition of function first.      *
20 // This function displays a message.  *
21 //*************************************
22
23 void first()
24 {
25    cout << "I am now inside the function first.\n";
26 }
27
28 //*************************************
29 // Definition of function second.     *
30 // This function displays a message.  *
31 //*************************************
32
33 void second()
34 {
35    cout << "I am now inside the function second.\n";
36 }
```

Sending Data into a Function

- Known as passing arguments (or actual parameters) in the function call
- This is received by parameters (or formal parameters) in the function header
- void displayValue(int num)
  - int num is a parameter
- displayValue(5)
  - 5 is an argument
- The above function call causes the argument 5 to get stored in the parameter num
- They must match data types

Sending Data into a Function

```
1  // This program demonstrates a function with a parameter.
2  #include <iostream>
3  using namespace std;
4
5  // Function Prototype
6  void displayValue(int);
7
8  int main()
9  {
10    cout << "I am passing 5 to displayValue.\n";
11    displayValue(5);  // Call displayValue with argument 5
12    cout << "Now I am back in main.\n";
13    return 0;
14 }
15
16 //*********************************************************
17 // Definition of function displayValue.                   *
18 // It uses an integer parameter whose value is displayed. *
19 //*********************************************************
20
21 void displayValue(int num)
22 {
23    cout << "The value is " << num << endl;
24 }
```

Sending Data into a Function

```
1  // This program demonstrates a function with a parameter.
2  #include <iostream>
3  using namespace std;
4
5  // Function Prototype
6  void displayValue(int);
7
8  int main()
9  {
10    cout << "I am passing several values to displayValue.\n";
11    displayValue(5);  // Call displayValue with argument 5
12    displayValue(10); // Call displayValue with argument 10
13    displayValue(2);  // Call displayValue with argument 2
14    displayValue(16); // Call displayValue with argument 16
15    cout << "Now I am back in main.\n";
16    return 0;
17 }
18
19 //*************************************************************
20 // Definition of function displayValue.                       *
21 // It uses an integer parameter whose value is displayed.     *
22 //*************************************************************
23
24 void displayValue(int num)
25 {
26    cout << "The value is " << num << endl;
27 }
```

Sending Data into a Function

```
1  // This program demonstrates a function with three parameters.
2  #include <iostream>
3  using namespace std;
4
5  // Function Prototype
6  void showSum(int, int, int);
7
8  int main()
9  {
10    int value1, value2, value3;
11
12    // Get three integers.
13    cout << "Enter three integers and I will display ";
14    cout << "their sum: ";
15    cin >> value1 >> value2 >> value3;
16
17    // Call showSum passing three arguments.
18    showSum(value1, value2, value3);
19    return 0;
20 }
21
22 //*************************************************************
23 // Definition of function showSum.                            *
24 // It uses three integer parameters. Their sum is displayed.  *
25 //*************************************************************
26
27 void showSum(int num1, int num2, int num3)
28 {
29    cout << (num1 + num2 + num3) << endl;
30 }
```

Sending Data into a Function

- Notes:
  - Do not put the data type in the function call
    - displayValue(int x); // Wrong
  - In the function prototype, the variable name is optional
    - void showSum(int num1, int num2, int num3);
  - The prototype must list the data type for each parameter though
  - Arguments go into Parameters in the order in which they are placed (int n1, int n2) (4, 5)
  - Parameters, like all variables, have scope limited to the body of the function

Passing Data by Value

- By default, an argument is passed to a parameter by value
- This means the parameter becomes a copy of the argument
- Which means changing the parameter does not affect the argument
  - Later, you will learn how to create this behavior

Passing Data by Value

```
1  // This program demonstrates that changes to a function parameter
2  // have no affect on the original argument.
3  #include <iostream>
4  using namespace std;
5
6  // Function Prototype
7  void changeMe(int);
8
9  int main()
10 {
11    int number = 12;
12
13    // Display the value in number.
14    cout << "number is " << number << endl;
15
16    // Call changeMe, passing the value in number
17    // as an argument.
18    changeMe(number);
19
20    // Display the value in number again.
21    cout << "Now back in main again, the value of ";
22    cout << "number is " << number << endl;
23    return 0;
24 }
25
26 //*************************************************************
27 // Definition of function changeMe.                           *
28 // This function changes the value of the parameter myValue.  *
29 //*************************************************************
30
31 void changeMe(int myValue)
32 {
33    // Change the value of myValue to 0.
34    myValue = 0;
35
36    // Display the value in myValue.
37    cout << "Now the value is " << myValue << endl;
38 }
```

Menu Example

```
1  // This is a menu-driven program that makes a function call
2  // for each selection the user makes.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  // Function prototypes
8  void showMenu();
9  void showFees(double, int);
10
11 int main()
12 {
13    int choice;        // To hold a menu choice
14    int months;        // To hold a number of months
15
16    // Constants for membership rates
17    const double ADULT = 40.0;
18    const double SENIOR = 30.0;
19    const double CHILD = 20.0;
20
21    // Set up numeric output formatting.
22    cout << fixed << showpoint << setprecision(2);
23
24    do
25    {
26       // Display the menu and get the user's choice.
27       showMenu();
28       cin >> choice;
```

Menu Example

```
30          // Validate the menu selection.
31          while (choice < 1 || choice > 4)
32          {
33              cout << "Please enter 1, 2, 3, or 4: ";
34              cin >> choice;
35          }
36
37          if (choice != 4)
38          {
39              // Get the number of months.
40              cout << "For how many months? ";
41              cin >> months;
42
43              // Display the membership fees.
44              switch (choice)
45              {
46                  case 1:  showFees(ADULT, months);
47                           break;
48                  case 2:  showFees(CHILD, months);
49                           break;
50                  case 3:  showFees(SENIOR, months);
51              }
52          }
53      } while (choice != 4);
54      return 0;
55  }
```

Menu Example

```
57  //*************************************************************
58  // Definition of function showMenu which displays the menu.    *
59  //*************************************************************
60
61  void showMenu()
62  {
63     cout << "\n\t\tHealth Club Membership Menu\n\n";
64     cout << "1. Standard Adult Membership\n";
65     cout << "2. Child Membership\n";
66     cout << "3. Senior Citizen Membership\n";
67     cout << "4. Quit the Program\n\n";
68     cout << "Enter your choice: ";
69  }
70
71  //*************************************************************
72  // Definition of function showFees. The memberRate parameter    *
73  // the monthly membership rate and the months parameter holds the *
74  // number of months. The function displays the total charges.   *
75  //*************************************************************
76
77  void showFees(double memberRate, int months)
78  {
79      cout << "The total charges are $"
80           << (memberRate * months) << endl;
81  }
```

Checkpoint

3. Indicate whether the following are function prototypes, headers, or calls:
   A. void showNum(float num)
   B. void showNum(float num);
   C. void showNum(float);
   D. showNum(num)
   E. showNum(num);
   F. showNum(45.67);

Checkpoint

4. What is the output of the following program:

```
void showDouble(int);
int main()
{
    for (int num = 0; num < 10; num++)
        showDouble(num);
    return 0;
}
void showDouble(int value)
{
    cout << value << "\t" << (value * 2) << endl;
}
```

Checkpoint

5. What is the output of the following program:

```
void func1(double, int);
int main()
{
        int x = 0; double y = 1.5;
        cout << x << " " << y << endl;
        func1(y, x);
        cout << x << " " << y << endl;
        return 0;
}
void func1(double a, int b)
{
        cout << a << " " << b << endl;
        a = 0.0; b = 10;
        cout << a << " " << b << endl;
}
```

The return Statement

- Causes the function to end immediately
- You've utilized this behavior with main
  - That ends the program, but only because main is a special function
  - Other functions will return to the point at which they were called
- This can be used even if the function is void
  - Which means it doesn't return a value, but return still ends the function
  - In this case (void) the return is optional
  - Other types must include a return though

The return Statement

```
1  // This program uses a function to perform division. If division
2  // by zero is detected, the function returns.
3  #include <iostream>
4  using namespace std;
5
6  // Function prototype.
7  void divide(double, double);
8
9  int main()
10 {
11    double num1, num2;
12
13    cout << "Enter two numbers and I will divide the first\n";
14    cout << "number by the second number: ";
15    cin >> num1 >> num2;
16    divide(num1, num2);
17    return 0;
18 }
19
20 //*************************************************************
21 // Definition of function divide.                             *
22 // Uses two parameters: arg1 and arg2. The function divides arg1*
23 // by arg2 and shows the result. If arg2 is zero, however, the  *
24 // function returns.                                          *
25 //*************************************************************
26
27 void divide(double arg1, double arg2)
28 {
29    if (arg2 == 0.0)
30    {
31       cout << "Sorry, I cannot divide by zero.\n";
32       return;
33    }
34    cout << "The quotient is " << (arg1 / arg2) << endl;
35 }
```

Returning a Value

- The function must be of non-void type
- You must use a return statement
- You can only return a single value
  - Until we learn more in Chapter 11
- The value returned is a copy of the data in the function
  - This is because all data within a function is destroyed after it is finished executing
- The value replaces the call to the function in the code
- You could even use the return value as the argument to another function

Returning a Value

```
1  // This program uses a function that returns a value.
2  #include <iostream>
3  using namespace std;
4
5  // Function prototype
6  int sum(int, int);
7
8  int main()
9  {
10    int value1 = 20,    // The first value
11        value2 = 40,    // The second value
12        total;          // To hold the total
13
14    // Call the sum function, passing the contents of
15    // value1 and value2 as arguments. Assign the return
16    // value to the total variable.
17    total = sum(value1, value2);
18
19    // Display the sum of the values.
20    cout << "The sum of " << value1 << " and "
21         << value2 << " is " << total << endl;
22    return 0;
23 }
24
25 //*********************************************************
26 // Definition of function sum. This function returns     *
27 // the sum of its two parameters.                         *
28 //*********************************************************
29
30 int sum(int num1, int num2)
31 {
32    return num1 + num2;
33 }
```

Using a Return as an Argument

```
1  // This program demonstrates two value-returning functions.
2  // The square function is called in a mathematical statement.
3  #include <iostream>
4  #include <iomanip>
5  using namespace std;
6
7  //Function prototypes
8  double getRadius();
9  double square(double);
10
11 int main()
12 {
13    const double PI = 3.14159; // Constant for pi
14    double area;               // To hold the circle's area
15
16    // Set the numeric output formatting.
17    cout << fixed << showpoint << setprecision(2);
18
19    // Get the radius of the circle.
20    cout << "This program calculates the area of ";
21    cout << "a circle.\n";
22
23    // Caclulate the area of the circle.
24    area = PI * square(getRadius());
25
26    // Display the area.
27    cout << "The area is " << area << endl;
28    return 0;
29 }
```

Using a Return as an Argument

```
31  //*********************************************************
32  // Definition of function getRadius.                      *
33  // This function asks the user to enter the radius of     *
34  // the circle and then returns that number as a double.   *
35  //*********************************************************
36
37  double getRadius()
38  {
39     double rad;
40
41     cout << "Enter the radius of the circle: ";
42     cin >> rad;
43     return rad;
44  }
45
46  //*********************************************************
47  // Definition of function square.                         *
48  // This function accepts a double argument and returns    *
49  // the square of the argument as a double.                *
50  //*********************************************************
51
52  double square(double number)
53  {
54     return number * number;
55  }
```

Returning a Boolean Value

```
1  // This program uses a function that returns true or false.
2  #include <iostream>
3  using namespace std;
4
5  // Function prototype
6  bool isEven(int);
7
8  int main()
9  {
10    int val;
11
12    // Get a number from the user.
13    cout << "Enter an integer and I will tell you ";
14    cout << "if it is even or odd: ";
15    cin >> val;
16
17    // Indicate whether it is even or odd.
18    if (isEven(val))
19       cout << val << " is even.\n";
20    else
21       cout << val << " is odd.\n";
22    return 0;
23 }
24
25 //*************************************************************
26 // Definition of function isEven. This function accepts an    *
27 // integer argument and tests it to be even or odd. The function  *
28 // returns true if the argument is even or false if the argument  *
29 // is odd. The return value is an bool.                       *
30 //*************************************************************
31
32 bool isEven(int number)
33 {
34    bool status;
35
36    if (number % 2)
37       status = false;    // number is odd if there's a remainder.
38    else
39       status = true;     // Otherwise, the number is even.
40    return status;
41 }
```

Checkpoint

6. How many return values can a function have?
7. Write a header for a function named distance. The function should return a double and have two double parameters: rate and time.
8. Write a header for a function named days. The function should return an int and have three int parameters: years, months, and weeks.

Checkpoint

9. Write a header for a function named getKey. The function should return a char and use no parameters.
10. Write a header for a function named lightYears. The function should return a long and have one long parameter: miles.

Local Variables

- Variables which are inside or local to a function
- These are the variables you have been using so far
- As long as they are contained within different sets of braces, they can even share the same name

Local Variables

```
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 }
```

Local Variables

- Think of the int num in main as main::num and the int num in anotherFunction as anotherFunction::num
  - In this regard, they are different
- Recall that anotherFunction::num's lifetime is only as long as we are running anotherFunction
- Also think of parameters are local variables
  - int sum(int num1, int num2)
    {
    return num1 + num2;
    }

Global Variables

- Variables defined outside of any function
- The scope of these variables is the entire program
- Which means that all code can access and change the value of global variables
- You must use these only when absolutely necessary or (sometimes) when they are constant
  - You can run into a lot of problems if you aren't careful with your global usage
- Unlike local variables, they are automatically initialized to 0 if you do not initialize them

Global Variables

```
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 }
```

Global Variables

```
1  // This program calculates gross pay.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  // Global constants
7  const double PAY_RATE = 22.55;    // Hourly pay rate
8  const double BASE_HOURS = 40.0;   // Max non-overtime hours
9  const double OT_MULTIPLIER = 1.5; // Overtime multiplier
10
11 // Function prototypes
12 double getBasePay(double);
13 double getOvertimePay(double);
14
15 int main()
16 {
17    double hours,           // Hours worked
18           basePay,         // Base pay
19           overtime = 0.0,  // Overtime pay
20           totalPay;        // Total pay
21
22    // Get the number of hours worked.
23    cout << "How many hours did you work? ";
24    cin >> hours;
25
26    // Get the amount of base pay.
27    basePay = getBasePay(hours);
28
29    // Get overtime pay, if any.
30    if (hours > BASE_HOURS)
31       overtime = getOvertimePay(hours);
32
33    // Calculate the total pay.
34    totalPay = basePay + overtime;
35
36    // Set up numeric output formatting.
37    cout << setprecision(2) << fixed << showpoint;
38
39    // Display the pay.
40    cout << "Base pay: $" << basePay << endl;
41    cout << "Overtime pay $" << overtime << endl;
42    cout << "Total pay $" << totalPay << endl;
43    return 0;
44 }
```

Global Variables

```
46  //************************************************
47  // The getBasePay function accepts the number of *
48  // hours worked as an argument and returns the   *
49  // employee's pay for non-overtime hours.        *
50  //************************************************
51
52  double getBasePay(double hoursWorked)
53  {
54     double basePay; // To hold base pay
55
56     // Determine base pay.
57     if (hoursWorked > BASE_HOURS)
58        basePay = BASE_HOURS * PAY_RATE;
59     else
60        basePay = hoursWorked * PAY_RATE;
61
62     return basePay;
63  }
64
65  //************************************************
66  // The getOvertimePay function accepts the number *
67  // of hours worked as an argument and returns the *
68  // employee's overtime pay.                       *
69  //************************************************
70
71  double getOvertimePay(double hoursWorked)
72  {
73     double overtimePay; // To hold overtime pay
74
75     // Determine overtime pay.
76     if (hoursWorked > BASE_HOURS)
77     {
78        overtimePay = (hoursWorked - BASE_HOURS) *
79                      PAY_RATE * OT_MULTIPLIER;
80     }
81     else
82        overtimePay = 0.0;
83
84     return overtimePay;
85  }
```

Mixing Local and Global

- Two local variables in the same scope cannot share the same name
- But a local variable can share a name with a global variable
- When this happens, the local variable shadows the global
  - This means that while the local variable is in scope, it's value will be used instead of the global

Mixing Local and Global

```
1  // This program demonstrates how a local variable
2  // can shadow the name of a global constant.
3  #include <iostream>
4  using namespace std;
5
6  // Gobal constant.
7  const int BIRDS = 500;
8
9  // Function prototype
10 void california();
11
12 int main()
13 {
14    cout << "In main there are " << BIRDS
15         << " birds.\n";
16    california();
17    return 0;
18 }
19
20 //*********************************************
21 // california function                        *
22 //*********************************************
23
24 void california()
25 {
26    const int BIRDS = 10000;
27    cout << "In california there are " << BIRDS
28         << " birds.\n";
29 }
```

Static Local Variables

- Recall that once a function is finished executing, that all local variables are destroyed
- If you want to prevent this effect, you can declare the local variables as static
- Their initialization happens only the first time they are run
- And their values are maintained throughout the course of the program running

Static Local Variables

```
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 }
```

Static Local Variables

```
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 }
```

Static Local Variables

```
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 }
```

Checkpoint

11. What is the difference between a local variable, static local variable, and a global variable?

Checkpoint

12. What will the following display:

```
void myFunc()
{
        int var = 50;
        cout << var << endl;
}

int main()
{
        int var = 100;
        cout << var << endl;
        myFunc();
        cout << var << endl;
        return 0;
}
```

Checkpoint

13. What will the following display:

```
void showVar()
{
        static int var = 50;
        cout << var++ << endl;
}

int main()
{
        for (int count = 0; count < 10; count++)
                showVar();

        return 0;
}
```

Default Arguments

- Arguments which are passed to the parameters of a function if no arguments are provided (must be literal or constant)
- They are included in the function prototype, or in the function header if there is no prototype, but not both
  - Whichever occurs first
- All, some, or none of the parameter list may contain default arguments
  - If only some of the parameters have default values, they must the ones that appear last
    void calcPay(int empNum, double hours = 10);

Default Arguments

```
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 }
```

Reference Variables

- We said that arguments are passed by value and make a copy
- Reference variables are those that are passed by reference, which means they can be changed by the function
- It's just like using a regular parameter, but you place a & after the data type
- This is done in the function prototype and header, but not in the call
- This passes the memory address, not the value

Reference Variables

```
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 }
```

Reference Variables

```
1  // This program uses reference variables as function parameters.
2  #include <iostream>
3  using namespace std;
4
5  // Function prototypes. Both functions use reference variables
6  // as parameters.
7  void doubleNum(int &);
8  void getNum(int &);
9
10 int main()
11 {
12    int value;
13
14    // Get a number and store it in value.
15    getNum(value);
16
17    // Double the number stored in value.
18    doubleNum(value);
19
20    // Display the resulting number.
21    cout << "That value doubled is " << value << endl;
22    return 0;
23 }
```

Reference Variables

```
25  //*********************************************************
26  // Definition of getNum.                                  *
27  // The parameter userNum is a reference variable. The user is *
28  // asked to enter a number, which is stored in userNum.   *
29  //*********************************************************
30
31  void getNum(int &userNum)
32  {
33     cout << "Enter a number: ";
34     cin >> userNum;
35  }
36
37  //*********************************************************
38  // Definition of doubleNum.                               *
39  // The parameter refVar is a reference variable. The value *
40  // in refVar is doubled.                                  *
41  //*********************************************************
42
43  void doubleNum (int &refVar)
44  {
45     refVar *= 2;
46  }
```

Reference Variables

- Notes:
  - Only variables may be passed by reference
    - doubleNum(5) // Wrong
    - doubleNum(userNum + 10) // Wrong
  - Don’t get carried away with reference variables
    - You only want to use them when the function actually needs access to change the variable
    - You can run into hard to fix bugs if you aren’t careful

Checkpoint

14. What kinds of values may be specified as default arguments?
15. Write the prototype and header for a function called compute. The function should have three parameters: an int, a float, and a long (not necessarily in that order). The int parameter should have a default argument of 5, and the long parameter should have a default argument of 65536. The float parameter should not have a default argument.

Checkpoint

16. Write the prototype and header for a function called calculate. The function should have three parameters: an int, a reference to a float, and a long (not necessarily in that order). Only the int parameter should have a default argument, which is 47.

Checkpoint

17. What will the following display:

```
void test(int first = 2, int second = 4, int third = 6)
{
        first += 3;
        second += 6;
        third += 9;
        cout << first << " " << second << " "  << third << endl;
}
int main()
{
        test();
        test(6);
        test(3, 9);
        test(1, 5, 7);
        return 0;
}
```

Checkpoint

18. What will the following display:

```
void func1(int& a, int& b)
{
        cout << “Enter two numbers: “;
        cin >> a >> b;
}
void func2(int& a, int& b, int& c)
{
        b++; c--; a = b + c;
}
void func3(int a, int b, int c)
{
        a = b – c;
}
```

-continued on next page-

Checkpoint

```
int main()
{
        int x = 0, y = 0, z = 0;
        cout << x << " " << y << " " << z << endl;
        func1(x, y);
        cout << x << " " << y << " " << z << endl;
        func2(x, y, z);
        cout << x << " " << y << " " << z << endl;
        func3(x, y, z);
        cout << x << " " << y << " " << z << endl;
        return 0;
}
```

Overloading Functions

- Like local variables, functions cannot share the same name
- Unless their parameter lists are different
  - The number and/or data types must vary
- This can be extremely useful however, unlike with local variables
  - Certain math functions need to be modified based on what type of number it is (int, float, etc.)
  - Or you don't know how many parameters you'll need
    - int sum(int num1, int num2, int num3, …);
  - But you just want one function call that "works"

Overloading Functions

```
1  // This program uses overloaded functions.
2  #include <iostream>
3  #include <iomanip>
4  using namespace std;
5
6  // Function prototypes
7  int square(int);
8  double square(double);
9
10 int main()
11 {
12    int userInt;
13    double userFloat;
14
15    // Get an int and a double.
16    cout << fixed << showpoint << setprecision(2);
17    cout << "Enter an integer and a floating-point value: ";
18    cin >> userInt >> userFloat;
19
20    // Display their squares.
21    cout << "Here are their squares: ";
22    cout << square(userInt) << " and " << square(userFloat);
23    return 0;
24 }
```

Overloading Functions

```
26  //*************************************************************
27  // Definition of overloaded function square.                  *
28  // This function uses an int parameter, number. It returns the *
29  // square of number as an int.                                *
30  //*************************************************************
31
32  int square(int number)
33  {
34      return number * number;
35  }
36
37  //*************************************************************
38  // Definition of overloaded function square.                  *
39  // This function uses a double parameter, number. It returns  *
40  // the square of number as a double.                          *
41  //*************************************************************
42
43  double square(double number)
44  {
45      return number * number;
46  }
```

Overloading Functions

```
// This program demonstrates overloaded functions to calculate
// the gross weekly pay of hourly-paid or salaried employees.
#include <iostream>
#include <iomanip>
using namespace std;

// Function prototypes
void getChoice(char &);
double calcWeeklyPay(int, double);
double calcWeeklyPay(double);

int main()
{
   char selection;   // Menu selection
   int worked;       // Hours worked
   double rate;      // Hourly pay rate
   double yearly;    // Yearly salary

   // Set numeric output formatting.
   cout << fixed << showpoint << setprecision(2);

   // Display the menu and get a selection.
   cout << "Do you want to calculate the weekly pay of\n";
   cout << "(H) an hourly-paid employee, or \n";
   cout << "(S) a salaried employee?\n";
   getChoice(selection);

   // Process the menu selection.
   switch (selection)
   {
      // Hourly-paid employee
      case 'H' :
      case 'h' :  cout << "How many hours were worked? ";
                  cin >> worked;
                  cout << "What is the hour pay rate? ";
                  cin >> rate;
                  cout << "The gross weekly pay is $";
                  cout << calcWeeklyPay(worked, rate) << endl;
                  break;

      // Salaried employee
      case 'S' :
      case 's' :  cout << "What is the annual salary? ";
                  cin >> yearly;
                  cout << "The gross weekly pay is $";
                  cout << calcWeeklyPay(yearly) << endl;
                  break;
   }
   return 0;
}
```

Overloading Functions

```
52  //*****************************************************************
53  // Definition of function getChoice.                              *
54  // The parameter letter is a reference to a char.                 *
55  // This function asks the user for an H or an S and returns       *
56  // the validated input.                                           *
57  //*****************************************************************
58
59  void getChoice(char &letter)
60  {
61     // Get the user's selection.
62     cout << "Enter your choice (H or S): ";
63     cin >> letter;
64
65     // Validate the selection.
66     while (letter != 'H' && letter != 'h' &&
67            letter != 'S' && letter != 's')
68     {
69        cout << "Please enter H or S: ";
70        cin >> letter;
71     }
72  }
73
74  //*****************************************************************
75  // Definition of overloaded function calcWeeklyPay.               *
76  // This function calculates the gross weekly pay of               *
77  // an hourly-paid employee. The parameter hours holds the         *
78  // number of hours worked. The parameter payRate holds the        *
79  // hourly pay rate. The function returns the weekly salary.       *
80  //*****************************************************************
81
82  double calcWeeklyPay(int hours, double payRate)
83  {
84     return hours * payRate;
85  }
86
87  //*****************************************************************
88  // Definition of overloaded function calcWeeklyPay.               *
89  // This function calculates the gross weekly pay of               *
90  // a salaried employee. The parameter holds the employee's        *
91  // annual salary. The function returns the weekly salary.         *
92  //*****************************************************************
93
94  double calcWeeklyPay(double annSalary)
95  {
96     return annSalary / 52;
97  }
```

The exit() Function

- Return causes the function to stop executing
  - Which in the case of main causes the program to end
- But sometimes you want this functionality within a function, exit() does just that
- Requires #include <cstdlib>
- You can pass it an integer argument to indicate an exit code
  - Usually 0 or EXIT_SUCCESS
  - Or some negative value
  - But this is only needed if your program is tested outside of main

The exit() Function

```
1  // This program shows how the exit function causes a program
2  // to stop executing.
3  #include <iostream>
4  #include <cstdlib>   // For exit
5  using namespace std;
6
7  void function();  // Function prototype
8
9  int main()
10 {
11    function();
12    return 0;
13 }
14
15 //*********************************************************
16 // This function simply demonstrates that exit can be used  *
17 // to terminate a program from a function other than main.  *
18 //*********************************************************
19
20 void function()
21 {
22    cout << "This program terminates with the exit function.\n";
23    cout << "Bye!\n";
24    exit(0);
25    cout << "This message will never be displayed\n";
26    cout << "because the program has already terminated.\n";
27 }
```

Checkpoint

19. What will the following output:

```
void showVals(double, p1, double p2)
{
        cout << p1 << endl;
        exit();
        cout << p2 << endl;
}
int main()
{
        double x = 1.2, y = 4.5;
        showVals(x, y);
        return 0;
}
```

Checkpoint

20. What will the following output:

```
int manip(int val)
{
        return val + val * 2;
}
int manip(int val1, int val2)
{
        return (val1 + val2) * 2;
}
int manip(int val1, double val2)
{
        return val1 * static_cast<int>(val2);
}
```

-continued on next page-

Checkpoint

```
int main()
{
        int x = 2, y = 4, z;
        double a = 3.1;

        z = manip(x) + manip(x, y) + manip(y, a);

        cout << z << endl;

        return 0;
}
```

Stubs and Drivers

- Tools for testing your code in a large or team environment
- Stubs are dummy functions used to test your main
- Drivers are dummy mains used to test your functions
- This is useful if you want to focus on finishing your main before writing your functions or vice versa
- Or if a member of your team is writing the function and you need to test your main or vice versa

Stubs and Drivers

```
1  // This program is a driver for testing the showFees function.
2  #include <iostream>
3  using namespace std;
4
5  // Prototype
6  void showFees(double, int);
7
8  int main()
9  {
10    // Constants for membership rates
11    const double ADULT = 40.0;
12    const double SENIOR = 30.0;
13    const double CHILD = 20.0;
14
15    // Perform a test for adult membership.
16    cout << "Testing an adult membership...\n"
17         << "Calling the showFees function with arguments "
18         << ADULT << " and 10.\n";
19    showFees(ADULT, 10);
20
21    // Perform a test for senior citizen membership.
22    cout << "\nTesting a senior citizen membership...\n"
23         << "Calling the showFees function with arguments "
24         << SENIOR << " and 10.\n";
25    showFees(SENIOR, 10);
26
27    // Perform a test for child membership.
28    cout << "\nTesting a child membership...\n"
29         << "\nCalling the showFees function with arguments "
30         << CHILD << " and 10.\n";
31    showFees(CHILD, 10);
32    return 0;
33 }
34
35 //*********************************************************************
36 // Definition of function showFees. The memberRate parameter      *
37 // the monthly membership rate and the months parameter holds the *
38 // number of months. The function displays the total charges.     *
39 //*********************************************************************
40
41 void showFees(double memberRate, int months)
42 {
43     cout << "The total charges are $"
44          << (memberRate * months) << endl;
45 }
```