Functions

In this laboratory session you will:

1. Learn how to write and use functions.
2. Learn how to transfer information to functions by means of parameters.
3. Learn how to retrieve information from a function using a return statement.

Your instructor will tell you which of the proposed experiments you are to perform.

In preparation for this laboratory session, you should read Sections 4.2, 4.4, 5.2, 5.3 and 6.3 of Computer Science: An Overview.
A Simple C++ Function

A C++ program can be presented in a modular fashion by means of program units called functions. You have already met this feature of the C++ language in the form of the functions `main` and `toascii`.

The following is an example of a function definition in C++.

```cpp
void rect_area(int base, int height)
{
    int area;
    area = base * height;
    cout << "The area of the rectangle is ";
    cout << area << endl;
}
```

The first line is the function header. It begins with the type of data returned by the function or the word `void` if nothing is returned. (We'll investigate these ideas later.) Following the type designator is the function's name (in our example `rect_area`), and the function's parameter list enclosed in parentheses. A function name consists of a sequence of letters, underscores, or digits, but may not start with a digit.

The entries in a parameter list are separated by commas. Each entry consists of the data type of the pertinent parameter followed by the name that is used to refer to that parameter within the function. These names are essentially placeholders for values that will be provided when the function's services are requested. They are often called formal parameters to distinguish them from the values actually provided, which are called actual parameters. Our example has two (formal) parameters, `base` and `height`, each of type `int`. If a function has no parameters, then the parenthetical phase following the function's name is left empty.

The body of the function follows the function header and is enclosed in braces. It consists of declarations followed by the procedural part of the function. In the example above, the declarative part consists of the statement

```cpp
int area;
```

that declares the variable named `area` to be of type `int`. The procedural part of the function consists of the statements

```cpp
area = base * height;
cout << "The area of the rectangle is ";
cout << area << endl;
```

that compute and report the value of `area`. As in the case of `main`, variables can be initialized when they are first declared within a function.

Once a function has been defined, its services can be requested by a function call, which consists of the function's name followed by a list of values to be used as the function's actual parameters. The location at which a function is called is often referred to as the calling environment. When a function is called, the following sequence of events takes place: The actual parameters are used to initialize the formal parameters, control is passed to the function, the function performs its task, and control is returned to the calling environment, where execution continues at the point following the function call.

In particular, execution of the following program begins with the function `main`. This function obtains the dimensions of a rectangle from the keyboard and then calls the function `rect_area` to compute and report the rectangle's area.
# include <iostream.h>
void rect_area(int, int);
int main()
{
    int x, y;
    cout << "What is the length of the rectangle?\n";
    cin >> x;
    cout << "What is the width of the rectangle?\n";
    cin >> y;
    rect_area(x, y);
    return 0;
}

void rect_area(int base, int height)
{
    int area;
    area = base * height;
    cout << "The area of the rectangle is ";
    cout << area << endl;
}

Notice that it appears that we have written the function header twice. The first occurrence,
void rect_area(int, int);

is called a function prototype. A function prototype provides enough information to the compiler
so that calls to the function can be properly constructed, even though the compiler has not yet
seen the function itself. (Note that the function prototype does not contain the names of the
formal parameters but merely their types.)

Had we not included the function prototype before presenting the function main, the
compiler would not understand the statement
rect_area(x, y);
since, at that point in reading the program, the name rect_area would not have been defined.
Another solution to this problem would be to present the entire definition of the function
rect_area in the program before presenting the function main. However, most programmers
find it convenient to present the function main at the beginning of a program where it can be
easily found. Moreover, in a later laboratory session we will learn that in some cases a function
cannot be fully defined before it is required in a program and thus a function prototype is the
only solution.

We will discuss function prototypes in more detail in a later laboratory session. For now, just
remember that the function prototype is essentially a copy of the function header followed by a
semicolon and either it or the complete definition of the function must precede any point in the
program at which the function is called.

**Experiment 6.1**

**Step 1.** Execute the program listed above (CP06E01) to make sure it performs correctly. Then,
change the program so that the measurements involved are of type float rather than int.
Summarize your modifications below.

__________________________________________________________________________________
__________________________________________________________________________________
Step 2. Add a function named `tri_area` to the program produced in Step 1 that computes and reports the area of a triangle with the given base and height. Then, modify the function `main` so that it the user to specify the shape (rectangle or triangle) and its dimensions and then calls the appropriate function to compute the area. Summarize your work below.
Experiment 6.2

Step 1. What happens when you try to compile and execute the following program CP06E02)?

```cpp
// This program contains an error.
#include <iostream.h>
int main()
{
    beginning();
    return 0;
}
void beginning();
void beginning()
{
    cout << "It was the best of times,\n";
    cout << "it was the worst of times, ...\n";
    ending();
}
void ending();
void ending()
{
    cout << "... it is a far, far better rest that I go to\n";
    cout << "than I have ever known.\n";
}
```

Step 2. What is wrong with the program?
Step 3. Fix the program and execute it. By the way, what is the syntax for calling a function that has no parameters?

Experiment 6.3

Step 1. Execute the following program (CP06E03) and record the results.

```cpp
#include <iostream.h>
void fun(int, int, int);
int main()
{
    int a = 1, b = 2, c = 3
    cout << "The values of a, b, and c are: " << a << " " << b << " " << c << endl;
    fun(a, b, c);
    return 0;
}
void fun(int c, int b, int a)
{
    cout << "The values of a, b, and c within the function are: " << a << " " << b << " " << c << endl;
}
```
Step 2. Based on the results in Step 1, what is the rule for determining which actual parameters are associated with which formal parameters?

__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________

Passing Parameters

Parameters can be passed (or transferred) to a function in two ways, "by value" and "by reference." To pass a parameter to a function by value means that a copy of the parameter is made and given to the function. In turn, any operations performed by the function will only affect the function's copy of the parameter and not the original data residing in the calling environment. That is, if the function modifies the parameter value, those modifications are performed on the function's copy—not on the data in the calling environment. Thus, when the function has completed its task, the original value will still be intact in the calling environment. This technique ensures that a function cannot cause unforeseen alterations in the program unit that requests its services.

In contrast, passing a parameter to a function by reference means that the function is given access to the parameter as it resides in the calling environment. Thus, any alterations made by the function will be reflected in the calling environment.

Parameters are passed by value in a C++ program. We will learn how to simulate passing parameters by reference in Session 10 when we learn about pointers.

Experiment 6.4

Step 1. Execute the following program (CP06E04), and record the results.

```cpp
#include <iostream.h>
void rect_area(float, float, float);
int main()
{
    float width = 2.5, height = 3.1, area = 0.0;
    rect_area(width, height, area);
    cout << "The area in main is " << area << endl;
    return 0;
}
void rect_area(float width, float height, float area)
{
    area = width * height;
    cout << "The area in the function is " << area << endl;
}"
```
The return Statement

The execution of a function continues until the end of the function (in its written form) is reached or until a return statement is executed. A return statement consists of the word return followed by an expression, as in

```plaintext
return area;
```

and

```plaintext
return(length * width);
```

(Parentheses can be used to enhance readability.) When a return statement is encountered, execution of the function is terminated and the value of the expression is transferred to the calling environment as the value of the function. The type of the value being returned must be the type that is specified at the beginning of the function’s header. For example, if the function `rect_area` is to return a value of type `float`, its definition would begin with

```plaintext
float rect_area(...)
```

whereas if it is to return an integer value, its definition would begin with

```plaintext
int rect_area(...)
```
As an example, consider the following program.

```cpp
#include <iostream.h>
float rect_area(float, float);
int main()
{
    float width = 2.5, height = 3.1;
    float area;
    area = rect_area(width, height);
    cout << "The area is " << area << endl;
    return 0;
}
float rect_area(float width, float height)
{
    return(width * height);
}
```

Here the function `main` calls the function `rect_area` to obtain the area of the rectangle with dimensions `width` and `height`. The result is returned as the value of the function, assigned to the variable `area`, and then printed.

**Experiment 6.5**

Step 1. Execute the following program (CP06E05), and record the results.

```cpp
#include <iostream.h>
int abs_value(int);
int main()
{
    int i;
    cout << "Enter an integer.\n";
    cin >> i;
    cout << "\nThe absolute value of " << i;
    cout << " is " << abs_value(i) << endl;
    return 0;
}
int abs_value(int x)
{
    if (x < 0)
        return(-x); // Terminate the function.
    else
        return(x); // Terminate the function.
}
```
Step 2. Modify the program in Step 1 to find absolute values of values of type float. Summarize your work below.

__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________

Experiment 6.6

Step 1. Execute the following program (CP06E06A), and record the results.

```cpp
#include <iostream.h>
int fun(int);  
int main()
{  
  int k = 1;  
  cout << "The value returned is " << fun(k++) << endl;  
  return 0;
}
int fun(int x)
{
  cout << "The value within the function is " << x << endl;  
  return (x++);
}
```
Step 2. Execute the following program (CP06E06B), and record the results.

```cpp
#include <iostream.h> 
int fun(int);
int main()
{
    int k = 1;
    cout << "The value returned is " << fun(++k) << endl;
    return 0;
}
int fun(int x)
{
    cout << "The value within the function is " << x << endl;
    return (++x);
}
```

Step 3. Explain the differences in the results from Steps 1 and 2.
**Experiment 6.7**

**Step 1.** Execute the following program (CP06E07) several times with different input values and record the results. What does the function `mystery` do?

```cpp
#include <iostream.h>
char mystery(char);
int main()
{
  char letter;
  cout << "Type a letter.\n";
  cin >> letter;
  cout << \nThe mystery is " << mystery(letter) << endl;
  return 0;
}
char mystery(char x)
{
  if ('a' <= x && x <= 'z')
    return (char (x - 32));
  else
    return (x);
}
```

**Step 2.** Explain the coercion that must be performed when executing the `return` statement in the preceding program.
**Step 3.** Based on the model provided by the program in Step 1, write and test a function that converts uppercase letters into lowercase letters. Summarize your work below.
Post-Laboratory Problems

6.1. Write a program that converts Fahrenheit temperatures into Celsius and vice versa. Use two functions—one for converting from Fahrenheit to Celsius and the other for converting from Celsius to Fahrenheit. \( \text{Celsius} = \frac{(\text{Fahrenheit} - 32) \times 5}{9} \)

6.2. Write a program with a modular structure that asks the user for two points on a line, calculates the slope of the line, and determines whether the line is horizontal, vertical, or slopes to the left or right.

6.3. Write a program that makes use of at least six functions to convert kilometers to miles, kilometers to nautical miles, miles to kilometers, miles to nautical miles, nautical miles to kilometers, and nautical miles to miles. Your program should display a menu that allows the user to choose which conversion they would like to do.

6.4. Write a modular program that determines whether or not a year supplied by the user is a leap year.

6.5. Write a program that presents the user with a menu of poets from which the user may select. Your program should then use a switch statement to produce a short poem by the selected poet. Isolate the initial user dialogue in a function. Also use a separate function to print each of the poems.

Next, rewrite the program so that one common function is used to print the poems, with the particular poem to be printed being indicated by a parameter.