A Technical Treatment of Functions and Parameters

A function is a reusable piece of code that can be *called* or invoked by other modules. In its simplest form, a function does not share any data with the function that called it. The technique of *parameter passing* allows a function to give information to another function as it calls it. In C++, there are two types of parameters: *value parameters* and *reference parameters*. When a value parameter is passed to a function, any changes made to the variable inside the function disappear as soon as the function ends. In other words, a value parameter is really a copy of the parameter, not the variable itself. In contrast, a function that receives a reference parameter receives the variable itself, and any changes made inside the function will remain once the function ends.

A function's *declaration* (its first line) defines how many parameters it will expect, what their data types will be, and their parameter type (value or reference). The parameters are given as a *parameter list* in parentheses, listing expected parameters and their types. See the example below for an illustration of this concept.

Every function can optionally "return a value" to the calling function. The return type of a function is determined by the first word in the function's header—`int`, `double`, `char`, or `void` for no return at all. The return value of a function is specified by the `return` keyword, followed by the return value. If a function's body has several `return` statements, the function is terminated the first time a `return` statement is encountered. Therefore, a function can be conditionally terminated in the middle of its body.

In C++, the header (declaration) of a function called by `main` is often placed before the `main` function as a *prototype*. Then, the header plus the code itself would be placed after `main`.

On the next two pages are examples of all the topics discussed above.

Function Example #1

A program that demonstrates different types of functions

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

//*****************Function prototypes*************************
void title();                 //Takes no parameters and no return
int getOneNum();              //Returns a single value
void getData(int &x, int &y); //Reference parameters and no return
int sumData(int x, int y);    //Takes parameters and returns a value
void displayResult(int s);    //Value parameter and no return

//*****************Function main*******************************
int main()
{
  title();                                  //This calls the title function,
                                            //jumping to its code below
  int numOne, numTwo;
  getData(numOne, numTwo);                  //This calls the getData function and
                                            //passes in two variables

  int sum = sumData(numOne, numTwo);        //This calls the sumData function and
                                            //pass in two variables. Then it
                                            //returns with a value to store in sum

  displayResult(sum);                       //This calls the displayResult function
  return 0;
}

//*****************Function definitions************************

void title()
{
  cout << "Want to add some numbers?" << endl;
}

int getOneNum()
{
  cout << "Enter a number: ";
  int n;
  cin >> n;
  return n;
}

void getData(int &x, int &y)
{
  x = getOneNum();
  y = getOneNum();
}

int sumData(int x, int y)
{
  return (x + y);
}

void displayResult(int s)
{
  cout << "The answer is " << s << "." << endl;
}
```

Functions Example #2

A program that demonstrates that one function can be called several times with different variables

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

double half(double p, double q);    // function has 2 double value parameters

int main()
{
    double first, second, avg, x1, y1, x2, y2, xmid, ymid,
           loc1, loc2, loc_half, loc_quarter;

    // Finding an average is to find a number halfway in between
    cout << "Enter two numbers to average: ";
    cin >> first >> second;
    avg = half(first, second);
    cout << endl << "The average of " << first << " and " << second << " is "
         << avg << endl << endl;

    // Finding a midpoint also needs values halfway in between
    cout << "Enter the coordinates of one point (x, y): ";
    cin >> x1 >> y1;
    cout << "Enter the coordinates of another point (x, y): ";
    cin >> x2 >> y2;
    xmid = half(x1, x2);
    ymid = half(y1, y2);
    cout << "The midpoint between those two points is (" << xmid << ", "
         << ymid << ")" << endl << endl;

    // If you find a number halfway and then halfway again,
    // then you have found the point a quarter of the way
    cout << "Enter two locations on the number line: ";
    cin >> loc1 >> loc2;
    loc_half = half(loc1, loc2);
    loc_quarter = half(loc1, loc_half);
    cout << loc_quarter << " is a quarter of the way from " << loc1 << " to "
         << loc2 << " on the number line." << endl;

    return 0;
}


double half(double a, double b)
{
    return ((a + b) / 2.0);
}
```