Programming Technique

Lab 7 – Function

1 Expected outcomes

- Know how to declare function correctly based on a task.
- Know how to write function definition.
- Understand the difference among different parameter passing methods.
- Know how to use array and pointer with a function.
- Know how to use recursion to solve a practical solution.

2 Mandatory exercises

Exercise 1. Given the following program:

```
#include <iostream>
using namespace std;
int main(){
	int a = 9;
	cout << "before: " << "a = " << a << endl;
	change1(a);
	change2(&a);
	cout << "after: " << "a = " << a << endl;

	return 0;
}
void change1(int& x){
	x = 100;
}
void change2(int* x){
	*x = 100;
}
```

If you try to compile the program, it will show an error telling that identifiers of `change1(a)` and `change2(&a)` are not found.

a) One way to fix this error is to move `change1` and `change2` above the main function. Is there another way to fix it?

b) If we remove the ampersand symbol (`&`) in the declaration of `change1`, what will happen to `change1(a)`.

c) Replace `int a = 9;` to `int* a = new int(9);` and insert `delete a;` before `return`.

Change `change(a);` and `change(&a)` so that the function will change the value a is pointing to (currently, it's 9).

d) If we replace `change1` and `change2` by `change`, will there be a compilation error because there are two functions of the same name? What is this concept called?
e) In addition to the d), add another `change` function that use pass-by-value method. Will there be an error? Why?

Exercise 2.

a) Define a function using the following guidelines:

- Function name: use a name of your choice.
- Input parameters (the function only reads these parameters and does not change them): an array of type double => you must decide the appropriate parameter passing method here so that **the array can't be changed in the function**.
- Output parameters (the function must write the results to these parameters before finishing): average value, standard deviation and an array storing a histogram just like Assignment 1.
    - You can create more utility functions to calculate the average value, standard deviation and the histogram.
    - You must use the appropriate parameter-passing method for these output parameters.
    - To store the histogram, this function assumes that the caller function had prepared an array of the correct size to store this histogram. For example, if you want to calculate a histogram of 10 elements, the caller must initialize an array of 10 elements and pass it to this function.
- Return type is void.

b) Write a main function that does the following tasks:

- Generate random data according to the normal distribution, just like the previous lab: http://www.cplusplus.com/reference/random/normal_distribution/
- Call the function defined in a) to calculate the average value, the standard deviation and the histogram.

Exercise 3. Assume that we have the following 2D matrix struct:

```
typedef struct{
	double data[N][N];
}  Matrix;
```

This struct stores a 2D N x N matrix. You must define N using macro. Create the following operators: addition (+), subtraction (-), multiplication (*) between:

- **A matrix and a matrix**: you should know how to add, subtract and multiply 2 matrices.
- **A matrix and a scala (a single number):** you can define your own calculations in this case. For example, an addition between a matrix and a scala equals to every element of that matrix added with the scala.

After writing such operators, you can use the following block of codes (or write your own) to check if the operators were defined correctly:

```
Matrix a = {/*initialization*/} // You can use a random generator here
Matrix b = {/*initialization*/} // You can use a random generator here
Matrix c, d;

c = a + b + 0.5;
cout << c << endl;
d = 1.5 + a*c;
cout << d << endl;
```

Guidelines:

- Use the #define macro to define the size of the array.
- Using operator overloading to define addition, subtraction, multiplication. Here is an example for the add operator:

```
Matrix operator+(Matrix x, Matrix y)
```

- For operators between a matrix and a scala, we can also use the same overloading method. However, we need to define two functions in this case because the scala can be on the left or the right of the operator:

```
Matrix operator+(Matrix x, double y)
Matrix operator+(double y, Matrix x)
```

- Use overloading to create an output function for a matrix:

```
ostream& operator<<(ostream &os, Matrix x)
```

Exercise 4. Similar to exercise 3, we can define operators such as +, -, * and / for complex numbers.

Guidelines:

- Use struct to create a data type for complex number. This struct should have two parts: real (double typed) and imaginary (double typed).
- Use overloading to define +, -, * and / for this newly-created data type.

- Use overloading to create a print function for complex numbers.

Exercise 5. Use recursive technique to:

a) Calculate the sum and the product of every elements in an array.
b) Print all elements in an array:
- In a forward direction.
- In reverse.

For example, if the array is {1,2,3,4,5}, you should be able to print 1 2 3 4 5 and 5 4 3 2 1.

c) Evaluate a N-th order polynomial. The coefficients of the polynomial should be stored in an array.
d) Convert a number in binary to decimal.
e) Calculate all numbers in a binary tree.
f) Check if all trees in the binary tree satisfy the following property: all values of the left sub-tree are smaller than the root, all values of the right sub-tree are bigger than the root.

Exercise 5. Define functions from Exercise 5 to 10 in the previous lab (pointer). Note that there should be more than one function for each exercise. You should be able to apply all learned programming techniques to create solid solutions.