Part 6: Functions and Parameters

Dr. Andi Toce
Lecture Notes for MAC 101 (Introduction to Computer Science)

Last updated / viewed: October 5, 15

Table of Contents

1. What are Functions?

Functions are important building blocks of computer programs. Functions are also referred to as procedures, subroutines or methods. Each function is a group of statements that accomplish a specific task. We will use a number of predefined functions available with C++ and we will also design our own functions.

Components of a function:

return_type function_name (argument_list){
statements
…….
return statement; (if applicable)
}

Example: Write a program that defines and tests a function that calculates the average of two numbers.

AverageFunction.cpp

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

// Function must be declared before being used.
double average(double x, double y);

int main() {

        double a = 0.0;
        double b = 0.0;

        cout << "Enter first number and press ENTER: ";
        cin >> a;
        cout << "Enter second number and press ENTER: ";
        cin >> b;

        // Call the function average().
        cout << "Average is: " << average(a, b) << endl;

        return 0;

} // end main() function

// Average-number function definition

double average(double x, double y) {

        return (x + y)/2;

} //end average() function
```

Output

```
Enter first number and press ENTER:
25.4
Enter second number and press ENTER:
58.2
Average is: 41.8
```

Example: Write a program that defines and tests a factorial function.

FactorialFunction.cpp

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

int factorial(int x);

int main() {

	int number;

	cout << "Enter an integer for the desired factorial: ";
	cin >> number;

	cout << number << "! = " << factorial(number) << endl;

	return 0;

} // end main() function

int factorial(int x) {

	int f=1; //use to store the factorial value

	for(int i=2; i<=x; i++){
		f *= i;
	}

	return f;

} //end average() function
```

Output

```
Enter an integer for the desired
factorial: 5
5! = 120
```

Example: Write a program that defines and tests a function that prints the first n integers.

PrintIntegersFunction.cpp

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

void printAllIntegers(int x);

int main() {

	int number;

	cout << "Enter an Integer: ";
	cin >> number;

	cout << "The first "<< number << " positive integers are listed
below:" <<endl;
	printAllIntegers(number);

	return 0;

} // end main() function

void printAllIntegers(int x) {

	for(int i=1; i<=x; i++){
		cout << i << endl;
	}
} //end printAllIntegers() function
```

Output

```
Enter an Integer: 6
The first 6 positive integers are
listed below:
1
2
3
4
5
6
```

4 [PART 6: FUNCTIONS AND PARAMETERS]

Try now: Write a program LargestIntegerFunction.cpp that defines and tests a function ***largest(….)*** that takes three integers and returns the largest integer.
Example:

Input: Enter three integers:
6
15
8
Output: The largest integer is: 15

Example: A program that checks whether a number is prime.

PrimeNumberFunction.cpp

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

// Function must be declared before being used.
bool prime(int n);

int main() {
        int i;

        // Set up an infinite loop; break if user enters 0.
        // Otherwise, evaluate n from prime-ness.
        while (true) {
                cout << "Enter number (0 = exit) and press ENTER: ";
                cin >> i;
                if (i == 0) // If user entered 0, EXIT
                        break;
                if (prime(i)) // Call prime(i)
                        cout << i << " is prime" << endl;
                else
                        cout << i << " is not prime" << endl;
        }// end while

        return 0;
} // end main

// Prime-number function. Return false if a divisor
// found; otherwise, return true.
bool prime(int n) {
        int i;
        for (i = 2; i < n; i++) {
                if (n % i == 0) // If i divides n evenly,
                        return false; // n is not prime.
                }
        return true; // If no divisor found, n is prime.
} // end prime()
```

Output

```
Enter number (0 = exit) and press
ENTER: 2
2 is prime
Enter number (0 = exit) and press
ENTER: 14
14 is not prime
Enter number (0 = exit) and press
ENTER: 23
23 is prime
Enter number (0 = exit) and press
ENTER:
```

Try now: Optimize the prime-number function. Can we reduce the current number of operations.

Try now: Rewrite **main** so that it tests all the numbers from 2 to 20 and prints out the results, each on a separate line. (Hint: Use a **for** loop, with i running from 2 to 20.)

Try now: Write a program PowerFunction.cpp that defines and tests a function ***power(base, exponent)*** that takes two integers b and x and returns b^x by using only multiplication.
Example:

```
Enter base: 2
Enter exponent: 3
2^3 = 8
```

2. Passing a Parameter by Value or by Reference

There are two ways to pass arguments to a function (by value or by reference). What we have done so far is to pass the value of the parameter. We illustrate both possibilities via examples.

AddOneFunctionByValue.cpp

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

int addOneByValue(int);

int main(){

        int number = 3;

        cout << "Integer number before calling function addOneByValue() is: " << number << endl;
        cout << "Value returned by the function addOneByValue() is: " << addOneByValue(number) << endl;
        cout << "Integer number after calling function addOneByValue() is: " << number << endl;
}

int addOneByValue(int value){
        value++;
        return value;
}
```

```
Integer number before calling function addOneByValue() is: 3
Value returned by the function addOneByValue() is: 4
Integer number after calling function addOneByValue() is: 3
```

AddOneFunctionByReference.cpp

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

int addOneByReference(int &);

int main(){

        int number = 3;

        cout << "Integer number before calling function addOneByReference() is: " << number << endl;
        cout << "Value returned by the function addOneByReference() is: " << addOneByReference(number) << endl;
        cout << "Integer number after calling function addOneByReference() is: " << number << endl;
}

int addOneByReference(int &value){
        value++;
        return value;
}
```

```
Integer number before calling function addOneByReference() is: 3
Value returned by the function addOneByReference() is: 4
Integer number after calling function addOneByReference() is: 4
```

Question: Which are the pros and cons of each method?

3. Default Function Arguments

It is possible to assign default values to each of the function arguments. This allows the caller to omit argument values. If the value is not specified by the caller, the default value will be used. See examples below.

AreaDefaultParameters.cpp

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

int rectangleArea(int length = 2, int width = 1);

int main(){

        cout << "Default rectangle area : " << rectangleArea() << endl;
        cout << "Area with length=10 and default width=1 : " << rectangleArea(10) << endl;
        cout << "Area with length=10 and width=5 : " << rectangleArea(10, 5) << endl;

        return 0;
}

int rectangleArea(int length, int width){

    return length*width;
}
```

```
Default rectangle area : 2
Area with length=10 and default width=1 : 10
Area with length=10 and width=5 : 50
```

InterestDefaultParameters.cpp

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

//Simple Interest Formula
// I = Principal * Rate * Time

double interest(double principle = 1000, double rate = 0.05, double time = 1);

int main(){

        cout << "Interest if we use interest(3000, 0.015, 5): " << interest(3000, 0.015, 5) << endl;
        cout << "Interest if we use interest(3000, 0.015): " << interest(3000, 0.015) << endl;
        cout << "Interest if we use interest(3000): " << interest(3000) << endl;
        cout << "Interest if we use interest(): " << interest() << endl;

        return 0;
}

double interest(double p, double r, double t){

        return p*r*t;
}
```

```
Interest if we use interest(3000, 0.015, 5): 225
Interest if we use interest(3000, 0.015): 45
Interest if we use interest(3000): 150
Interest if we use interest(): 50
```

4. Function Overloading

It is possible to declare different functions using the same function identifier. These functions however have different signatures, typically different data types. The C++ compiler will select the version of the function that matches with the signature.
Example:

OverloadingExample.cpp

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

// Illustrate Function Overloading

int addNumbers(int, int);
double addNumbers(double, double);

int main ()
{
    int xInt = 5;
    int yInt = 7;
    double xDouble = 3.3;
    double yDouble = 2.4;

    cout << "Calling the integer version." << endl;
    cout << "Sum is: " << addNumbers(xInt, yInt) << endl << endl;
    cout << "Calling the double version." << endl;
    cout << "Sum is: " << addNumbers(xDouble, yDouble) << endl;

    return 0;
}

int addNumbers(int first, int second){
        cout << "Inside the integer version." << endl;
        return first + second;
}

double addNumbers(double first, double second){
        cout << "Inside the double version." << endl;
        return first + second;
}
```

Output

```
Calling the integer version.
Inside the integer version.
Sum is: 12

Calling the double version.
Inside the double version.
Sum is: 5.7
```

Try now: Reuse the asterisk diamond program to create a function that takes two values, an integer for the side of the diamond and a char value for the character printed and outputs the corresponding diamond shape. The function has default values, 5 for the side and * for the character.