Chapter 6 Functions

6.1 Focus on Software Engineering: Modular Programming

Concept:

A program may be broken up into manageable functions.

- A **function** is a set of statements designed to perform a specific task.
- You've already used the `main` function and library functions like `pow`.
- Breaking a large problem into smaller, manageable functions is a strategy known as **divide and conquer**. This is called modular programming.
- Functions also promote **code reuse**, allowing you to write a task's code once and call it multiple times.

6.2 Defining and Calling Functions

Concept:

A function call is a statement that causes a function to execute. A function definition contains the statements that make up the function.

- A function **definition** consists of several parts:

Return type:	The data type of the value the function sends back.
Name:	A descriptive name for the function.
Parameter list:	A list of variables to hold values passed into the function.
Body:	The statements that perform the function's task, enclosed in braces `{}`.

- The first line of a function definition, like `int main()`, is the **function header**.

`void` Functions

- Functions that do not return a value are called `void` **functions**.
- They use `void` as their return type and do not require a `return` statement.

```
void displayMessage()
{
   cout << "Hello from the function displayMessage.\n";
}
```

Calling a Function

- A function is executed when it is **called**.
- A function call statement causes the program to branch to the function, execute its body, and then return to the point of the call.
- The compiler must know a function's details before it is called, so the function definition should appear before any calls to it.

Program 6-1

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


void displayMessage()
{
    cout << "Hello from the function displayMessage.\n";
}


int main()
{
    cout << "Hello from main.\n";
    displayMessage();
    cout << "Back in function main again.\n";
    return 0;
}
```

Program Output

- A function call is a statement and must end with a semicolon, unlike a function header.

Note:

Later in this chapter, you will see how data can be passed into a function by being listed inside the parentheses.

Note:

You should always document your functions by writing comments that describe what they do. These comments should appear just before the function definition.

- Function calls can be placed inside control structures like loops.

Program 6-2

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


void displayMessage()
{
    cout << "Hello from the function displayMessage.\n";
}


int main()
{
    cout << "Hello from main.\n";
    for (int count = 0; count < 5; count++)
        displayMessage();
    cout << "Back in function main again.\n";
    return 0;
}
```

Program Output

- A program can contain many functions that are called in sequence.

Program 6-3

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


void first()
{
    cout << "I am now inside the function first.\n";
}


void second()
{
    cout << "I am now inside the function second.\n";
}


int main()
{
    cout << "I am starting in function main.\n";
    first();
    second();
    cout << "Back in function main again. \n";
    return 0;
}
```

Program Output

- Functions can also be called in a layered or hierarchical fashion, where one function calls another.

Program 6-4

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


void deeper()
{
    cout << "I am now inside the function deeper.\n";
}


void deep()
{
    cout << "I am now inside the function deep.\n";
    deeper();
    cout << "Now I am back in deep.\n";
}


int main()
{
    cout << "I am starting in function main.\n";
    deep();
    cout << "Back in function main again.\n";
    return 0;
}
```

Program Output

Checkpoint

6.1 Is the following a function header or a function call?

```
calcTotal();
```

6.2 Is the following a function header or a function call?

```
void showResults()
```

6.3 What will the output of the following program be if the user enters 10?

```
#include <iostream>
using namespace std;
void func1()
{
   cout << "Able was I\n";
}
void func2()
{
   cout << "I saw Elba\n";
}
int main()
{
   int input;
   cout << "Enter a number: ";
   cin >> input;
   if (input < 10)
   {
      func1();
      func2();
   }
   else
   {
      func2();
      func1();
   }
   return 0;
}
```

6.4 The following program skeleton determines whether a person qualifies for a credit card. To qualify, the person must have worked on his or her current job for at least 2 years, and make at least $17,000 per year. Finish the program by writing the definitions of the functions `qualify` and `noQualify`. The function `qualify` should explain that the applicant qualifies for the card, and that the annual interest rate is 12 percent. The function `noQualify` should explain that the applicant does not qualify for the card and give a general explanation why.

```cpp
#include <iostream>
using namespace std;
int main()
{
   double salary;
   int years;
   cout << "This program will determine if you qualify\n";
   cout << "for our credit card.\n";
   cout << "What is your annual salary? ";
   cin >> salary;
   cout << "How many years have you worked at your ";
   cout << "current job? ";
   cin >> years;
   if (salary >= 17000.0 && years >= 2)
      qualify();
   else
      noQualify();
   return 0;
}
```

6.3 Function Prototypes

Concept:

A function prototype eliminates the need to place a function definition before all calls to the function.

- A **function prototype** (or function declaration) tells the compiler a function's name, return type, and parameter details before it sees the function's definition.
- This allows you to place function definitions in any order, such as placing `main` first.
- A prototype looks like a function header but ends with a semicolon.

```cpp
void displayMessage();
```

Note:

Function prototypes are also known as *function declarations*.

Warning!

You must place either the function definition or the function prototype ahead of all calls to the function. Otherwise, the program will not compile.

- Prototypes are typically placed at the top of the program.

Program 6-5

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

void first();
void second();

int main()
{
    cout << "I am starting in function main.\n";
    first();
    second();
    cout << "Back in function main again.\n";
    return 0;
}


void first()
{
    cout << "I am now inside the function first.\n";
}


void second()
{
    cout << "I am now inside the function second.\n";
}
```

Program Output

(The program's output is the same as the output of Program 6-3.)

Note:

Although some programmers make `main` the last function in the program, many prefer it to be first because it is the program's starting point.

6.4 Sending Data into a Function

Concept:

When a function is called, the program may send values into the function.

Functions and Arguments

- **Arguments** are values sent *into* a function when it is called.
- A **parameter** is a special variable in a function's definition that holds the value of an argument.

```
void displayValue(int num)
{
   cout << "The value is " << num << endl;
}
```

Note:

In this text, the values that are passed into a function are called arguments, and the variables that receive those values are called parameters. There are several variations of these terms in use. Some call the arguments *actual parameters* and call the parameters *formal parameters*. Others use the terms *actual argument* and *formal argument*. Regardless of which set of terms you use, it is important to be consistent.

Program 6-6

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

void displayValue(int);

int main()
{
    cout << "I am passing 5 to displayValue.\n";
    displayValue(5);
    cout << "Now I am back in main.\n";
    return 0;
}


void displayValue(int num)
```

```
{
    cout << "The value is " << num << endl;
}
```

Program Output

- A function prototype only needs the data types of the parameters, not their names.
- When calling a function, the argument's value is copied into the corresponding parameter variable.
- A function can be called multiple times with different arguments.

Program 6-7

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

void displayValue(int);

int main()
{
    cout << "I am passing several values to displayValue.\n";
    displayValue(5);
    displayValue(10);
    displayValue(2);
    displayValue(16);
    cout << "Now I am back in main.\n";
    return 0;
}


void displayValue(int num)
{
    cout << "The value is " << num << endl;
}
```

Program Output

Warning!

When passing a variable as an argument, simply write the variable name inside the parentheses of the function call. Do not write the data type of the argument variable in the function call. For example, the following function call will cause an error:

```
displayValue(int x);
```

The function call should appear as

```
displayValue(x);
```

- A function can have multiple parameters, separated by commas in the function header and prototype.
- When calling a function with multiple parameters, the arguments are passed to the parameters in order.

Program 6-8

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

void showSum(int, int, int);

int main()
{
    int value1, value2, value3;

    cout << "Enter three integers and I will display ";
    cout << "their sum: ";
    cin >> value1 >> value2 >> value3;

    showSum(value1, value2, value3);
    return 0;
}
```

```


void showSum(int num1, int num2, int num3)
{
    cout << (num1 + num2 + num3) << endl;
}
```

Program Output

Warning!

Each parameter variable in a parameter list must have a data type listed before its name. For example, a compiler error would occur if the parameter list for the `showSum` function were defined as shown in the following header:

```
void showSum(int num1, num2, num3)
```

A data type for all three of the parameter variables must be listed, as shown here:

```
void showSum(int num1, int num2, int num3)
```

Note:

The function prototype must list the data type of each parameter.

Note:

Like all variables, parameters have a scope. The scope of a parameter is limited to the body of the function that uses it.

6.5 Passing Data by Value

Concept:

When an argument is passed into a parameter, only a copy of the argument's value is passed. Changes to the parameter do not affect the original argument.

- When an argument is **passed by value**, the function receives only a copy of the argument's value.
- The function cannot access or modify the original argument variable.
- Any changes made to the parameter inside the function are lost when the function terminates.

Program 6-9

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

void changeMe(int);

int main()
{
    int number = 12;

    cout << "number is " << number << endl;

    changeMe(number);

    cout << "Now back in main again, the value of ";
    cout << "number is " << number << endl;
    return 0;
}


void changeMe(int myValue)
{
    myValue = 0;

    cout << "Now the value is " << myValue << endl;
}
```

Program Output

Note:

Later in this chapter, you will learn ways to give a function access to its original arguments.

6.6 Focus on Software Engineering: Using Functions in a Menu-Driven Program

Concept:

Functions are ideal for use in menu-driven programs. When the user selects an item from a menu, the program can call the appropriate function.

- A **modular program** uses functions to perform specific tasks, making it organized and easy to manage.
- In a menu-driven program, each menu choice can correspond to a call to a specific function that handles that choice.

Program 6-10

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

void showMenu();
void showFees(double, int);

int main()
{
    int choice;
    int months;

    const int ADULT_CHOICE = 1,
              CHILD_CHOICE = 2,
              SENIOR_CHOICE = 3,
              QUIT_CHOICE = 4;

    const double ADULT = 40.0,
                 CHILD = 20.0;
                 SENIOR = 30.0,

    cout << fixed << showpoint << setprecision(2);

    do
    {
        showMenu();
        cin >> choice;

        while (choice < ADULT_CHOICE || choice > QUIT_CHOICE)
        {
            cout << "Please enter a valid menu choice: ";
            cin >> choice;
        }

        if (choice != QUIT_CHOICE)
        {
            cout << "For how many months? ";
            cin >> months;

            switch (choice)
            {
                case ADULT_CHOICE:
                    showFees(ADULT, months);
                    break;
                case CHILD_CHOICE:
                    showFees(CHILD, months);
                    break;
                case SENIOR_CHOICE:
                    showFees(SENIOR, months);
            }
        }
    } while (choice != QUIT_CHOICE);
    return 0;
}


void showMenu()
{
    cout << "\n\t\tHealth Club Membership Menu\n\n"
        << "1. Standard Adult Membership\n"
        << "2. Child Membership\n"
        << "3. Senior Citizen Membership\n"
        << "4. Quit the Program\n\n"
        << "Enter your choice: ";
}


```

```
void showFees(double memberRate, int months)
{
    cout  << "The total charges are $"
         << (memberRate * months) << endl;
}
```

Program Output

Checkpoint

6.5 Indicate which of the following is the function prototype, the function header, and the function call:

```
void showNum(double num)
void showNum(double);
showNum(45.67);
```

6.6 Write a function named `timesTen`. The function should have an integer parameter named `number`. When `timesTen` is called, it should display the product of `number` times ten. (*Note*: Just write the function. Do not write a complete program.)

6.7 Write a function prototype for the `timesTen` function you wrote in Question 6.6.

6.8 What is the output of the following program?

```
#include <iostream>
using namespace std;
void showDouble(int);
```

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

6.9 What is the output of the following program?

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

6.10 The following program skeleton asks for the number of hours you've worked and your hourly pay rate. It then calculates and displays your wages. The function `showDollars`, which you are to write, formats the output of the wages.

```
#include <iostream>
using namespace std;
void showDollars(double);
int main()
{
   double payRate, hoursWorked, wages;
```

```
    cout << "How many hours have you worked? "
    cin >> hoursWorked;
    cout << "What is your hourly pay rate? ";
    cin >> payRate;
    wages = hoursWorked * payRate;
    showDollars(wages);
    return 0;
}
```

6.7 The `return` Statement

Concept:

The `return` statement causes a function to end immediately.

- When a `return` statement is encountered in a function, the function terminates immediately.
- Control of the program returns to the statement that followed the function call.
- This is useful for exiting a function early, such as when an error condition is detected.

Program 6-11

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

void divide(double, double);

int main()
{
    double num1, num2;

    cout << "Enter two numbers and I will divide the first\n";
    cout << "number by the second number: ";
    cin >> num1 >> num2;
    divide(num1, num2);
    return 0;
}


void divide(double arg1, double arg2)
{
    if (arg2 == 0.0)
```

```
        {
           cout << "Sorry, I cannot divide by zero.\n";
           return;
        }
        cout << "The quotient is " << (arg1 / arg2) << endl;
    }
```

Program Output

6.8 Returning a Value from a Function

Concept:

A function may send a value back to the part of the program that called the function.

- **Value-returning functions** send a value back to the part of the program that called them.
- A function can accept multiple arguments but can only return a single value.

Value-Returning Functions

Note:

It is possible to return multiple values from a function, but they must be "packaged" in such a way that they are treated as a single value. This will be a topic in Chapter 11.

Defining a Value-Returning Function

- To define a value-returning function, you specify its return type (e.g., `int`, `double`) in the function header.
- The function must use a `return` statement to send a value back. The format is `return expression;`.
- Variables defined inside a function are called **local variables**.

```
int sum(int num1, int num2)
{
    return num1 + num2;
}
```

Calling a Value-Returning Function

- The value returned by a function can be used in various ways:
  - Assigned to a variable.
  - Used in a mathematical expression.
  - Passed to another function, such as `cout`.

Program 6-12

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

int sum(int, int);

int main()
{
    int value1  = 20,
        value2 = 40,
        total;

    total = sum(value1, value2);

    cout << "The sum of " << value1 << " and "
         << value2 << " is " << total << endl;
    return 0;
}


int sum(int num1, int num2)
{
    return num1 + num2;
}
```

Program Output

Program 6-13

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

double getRadius();
double square(double);

int main()
{
    const double PI = 3.14159;
    double radius;
    double area;

    cout << fixed << showpoint << setprecision(2);

    cout << "This program calculates the area of ";
    cout << "a circle.\n";
    radius = getRadius();

    area = PI * square(radius);

    cout << "The area is " << area << endl;
    return 0;
}


double getRadius()
{
    double rad;

    cout << "Enter the radius of the circle: ";
    cin >> rad;
    return rad;
}


double square(double number)
{
    return number * number;
}
```

Program Output

In the Spotlight:

Using Functions

Your friend Michael runs a catering company. Some of the ingredients that his recipes require are measured in cups. When he goes to the grocery store to buy those ingredients, however, they are sold only by the fluid ounce. He has asked you to write a simple program that converts cups to fluid ounces.

You design the following algorithm:

1. *Display an introductory screen that explains what the program does.*
2. *Get the number of cups.*
3. *Convert the number of cups to fluid ounces and display the result.*

This algorithm lists the top level of tasks that the program needs to perform and becomes the basis of the program's `main` function. The hierarchy chart shown in Figure 6-13 shows how the program will be broken down into functions.

As shown in the hierarchy chart, the `main` function will call three other functions. Here are summaries of those functions:

- `showIntro`—This function will display a message on the screen that explains what the program does.
- `getCups`—This function will prompt the user to enter the number of cups, then will return that value as a `double`.
- `cupsToOunces`—This function will accept the number of cups as an argument, then return an equivalent number of fluid ounces as a `double`.

Program 6-14

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

void showIntro();
```

```cpp
double getCups();
double cupsToOunces(double);

int main()
{
    double cups, ounces;

    cout << fixed << showpoint << setprecision(1);

    showIntro();

    cups = getCups();

    ounces = cupsToOunces(cups);

    cout  << cups << " cups equals "
         << ounces << " ounces.\n";

    return 0;
}


void showIntro()
{
     cout << "This program converts measurements\n"
         << "in cups to fluid ounces. For your\n"
         << "reference the formula is:\n"
         << " 1 cup = 8 fluid ounces\n\n";
}


double getCups()
{
    double numCups;

    cout << "Enter the number of cups: ";
    cin >> numCups;
    return numCups;
}


double cupsToOunces(double numCups)
{
    return numCups * 8.0;
}
```

Program Output

6.9 Returning a Boolean Value

Concept:

Functions may return `true` or `false` values.

- Functions can be written to return a `bool` value (`true` or `false`).
- These are useful for testing a condition and indicating whether it exists.
- The return value can be used directly in control structures like `if/else` statements.

```
bool isValid(int number)
{
   bool status;
   if (number >= 1 && number <= 100)
      status = true;
   else
      status = false;
   return status;
}
```

```
int value = 20;
if (isValid(value))
   cout << "The value is within range.\n";
else
   cout << "The value is out of range.\n";
```

Program 6-15

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

bool isEven(int);

int main()
{
    int val;

    cout << "Enter an integer and I will tell you ";
    cout << "if it is even or odd: ";
    cin >> val;

    if (isEven(val))
       cout << val << " is even.\n";
    else
       cout << val << " is odd.\n";
    return 0;
}


bool isEven(int number)
{
    bool status;

    if (number % 2 == 0)
       status = true;
    else
       status = false;
    return status;
}
```

Program Output

Checkpoint

6.11 How many return values may a function have?

6.12 Write a header for a function named `distance`. The function should return a `double` and have two `double` parameters: `rate` and `time`.

6.13 Write a header for a function named `days`. The function should return an `int` and have three `int` parameters: `years`, `months`, and `weeks`.

6.14 Write a header for a function named `getKey`. The function should return a `char` and use no parameters.

6.15 Write a header for a function named `lightYears`. The function should return a `long` and have one `long` parameter: `miles`.

6.10 Local and Global Variables

Concept:

A local variable is defined inside a function and is not accessible outside the function. A global variable is defined outside all functions and is accessible to all functions in its scope.

Local Variables

- Variables defined inside a function are **local** to that function.
- They are hidden from other functions, which cannot access them.
- Different functions can have distinct local variables with the same name.

Program 6-16

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

void anotherFunction();

int main()
{
    int num = 1;

    cout << "In main, num is " << num << endl;
    anotherFunction();
    cout << "Back in main, num is " << num << endl;
    return 0;
}


void anotherFunction()
```

```
{
    int num = 20;

    cout << "In anotherFunction, num is " << num << endl;
}
```

Program Output

Local Variable Lifetime

- A local variable’s **lifetime** is the time the function is executing.
- Local variables are created when the function begins and destroyed when it ends.
- Values stored in local variables are lost between function calls.

Initializing Local Variables with Parameter Values

- It is possible to use a parameter’s value to initialize a local variable within the same function.

```
int sum(int num1, int num2)
{
   int result = num1 + num2;
   return result;
}
```

Global Variables

- A **global variable** is defined outside of all functions.
- Its scope runs from its definition to the end of the program.
- It can be accessed by any function that is defined after it.

Program 6-17

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

void anotherFunction();
int num = 2;

int main()
{
    cout << "In main, num is " << num << endl;
    anotherFunction();
    cout << "Back in main, num is " << num << endl;
    return 0;
}


void anotherFunction()
{
    cout << "In anotherFunction, num is " << num << endl;
    num = 50;
    cout << "But, it is now changed to " << num << endl;
}
```

Program Output

- Uninitialized numeric global variables are automatically initialized to zero.

Program 6-18

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

int globalNum;

int main()
{
    cout << "globalNum is " << globalNum << endl;
    return 0;
}
```

Program Output

- You should avoid using global variables because they:

    - Make debugging difficult, as any function can modify them.
    - Make functions less portable and dependent on external variables.
    - Make programs harder to understand.
- It is better to use local variables and pass them as arguments.

Global Constants

- It is generally acceptable to use **global constants**.
- Since their value cannot be changed, they do not present the same dangers as global variables.
- They are useful for representing unchanging values needed throughout a program, which simplifies maintenance.

Program 6-19

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

const double PAY_RATE = 22.55;
const double BASE_HOURS = 40.0;
const double OT_MULTIPLIER = 1.5;

double getBasePay(double);
double getOvertimePay(double);

int main()
{
    double hours,
           basePay,
           overtime = 0.0,
           totalPay;

    cout << "How many hours did you work? ";
    cin >> hours;

    basePay = getBasePay(hours);

    if (hours > BASE_HOURS)
       overtime = getOvertimePay(hours);

    totalPay = basePay + overtime;

    cout << setprecision(2) << fixed << showpoint;

    cout << "Base pay: $" << basePay << endl
           << "Overtime pay $" << overtime << endl
           << "Total pay $" << totalPay << endl;
    return 0;
}


double getBasePay(double hoursWorked)
{
    double basePay;

    if (hoursWorked > BASE_HOURS)
       basePay = BASE_HOURS * PAY_RATE;
    else
       basePay = hoursWorked * PAY_RATE;

    return basePay;
}


double getOvertimePay(double hoursWorked)
{
    double overtimePay;

    if (hoursWorked > BASE_HOURS)
    {
       overtimePay = (hoursWorked 2 BASE_HOURS) *
               PAY_RATE * OT_MULTIPLIER;
    }
    else
       overtimePay = 0.0;

    return overtimePay;
}
```

Program Output

Local and Global Variables with the Same Name

- A local or parameter variable can have the same name as a global variable.
- When this happens, the local variable's name **shadows** the global variable's name.
- Inside the function, the name refers to the local variable, not the global one.

Program 6-20

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

const int BIRDS = 500;

void california();

int main()
{
    cout  << "In main there are " << BIRDS
          << " birds.\n";
    california();
    return 0;
}


void california()
{
    const int BIRDS = 10000;
    cout << "In california there are " << BIRDS
          << " birds.\n";
}
```

Program Output

6.11 Static Local Variables

- Normally, a function's local variables are destroyed when the function returns and do not retain their values between calls.

🗐 **Program 6-21**

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

void showLocal();

int main()
{
    showLocal();
    showLocal();
    return 0;
}


void showLocal()
{
    int localNum = 5;

    cout << "localNum is " << localNum << endl;
    localNum = 99;
}
```

💻 **Program Output**

- A **static local variable** is not destroyed when a function returns.
- It exists for the entire lifetime of the program, retaining its value between function calls.
- Static local variables are initialized to zero by default if not explicitly initialized.

🗐 **Program 6-22**

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

void showStatic();

int main()
{
    for (int count = 0; count < 5; count++)
```

```
        showStatic();
    return 0;
}


void showStatic()
{
    static int statNum;

    cout << "statNum is " << statNum << endl;
    statNum++;
}
```

Program Output

- If you provide an initialization value for a static local variable, the initialization only happens once.

Program 6-23

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

void showStatic();

int main()
{
    for (int count = 0; count < 5; count++)
        showStatic();
    return 0;
}


void showStatic()
{
    static int statNum = 5;

    cout << "statNum is " << statNum << endl;
```

```
        statNum++;
    }
```

Program Output

Checkpoint

6.16 What is the difference between a static local variable and a global variable?

6.17 What is the output of the following program?

```
#include <iostream>
using namespace std;
void myFunc();
int main()
{
   int var = 100;
   cout << var << endl;
   myFunc();
   cout << var << endl;
   return 0;
}
void myFunc()
{
   int var = 50;
   cout << var << endl;
}
```

6.18 What is the output of the following program?

```
#include <iostream>
using namespace std;
void showVar();
int main()
{
   for (int count = 0; count < 10; count++)
      showVar();
```

```
    return 0;
}
void showVar()
{
    static int var = 10;
    cout << var << endl;
    var++;
}
```

6.12 Default Arguments

Concept:

Default arguments are passed to parameters automatically if no argument is provided in the function call.

- A **default argument** is a value that is automatically passed to a parameter if an argument is not provided in the function call.
- Default arguments are typically specified in the function prototype.

```
void showArea(double length = 20.0, double width = 10.0);
```

- When calling the function, you can omit arguments that have defaults. The provided arguments fill the parameters from left to right.

```
showArea();
showArea(12.0);
showArea(12.0, 5.5);
```

Note:

If a function does not have a prototype, default arguments may be specified in the function header. The `showArea` function could be defined as follows:

```
void showArea(double length = 20.0, double width = 10.0)
{
    double area = length * width;
    cout << "The area is " << area << endl;
}
```

Warning!

A function’s default arguments should be assigned in the earliest occurrence of the function name. This will usually be the function prototype.

Program 6-24

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

void displayStars(int = 10, int = 1);

int main()
{
    displayStars();
    cout << endl;
    displayStars(5);
    cout << endl;
    displayStars(7, 3);
    return 0;
}


void displayStars(int cols, int rows)
{
    for (int down = 0; down < rows; down++)
    {
        for (int across = 0; across < cols; across++)
            cout << "*";
        cout << endl;
    }
}
```

Program Output

- Important rules for default arguments:
    - The default value must be a literal or a named constant.

- When an argument is left out, all subsequent arguments must also be left out.
- Parameters with default arguments must be declared last in the parameter list.

6.13 Using Reference Variables as Parameters

Concept:

When used as parameters, reference variables allow a function to access the parameter's original argument. Changes to the parameter are also made to the argument.

- A **reference variable** acts as an alias for another variable.
- When used as a function parameter, it allows the function to access and modify the original argument variable that was passed to it. This is known as **pass by reference**.
- A reference variable is declared by placing an ampersand (`&`) after the data type in the parameter list.

```
void doubleNum(int &refVar)
{
   refVar *= 2;
}
```

- The `&` must appear in the function prototype as well as the function header.

```
void doubleNum(int &);
```

Note:

The variable `refVar` is called "a reference to an `int`."

Note:

Some programmers prefer not to put a space between the data type and the ampersand. The following prototype is equivalent to the one above:

```
void doubleNum(int &);
```

Note:

The ampersand must appear in both the prototype and the header of any function that uses a reference variable as a parameter. It does not appear in the function call.

Program 6-25

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

void doubleNum(int &);

int main()
{
    int value = 4;

    cout << "In main, value is " << value << endl;
    cout << "Now calling doubleNum..." << endl;
    doubleNum(value);
    cout << "Now back in main. value is " << value << endl;
    return 0;
}


void doubleNum (int &refVar)
{
    refVar *= 2;
}
```

Program Output

Program 6-26

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

void doubleNum(int &);
void getNum(int &);

int main()
```

```
{
    int value;

    getNum(value);

    doubleNum(value);

    cout << "That value doubled is " << value << endl;
    return 0;
}


void getNum(int &userNum)
{
    cout << "Enter a number: ";
    cin >> userNum;
}


void doubleNum (int &refVar)
{
    refVar *= 2;
}
```

Program Output

Note:

Only variables may be passed by reference. If you attempt to pass a nonvariable argument, such as a literal, a constant, or an expression, into a reference parameter, an error will result. Using the `doubleNum` function as an example, the following statements will generate an error:

```
doubleNum(5);
doubleNum(userNum + 10);
```

Warning!

Don't get carried away with using reference variables as function parameters. Any time you allow a function to alter a variable that's outside the function, you are creating potential debugging problems. Reference variables should only be used as parameters when the situation requires them.

Checkpoint

6.19 What kinds of values may be specified as default arguments?

6.20 Write the prototype and header for a function called `compute`. The function should have three parameters: an `int`, a `double`, 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 `double` parameter should not have a default argument.

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

6.22 What is the output of the following program?

```cpp
#include <iostream>
using namespace std;
void test(int = 2, int = 4, int = 6);
int main()
{
   test();
   test(6);
   test(3, 9);
   test(1, 5, 7);
   return 0;
}
void test (int first, int second, int third)
{
   first += 3;
   second += 6;
   third += 9;
   cout << first << " " << second << " " << third << endl;
}
```

6.23 The following program asks the user to enter two numbers. What is the output of the program if the user enters 12 and 14?

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

```
void func1(int &, int &);
void func2(int &, int &, int &);
void func3(int, int, int);
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;
}
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;
}
```

6.14 Overloading Functions

Concept:

Two or more functions may have the same name, as long as their parameter lists are different.

- In C++, you can **overload** functions by giving the same name to multiple functions.
- This is allowed as long as each function has a different parameter list (either different numbers of parameters or different data types).

- The compiler determines which version of the function to call based on the **function signature** (the function's name and the data types of its parameters).
- The function's return type is *not* part of its signature.

Program 6-27

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

int square(int);
double square(double);

int main()
{
    int userInt;
    double userFloat;

    cout << fixed << showpoint << setprecision(2);
    cout << "Enter an integer and a floating-point value: ";
    cin >> userInt >> userFloat;

    cout << "Here are their squares: ";
    cout << square(userInt) << " and " << square(userFloat);
    return 0;
}


int square(int number)
{
    return number * number;
}


double square(double number)
{
    return number * number;
}
```

Program Output

Program 6-28

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

void getChoice(char &);
double calcWeeklyPay(int, double);
double calcWeeklyPay(double);

int main()
{
    char selection;
    int worked;
    double rate;
    double yearly;

    cout << fixed << showpoint << setprecision(2);

    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);

    switch (selection)
    {
            case 'H' :
            case 'h' : cout << "How many hours were worked? ";
                   cin >> worked;
                   cout << "What is the hourly pay rate? ";
                   cin >> rate;
                   cout << "The gross weekly pay is $";
                   cout << calcWeeklyPay(worked, rate) << endl;
                   break;

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

```cpp
                    break;
    }
    return 0;
}


void getChoice(char & letter)
{
    cout << "Enter your choice (H or S): ";
    cin >> letter;

    while (letter != 'H' && letter != 'h' &&
          letter != 'S' && letter != 's')
    {
        cout << "Please enter H or S: ";
        cin >> letter;
    }
}


double calcWeeklyPay(int hours, double payRate)
{
    return hours * payRate;
}


double calcWeeklyPay(double annSalary)
{
    return annSalary / 52;
}
```

Program Output

Program Output

6.15 The `exit()` Function

Concept:

The `exit()` function causes a program to terminate, regardless of which function or control mechanism is executing.

- The `exit()` function immediately terminates the program, no matter which function it is called from.
- To use `exit()`, you must include the `<cstdlib>` header file.
- It takes an integer argument, called an **exit code**, which is passed to the operating system.
- Common exit codes are `0`, `EXIT_SUCCESS`, or `EXIT_FAILURE`.

Program 6-29

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

void function();

int main()
{
   function();
   return 0;
}


void function()
{
    cout << "This program terminates with the exit function.\n";
    cout << "Bye!\n";
    exit(0);
    cout << "This message will never be displayed\n";
```

```
        cout << "because the program has already terminated.\n";
    }
```

Program Output

Note:

Generally, the exit code is important only if you know it will be tested outside the program. If it is not used, just pass zero, or `EXIT_SUCCESS`.

Warning!

The `exit()` function unconditionally shuts down your program. Because it bypasses a program's normal logical flow, you should use it with caution.

Checkpoint

6.24 What is the output of the following program?

```
#include <iostream>
#include <cstdlib>
using namespace std;
void showVals(double, double);
int main()
{
   double x = 1.2, y = 4.5;
   showVals(x, y);
   return 0;
}
void showVals(double p1, double p2)
{
   cout << p1 << endl;
   exit(0);
   cout << p2 << endl;
}
```

6.25 What is the output of the following program?

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

```
int manip(int);
int manip(int, int);
int manip(int, double);
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;
}
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);
}
```

6.16 Stubs and Drivers

- **Stubs** and **drivers** are tools for testing and debugging programs with functions.
- A **stub** is a dummy function that stands in for a real function. It typically just displays a message to confirm it was called and shows the arguments it received. This allows you to test the calling logic without needing a working function.

```
void showFees(double memberRate, int months)
{
   cout << "The showFees function was called with "
        << "the following arguments:\n"
        << "memberRate: " << memberRate << endl
        << "months: " << months << endl;
}
```

- A **driver** is a program that tests a function by calling it. It passes test data to the function and, if applicable, displays the return value. This allows you to test a function in isolation before integrating it into the main program.

Program 6-30

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

void showFees(double, int);

int main()
{
    const double ADULT = 40.0;
    const double SENIOR = 30.0;
    const double CHILD = 20.0;

    cout << "Testing an adult membership...\n"
           << "Calling the showFees function with arguments "
           << ADULT << " and 10.\n";
    showFees(ADULT, 10);

    cout << "\nTesting a senior citizen membership...\n"
           << "Calling the showFees function with arguments "
           << SENIOR << " and 10.\n";
    showFees(SENIOR, 10);

    cout << "\nTesting a child membership...\n"
           << "Calling the showFees function with arguments "
           << CHILD << " and 10.\n";
    showFees(CHILD, 10);
    return 0;
}


void showFees(double memberRate, int months)
{
    cout << "The total charges are $"
         << (memberRate * months) << endl;
}
```

Program Output

Case Study: See High Adventure Travel Agency Part 1 Case Study on the Computer Science Portal at www.pearsonhighered.com/gaddis.

Review Questions and Exercises

Short Answer

1. Why do local variables lose their values between calls to the function in which they are defined?
2. What is the difference between an argument and a parameter variable?
3. Where do you define parameter variables?
4. If you are writing a function that accepts an argument and you want to make sure the function cannot change the value of the argument, what do you do?
5. When a function accepts multiple arguments, does it matter in what order the arguments are passed?
6. How do you return a value from a function?
7. What is the advantage of breaking your application's code into several small procedures?
8. How would a `static` local variable be useful?
9. Give an example where passing an argument by reference would be useful.

Fill-in-the-Blank

1. The _______ is the part of a function definition that shows the function name, return type, and parameter list.
2. If a function doesn't return a value, the word _______ will appear as its return type.
3. Either a function's _______ or its _______ must precede all calls to the function.

4. Values that are sent into a function are called ________.
5. Special variables that hold copies of function arguments are called ________.
6. When only a copy of an argument is passed to a function, it is said to be passed by ________.
7. A(n) ________ eliminates the need to place a function definition before all calls to the function.
8. A(n) ________ variable is defined inside a function and is not accessible outside the function.
9. ___________ variables are defined outside all functions and are accessible to any function within their scope.
10. ___________ variables provide an easy way to share large amounts of data among all the functions in a program.
11. Unless you explicitly initialize global variables, they are automatically initialized to ________.
12. If a function has a local variable with the same name as a global variable, only the ________ variable can be seen by the function.
13. ___________ local variables retain their value between function calls.
14. The ________ statement causes a function to end immediately.
15. ___________ arguments are passed to parameters automatically if no argument is provided in the function call.
16. When a function uses a mixture of parameters with and without default arguments, the parameters with default arguments must be defined ________.
17. The value of a default argument must be a(n) ________.
18. When used as parameters, ________ variables allow a function to access the parameter's original argument.
19. Reference variables are defined like regular variables, except there is a(n) ________ in front of the name.
20. Reference variables allow arguments to be passed by _______________.

21. The _______ function causes a program to terminate.

22. Two or more functions may have the same name, as long as their _______ are different.

Algorithm Workbench

1. Examine the following function header, then write an example call to the function:

```
void showValue(int quantity)
```

2. The following statement calls a function named `half`. The `half` function returns a value that is half that of the argument. Write the function.

```
result = half(number);
```

3. A program contains the following function:

```
int cube(int num)
{
   return num * num * num;
}
```

   Write a statement that passes the value 4 to this function and assigns its return value to the variable `result`.

4. Write a function named `timesTen` that accepts an argument. When the function is called, it should display the product of its argument multiplied by 10.

5. A program contains the following function:

```
void display(int arg1, double arg2, char arg3)
{
   cout  << "Here are the values: "
         << arg1 << " " << arg2 << " "
         << arg3 << endl;
}
```

   Write a statement that calls the procedure and passes the following variables to it:

```
int age;
double income;
char initial;
```

6. Write a function named `getNumber` that uses a reference parameter variable to accept an integer argument. The function should prompt the user to enter a number in the range of 1 through 100. The input should be validated and stored in the parameter variable.

True or False

1. T F Functions should be given names that reflect their purpose.
2. T F Function headers are terminated with a semicolon.
3. T F Function prototypes are terminated with a semicolon.
4. T F If other functions are defined before `main`, the program still starts executing at function `main`.
5. T F When a function terminates, it always branches back to `main`, regardless of where it was called from.
6. T F Arguments are passed to the function parameters in the order they appear in the function call.
7. T F The scope of a parameter is limited to the function that uses it.
8. T F Changes to a function parameter always affect the original argument as well.
9. T F In a function prototype, the names of the parameter variables may be left out.
10. T F Many functions may have local variables with the same name.
11. T F Overuse of global variables can lead to problems.
12. T F Static local variables are not destroyed when a function returns.
13. T F All static local variables are initialized to −1 by default.
14. T F Initialization of static local variables only happens once, regardless of how many times the function in which they are defined is called.
15. T F When a function with default arguments is called and an argument is left out, all arguments that come after it must be left out as well.
16. T F It is not possible for a function to have some parameters with default arguments and some without.
17. T F The `exit` function can only be called from `main`.

18. T F A stub is a dummy function that is called instead of the actual function it represents.

Find the Errors

Each of the following functions has errors. Locate as many errors as you can.

1.
```
void total(int value1, value2, value3)
{
   return value1 + value2 + value3;
}
```

2.
```
double average(int value1, int value2, int value3)
{
   double average;
   average = value1 + value2 + value3 / 3;
}
```

3.
```
void area(int length = 30, int width)
{
   return length * width;
}
```

4.
```
void getValue(int value&)
{
   cout << "Enter a value: ";
   cin >> value&;
}
```

5. *(Overloaded functions)*

```
int getValue()
{
   int inputValue;
   cout << "Enter an integer: ";
   cin >> inputValue;
   return inputValue;
}
double getValue()
{
   double inputValue;
   cout << "Enter a floating-point number: ";
   cin >> inputValue;
```

```
    return inputValue;
}
```

Programming Challenges

VideoNote

Solving the Markup Problem

1. Markup

   Write a program that asks the user to enter an item's wholesale cost and its markup percentage. It should then display the item's retail price. For example:

   - If an item's wholesale cost is 5.00 and its markup percentage is 100 percent, then the item's retail price is 10.00.
   - If an item's wholesale cost is 5.00 and its markup percentage is 50 percent, then the item's retail price is 7.50.

   The program should have a function named `calculateRetail` that receives the wholesale cost and the markup percentage as arguments and returns the retail price of the item.

   Input Validation: Do not accept negative values for either the wholesale cost of the item or the markup percentage.

2. Rectangle Area—Complete the Program

   If you have downloaded this book's source code, you will find a partially written program named AreaRectangle.cpp in the Chapter 06 folder. Your job is to complete the program. When it is complete, the program will ask the user to enter the width and length of a rectangle, then display the rectangle's area. The program calls the following functions, which have not been written:

   - `getLength`—This function should ask the user to enter the rectangle's length then return that value as a `double`.
   - `getWidth`—This function should ask the user to enter the rectangle's width then return that value as a `double`.

   - `getArea`—This function should accept the rectangle’s length and width as arguments and return the rectangle’s area. The area is calculated by multiplying the length by the width.
   - `displayData`—This function should accept the rectangle’s length, width, and area as arguments and display them in an appropriate message on the screen.
3. Winning Division

   Write a program that determines which of a company’s four divisions (Northeast, Southeast, Northwest, and Southwest) had the greatest sales for a quarter. It should include the following two functions, which are called by `main`:

   - `double getSales()` is passed the name of a division. It asks the user for a division’s quarterly sales figure, validates the input, then returns it. It should be called once for each division.
   - `void findHighest()` is passed the four sales totals. It determines which is the largest and prints the name of the high-grossing division, along with its sales figure.

   Input Validation: Do not accept dollar amounts less than $0.00.

4. Safest Driving Area

   Write a program that determines which of five geographic regions within a major city (north, south, east, west, and central) had the fewest reported automobile accidents last year. It should have the following two functions, which are called by `main`:

   - `int getNumAccidents()` is passed the name of a region. It asks the user for the number of automobile accidents reported in that region during the last year, validates the input, then returns it. It should be called once for each city region.
   - `void findLowest()` is passed the five accident totals. It determines which is the smallest and prints the name of the region, along with its accident figure.

   Input Validation: Do not accept an accident number that is less than 0.

5. Falling Distance

   When an object is falling because of gravity, the following formula can be used to determine the distance the object falls in a specific time period:

$$d = {}^{1}/_{2}gt^2$$

The variables in the formula are as follows: *d* is the distance in meters, *g* is 9.8, and *t* is the amount of time, in seconds, that the object has been falling.

Write a function named `fallingDistance` that accepts an object's falling time (in seconds) as an argument. The function should return the distance, in meters, that the object has fallen during that time interval. Write a program that demonstrates the function by calling it in a loop that passes the values 1 through 10 as arguments and displays the return value.

6. Kinetic Energy

   In physics, an object that is in motion is said to have kinetic energy. The following formula can be used to determine a moving object's kinetic energy:

$$KE = {}^{1}/_{2}\, mv^2$$

   The variables in the formula are as follows: *KE* is the kinetic energy, *m* is the object's mass in kilograms, and *v* is the object's velocity, in meters per second.

   Write a function named `kineticEnergy` that accepts an object's mass (in kilograms) and velocity (in meters per second) as arguments. The function should return the amount of kinetic energy that the object has. Demonstrate the function by calling it in a program that asks the user to enter values for mass and velocity.

7. Celsius Temperature Table

   The formula for converting a temperature from Fahrenheit to Celsius is

$$C = \frac{5}{9}(F - 32)$$

   where *F* is the Fahrenheit temperature and *C* is the Celsius temperature. Write a function named `celsius` that accepts a Fahrenheit temperature as an argument. The function should return the temperature, converted to Celsius. Demonstrate the function by calling it in a loop that displays a table of the Fahrenheit temperatures 0 through 20 and their Celsius equivalents.

8. Coin Toss

   Write a function named `coinToss` that simulates the tossing of a coin. When you call the function, it should generate a random number in the range of 1 through 2. If the random number is 1, the function should display "heads." If the random number is 2, the function should display "tails." Demonstrate the function in a program that asks the user how many times the coin should be tossed, then simulates the tossing of the coin that number of times.

9. Present Value

   Suppose you want to deposit a certain amount of money into a savings account and then leave it alone to draw interest for the next 10 years. At the end of 10 years, you would like to have $10,000 in the account. How much do you need to deposit today to make that happen? You can use the following formula, which is known as the present value formula, to find out:

   $$P = \frac{F}{(1+r)^n}$$

   The terms in the formula are as follows:

   - *P* is the **present value**, or the amount that you need to deposit today.
   - *F* is the **future value** that you want in the account. (In this case, *F* is $10,000.)
   - *r* is the **annual interest rate**.
   - *n* is the **number of years** that you plan to let the money sit in the account.

   Write a program that has a function named `presentValue` that performs this calculation. The function should accept the future value, annual interest rate, and number of years as arguments. It should return the present value, which is the amount that you need to deposit today. Demonstrate the function in a program that lets the user experiment with different values for the formula's terms.

10. Future Value

    Suppose you have a certain amount of money in a savings account that earns compound monthly interest, and you want to calculate the amount that you will have after a specific number of months. The formula, which is known as the future value formula, is

    $$F = P \times (1+i)^t$$

    The terms in the formula are as follows:

    - *F* is the **future value** of the account after the specified time period.
    - *P* is the **present value** of the account.

- *i* is the **monthly interest rate**.
- *t* is the **number of months**.

Write a program that prompts the user to enter the account's present value, monthly interest rate, and the number of months that the money will be left in the account. The program should pass these values to a function named `futureValue` that returns the future value of the account, after the specified number of months. The program should display the account's future value.

11. Lowest Score Drop

    Write a program that calculates the average of a group of test scores, where the lowest score in the group is dropped. It should use the following functions:

    - `void getScore()` should ask the user for a test score, store it in a reference parameter variable, and validate it. This function should be called by `main` once for each of the five scores to be entered.
    - `void calcAverage()` should calculate and display the average of the four highest scores. This function should be called just once by `main` and should be passed the five scores.
    - `int findLowest()` should find and return the lowest of the five scores passed to it. It should be called by `calcAverage`, which uses the function to determine which of the five scores to drop.

    Input Validation: Do not accept test scores lower than 0 or higher than 100.

12. Star Search

    A particular talent competition has five judges, each of whom awards a score between 0 and 10 to each performer. Fractional scores, such as 8.3, are allowed. A performer's final score is determined by dropping the highest and lowest score received, then averaging the three remaining scores. Write a program that uses this method to calculate a contestant's score. It should include the following functions:

    - `void getJudgeData()` should ask the user for a judge's score, store it in a reference parameter variable, and validate it. This function should be called by `main` once for each of the five judges.
    - `void calcScore()` should calculate and display the average of the three scores that remain after dropping the highest and lowest scores the performer received. This

function should be called just once by `main` and should be passed the five scores.

The last two functions, described below, should be called by `calcScore`, which uses the returned information to determine which of the scores to drop.

- `int findLowest()` should find and return the lowest of the five scores passed to it.
- `int findHighest()` should find and return the highest of the five scores passed to it.

Input Validation: Do not accept judge scores lower than 0 or higher than 10.

13. Days Out

Write a program that calculates the average number of days a company's employees are absent. The program should have the following functions:

- A function called by `main` that asks the user for the number of employees in the company. This value should be returned as an `int`. (The function accepts no arguments.)
- A function called by `main` that accepts one argument: the number of employees in the company. The function should ask the user to enter the number of days each employee missed during the past year. The total of these days should be returned as an `int`.
- A function called by `main` that takes two arguments: the number of employees in the company and the total number of days absent for all employees during the year. The function should return, as a `double`, the average number of days absent. (This function does not perform screen output and does not ask the user for input.)

Input Validation: Do not accept a number less than 1 for the number of employees. Do not accept a negative number for the days any employee missed.

14. Order Status

The Middletown Wholesale Copper Wire Company sells spools of copper wiring for $100 each. Write a program that displays the status of an order. The program should have a function that asks for the following data:

- The number of spools ordered
- The number of spools in stock

- Whether there are special shipping and handling charges

(Shipping and handling is normally $10 per spool.) If there are special charges, the program should ask for the special charges per spool.

The gathered data should be passed as arguments to another function that displays:

- The number of spools ready to ship from current stock
- The number of spools on backorder (if the number ordered is greater than what is in stock)
- Subtotal of the portion ready to ship (the number of spools ready to ship times $100)
- Total shipping and handling charges on the portion ready to ship
- Total of the order ready to ship

The shipping and handling parameter in the second function should have the default argument 10.00.

Input Validation: Do not accept numbers less than 1 for spools ordered. Do not accept a number less than 0 for spools in stock or shipping and handling charges.

15. Overloaded Hospital

Write a program that computes and displays the charges for a patient's hospital stay. First, the program should ask if the patient was admitted as an inpatient or an outpatient. If the patient was an inpatient, the following data should be entered:

- The number of days spent in the hospital
- The daily rate
- Hospital medication charges
- Charges for hospital services (lab tests, etc.)

The program should ask for the following data if the patient was an outpatient:

- Charges for hospital services (lab tests, etc.)
- Hospital medication charges

The program should use two overloaded functions to calculate the total charges. One of the functions should accept arguments for the inpatient data, while the other function accepts arguments for outpatient information. Both functions should return the total charges.

Input Validation: Do not accept negative numbers for any data.

16. Population

    In a population, the birth rate is the percentage increase of the population due to births, and the death rate is the percentage decrease of the population due to deaths. Write a program that displays the size of a population for any number of years. The program should ask for the following data:

    - The starting size of a population
    - The annual birth rate
    - The annual death rate
    - The number of years to display

    Write a function that calculates the size of the population for a year. The formula is

    ```
    N = P + BP - DP
    ```

    where `N` is the new population size, `P` is the previous population size, `B` is the birth rate, and `D` is the death rate.

    Input Validation: Do not accept numbers less than 2 for the starting size. Do not accept negative numbers for birth rate or death rate. Do not accept numbers less than 1 for the number of years.

17. Transient Population

    Modify Programming Challenge 16 to also consider the effect on population caused by people moving into or out of a geographic area. Given as input a starting population size, the annual birth rate, the annual death rate, the number of individuals who typically move into the area each year, and the number of individuals who typically leave the area each year, the program should project what the population will be `numYears` from now. You can either prompt the user to input a value for `numYear`s, or you can set it within the program.

Input Validation: Do not accept numbers less than 2 for the starting size. Do not accept negative numbers for birth rate, death rate, arrivals, or departures.

18. Paint Job Estimator

    A painting company has determined that for every 110 square feet of wall space, 1 gallon of paint and 8 hours of labor will be required. The company charges $25.00 per hour for labor. Write a modular program that allows the user to enter the number of rooms that are to be painted and the price of the paint per gallon. It should also ask for the square feet of wall space in each room. It should then display the following data:

    - The number of gallons of paint required
    - The hours of labor required
    - The cost of the paint
    - The labor charges
    - The total cost of the paint job

    Input validation: Do not accept a value less than 1 for the number of rooms. Do not accept a value less than $10.00 for the price of paint. Do not accept a negative value for square footage of wall space.

19. Using Files—Hospital Report

    Modify Programming Challenge 15 (Overloaded Hospital) to write the report it creates to a file.

20. Stock Profit

    The profit from the sale of a stock can be calculated as follows:

$$\text{Profit} = ((NS \times SP) - SC) - ((NS \times PP) + PC)$$

    where *NS* is the number of shares, *SP* is the sale price per share, *SC* is the sale commission paid, *PP* is the purchase price per share, and *PC* is the purchase commission paid. If the calculation yields a positive value, then the sale of the stock resulted in a profit. If the calculation yields a negative number, then the sale resulted in a loss.

    Write a function that accepts as arguments the number of shares, the purchase price per share, the purchase commission paid, the sale price per share, and the sale commission paid. The function should return the profit (or loss) from the sale of stock.

Demonstrate the function in a program that asks the user to enter the necessary data and displays the amount of the profit or loss.

21. Multiple Stock Sales

    Use the function that you wrote for Programming Challenge 20 (Stock Profit) in a program that calculates the total profit or loss from the sale of multiple stocks. The program should ask the user for the number of stock sales and the necessary data for each stock sale. It should accumulate the profit or loss for each stock sale, then display the total.

22. `isPrime` Function

    A prime number is a number that is only evenly divisible by itself and 1. For example, the number 5 is prime because it can only be evenly divided by 1 and 5. The number 6, however, is not prime because it can be divided evenly by 1, 2, 3, and 6.

    Write a function name `isPrime`, which takes an integer as an argument and returns `true` if the argument is a prime number, or `false` otherwise. Demonstrate the function in a complete program.

    Tip:

    Recall that the % operator divides one number by another, and returns the remainder of the division. In an expression such as `num1 % num2`, the % operator will return 0 if `num1` is evenly divisible by `num2`.

23. Prime Number List

    Use the `isPrime` function that you wrote in Programming Challenge 22 (`isPrime function)` in a program that stores a list of all the prime numbers from 1 through 100 in a file.

24. Rock, Paper, Scissors Game

    Write a program that lets the user play the game of Rock, Paper, Scissors against the computer. The program should work as follows:

    1. When the program begins, a random number in the range of 1 through 3 is generated. If the number is 1, then the computer has chosen rock. If the number is 2, then the computer has chosen paper. If the number is 3, then the computer has chosen scissors. (Don't display the computer's choice yet.)

2. The user enters his or her choice of “rock”, “paper”, or “scissors” at the keyboard. (You can use a menu if you prefer.)
3. The computer’s choice is displayed.
4. A winner is selected according to the following rules:
    - If one player chooses rock and the other player chooses scissors, then rock wins. (The rock smashes the scissors.)
    - If one player chooses scissors and the other player chooses paper, then scissors wins. (Scissors cuts paper.)
    - If one player chooses paper and the other player chooses rock, then paper wins. (Paper wraps rock.)
    - If both players make the same choice, the game must be played again to determine the winner.

Be sure to divide the program into functions that perform each major task.

Group Project

1. Travel Expenses

    This program should be designed and written by a team of students. Here are some suggestions:

    - One student should design function `main`, which will call the other functions in the program. The remainder of the functions will be designed by other members of the team.
    - The requirements of the program should be analyzed so each student is given about the same workload.
    - The parameters and return types of each function should be decided in advance.
    - Stubs and drivers should be used to test and debug the program.
    - The program can be implemented as a multi-file program, or all the functions can be cut and pasted into the main file.

Here is the assignment: Write a program that calculates and displays the total travel expenses of a businessperson on a trip. The program should have functions that ask for and return the following:

- The total number of days spent on the trip
- The time of departure on the first day of the trip, and the time of arrival back home on the last day of the trip
- The amount of any round-trip airfare
- The amount of any car rentals
- Miles driven, if a private vehicle was used. Calculate the vehicle expense as $0.27 per mile driven
- Parking fees (The company allows up to $6 per day. Anything in excess of this must be paid by the employee.)
- Taxi fees, if a taxi was used anytime during the trip (The company allows up to $10 per day, for each day a taxi was used. Anything in excess of this must be paid by the employee.)
- Conference or seminar registration fees
- Hotel expenses (The company allows up to $90 per night for lodging. Anything in excess of this must be paid by the employee.)
- The amount of *each* meal eaten. On the first day of the trip, breakfast is allowed as an expense if the time of departure is before 7 a.m. Lunch is allowed if the time of departure is before 12 noon. Dinner is allowed on the first day if the time of departure is before 6 p.m. On the last day of the trip, breakfast is allowed if the time of arrival is after 8 a.m. Lunch is allowed if the time of arrival is after 1 p.m. Dinner is allowed on the last day if the time of arrival is after 7 p.m. The program should only ask for the amounts of allowable meals. (The company allows up to $9 for breakfast, $12 for lunch, and $16 for dinner. Anything in excess of this must be paid by the employee.)

The program should calculate and display the total expenses incurred by the businessperson, the total allowable expenses for the trip, the excess that must be reimbursed by the businessperson, if any, and the amount saved by the businessperson if the expenses were under the total allowed.

Input Validation: Do not accept negative numbers for any dollar amount or for miles driven in a private vehicle. Do not accept numbers less than 1 for the number of days. Only accept valid times for the time of departure and the time of arrival.