Homework #1

Before starting with the first programming assignment, you need to get familiar with the C++ programming environment that we will use throughout the course. We will be using the Linux machines at the College of Engineering (CoE), which can be accessed physically at the CoE labs, or remotely through an SSH client. You can find more information on how to access the CoE machines on the course syllabus.

For this course, we will assume a basic understanding of the following UNIX command-line tools:

- **ls** – List the content of the current directory.
- **mkdir** – Create a directory.
- **pwd** – Return the current path.
- **cd** – Change current working directory.
- **nano** – Basic word editor for plain-text files.
- **vi** – Alternative word editor, more powerful than **nano**, but harder to use for the first time.
- **cp**, **mv**, **rm** – Copy, move, or remove files/directories.

If you are not familiar with these commands, you can find a simple UNIX tutorial here:

http://www.coe.neu.edu/computer/UNIXhelp/

Let us start writing a simple *hello world* program in C++. The source code will be created in file `hello.cc` in directory `~/fce/hw1` (character "~" in directories is used to refer to the home folder). Type the following commands:

```
$ mkdir fce
$ cd fce
$ mkdir hw1
$ cd hw1
$ nano hello.cc
```

Type the following code:

```
#include <iostream>

int main()
{
        std::cout << "Hello, world!" << std::endl;
        return 0;
}
```

The header file `<iostream>` provides a standard name space `std` with a pre-defined object `cout`. This object can be used with operator "`<<`" followed by a string in order to dump text into the standard output. We will present all details in class needed to thoroughly understand this code. For now, we just need to know that this is the pattern used to write text into the screen. The following command can be used to compile the code:

```
$ g++ hello.cc -o hello
```

This command takes file `hello.cc` as an input and produces an executable file named `hello`. The program can be executed with the following command:

```
$ ./hello
```

If you want to import the source file into your local machine (e.g., in order to upload them on Blackboard as part of your homework submission), you need a tool to transfer files through an SSH connection. On Windows, you can use WinSCP, available for free at http://winscp.net. On Linux/Mac, you can use the following `scp` command:

```
$ scp gateway.coe.neu.edu:fce/hw1/hello.cc .
```

The command above copies file `hello.cc` from the file system at *gateway.coe.neu.edu* into the current directory of your local machine.

Once our first program is running correctly, let us implement the insertion sort algorithm presented in class. This program is composed of a function running the algorithm itself, a function to print the unsorted and sorted versions of the vector, and a main program calling these functions.

```
#include <iostream>

void insertion_sort(int v[], int n)
{
        int value;
        int i, j;

        for (i = 1; i < n; i++)
        {
                value = v[i];
                j = i - 1;
                while (j >= 0 && v[j] > value)
                {
                        v[j + 1] = v[j];
                        j--;
                }
                v[j + 1] = value;
        }
}


void print_vector(int v[], int n)
{
        int i;

        std::cout << "Vector:";
        for (i = 0; i < n; i++)
                std::cout << " " << v[i];
        std::cout << std::endl;
}


int main()
{
        int v[] = { 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 };
        print_vector(v, 10);
        insertion_sort(v, 10);
        print_vector(v, 10);
}
```

This homework assignment, as well as the material presented in class, is based on the assumption that you have a good understanding of the code presented above. For further reading, the tutorial found in

http://www.cplusplus.com/files/tutorial.pdf

offers a good introduction to C++. The material considered as a prerequisite for the course is well captured in the first four chapters of this tutorial, titled *Introduction*, *Basics of C++*, *Control Structures*, and *Compound Data Types*.

Question 1 (4 pt.)

Write a program that sorts a vector using the *merge sort* algorithm. The structure of the program can be the same as the one shown in the previous example for *insertion sort*. The program should print the vector before and after it is sorted. Make sure that your C++ program runs correctly on the COE machines, and include the code in your answer.

Upload your code in a file named `q1.cc` as part of your submission on Blackboard. This file should compile and run without errors on the CoE Linux machines.

Question 2 (3 pt.)

Modify the merge sort program in such a way that it accepts the size of the vector as a command-line argument. You should be able to run the program as follows:

```
$ ./merge-sort 20
```

Here are some tips:

- First, you need to modify the header of the `main()` function to collect the arguments given in the command line: `int main(int argc, char *argv[])`. Within the body of `main()`, you can access the number of arguments through variable `argc`, while `argv[0]`, `argv[1]`, ... contain strings with the arguments themselves.

- To convert a numeric value in a string into an integer, you can use function `atoi()`. This function is declared in header file `<cstdlib>`, which you need to add in the beginning of your code using an `#include` directive.

- The memory for the vector must now be allocated dynamically. For this purpose, you need to declare `v` as a pointer to an integer "`int *v`". Then, allocate memory for the vector with "`v = new int[size];`", where `size` is the number of elements. At the end of the program, free the memory associated with the vector with "`delete v;`". Chapter *Compound Data Types* of the C++ tutorial presented above contains the details of these calls.

- The elements of the vector must now be initialized dynamically as well. You can do this with a `for` loop that initializes it in a reverse order. For example, a vector of 20 elements would be initialized as { 19, 18, 17, …, 0 }.

Upload your code in a file named `q2.cc` as part of your submission on Blackboard. This file should compile and run without errors on the CoE Linux machines.

Question 3 (3 pt.)

Even though *merge sort* has a lower asymptotic cost than *insertion sort*, the real cost of merge sort's recursion for small problem sizes can be higher than sorting that same small vector iteratively with insertion sort. As a trade-off, we could use merge sort to solve large problem instances, and switch to insertion sort once the sub-problem is small enough.

Write a hybrid sort algorithm that uses merge sort in the generic case, but solves sub-problem instances of size smaller than 10 with insertion sort. The program should print the input vector before and after it is sorted.

Upload your code in a file named `q3.cc` as part of your submission on Blackboard. This file should compile and run without errors on the CoE Linux machines.

Question 4 (open-ended, +2pt.)

Edit your insertion sort, merge sort, and hybrid sort programs to avoid printing the vectors when the problem size is greater than 20. Now measure the execution time for increasing problem sizes, and compare these times with the asymptotic costs studied in class, plotting the results if necessary. You can obtain the execution time of a command by prepending it with command `time` as follows:

```
$ time ./merge-sort 100000
```

Upload your answer to this question in a PDF file named `q4.pdf` that includes your written answer, and any optional additional plots.