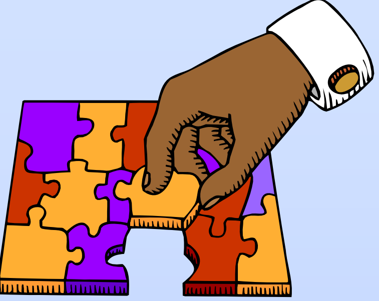
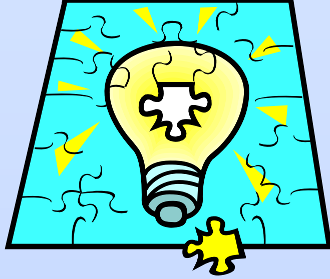



Unit 3 Lesson 11

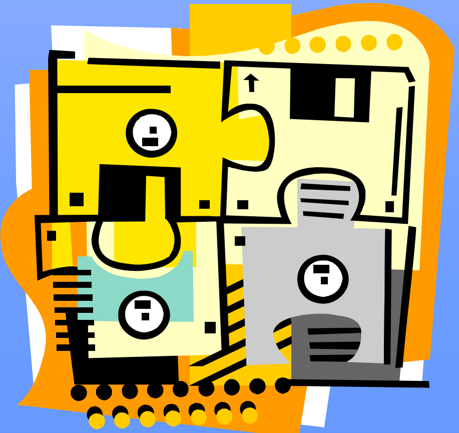
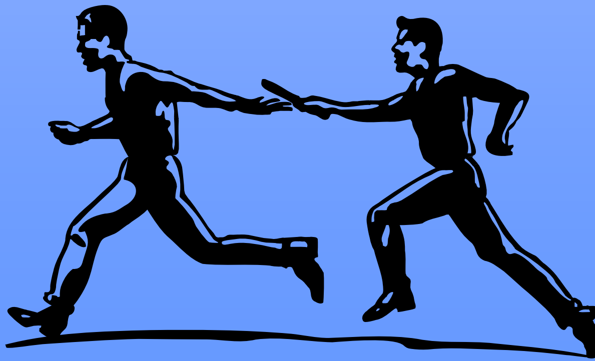
Passing Data and Using Library Functions

Textbook Authors:

Knowlton, Barksdale, Turner, & Collings

PowerPoint Lecture

by Dave Clausen

Function Order Within a Program

Comments

Preprocessor Directives (#include …)

Constant Declaration

Function Declarations (Prototypes)

```
int main( )
{
   //main program
   // Function calls
}
```

Function Implementations

Function Declarations

- Function Declaration
  - Function name, number and type of arguments it expects and type of value it returns (if any).
  - General Form:
    - <return type> <Function_Name> (<arguments>);
    - function name is descriptive valid identifier
    - return type, declares the one data type returned by the function
    - Style: include comments regarding input & output.

Function Declaration Example

```
//Function: Square
//Compute the square of a double precision floating point number
//
//Input: a double precision floating point number
//Output: a double precision floating point number that
//represents the square of the number received.

double Square (double x);  //Semicolon here
```

Function Implementations

- Function Implementation
  - A complete description of the function
  - All the commands necessary to fulfill the purpose of the function.

```
double Square (double x)      //No semicolon here
{
  return x*x;
}
```

Function Headings

- Function heading
  - the first line of the function implementation containing the function’s name, parameter declarations, and return type
  - looks like the function prototype (declaration) minus the semicolon
  - function heading must match the function declaration in type and parameter list types and order of variables.
  - I recommend copying the prototype (minus the semicolon) and pasting for the function implementation heading.

Function Headings 2

- Formal Parameters (Parameters)
  - The arguments in the **function heading and function prototype (declaration)**.
  - Describes the **form** that the variables must take (int, double, char, etc.)
- Actual Parameters (Arguments)
  - The arguments in the **function call (do not list the data type here: int, double, char etc.)**.
- The number of formal and actual parameters must match and should match in order and type.

Function Heading General Form

- General Form

  <return type> <Function_Name> (<formal parameter list>)

  – Function name is a valid identifier
    - use descriptive names
    - a value may need to be returned if a return type is declared

  – Return type declares the data type of the function

  – The formal parameters are pairs of data types and identifier names separated by commas.

  – No semicolon at the end of the header.

Getting Data to and from Functions

- You have learned that the parentheses following a function's name let the complier know that it is a function.
- The parentheses can serve another purpose as well.
- That is, parentheses can be used to *pass* data ***to a function*** and in some cases to return data ***from a function***.

Getting Data to and from Functions 2

- When a function is called, the data in the parentheses (called the *argument* or *actual parameter*) is passed into the receiving function.
- There are two ways to pass data to functions: *passing by value* and *passing by reference*.

Value Parameters

- Value Parameters (Passed by Value)
  - values passed only from caller to a function
  - think of this as a one way trip
  - a copy of the actual parameters are made and sent to the function where they are known locally.
  - any changes made to the formal parameters will leave the actual parameters unchanged
  - at the end of the function, memory is deallocated for the formal parameters

Passing by Value

- When you pass a variable to a function *by value*, a copy of the value in the variable is given to the function for it to use.
  - These are called *value parameters*.
- If the variable is changed within the function, the original copy of the variable in the calling function remains the same.
- Code List 11-1 is an example of a function that accepts data using the passing by value technique.

Code List 11-1

```
void Print_True_Or_False ( bool true_false)
{
    if true_false  // If true_false is true, display the word TRUE, else FALSE.
            cout << “TRUE\n”;
    else
            cout << “FALSE\n”;
}
```

- A value comes into the function through the parentheses, and the copy of the value will be placed in the variable true_false.
- The variable true_false is called a ***parameter (formal parameter)***.
- When you call the function you pass the variable in the parentheses to the function.
- The data types, and order of variables must be the same that you send to the function.

Code List 11-2

- Here are some sample calls to the function Print_True_Or_False.

```
Print_True_Or_False (complete);      //passes a variable
Print_True_Or_False(true);           //passes a literal
Print_True_Or_False(j = = 3 && k = = 2);    // passes the result
                                            // of an expression
```

- Most of the time you will pass a variable to a function.
- Notice that in the function call, you only list the variable name and not it’s data type.
- This is called the argument or actual parameter.

Code List 11-3

```
// passval.cpp                              passval.txt
#include <iostream.h>
void Print_Value (int j);    // function prototype or declaration: parameter or formal parameter
//=============================================================
int main ( )
{
        int i = 2;
        cout << "The value before the function is " << i << endl;
        Print_Value (i);  //argument or actual parameter
        cout << "the value after the function exists is " << i << endl;
        return 0;
}
//------------------------------------------------------------------------------------------------
void Print_Value (int j)  //parameter or formal parameter
{
        cout << "The value passed to the function is " << j << endl;
        j = j * 2; // the value in the variable i is doubled
        cout << "The value at the end of the function is " << j << endl;
}
```

Reference Parameters

- Reference Parameters (Pass by Reference)
  - When you want a function to return more than one value:
    - Use a void function with reference parameters.
    - Like a two way trip for more than one parameter.
  - The value of the parameter can be changed by the subprogram.
  - No copy of the parameters are made

Page147Swap.cpp **P147Swap.txt**

Reference Parameters 2

- Only the "address" of the actual parameter is sent to the subprogram.
- Format for formal reference parameters
  - <type name> **&**<formal parameter name>
  - void Get_Data(int &length, int &width);
- Remember:
  - avoid reference parameters in value returning functions
  - don't send constants to a function with reference parameters

Passing by Reference

- Functions that pass variables *by reference* will pass any changes you make to the variables back to calling function.
- For example, suppose you need a function that gets input from the user.
- The function in Code List 11-4 uses *passing by reference* to get two values from the user and pass them back through parentheses and *reference parameters*.

Code List 11-4

- The use of the ampersand symbol (**&**) means that the variables are reference parameters.
- Think of these variables as making a round trip.
  - Information is sent in to the function and returned from the function (no return statement necessary).

```
void Get_Values (float &income, float &expense)  //reference parameters
{
      cout << “Enter this month’s income amount: $”;
      cin >> income;
      cout << “Enter this month’s expense amount: $”;
      cin >> expense;
}
```

Value Returning Functions

- Declare function type of value to be returned
  - double, int, char, etc.
- Only ONE value is returned
  - use return command as last line of function
  - don’t use pass by reference parameters
  - don’t use cout statements
  - like mathematical function, calculates 1 answer
  - Style: Use Noun names that start with a capital letter

area.cpp area.txt

Function Example

```
//Function:  Cube
// Computes the cube of an integer
//
//Input: a number
//Output: a number representing the cube of the input number

double Cube (double x);

double Cube (double x)
{
  return x*x*x;
}
```

Function Declaration

(before int main)

Function Implementation

(after int main)

Returning Values Using *Return*

- As you learned earlier in this lesson, unless a function is declared with the keyword void, the function will return a value.
- In the case of the main function, it returns a value to the operating system.
- Other functions, however, return a value to the function that called them.
- The value to be returned is specified using the *return* statement.
- The function in Code List 11-5 is an example of a function that returns a value of type double.

Code List 11-5

- Functions that return a value using the return statement can return only one value.
- If you need more than one value returned, use a void function with reference parameters (&).
- **Value returning functions need to be called in an assignment statement.**

```
double Celsius_To_Fahrenheit (double celsius)
{
    double fahrenheit;                    // local variable
    fahrenheit = celsius * (9.0/5.0) + 32.0;
    return (fahrenheit);
}
```

Value Returning Functions

- Any function that is not declared as void should include a return statement.
- The value or expression in the return statement is returned to the calling function.
- In the Celsius_To_Fahrenheit function, the value store in fahrenheit is returned to the calling function.
- The program in Code List 11-6 shows how you can use this function.

Code List 11-6

```
// ctof.cpp                    ctof.txt
#include <iostream.h>
int main ( )
{
        double fahrenheit;
        double celsius = 22.5;

        fahrenheit = Celsius_To_Fahrenheit (celsius);
//value returning functions need to be called in an assignment statement
        cout << celsius << " C = " << fahrenheit << "F\n";
        return 0;
}
```

Code List 11-7

- Compare the code below with code list 11-5.
- You can declare a local variable, use it for calculations and then return that variable, as in code list 11-5.
- Or you can perform the calculations in the return statement as shown below.

```
double Celsius_To_Fahrenheit (double celsius)
{
      return (celsius * (9.0/5.0) + 32.0);
}
```

Return Statement

When using the return statement, keep the following points in mind:

1. The *return* statement does not require that the value being returned be placed in parentheses. You may, however, want to get into the habit of placing variables and expressions in parentheses to make the code more readable.
2. A function can return **only one value** using ***return***. Use passing by reference to return multiple values from a function.
3. When a *return* statement is encountered, the function will exit and return the value specified, even if other program lines exist below the *return.*

Return Statement 2

4. A function can have more than one *return* statement to help simplify an algorithm. For example, a *return* statement could be in an if structure allowing a function to return early if a certain condition is met.
5. The calling function isn't required to use or even to capture the value returned from a function it calls (See point #8).
6. You can call a value returning function in a cout statement or an assignment statement.
7. **Your instructor prefers that you call value returning functions in assignment statements.**
8. If you call a value returning function in the way that you call a void function, you lose the value returned from the function.
9. When the last line of a function is reached, or a return ( ) statement is executed, the function ends and the program returns to the calling function and begins executing statements from where it left off.

Choosing a Parameter Method

- When the actual parameter must be changed, or you need a "two way" trip use a reference parameter (&).
- When the value should NOT be changed (a "one way trip") and the data size is small, declare as a value parameter.

More About Function Prototypes

- A function prototype consists of the function return type, name and argument list. The prototype for the Celsius_To_Fahrenheit function could be written as:

double Celsius_To_Fahrenheit (double);

- The prototype for the Get_Values function could be written as:

void Get_Values (float &, float &);

- **Your instruction doesn’t consider this good programming style, so you will lose points if you do this.**

Formal and Actual Parameters

[Area.cpp]

- Formal parameters
  - – the parameters listed in the function declaration and implementation
  - – they indicate the “form” that the parameters will take, a data type and identifier paired together

[Area.txt]

- Actual parameters
  - the parameters in the function call which must match the formal parameter list in order & type
  - choose synonyms for names of actual & formal or use the same name

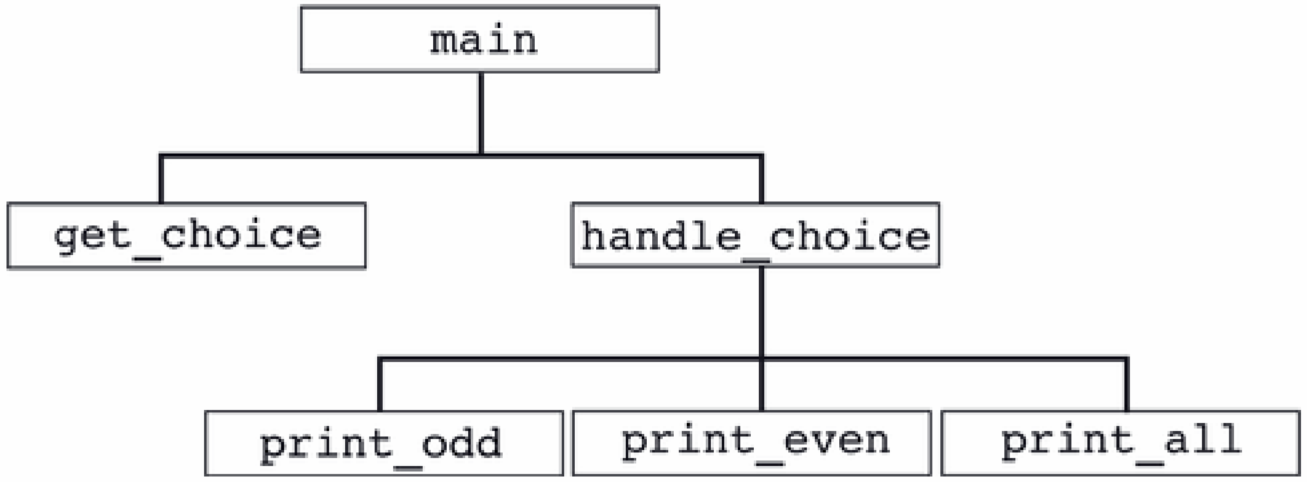
Dividing the Series Program into Functions

- Now that you have practiced creating functions and moving data to and from them, let us take another look at the program from Step-by-Step 11-1.
- In lesson 10, you studied a VTOC of the program divided into functions.
- That VTOC appears again in Figure 11-1.

Figure 11-1

FIGURE 11-1

This Visual Table of Contents shows the series program divided into functions.



The Series Program

- Let’s look at the series program source code, rewritten to use functions.
- Your instructor has modified the programming style to be an example for the programs that you will write for this class.

Series2.cpp Series2.txt

Stub Programming

- Stub Programming
  - the use of incomplete functions to test data transmission between them.
  - Contains declarations and rough implementations of each function.
  - Tests your program logic and values being passed to and from subprograms.

Roots1.cpp	Roots1.txt
Roots2.cpp	Roots2.txt
Roots.cpp	Roots.txt

Main Driver

- Main Driver
  - another name for the main function when subprograms are used.
  - The driver can be modified to test functions in a sequential fashion.
  - The main driver is a “function factory”
    - add a function declaration, then a rough implementation
    - test the function stub by calling the function from the main driver

Driver Example

// Program file: driver.cpp

```
#include <iostream.h>

int main()
{
    int data;

    cout << "Enter an integer: ";
    cin >> data;
    return 0;
}
```

//Start with a simple program

Driver Example 2

// Program file: driver.cpp   driver.txt

```
#include <iostream.h>

int main()
{
      int data;

      cout << "Enter an integer: ";
      cin >> data;
      cout << "The square is " << Sqr(data) << endl;
      return 0;
}
```

Add a function declaration, rough implementation, and a function call to test one function.

Driver: Test and Fill in Details

- Once you have tested the function declaration, call, and implementation by sending and returning the same value, fill in the details of the function.
- In the driver.cpp example, this means replace:
  - return x   with
  - return x*x

driver2.cpp   driver2.txt

Putting it all together

- Sample Program
  - Comparing the cost of pizzas by square inch

Pizza.cpp

Pizza.txt

Using Library Functions

- Library functions are just like functions you create and may be used in the same way.
- The difference is that the source code for library functions does not appear in your program.
- The prototypes for library functions are provided to your program using the #include complier directive.
- Let us examine a common C++ library function, pow ( ), which is used to raise a value (x) by a designated power (y). The pow ( ) function prototype is shown below and is included in math.h

```
double pow (double x, double y);
```

- The function pow receives two values or expressions of type double and returns the result as a double. Below is an example of a call to the pow function

```
z = pow (x, y);   // z equals x raised to the y power
```

Pow Function

- In order to use the pow function, you must include the math.h header file using the compiler directive below.
- A *header file* is a text file that provides the prototypes of a group of library functions. The ***linker*** uses the information in the header file to properly link your program with the function you want to use.

```
#include <math.h>
```

Code List 11-8

```
// power.cpp                      power.txt
#include <iostream.h>
#include <math.h>
int main ( )
{
        double base;
        double exponent;
        double answer;

        cout << “Enter the base: “;            // prompt user for base
        cin >> base;
        cout << “Enter the exponent: “;        // prompt user for exponent
        cin >> exponent;

        answer = pow (base, exponent);         // calculate answer

        cout << “The answer is” << answer << end1;

        return 0;
}
```

Popular Math Functions

- Many C++ compilers provide basic math functions, such as the one you used to calculate x^y. (error in textbook page 366 is listed as xy, but it should be x^y)

- Table 11-1 describes some basic math functions and shows their prototypes and their purpose.

Table 11-1

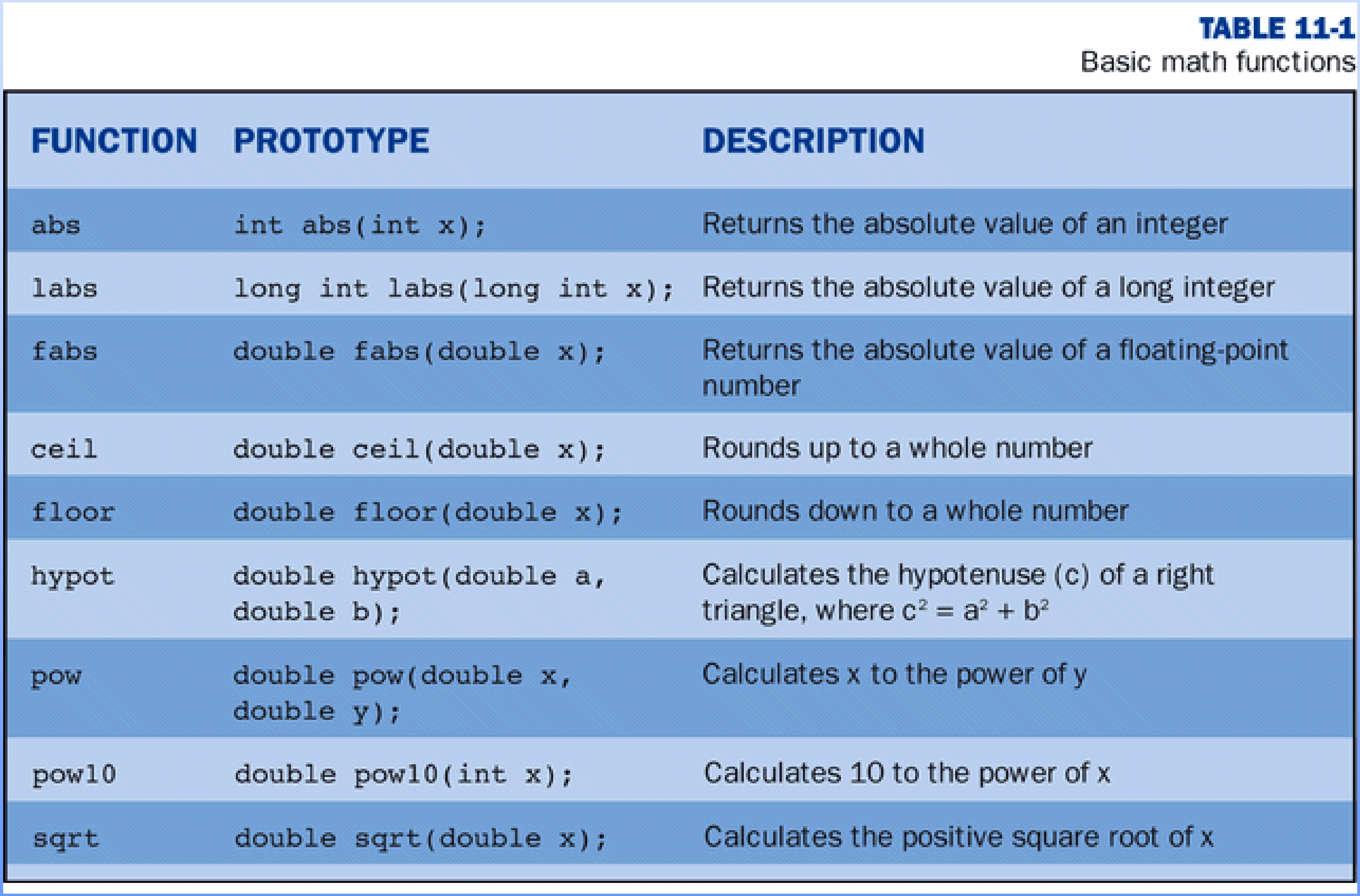


TABLE 11-1
Basic math functions

FUNCTION	PROTOTYPE	DESCRIPTION
`abs`	`int abs(int x);`	Returns the absolute value of an integer
`labs`	`long int labs(long int x);`	Returns the absolute value of a long integer
`fabs`	`double fabs(double x);`	Returns the absolute value of a floating-point number
`ceil`	`double ceil(double x);`	Rounds up to a whole number
`floor`	`double floor(double x);`	Rounds down to a whole number
`hypot`	`double hypot(double a, double b);`	Calculates the hypotenuse (c) of a right triangle, where $c^2 = a^2 + b^2$
`pow`	`double pow(double x, double y);`	Calculates x to the power of y
`pow10`	`double pow10(int x);`	Calculates 10 to the power of x
`sqrt`	`double sqrt(double x);`	Calculates the positive square root of x

Code List 11-9

- Here is an example of using one of the math functions included in the math.h library.
- Notice that you use these “value returning functions” in a cout statement or in an assignment statement.

```
cout << “The absolute value of ” << x << “ is ” << fabs (x) << endl;
answer = sqrt (radicand);
cout << “The square root of ” << radicand << “ is ” << answer << endl;
```

Functions for Working with Characters

- C++ compilers also include many functions for analyzing and changing characters.
- The header file ctype.h must be included for a calling program to use the functions listed in Table 11-2.
- The conditional functions in the table return a nonzero integer if the condition is true and zero if the condition if false.

Table 11-2

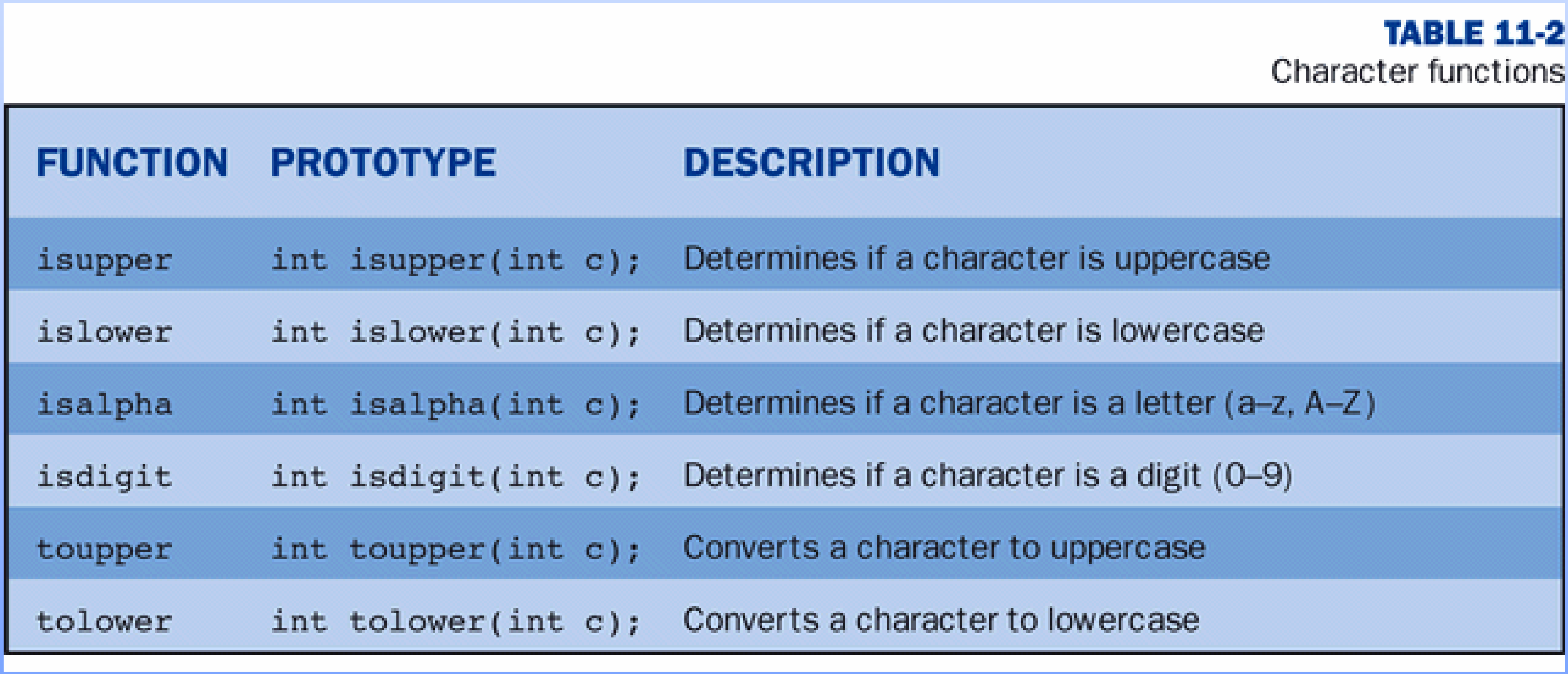


TABLE 11-2
Character functions

FUNCTION	PROTOTYPE	DESCRIPTION
`isupper`	`int isupper(int c);`	Determines if a character is uppercase
`islower`	`int islower(int c);`	Determines if a character is lowercase
`isalpha`	`int isalpha(int c);`	Determines if a character is a letter (a–z, A–Z)
`isdigit`	`int isdigit(int c);`	Determines if a character is a digit (0–9)
`toupper`	`int toupper(int c);`	Converts a character to uppercase
`tolower`	`int tolower(int c);`	Converts a character to lowercase

Code List 11-10 Description

- The Code List 11-10 demonstrates the use of isalpha, isupper, and isdigit functions.
- The program asks for a character and then reports to the user whether the character is uppercase or lowercase.
- If the user enters a numeral, the program detects and reports that as well.

Code List 11-10

```
// charfun.cpp                    charfun.txt
#include <iostream.h>
#include <ctype.h>
int main ( )
{
      char c;
      cout << “Enter a character\n ”;
      cin >> c;
      if (isalpha( c))
        {
          if (isupper( c))
                cout << c << “is an uppercase letter\n”;
          else
                cout << c << “is a lower case letter\n”;
        }
```

Code List 11-10 Cont.

```
  if (isdigit( c))
    cout << c << “is a number\n”;

  if ( ! (isdigit (c) || is alpha( c)))
    cout << c <<“ is neither a letter nor a number\n”;

  return 0;
}
```