Functions

1. Functions

A function is a collection of statements that performs a specific task. Some functions that you might have used so far are `pow( )`, `strlen( )`, `rand( )`, and `sqrt( )`. Data can be passed into the function through the use of parameters, and data can be returned from the function through its return value or parameters.

2. Simple example of a function

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

/* the sum function */
int sum (int x, int y) { // declaring the function
    int s;
    s = x + y;
    return s; // returning a value for the function
}

/* the main function */
int main () {
    int s;

    s = sum(5, 4); // calling the function
    cout << "The sum of " <<m<< " and " <<n<< " is "<<s<< endl;

    return 0;
}
```

The line

```
int sum(int x, int y)
```

declares a function called `sum` that takes two integer parameters `x` and `y`. Changes made in the function to the value stored in these two variables `x` and `y` are not reflected back in the main routine. In other words, if the function modifies the value stored in `x`, the calling program **will not** see this new value, but will still see the original value. This parameter passing scheme is referred to as *call by value*. The `int` keyword in front of the function name signifies that the function will return an integer value back to the calling program.

3. Putting functions in a separate file

The following listing shows three function examples: `sum`, `swap` and `changeArray`. The functions are first declared in the myFunctions.h file. The actual definition (coding) of the functions are in the myFunctions.cpp file. You will also have your usual main.cpp file where you have your main program that calls the functions.

Next are the three functions **declared** in the myFunctions.h file.

```
/*
 *  This is the myFunctions.h with just the declaration of the
 *  three functions.
 */

#ifndef _MYFUNCTIONS_
#define _MYFUNCTIONS_

int sum(int x, int y);
void swap(int &x, int &y);
void changeArray(int a[], int arraySize);

#endif
```

The line

```
int sum(int x, int y);
```

declares a function called `sum` that takes two integer parameters `x` and `y`. Changes made in the function to the value stored in the two variables `x` and `y` are not reflected back in the calling program. In other words, if the function modifies the value stored in `x`, the calling program **will not** see this new value, but will still see the original value. This parameter passing scheme is referred to as *call by value*. The `int` keyword in front of the function name signifies that the function will return an integer value back to the calling program.

The line

```
void swap(int &x, int &y);
```

declares a function called `swap` that also takes two integer parameters `x` and `y`. The pointer symbol & placed in front of the variable denotes that changes made to the value stored in that variable are reflected back in the calling program. In other words, if the function modifies the value stored in `x`, the calling program **will** see this new value. This parameter passing scheme is referred to as *call by reference*. The `void` keyword in front of the function name signifies that the function will not return any value back to the calling program.

The line

```
void changeArray(int a[], int arraySize);
```

declares a function called `changeArray` that takes two parameters – an integer array and an integer. Passing an array as a parameter always uses the *call by reference* scheme and the & symbol is not needed. If the function modifies a value in the array, the calling program will see the change. When passing an array, the size of the array must also be passed to the function, otherwise, the function will not know what the array size is.

The three lines starting with the # symbol are called compiler directives. They are *not* C++ commands, but rather commands to tell the compiler how to compile (translate) the file.

```
#ifndef _MYFUNCTIONS_
#define _MYFUNCTIONS_

…

#endif
```

The #ifndef line says that if the user identifier that follows (in this case, _MYFUNCTIONS_) has not already been defined then do the code inside the block until the #endif line. So the first time that the compiler sees this, the user identifier has not been defined, so it will look at the code inside the block, which includes the #define line to define the user identifier. The next time the compiler sees this code, the user identifier has now been defined, so it will skip the whole block. This way, even if you #include this .h file more than once, you will not get a compilation error that says you are re-declaring the functions and/or variables.

Next are the three functions **defined** in the myFunctions.cpp file.

```
/*
 *  This is the myFunctions.cpp file containing the definition (actual
 *  coding) of the three functions.
 */

#include "myFunctions.h"

/*************************************************************
 * the sum function */
int sum(int x, int y){
    return x + y;
}

/*************************************************************
 * the swap function */
void swap(int &x, int &y){
    int temp;
    temp = x;
    x = y;
    y = temp;
}

/*************************************************************
 * the changeArray function */
void changeArray(int a[], int size){
    for (int i = 0; i<size; i++) {
      a[i] = a[i]*2;
    }
}
```

Next is the main program showing some examples of how the three functions are called (used).

```
/*
 *  This is the main.cpp file with code to call the functions
 */

#include <iostream>
#include "myFunctions.h"
using namespace std;

int main (int argc, const char * argv[]) {
    int m=5, n=9;
    int a[5]={1,2,3,4,5};
    int s;
    int i;

    // sum function example
    s = sum(m,n);
    cout << "The sum of " <<m<< " and " <<n<< " is "<<s<< endl<<endl;

    // swap function example
    cout << "Before swapping m is " << m << " and n is " << n << endl;
    swap(m,n);
    cout << "After swapping m is " << m << " and n is " << n << endl;

    // changeArray function example
    cout << endl << "The original array is: ";
    for(i=0;i<5;i++) {
      cout << a[i] << " ";
    }

    changeArray(a,5);

    cout << endl << "The changed array is: ";
    for(i=0;i<5;i++) {
      cout << a[i] << " ";
    }

    return 0;
}
```

Sample output:

```
The sum of 5 and 9 is 14

Before swapping m is 5 and n is 9
After swapping m is 9 and n is 5

The original array is: 1 2 3 4 5
The changed array is: 2 4 6 8 10
```

Notice in the main program that it prints out the array twice, once before the changeArray call and once after the changeArray call. Since you are duplicating the code for printing the array, this immediately suggests that you should move the code for printing the array into a function. Write a function named printArray with two parameters, the array and the size of the array, to print out the contents of the array. Replace the code in the main program to call this function to print out the array.

How would you include the two different messages (`"The original array is: "` and `"The changed array is: "`) that are printed inside the function?

Just like with naming variables, you want to use meaningful names for your functions. A good practice and convention to follow is to use action words for a function name. The action words will concisely describe what the function will do. For example, if I want to write a function to count the number of words in a given passage, which of the following names best describes what the function does?

1. words
2. countWords
3. wordCount

Using the name "words" is totally meaningless. "countWords" is the best. Do you see the difference in the meaning between countWords and wordCount? countWords is an action that means "count the number of words", whereas wordCount means "the number of words" you have which is not an action.

4. Exercises (Problems with an asterisk are more difficult)

In all of the following questions, you need to write the function and the main program to test out the function.

1. Implement the three functions and the main program from section 1 above and make sure that they work.

2. Modify the sum function from above to assign new values to x and y inside the function. Convince yourself that the changes you made to x and y inside the function are not reflected in the outside variables m and n by printing out m and n.

3. Remove both of the &'s in the swap function from above. Did the two numbers get swapped?

4. Implement the following function:

```
void changes(int &x, int y){
   x = x * 2;
   y = y * 2;
}
```

   In the main routine, type in:

```
   int m=3, n=7;

   cout << "Before calling changes, m is " << m << " and n is " << n
<< endl;
   changes(m,n);
   cout << "After calling changes, m is " << m << " and n is " << n
<< endl;
```

   Which variable got changed? Which variable did not change? Why?

5. Write a function named `timesTen`. The function should have an integer parameter named `number`. When `timesTen` is called, it should display the product of `number` times 10. The function does not return anything.

6. Write a function named `timesTen`. The function should have an integer parameter named `number`. When `timesTen` is called, it calculates the product of `number` times 10. The function returns this product.

7. Write a function that prints out the larger of two numbers. The two numbers are passed into the function. The function does not return anything.

8. Write a function that determines the larger of two numbers. The two numbers are passed into the function and the function returns the largest number.

9. Write a function that calculates the average of two numbers. The two numbers are passed into the function and the function returns the average.

10. * Write a function named `twoSine` that calculates the sine of two numbers. The two numbers are passed into the function and the function returns two numbers which are the sine of these two numbers. You can use the trig function sin(x). You need to #include <math.h>.

11. * Write a function that calculates the average of numbers in an array. The array is passed into the function and the function returns the average.

12. **** Write a function to sort numbers in an array. The array is passed into the function and the function returns the sorted array.

