資料結構與程式設計

(Data Structure and Programming)

108 學年上學期複選必修課程 901 31900

Homework #1 (Due: 9:00pm, Tuesday, Sep 24, 2019)

Objectives

1. Getting familiar with Linux programming environment.

2. Review basic C++ syntax and concepts.

3. Implement a simple JSON reader.

Problems

1. In this course, all homework must be done in a Linux-compatible programming environment (e.g., Ubuntu or MacOS). Please ensure you have such an environment set up before the class begins.

   There are several convenient ways to obtain a Linux-compatible environment:

   (i) **Use a native Linux system** – Install Ubuntu, Fedora, or another distribution on your primary machine.

   (ii) **Dual-boot** – Set up your computer to boot either Linux or your existing OS.

   (iii) **Virtual Machine** – Run Linux inside a virtual machine on Windows or macOS using tools like VirtualBox, VMware Workstation, or KVM/QEMU.

   (iv) **Windows Subsystem for Linux (WSL2)** – If you use Windows 10 or later, you can install WSL2 and run a full Linux distribution alongside your Windows environment without a separate VM.

   (v) **Cloud or Remote Access** – Access a Linux system on a remote server (via SSH) or use cloud-based development environments such as GitHub Codespaces, GitPod, or Google Cloud Shell.

   (vi) **MacOS Terminal** – If you have a Mac, the built-in terminal provides a Unix-like environment that works for our homework assignments. You can also use Homebrew to install additional tools.

   In this problem, we will check if you have "g++" installed and if it supports C++17. Please follow the steps below, screenshot your "terminal", rename and save the file under the p1 directory.

   (i) In terminal, execute "cd p1" // change directory to "p1"

   (ii) Type "make" // it will evoke the "Makefile"

If it is executed correctly, you should see:

```
Compile success!!
Fruit: kiwi, Count: 5
Fruit: apple, Count: 3
program tested successfully !!
compiling using <yourCompiler>
compiler check pass !!
```

If you see any problem on C++17 compatibility or g++ compiler, fix it before you move on.

(iii) Screenshot your terminal, and save the screenshot as "p1.xxx", where "xxx" is the file extension of the image (e.g. p1.jpg).

(iv) Make sure "p1.xxx" is stored properly under the p1 directory.

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

Execute the above steps in the same terminal so that the screenshot can be fitted into a single image file.

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

   A JSON file is a lightweight data-interchange format. It is easy for humans to read and write, and it is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. Please refer to JSON official site for its complete spec.

   However, in this homework, to simplify the task, we will set the following limitations on our test JSON file format:

   - ✓ Each file contains one and only one JSON object.
   - ✓ Each JSON object is enclosed by opening and closing curly braces '{' and '}' located in the begin and end of the file, respectively, each in one line. No extra characters, except perhaps white spaces (e.g. ' ', '\t', '\b', '\r', '\n'), are in these two lines.
   - ✓ Within the curly braces is(are) zero, one, or more element(s). Each element is a key-value pair, where the key is of type "string" and is enclosed by double quote " ", and the value is a positive or negative C++ integer (i.e. type "int"). No object, array, string, or Boolean value will appear in our testcases.
   - ✓ Key is composed of one or more characters. The length of the key is virtually unlimited. However, it should be able to read in as a C++ "string".
   - ✓ Key cannot be empty nor white-space string.
   - ✓ At least a white space is inserted between the key and the column symbol ':', and between the column symbol ':' and the value.
   - ✓ Each element, except for the last one, is followed by a comma ','.
   - ✓ White spaces appeared outside the key string are ignored.
   - ✓ No two elements in a JSON object can have the same "key".

Valid examples of JSON files in this homework are like:

```
[Example-1]
{
   "Ric" : 100,
   "John" : 50
}

[Example-2]
{
"aa" : 3 ,
  "bbb" : 4
}

[Example-3]
 {
  "asf" : -123,
"qQ"     :  88  ,
   "j" : 0
 }

[Example-4]
{

}
```

A JSON file generator is provided in the directory "p2/jsonGen" for you to generate JSON testcases. Simply type "make json" in "p2" directory and you will get a generator called "jgen". Execute "jgen" followed by a number (i.e. the number of elements) and you will see a randomly generated JSON file on the screen (stdout). Use redirect operator '>' to save the output to a file (e.g. ./jgen 100 > 100.json). Please note that the generated JSON file contains keys of length smaller than 10 and with lower-case letters only, and its values are between -999 to 999. Please refer to the above rules/limitations for our real testcases.

You can assume that our testcases are all syntactically correct. You don't need to handle format errors.

Implement the following sub-problems using C++ in the subdirectory "p2" and write your codes in the files as specified in each sub-problem. To compile the codes, type "make" under "p2" directory to evoke "Makefile" to generate the executable "p2Run". In principle, you don't need to create new ".h" or ".cpp" files. However,

if you intend to do so, please make sure you understand how to incorporate the new files into the Makefile and the executable can be successfully generated by "make".

Under the success of compilation, type "./p2Run" to run the program.

~~We also provide reference programs for both Linux (ubuntu16) and Mac (macOS Mojave 10.14.3) environments as "p2Run-linux" and "p2Run-mac" under the "ref" directory, respectively. You should run them to compare their outputs with your results.~~

(a) Define the *main()* function in "p2Main.cpp" and declare a JSON object "`Json json`" in it.

In the beginning, your program should prompt the message to ask for the input of the .json file.

```
Please enter the file name: test1.json
```

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

```
File "test1.json" was read in successfully.
```

and you should see "`Enter command: `" on the screen waiting for the user to enter any of the commands as defined in sub-problems (c) – (f). Since we assume that there is no syntax error in the .json file, there is no need to handle the parsing error. Therefore, if your program does not halt and print out the above message, it is definitely something wrong in your code.

You can also assume that all the elements have unique keys.

(Hint) Suggested readings: iostream, fstream, string

(b) Define `class Json` and `class JsonElem` in the file "p2Json.h" to represent the data of a JSON object and its elements, respectively. Since the number of elements in a JSON object is not known in the beginning and may increase with the "ADD" command later, please use dynamic array "`vector<JsonElem>`" to record the elements. That is,

```
class Json {
  public:
    // TODO: define constructor & member functions on your own
    bool read(const string&);

  private:
    std::vector<JsonElem> _obj; // DO NOT change this definition.
                                // Use it to store JSON elements.
};
```

And for each element, define `string` and `int` as its data members:

```
class JsonElem {
  public:
    // TODO: define constructor & member functions on your own
    JsonElem() = default;
    JsonElem(std::string key, int value)
    : _key(std::move(key)), _value(value){}

    friend std::ostream &operator <<
```

```
    (std::ostream&, const JsonElem &);

  private:
    std::string _key;    // DO NOT change this definition.
                         // Use it to store key.
    int         _value;  // DO NOT change this definition.
                         // Use it to store value.
};
```

(Hint) Suggested readings: class / constructor, dynamic array, return by reference, operator overloading, const method/variable

For the commands in sub-problems (c) – (f) below, implement them as member functions of `class Json` and name them as "`print()`", "`add()`"… etc. accordingly. Evoke them in "p2Main.cpp" such as `json.print()`. Note that the commands are case sensitive (all uppercase letters) and you don't need to handle errors when parsing the commands/options.

(c) The command "`PRINT`" prints the entire JSON object. For example, for the example "test1.json" in the hw2 directory, you should see:

```
{
  "Ric" : 100,
  "John" : 50
}
```

Note that each element is printed in a row with indentation of 2 space characters in the beginning, followed by the key (enclosed by double quote `" "`), a space character, a column symbol, a space character, and the value. Put a comma ',' after the value if it is not the last element. For ease of grading, please DO NOT put any extra white space anywhere in the printout.

(d) The commands "SUM", "AVE", "MAX", and "MIN" compute the summation, average, maximum, and minimum of the values of different elements in the JSON object. Clearly, there will be NO argument for these commands.

For example, for the example "test3.json" in the hw2 directory, you should see:

```
Enter command: SUM
The summation of the values is: -35.
Enter command: AVE
The average of the values is: -11.7.
Enter command: MAX
The maximum element is: { "qQ" : 88 }.
Enter command: MIN
The minimum element is: { "asf" : -123 }.
```

For the `AVE` command, compute and round the result up to the first decimal digit (Use `fixed` and `setprecision(1)` to format the printing).

If there is no element in the JSON object, you should report:

```
Error: No element found!!
```

(e) The command "`ADD`" adds a new element (i.e. key-value pair) to the JSON object. The added key and value are provided as arguments to this command and are separated by space(s).

For example ---

```
ADD Mary 70   // add a new element "Mary" : 70
```

When an element is added successfully, nothing will be printed for this command. However, when there is already an element with the added key, print out an error (for example):

```
Error: Element with key "Mary" already exists!!
```

and do not insert it.

Use "`PRINT`" command to check the result.

Please note that the added element affects the JSON object in your program only. It will not change (i.e. write back to) the read-in file.

(f) The command "`EXIT`" exits the program.

Notes: Please pay attention to the homework rules below and announcements on the website/FB page. Failure to abide by the rules may result in deduction of your homework score.

[Homework Rules]

1. Name your files as specified in each problem. Put all your files and in a directory named "yourID_hw1", where "yourID" is your school ID in lower case and numbers (e.g. b06901888_hw1). Compress the directory by the command (on Linux or Mac Terminal):

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

   You should remove the executable and object files (if any) before submission to reduce the file size (e.g. type "make clean" under p2).

2. Submit your homework on Ceiba before due date.