資料結構與程式設計

(Data Structure and Programming)

104 學年上學期複選必修課程 901 31900

Homework #1 (Due: 9:00pm, Sep 22, 2015)

[Note]

1. You need to submit this homework in order to get the 授權碼 to register the course.
2. Although the deadline is on Tuesday Sep. 22, which is the second week of the semester, you are advised to submit it at your earlist convenience in order to get the 授權碼 earlier.
3. The instructor will quickly go over your submission of homework and qualify it ASAP. Plagiarism will be checked. You must pass the plagiarism check and have a "reasonable" submission in order to qualify for the 授權碼. (Note) "Reasonable" is a qualitative judgement, not a quantitative measurement, meaning that you should not turn in blank homework or something that is not abide by our requirements. Homework score is not a decision factor for it.
4. Please do pay attention to the homework submisstion rules at the end of this document. Failure to follow the rules will result in disqualification of getting 授權碼.
5. It's very likely you are NOT that familiar with some advanced C++ features required in this homework. For example, dynamic array, operator overloading, functional object, template class, etc. Nevertheless, "self-learning" by researching on the internet and then figuring out the solution is one of the key spirit in this course. You need to get used to it in order to enjoy the class.
6. The copyright of these homework problems belongs to the author, Prof. Chung-Yang (Ric) Huang, in National Taiwan University (NTU). These problems are used for the class "Data Structure and Programming" in NTU only. Use in any form without the permission of the author will be treated as violation of the copyright law. For permission requests, send e-mail to cyhuang@ntu.edu.tw addressed "Attention: Use of DSnP problems".

---

1. In this course, we require that all the homework be done in a Linux compatible programming environment. Therefore, we would like to make sure you have it before the class starts.

   There are several ways to obtain a Linux compatible environment, for example: (i) Have a computer that runs Linux OS, (ii) Have a dual-boot computer that you can choose to boot Linux when powering it on, (iii) Install a virtual machine (e.g.

VirtualBox) on a non-Linux environment (e.g. Windows) and then install the Linux OS on the virtual machine, (iv) Remotely login (i.e. telnet or ssh) to a Linux workstation (e.g. some server in your lab), and (v) Use "terminal" on Mac.

In this problem, you are asked to screenshot a "terminal" on your Linux environment showing the results of the following commands:

(a) whoami

(b) df

(c) ifconfig | grep "ether "

(d) ifconfig | grep "inet " | grep -v 127.0.0.1.

Feel free to mozaic parts of the screenshot if you have any security concern.

Name the file of this problem as "p1.xxx", where xxx can be any file extension as you choose. If there are multiple files, create a sub-directory "p1" and name the files properly with the prefix "p1".

2. In this problem, you are asked to read in a ".csv" file and perform some operations and/or statistics on its data.

   A .csv file is commonly used to represent a table of data where cells are separated by comma ',' and rows are separated by a newline symbol (i.e. ctrl-m, `^M`, ASCII code = 13).

   For example, for the following table ---

1	2
3	
5	6
	8

   Saving it as a .csv file, and you will see (e.g. in a text editor):

```
1,2^M3,^M5,6^M,8^M
```

   Implement the following sub-problems in a single C++ program and name the file as "p2.cpp". If multiple files are involved, create a sub-directory "p2" and place the files in it. The filenames should also be named properly with the prefix "p2".

   Note that: (1) All the cells in the table contain **integeral** data only and their values are between -99 and 100, if not empty. You don't need to handle other data type. (2) The number of columns won't be changed after the table is read in. However, user may add new rows to the table and thus the number of rows will increased. (3) Data won't be removed once it is read into the table. (4) You don't need to worry about input error handling. That is, you can assume that the input is always in a correct format.

   Key reviews: *Dynamic array, Standard Template Library (STL) vector, functional object, operator overloading, template function.*

   (a) Define a `class Data` to represent the data of a row. Since the number of columns is not known in the beginning and will be fixed after initialization, please use "`int *`" as its data member. That is,

```
class Data {
   const int operator[] (size_t i) const;
   int& operator[] (size_t i);
private:
   int  *_cols;
};
```

Note that the overloaded operator [] is to access the data in "`_cols`". There is a non-const version (i.e. return by reference) so that users can write data into the object using the [] operator. Please feel free to define more member functions in order to operate on the data. DO NOT change "`_cols`" to *public* and DO NOT use "`friend`" to bypass the data encalpusulation.

To define a table, please use STL dynamic array "`vector<Data> table`" in *main()* so that the number of rows can be increased on demand. With such design, to access the i^{th} row in table, you can use "`table[i]`", and to obtain the value of the j^{th} cell of the i^{th} row, you can use "`table[i][j]`".

(b) In the beginning, your program should prompt the message to ask for the input of the .csv file.

```
Please enter the file name: hw1_example.csv
Please enter the number of rows and columns: 4 2
```

Once the .csv file is properly read in, print the message (change the filename accordingly):

```
File "hw1_example.csv" was read in successfully.
```

and leave the cursor on the screen waiting for the user to enter any of the following commands (see the following sub-problem).

(c) The command "`PRINT`" prints the entire table content in a tabular format. For example, for the example in the previous page, you should see:

```
1   2
3
5   6
    8
```

Please use "`setw(4)`" and "`right`" for "`cout`" to control the output format.

(d) The command "`ADD`" adds a new row to the end of the table. The added data are provided as arguments to this command and are separated by space. For the column with intended null data, put a '-' symbol to represent it. Therefore, there must be exactly *#columns* arguments for this command.

For example ---

```
ADD 9 10   // add a new row with data 9, 10.
ADD - 12   // add a new row with data , 12.
ADD -3 -   // add a new row with data -3, .
```

Nothing will be printed for this command. Use "`PRINT`" command to check the result.

(e) The commands "SUM", "AVE", "MAX", "MIN" and "COUNT" compute the summation, average, maximum, minimum, and distinct count of data in the specified column. Clearly, there will be only one argument (i.e. the column index) for these commands. Null cells are ignored (not treated as '0') here.

For example ---

```
SUM   3   // compute the summation of data in column #3.
COUNT 2   // count the number of distinct data in column #2.
```

The output of these commands should be as the following ---

```
The summation of data in column #3 is 38.
The distinct count of data in column #2 is 10.
```

(f) The command "SORT" sorts the data in ascending order with respected to the specified series of column indices (note: column #0 is the first column). Null cells are assumed to be greater than any number in this column.

For example, for the table ---

3	
3	2
	6
-1	8

```
SORT 0 1   // sort the data ascendingly first by column #0,
           // and then by column #1
```

The sorted table will be ---

-1	8
3	2
3	
	6

Please note that in this sub-problem you are asked to use "sort()" algorithm in STL:

```
template <class RandomAccessIterator, class StrictWeakOrdering>
void sort(RandomAccessIterator first, RandomAccessIterator last,
        StrictWeakOrdering comp);
```

to sort the data, and define a functional object in the following format:

```
struct SortData {
   bool operator() (const Data& d1, const Data& d2);
   void pushOrder(size_t i);
   vector<size_t>  _sortOrder;
};
```

as `class StrictWeakOrdering` in template argument list.

In other words, in your program you should first create a functional object of `struct SortData`, and then push the sorting column indices to this object by the member function `SortData::pushOrder()`, according to the arguments in the "SORT" command. Finally, pass this functional object to the `sort()` algorithm and the data in the table will be sorted automatically.

Notes: Please pay attention to the homework rules below and announcements on the website. Failure to abide by the rules may result in disqualification of getting 授權碼.

[Homework Rules]

1. Open a text file "README" and provide the following information:

   姓名：

   系級：

   e-mail：

   Once the submission is qualified, we will send the 授權碼 to the e-mail address above. Disqualified homework will also received an e-mail notice to the above address. Therefore, please make sure your e-mail is functionable.

2. Put all your files and directories (README, p1, p2) in a directory named "yourID_hw1", where "yourID" is your school ID in lower case and numbers (e.g. b04901888_hw1). Compress the directory by the command:

   ```
   tar zcvf yourID_hw1.tgz yourID_hw1
   ```

   You should remove the executable and object files (if any) before submission to reduce the file size. Please make sure the file size of your homework is no larger than 10MB. Those larger than 10MB will be disqualified.

3. Send your homework to cyhuang@ntu.edu.tw titled:

   "Attention: yourID_hw1 for DSnP 授權碼".

4. Any update, problem fix, and announcement will be posted on FaceBook group NTU_DSnP: https://www.facebook.com/groups/NTUDSnP/. Please follow it closely.