C++ Programming: Functions

Some material taken from: C++ Programming: Program Design Including Data Structures

Objectives

- Learn about standard (predefined) functions and discover how to use them in a program
- Learn how to write your own functions
- Learn about actual and formal parameters
- Use arrays as input to functions

Functions

- A program is collection of modules called functions
- Functions we have seen so far consist of the main function in addition to library functions: sqrt, pow, …
- Using functions allows for code reuse
  - The same Selection-sort can be used to sort different arrays when wraped as a function

Functions (continued)

- Functions are like building blocks
- They allow complicated programs to be divided into manageable pieces
- Some advantages of functions:
  - A programmer can focus on just that part of the program and construct it, debug it, and perfect it
  - Different people can work on different functions simultaneously
  - Can be used in more than one place in a program or in different programs

Predefined (Library) Functions

- Some of the predefined functions are:
  - sqrt(x)
  - pow(x,y)
  - floor(x)
- Predefined functions are organized into separate libraries, e.g.,
  - Math functions are in <cmath> header
  - I/O functions are in <iostream> header
  - string functions are in <string> header

The Power Function (pow)

- pow(x,y) calculates x^y, pow(2,3) returns 8.0
- pow returns a value of the type double
- x and y are called the parameters (or arguments) of the function pow
- Function pow has two parameters

The sqrt and floor Functions

- The square root function sqrt(x)
  - Calculates the non-negative square root of x, for x >= 0.0
  - sqrt(2.25) returns 1.5
  - Type double and has only one parameter

The sqrt and floor Functions (continued)

- The floor function floor(x)
  - Calculates largest whole number not greater than x
  - floor(48.79) returns 48.0
  - Type double and has only one parameter

Table 6-1 Predefined Functions

Function	Standard Header File	Purpose	Parameter(s) Type	Result
`abs(x)`	`<cstdlib>`	Returns the absolute value of its argument: `abs(-7) = 7`	`int`	`int`
`ceil(x)`	`<cmath>`	Returns the smallest whole number that is not less than `x`: `ceil(56.34) = 57.0`	`double`	`double`
`cos(x)`	`<cmath>`	Returns the cosine of angle `x`: `cos(0.0) = 1.0`	`double` (radians)	`double`
`exp(x)`	`<cmath>`	Returns e^x, where `e = 2.718`: `exp(1.0) = 2.71828`	`double`	`double`
`fabs(x)`	`<cmath>`	Returns the absolute value of its argument: `fabs(-5.67) = 5.67`	`double`	`double`
`floor(x)`	`<cmath>`	Returns the largest whole number that is not greater than `x`: `floor(45.67) = 45.00`	`double`	`double`
`pow(x,y)`	`<cmath>`	Returns x^y; if `x` is negative, `y` must be a whole number: `pow(0.16, 0.5) = 0.4`	`double`	`double`

User-Defined Functions

Example: larger function

```
#include<iostream>
Using namespace std;
double  larger(double,double);  // function prototype or heading
int main()
{    ….
     double a = 3.0;
     double c = larger(a,7);  // calling the function
     ….
     return 0;
 }
double larger(double x, double y) //  body of the function
{
   double  max;
   if(x>y)   max = x;
   else   max = y;
   return max;
}
```

User-Defined Functions

- Two types of functions:
  - Void functions: do not return a value
  - Value-returning functions: return a value

- To create a new function:
  - you need to declare it: prototype heading
  - write the body of the function: definition
- Then you can use it: function call

Value-Returning Functions

- Because the value returned by a value-returning function is unique, we must:
  - Save the value for further calculation
  - Use the value in some calculation
- A value-returning function is used in an assignment or in an output statement

Function declaration

- Heading:
    1. Name of the function
    2. List of parameters: data type and name of each. Called Formal parameters (or arguments)
    3. Type of the function (void if it returns no value)
- Actual parameter (or arguments):
    - variable or expression listed in a call to a function
    - mapped by position to the formal parameters

Function Syntax

- The syntax for function heading/declaration

  functionReturnType functionName(formal parameter list);

- The syntax for function defintion:

  functionReturnType functionName(formal parameter list)

  {

  statements

  }

- Function return Type: type of the value returned by the function

Function Syntax (continued)

- The syntax of the formal parameter list is:

  dataType identifier, dataType identifier, ...

- The syntax for a function call is:

  functionName(actual parameter list)

- The syntax for the actual parameter list is:

  expression or variable,expression or variable, ...

Calling Functions

- To call a function:
  - Use its name, with the actual parameters (if any) in parentheses
  - There is a one-to-one correspondence between actual and formal parameters
- A function call in a program results in the execution of the body of the called function

The return Statement

- Once the function computes the value, the function returns the value via the return statement
- The syntax of the return statement is:

  return expression;

- When a return statement executes
  - Function immediately terminates
  - Control goes back to the caller
- When a return statement executes in the function main, the program terminates

Example

- Largest of three numbers
- Larger3.cpp

void Functions

- A void function does not return a value
- Syntax:
  - Prototype/heading:

    void functionName(formal parameter list);

  - Function definition:

    void functionName(actual parameter list) { … }

  - Function call:

    functionName(actual parameter list);

Example

- Illustrative example of void functions

```
void print(int n); // prototype
void print(int n) {
        for(int i=1; i<n; i++) cout<<“*”;
}
int main() {  print(7); return 0;}
```

- More interesting examples soon

Flow of Execution

- Execution always begins at
  - The first statement in the function main no matter where main is placed in the program
- Other functions are executed only when they are called

Flow of Execution (continued)

- A function call statement results in
  - Transfer of control to the first statement in the body of the called function
- After the last statement of the called function is executed
  - Control is passed back to the point immediately following the function call

Flow of Execution (continued)

- A value-returning function returns a value
- After executing the function
  - The value that the function returns replaces the function call statement

Passing arrays to functions: Example

```
int larger(int,int);
int arrayMax(int B[],int n);
int main() {
    int A[] = {7,5,2,31,12};
    int max = arrayMax(A,5); // a  reference to the array A is
                             // is passed to the function
                             // ArrayMax
    return 0;
}
int arrayMax(int B[],int n) {
     int max = B[0];
     for(int i=1;i<n;i++) max = larger(max,B[i]);
     return max;
}
int larger(int x, int y) {…}
```

Passing arrays to functions

- Typical syntax of array formal argument:

```
returnType functionName(arrayType arrayName[], int arraySize, other arguments);
```

  Typically you pass the array size also as an argument. For instance, in the above example, we need it in the loop.

- Unlike other arguments, arrays in C++ are passed by reference:
  - C++ does not make a copy of the whole array
  - Modifying the array inside the function modifies the original array

Passing arrays to functions (continued)

- Some functions do not modify the input array.

  Example: the above arrayMax function

- Other functions do

  Example: selection sort

Selection Sort function

```
void selectionSort(int A[], int   n)  {
    for(int i=0;i<n;i++) {
       // find the the index minIndex of the smallest element of A[i...n-1]
        int minIndex = i;
        for(int j=i+1; j<n; j++)
            if (A[minIndex] > A[j])
                    minIndex = j;
       // swap A[i] and A[minIndex]
       int temp = A[i];
       A[i] = A[minIndex];
       A[minIndex] = temp;
    }
}
```

Summary

- Functions enable you to divide a program into manageable tasks and allow for code reuse
- To define new functions: function prototype, function definition
- Function are arguments in function definition and heading are called formal parameters
- Expressions, variables, or constant values in a function call are called actual parameters

Summary

- A prototype is the function heading without the body of the function; prototypes end with the semicolon
- Prototypes are placed before every function definition, including main
- A prototype is the function heading without the body of the function; prototypes end with the semicolon

Summary

- In a call statement, specify only the actual parameters, not their data types
- void functions do not return a value
- Unlike other parameters, arrays are passed to functions by reference
- Modifying the array reference inside the function modifies the original array
- In addition to array name, pass the array size to the function