CPP Problem Design Example

Subject: Elevator Simulator

Contributor: Yuan, Wei-Cheng

Main testing concept: Pointer

Basics

■ C++ BASICS
■ FLOW OF CONTROL
□ FUNCTION BASICS
□ PARAMETERS AND OVERLOADING
□ ARRAYS
□ STRUCTURES AND CLASSES
□ CONSTRUCTORS AND OTHER TOOLS
□ OPERATOR OVERLOADING, FRIENDS,AND REFERENCES
□ STRINGS
■ POINTERS AND DYNAMIC ARRAYS

Functions

□ SEPARATE COMPILATION AND NAMESPACES
□ STREAMS AND FILE I/O
□ RECURSION
□ INHERITANCE
□ POLYMORPHISM AND VIRTUAL FUNCTIONS
□ TEMPLATES
□ LINKED DATA STRUCTURES
□ EXCEPTION HANDLING
□ STANDARD TEMPLATE LIBRARY
□ PATTERNS AND UML

Description:

In this task, please write a program to simulate the operations of a simple elevator as stated in the following:

1. There is only one button at each floor to call the elevator to your floor and open the door. There are no other buttons inside the elevator for additional input commands.
2. When the elevator is going up to its destination,
   (1) Pushing a button on a higher floor comparing to where the elevator currently is can make it keep going up and add the higher floor into its destination list. However, if the floor is already in the destination list, it is ignored.
   (2) Pushing the button when the elevator is on the same floor would make the newly inputted command neglected.
   (3) Pushing a button of a lower floor adds the floor to its destination list. The elevator first goes up to all higher destinations then goes down to the targeted floor. However, if the floor is already in the destination list, it will be ignored.
3. When it goes down to a destination,
   (1) Pushing a button of a lower floor comparing to where the elevator currently is can make it keep going down and add the lower floor into its destination list. However, if the floor is already in the destination list, it is ignored.
   (2) Pushing the button when the elevator is on the same floor would make the newly inputted command neglected.
   (3) Pushing a button of a higher floor adds the floor to its destination list. The elevator first goes down to all lower destinations then goes up to the targeted floor. However, if the floor is already in the destination list, it will be ignored.

In this program, you need to design an elevator system that can display the position of the elevator during operation. We define each floor moving as one minute.

1. Our elevator goes up or down one floor in one minute, and the time duration of the opening and closing of doors is neglected.
2. You do not need to deal with conflicts when the destination and the current floor are both the same. In other words, you can just neglect the command.
3. The elevator will start at the first floor.
4. When pushing a button, the elevator does not move immediately. It will move at next minute.
5. If there is no new destination, it stops at the final destination.

Input:

You can use standard input to test your code. First, we input an integer n as the count of total operations. Then the next n lines each indicates the time of the operation in minutes and which floor the button is pushed, separated by a space. Additionally, the order of operations is not necessarily listed in numerical order. The input ends with EOF(end of file).

Output:

List out the location of the elevator in which floor at every minute.

Error Handing:

Sample Input / Output：

Sample Input	Sample Output

5
2 10
3 6
8 2
15 4
20 8
	1 1
	2 1
	3 2
	4 3
	5 4
	6 5
	7 6
	8 7
	9 8
	10 9
	11 10
	12 9
	13 8
	14 7
	15 6
	16 5
	17 4
	18 3
	19 2
	20 2
	21 3
	22 4
	23 5
	24 6
	25 7
	26 8

□ **Easy,Only basic programming syntax and structure are required.**
■ **Medium,Multiple programming grammars and structures are required.**
□ **Hard,Need to use multiple program structures or complex data types.**

Expected solving time:

20 minutes

Other notes:

10902 CPP Final Exam

Contributor：Wen, Yong-Wei

Subject：Design a Train

Main testing concept：

Basics

■ C++ BASICS
■ FLOW OF CONTROL
■ FUNCTION BASICS
□ PARAMETERS AND OVERLOADING
□ ARRAYS
■ STRUCTURES AND CLASSES
■ CONSTRUCTORS AND OTHER TOOLS
□ OPERATOR OVERLOADING, FRIENDS, AND REFERENCES
□ STRINGS
■ POINTERS AND DYNAMIC ARRAYS

Functions

□ SEPARATE COMPILATION AND NAMESPACES
□ STREAMS AND FILE I/O
□ RECURSION
□ INHERITANCE
□ POLYMORPHISM AND VIRTUAL FUNCTIONS
□ TEMPLATES
■ LINKED DATA STRUCTURES
□ EXCEPTION HANDLING
□ STANDARD TEMPLATE LIBRARY
□ PATTERNS AND UML

Description：

In 2080, people attempt to stop global warming via climate engineering catastrophically backfires. The failure of the plan causes the world to enter the ice age. The remnants of humanity have taken to a circumnavigational train, the Snowpiercer, run by transportation magnate Wilford.

Mr. Wilford arranges different uses for each carriage to ensure the maintenance of the daily. The train is connected by the carriage. For each carriage, he records the carriage name, the number of passenger and the name of passengers.

You need to implement two classes to help Mr. Wilford to arrange his train.

Class: Train	Class: Carriage

■ `Carriage*: root`
■ function: `pushCarriage(string name, int passengerNum, string *passengerName)`
■ function: `checkPassenger(string name)`
■ function: `deleteCarriage(string name)`
■ function: `getCarriage(std::string name)`
■ default constructor()
	`public`
	■ function: `getPassengerNum()`
	■ function: `getName()`
	■ function: `printPassenger()`
	■ `constructor(string name, int passengerNum, string* passengerName)`
	■ `destructor()`
	`private:`
	■ `string: name`
	■ `int: passengerNum`
	■ `string*: passengerName`

`Carriage` Function Implement:

1. `int getPassengerNum()`: return the number of the passenger in current `Carriage`.
2. `string getName()`: return the name of the current `Carriage`.
3. `void printPassenger()`: print the name of each passenger separated by one space (" "). (i.e. "Melanie Wardell Kevin")
4. `constructor()`： create the data of Carriage.
5. `destructor()`： delete the member variable which you have allocated the memory and print "Delete Carriage!"

`Train` Function Implement:

1. `void pushCarriage()`: Give the information of the `Carriage`. Use this to build a `Carriage` and push it to the end of the `Train`.
2. `void checkPassenger()`: Give the name of the `Carriage`. Find the `Carriage` with the same name and call its printPassenger. If it does not exist, print "Carriage does not exist!"
3. `void deleteCarriage()`: Give the name of the `Carriage`. Delete the `Carriage` with the same name and re-connect the train. If it does not exist, print "Carriage does not exist!"
4. `Carriage* getCarriage()`: Give the name of the `Carriage`. Return the pointer of Carriage with the same name. If it does not exist, print "Carriage does not exist!"

Input：

You don't need to deal with input format. The given file main.cpp will deal with all input format and catching (see void CreateTrain() in Notes). You can use the given main.cpp and enter the Sample Input to test your program.

The introduction of the Input:

1. First, input a positive number n which represent the number of Carriages.
2. For each Carriage, the first line input the carriage name and the number of passenger.
   The second line input each passenger's name.

Notice again. The given main.cpp will deal with the catching of input. This is just a introduction of the input. You only need to implement the Train class and the Carriage class. You can directly using Sample Input to test your program.

The input rule.

1. The input will not give you the same name of the carriage.
2. The length of the Train will not less than 3 carriages.

Additionally, there are a few points about our Judge testing.

1. The main.cpp in your submission will be replaced when judging.
2. You can use the main.cpp in "Other Notes" to test your program.

Output：

- When calling the function of `checkPassenger()`, you have to output the name of each passengers separated by space (" "). If the `Carriage` does not exist, you have to output "Carriage does not exist!"
- When calling the function of `deleteCarriage()`. If the Carriage does not exist, you have to output "Carriage does not exist!"
- When calling the function of `getCarriage()`. If the Carriage does not exist, you have to output "Carriage does not exist!"

Sample Input / Output :

Sample Input	Sample Output
6 Engine 2 Melanie Bennett Hospitality 3 Melanie Wardell Kevin First_class 2 Robert Lilah Head_bar 1 Ferami Second_class 3 Till Jinju Miles Agricultural 2 Otto Sheila	Carriage name: Engine Passenger num: 2 Carriage name: Hospitality Passenger num: 3 Carriage name: First_class Passenger num: 2 Carriage name: Head_bar Passenger num: 1 Carriage name: Second_class Passenger num: 3 Carriage name: Agricultural Passenger num: 2 The passenger in Hospitality: Melanie Wardell Kevin Carriage does not exist! Delete Carriage! Carriage name: Engine Passenger num: 2 Carriage name: Hospitality Passenger num: 3 Carriage name: First_class Passenger num: 2 Carriage name: Head_bar Passenger num: 1 Carriage name: Agricultural Passenger num: 2

□ **Easy, only basic programming syntax and structure are required.**

□ **Medium, multiple programming grammars and structures are required.**

■ **Hard, need to use multiple program structures or complex data types.**

Expected solving time:

60 minutes

Other notes:

```
#include "Train.h"

void printTrain(const Train& train)
{
	Carriage* cur = train.root;
	while (cur != nullptr)
	{
		std::cout << "Carriage name: " << cur->getName() << "\tPassenger num: " << cur-
>getPassengerNum() << std::endl;
		cur = cur->next;
	}
}


void CreateTrain(Train& train)
{
	int num;
	std::cin >> num;
	for (int i = 0; i < num; i++)
	{
		std::string name;
		int passengerNum;
		std::cin >> name >> passengerNum;

		std::string* passengerName = new std::string[passengerNum];
		for (int j = 0; j < passengerNum; j++)
		{
			std::cin >> passengerName [j];
		}
		train.pushCarriage (name, passengerNum, passengerName);
	}
}
```

```
int main()
{
	Train train;
	CreateTrain(train);
	printTrain(train);

	std::string checkName = "Hospitality";
	std::cout << "The passenger in Hospitality:" << std::endl;
	train.checkPassenger (checkName);

	std::string deleteName = "Third_class";
	train.deleteCarriage (deleteName);


	deleteName = "Second_class";
	train.deleteCarriage(deleteName);

	printTrain(train);

	return 0;
}
```

CPP Problem Design Example

Subject: School Inheritance

Contributor: Jun-Yu Chen

Main testing concept: Class/Inheritance/Overloading/Virtual Functions

Basics

□ C++ BASICS
□ FLOW OF CONTROL
□ FUNCTION BASICS
□ PARAMETERS AND OVERLOADING
□ ARRAYS
■ STRUCTURES AND CLASSES
□ CONSTRUCTORS AND OTHER TOOLS
■ OPERATOR OVERLOADING, FRIENDS, AND REFERENCES
□ STRINGS
□ POINTERS AND DYNAMIC ARRAYS

Functions

□ SEPARATE COMPILATION AND NAMESPACES
□ STREAMS AND FILE I/O
□ RECURSION
■ INHERITANCE
■ POLYMORPHISM AND VIRTUAL FUNCTIONS
□ TEMPLATES
□ LINKED DATA STRUCTURES
□ EXCEPTION HANDLING
□ STANDARD TEMPLATE LIBRARY
□ PATTERNS AND UML

Description：

The governing of public and private schools is considered different as the two types of schools have different education goals while having different funding and student structures. For private schools the main income comes from students' paying high tuition fees comparing to public schools whose main income relies on funding from the government, as schools attracting more students with their good performance should receive more resources. However, even being almost totally different they are still closely related and could not be managed separately, thus an intermediate between the two systems, regarding student transfer and the increase or decrease of admissions of next year is created.

For this task, you have to write a program that creates three types of school (one from each class: `School`, `PrivateSchool`, and `PublicSchool`), transfers students from each to another and calculate the available admissions for the next year given governing rules.

Create a base class called **School** that has member variables and functions:

- the **name** of the school (a string)
- the **studentNumber** of the school currently
- **studentNumberNextYear** indicating the number of students the school could have next year, where when constructing is same as the student number this year.
- ***admissions(float number):*** adds the number (passed as a parameter) to the total student number this year (if the number is nonnegative),
- ***dropouts(float number):*** subtracts the number from the number of students this year (if the number is nonnegative and less than or equal to the student number)
- ***transfer(float number, School &toSchool):*** deducts from the student number of current and transfers them to another school (passed as a parameter), implemented calling ***dropouts(number)*** and ***toSchool.admissions(number).***

Also, create a class called **PrivateSchool** that is derived from **School**. In a **PrivateSchool,** while the first call of ***dropouts*** comes without any penalty, starting from the second call, every additional call of ***dropouts*** induces a penalty to deduce 100 from its **studentNumberNextYear**. Hence, the class must have a data member to keep track of the times of dropouts being called and override the ***dropouts*** function.

Finally, create a **PublicSchool** class derived from **School.** Additionally, please add a member variable, ***growing_rate*** **(=0.05),** and a member function, ***apply_growth()***, which increases number of students able to admit next year by **studentNumberNextYear *+= growing_rate*** * **studentNumberNextYear**. **PublicSchool** incur penalties when large number of students (>100) leave the school at once. A dropout of such a number induces a loss of

5% of **`studentNumberNextYear`**, truncating the decimal places. Again, the ***`dropouts`*** function must override the one in the base class.

For all 3 classes create constructors (default and with parameters) and overloaded <<
(output) operator, reuse constructors and operator << of the base class in the derived classes.

Input :

No Inputs.

Output :

Please refer to sample output for output format, consisting of name, studentNumber and studentNumberNextYear. Separated by tab('\t') as written in bold in the next line:

** **″name\tstudentNumber\tstudentNumberNextYear″**

Error handling :

If any violations stated in the description occurs (Ex. Subtracting more than existing number), we do not do the operation and output "ERROR\n".

Sample Input	Sample Output
According to the given main.cpp in Other Notes	NTUST 12500 12500 NTUT 85000 85000 FJCU 25000 25000 NTUST 12700 12500 NTUST 12500 12500 ERROR NTUST 12500 12500 FJCU 26000 25000 FJCU 25950 25000 FJCU 24950 24900 NTUT 86000 85000 NTUT 86000 89250 NTUT 85000 84787 NTUT 85000 84787 NTUT 84000 80547 NTUST 13500 12500 ERROR FJCU 24950 24900 NTUST 13500 12500

□ Easy, Only basic programming syntax and structure are required.
■ Medium, Multiple programming grammars, and structures are required.
□ Hard, Need to use multiple program structures or complex data types.

Expected solving time:

25 minutes

Other Notes:

```
#include <iostream>
#include "School.h"
#include <string>
```

```
using namespace std;

int main()
{
    //init 3 different account types
    School ntust("NTUST", 12500);
    PublicSchool ntut("NTUT", 85000);
    PrivateSchool fjcu("FJCU", 25000);

    //state info all 3
    cout<<ntust<<endl;
    cout<<ntut<<endl;
    cout<<fjcu<<endl;

    //test all methods on School
    ntust.admissions(200);
    cout<<ntust<<endl;

    ntust.dropouts(200);
    cout<<ntust<<endl;

    ntust.dropouts(100000);
    cout<<ntust<<endl;

    //test all methods on PrivateSchool
    fjcu.admissions(1000);
    cout<<fjcu<<endl;

    fjcu.dropouts(50);
    cout<<fjcu<<endl;

    fjcu.dropouts(1000);
    cout<<fjcu<<endl;

    //test all methods on PublicSchool
    ntut.admissions(1000);
    cout<<ntut<<endl;

    ntut.apply_growth();
    cout<< ntut <<endl;

    ntut.dropouts(1000);
    cout<< ntut <<endl;

    //Transfer method
    cout << ntut << endl;
    ntut.transfer(1000, ntust);
    cout << ntut << endl;
    cout << ntust << endl;

    fjcu.transfer(30000, ntust);
    cout << fjcu << endl;
    cout << ntust << endl;

    return 0;
}
```

CPP Problem Design Example

Contributor：Cheryl Huang, Jun-Yu Chen

Subject：The Translation Machine

Main testing concept：

Basics

■ C++ BASICS
■ FLOW OF CONTROL
■ FUNCTION BASICS
□ PARAMETERS AND OVERLOADING
■ ARRAYS
□ STRUCTURES AND CLASSES
□ CONSTRUCTORS AND OTHER TOOLS
□ OPERATOR OVERLOADING, FRIENDS, AND REFERENCES
□ STRINGS
□ POINTERS AND DYNAMIC ARRAYS

Functions

□ SEPARATE COMPILATION AND NAMESPACES
□ STREAMS AND FILE I/O
■ RECURSION
□ INHERITANCE
□ POLYMORPHISM AND VIRTUAL FUNCTIONS
□ TEMPLATES
□ LINKED DATA STRUCTURES
□ EXCEPTION HANDLING
□ STANDARD TEMPLATE LIBRARY
□ PATTERNS AND UML

Description：

Here is a translation machine. You are given the possible translations of letters which are a list of pairs of original and deciphered characters. Your task is to verify whether a word can be translated into another given translatable letter pairs. Two words match if the they have the same length and each character of the first word can be turned into a corresponding character of the second word after the translation process. For example, a word “sky”, given a translation pair making ‘k’ the same as ‘p’, yields the word ‘spy’. As a result, we deem ‘sky’ the same as ‘spy’, with the position of letters having to correspond to the other by using the available translations zero or more times.

Input：

Users input through standard input. The input contains several test cases, each of them as described below.

1. The first line of input contains two integers **m (1 ≤ m ≤ 500)** and **n (1 ≤ n ≤ 50)**, where **m** is the number of translations of letters and **n** is the number of word pairs. All given test inputs will fall in this range.
2. Each of the next **m** lines contains two distinct space-separated letters **a** and **b**, indicating that the letter **a** can be translated to the letter **b**. Each ordered pair of letters **(a, b)** appears at most once. Following this are **n** lines, each containing a word pair to check. Remember, there are chances where one letter could be translated into many others. For example, for the pairs (i, b), (l, i), (y, l) the word ‘y’ could be translated into either ‘l’, ‘i’ or ‘b’.
3. Translations and words use only lowercase letters ‘a’…‘z’ and each word contains at least 1 and at most 50 letters.
4. Exit the program while **m** and **n** are both **0**.

Output：

For each pair of words, display ‘**yes**’ if the two words match, and ‘**no**’ otherwise, on a line by itself.

Error handling：

Output “ERROR” when uppercase letters (i.e., ‘L’) or other invalid inputs (i.e., numbers) are read. However, the input still counts towards the input tally while the information is ignored. (The input counter adds one but the inputted information will not be used). Please refer to Sample Input / Output for a more thorough understanding regarding error handling.

Sample Input / Output :

Sample Input	Sample Output
10 5	
c t i r k p o c r o t e t f u h w p E c	ERROR
we we can the work people it of out the	yes no no yes yes
3 4	
a c b a a b	
aaa abc abc aaa acm bcm a3 bb	yes no yes ERROR
0 0	

□ **Easy, only basic programming syntax and structure are required.**

■ **Medium, multiple programming grammars and structures are required.**

□ **Hard, need to use multiple program structures or complex data types.**

Expected solving time:

40 minutes

Other notes：

CPP Problem Design Example

Contributor： Pin-Shao Chen

Subject：Student Course System

Main testing concept：

Basics

- ■ C++ BASICS
- ■ FLOW OF CONTROL
- ■ FUNCTION BASICS
- □ PARAMETERS AND OVERLOADING
- ■ ARRAYS
- □ STRUCTURES AND CLASSES
- □ CONSTRUCTORS AND OTHER TOOLS
- □ OPERATOR OVERLOADING, FRIENDS, AND REFERENCES
- ■ STRINGS
- □ POINTERS AND DYNAMIC ARRAYS

Functions

- □ SEPARATE COMPILATION AND NAMESPACES
- □ STREAMS AND FILE I/O
- □ RECURSION
- □ INHERITANCE
- □ POLYMORPHISM AND VIRTUAL FUNCTIONS
- □ TEMPLATES
- □ LINKED DATA STRUCTURES
- □ EXCEPTION HANDLING
- ■ STANDARD TEMPLATE LIBRARY
- □ PATTERNS AND UML

Description：

Design a course registration system, which can do the following things:

1. Add or delete a student.
2. Add or delete a course.
3. List the schedule of every student.
4. Error Handling

To be specific, please implement the commands in the Input/Output (each command will be separated by one space “ ”):

Input/Output：

Input some commands to operate course registration system. Each parameter separates by one space ” “. The following lists the legal commands and their format.

1. **AddStudent <Student_name>:** Add a student. If the student already exists, output “**The student's name is duplicate.**”
2. **AddCourse <Student_name> <Course_name> <class_time1> <class_time2>… :** Add a course for the student, if the course will conflict with the schedule, output “**Course conflict.**”
3. **DelStudent <Student_name>:** Delete a student.
4. **DelCourse <Student_name> <Course_name>:** Delete the course from the student's schedule.
5. **Print StudentList:** List all student's names in insertion order. Separate each student’s name by one space “ ”. Output “**The Students list is empty.**” if the StudentList is empty.
6. **Print <Student_name>:** Print the schedule for the student in insertion order. Output **empty schedule** if the schedule is empty.

Example of **empty schedule** (for each line output the day with colon):

```
M:
T:
W:
R:
F:
```

Example of schedule:

```
M: 2:Math 3:Math 4:Math
T:
W: 1:DeepLearning 2:DeepLearning
R: 4:DeepLearning
F:
```

** The pseudo code of output format.

```
cout << “M:”;
foreach course in Monday
{
    cout << “ ” << time << “:” << course;
}
cout << endl;
```

7. Print <Student_name> <Course_name>

Print information of the course. Output in the order from M to F.

The output format: <**Course_name**> **<class_time1> <class_time2>…**

Example:

```
>Print Jason Math
Math M2 M3 M4
```

Schedule format:

You can input the day of class_time from M to F, the time of class_time from 1 to 10.
(i.e., ‘R8 R9 R10’)

	M	**T**	**W**	**R**	**F**
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

If the input tries to deal with a non-existent student, output “**The student's name does not exist.**” If the input tries to deal with a course that is not on the schedule, output “**The course does not exist.**”

<Course_name> and <Student_name> will be one word and only contain alphabet.

Please output “Illegal command.” if the user inputs incomplete command, wrong command or too many parameters. Additionally, if the input parameter is not given a legal value such as “M11”, you should output “Illegal parameters.”

Check the illegal command first, then parameters.

Sample Input / Output :

Sample Input	Sample Output
AddStudent Jason Print Jason AddCourse Jason Math M2 M3 M4 AddCourse Jason DeepLearning W1 W2 R4 Print StudentList AddStudent Mike1 Print Jason Math Print Jason DelStudent Jason DelCourse Jason Math Print StudentList AddCourse Jason Science	M: T: W: R: F: Jason Illegal parameters. Math M2 M3 M4 M: 2:Math 3:Math 4:Math T: W: 1:DeepLearning 2:DeepLearning R: 4:DeepLearning F: The student's name does not exist. The Students list is empty. Illegal command.
AddStudent Mike AddStudent Mike Print Mike Print StudentList DelCourse Mike Math AddCourse Mike DeepLearning W1 W2 R4 AddCourse Mike Math W1 W2 W3 AddCourse Jason DeepLearning W1 W2 R4 Print Mike English Print Print Mike DelStudent Jason DelCourse Jason Math Print Mike DeepLearning M11	The student's name is duplicate. M: T: W: R: F: Mike The course does not exist. Course conflict. The student's name does not exist. The course does not exist. Illegal command. M: T: W: 1:DeepLearning 2:DeepLearning

	R: 4:DeepLearning F: The student's name does not exist. The student's name does not exist. Illegal command.

□ **Easy, only basic programming syntax and structure are required.**

■ **Medium, multiple programming grammars and structures are required.**

□ **Hard, need to use multiple program structures or complex data types.**

Expected solving time:

60 minutes。

Other notes：

10902 CPP Final Exam

Subject: Triangle Json File

Contributor：Yuan, Wei-Cheng

Main testing concept: File inout

Basics

■ C++ BASICS
■ FLOW OF CONTROL
□ FUNCTION BASICS
□ PARAMETERS AND OVERLOADING
□ ARRAYS
□ STRUCTURES AND CLASSES
□ CONSTRUCTORS AND OTHER TOOLS
■ OPERATOR OVERLOADING, FRIENDS,AND REFERENCES
■ STRINGS
■ POINTERS AND DYNAMIC ARRAYS

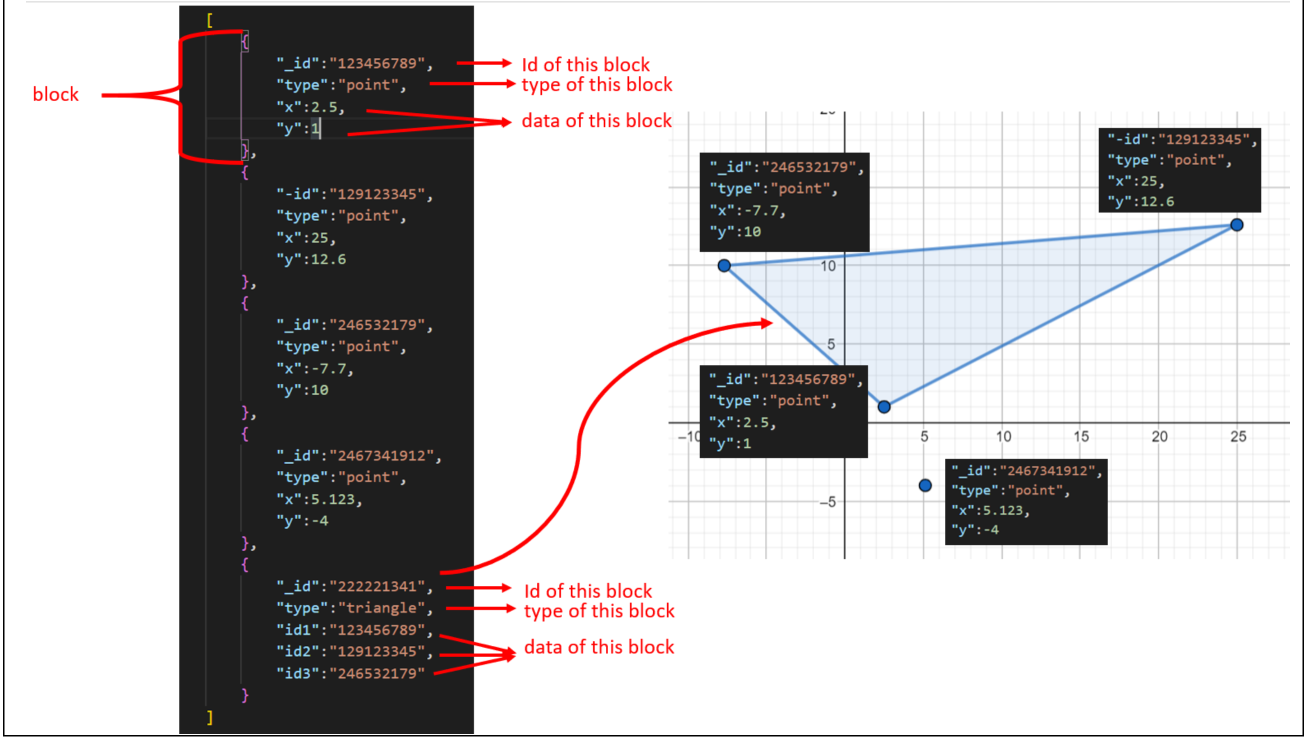
Functions

□ SEPARATE COMPILATION AND NAMESPACES
■ STREAMS AND FILE I/O
□ RECURSION
□ INHERITANCE
□ POLYMORPHISM AND VIRTUAL FUNCTIONS
□ TEMPLATES
□ LINKED DATA STRUCTURES
□ EXCEPTION HANDLING
□ STANDARD TEMPLATE LIBRARY
□ PATTERNS AND UML

Description:

CRUD is an acronym for create, read, update, and delete. These are the four basic functions of persistent storage. A pair (name: value) in a JSON file is composed of a name and a value. Each name is followed by a colon, and pairs are separated by a comma. In the sample JSON file, ***test.json,*** containing records of points and triangles in an array, each record is enclosed with curly brackets. Additionally, while points record the raw data of "id", "type", and the coordinate of "x", "y", triangles record the raw data of "id" and "type" and the index of its three points. In a record, each field is a name-value pair. A name must be a string while a value can be a string, number, boolean, etc. The figure below illustrates the structure of our triangle JSON file.

CRUD 是创建、读取、更新和删除的首字母缩写。这是持久性存储的四个基本功能。JSON 文件中的一对(名称:值)由名称和值组成。每个名称后面跟一个冒号，对之间用逗号分隔。在样例 JSON 文件中，测试。Json 包含数组中的点和三角形记录，每条记录都用大括号括起来。另外，点记录“id”、“type”的原始数据以及“x”、“y”的坐标，三角形记录“id”、“type”的原始数据及其三个点的索引。在记录中，每个字段是一个名值对。名称必须是字符串，而值可以是字符串、数字、布尔值等。下图展示了我们的三角形 JSON 文件的结构。



```
[
    {
        "_id":"123456789",
        "type":"point",
        "x":2.5,
        "y":1
    },
    {
        "-id":"129123345",
        "type":"point",
        "x":25,
        "y":12.6
    },
    {
        "_id":"246532179",
        "type":"point",
        "x":-7.7,
        "y":10
    },
    {
        "_id":"2467341912",
        "type":"point",
        "x":5.123,
        "y":-4
    },
    {
        "_id":"222221341",
        "type":"triangle",
        "id1":"123456789",
        "id2":"129123345",
        "id3":"246532179"
    }
]
```

block
Id of this block
type of this block
data of this block
Id of this block
type of this block
data of this block
"_id":"246532179", "type":"point", "x":-7.7, "y":10
"-id":"129123345", "type":"point", "x":25, "y":12.6
"_id":"123456789", "type":"point", "x":2.5, "y":1
"_id":"2467341912", "type":"point", "x":5.123, "y":-4
10
5
−5
−10
5
10
15
20
25

Given a JSON file containing the record of points and triangles, which has member functions to parse the file, manipulate (RUD) records in the given file, output manipulated records to a JSON file in the same format of input JSON file. Note that the class BasicJSON includes at least the following member functions:

给定一个包含点和三角形记录的 JSON 文件，该文件具有解析该文件的成员函数，操作给定文件中的(RUD)记录，以与输入 JSON 文件相同的格式将被操作的记录输出到 JSON 文件。注意，BasicJSON 类至少包含以下成员函数：

1. **`bool Parse(std::string InputFileName);`** Read a list of records from the file *InputFileName* and parse the contents in the file to construct your data with your selected structure. Also, if the file is parsed successfully, return `true`. Otherwise, return `false`.
   从文件 `InputFileName` 中读取记录列表，并解析文件中的内容，以使用所选的结构构造数据。此外，如果文件被成功解析，则返回 `true`。否则,返回 `false`。
2. **`void Write(std::string OutputFileName);`** Output all records to the file *OutputFileName* in JSON format listed above. Do not add tab or space in front of line. see reference in below.
   将所有记录输出到上面列出的 JSON 格式的 `OutputFileName` 文件。不要在行前添加制表符或空格。参见下面的参考资料。
3. **`void Delete(std::string Name);`** Delete a record of the passed `Name`(id). If a point is be deleted, you should remove all triangles which have the deleted point as one of their vertices.
   删除传递的 `Name(id)`的记录。如果一个点被删除，你应该删除所有的三角形，其中有删除的点作为他们的顶点之一。
4. **`float TotalArea();`** Return the sum of the area of all triangles in the JSON. If it does not contain any triangle, output “File dont have triangle\n” and return 0. The area of a triangle can be computed using Heron’s formula listed at the end of the file.
   返回 JSON 中所有三角形的面积之和。如果不包含任何三角形，输出“File don ' t have triangle\n”并返回 0。三角形的面积可以使用文件末尾列出的 Heron 公式来计算。
5. **`overload [] operator`** such that
   1. **JSONobject[*index*]**: an *index* is a non-negative integer for accessing an array element (record).索引是一个访问数组元素(记录)的非负整数。
   2. **JSONobject[*index*][*name*]**: get a value associated with given *index* and *name*.
      获取与给定索引和名称相关联的值。

Please note that:

1. We will provide main.cpp and products.json to test your class. Sample files input-main.cpp, test.json and output.json are shown as an example of testing cases, located under the same directory, e.g.

```
..\
 ├ Q5
   ... ├ CPP 程式設計題-JSON.pdf
       ├ input-main.cpp
```

├ output.json

└test.json

2. No comments are included in the JSON file.
3. The comma at the end of any last pair in an object or array is optional
4. A few redundant spaces and next-line characters are acceptable.
5. All coordinate is float type

1. 我们将提供主要的。cpp 和 `products.json` 测试你的类。示例文件 input-main.cpp, test。json 和输出。Json 显示为一个测试用例的例子，位于相同的目录下，例如。

2. JSON 文件中没有包含任何注释。

3. 对象或数组中最后一对结尾处的逗号是可选的

4. 一些多余的空格和下一行字符是可以接受的。

5. 所有坐标为浮点型

Input:

Substitution of your main function.

Output:

Please refer to the sample output.

Sample Input / Output：

Sample Input	Sample Output
According to the given main function and json file	point 5 129123345 37.5 File dont have triangle 0

□ **Easy, Only basic programming syntax and structure are required.**

□**Medium, Multiple programming grammars and structures are required.**

■ **Hard, Need to use multiple program structures or complex data types.**

Expected solving time:

60 minutes

Other notes:

Given main function:

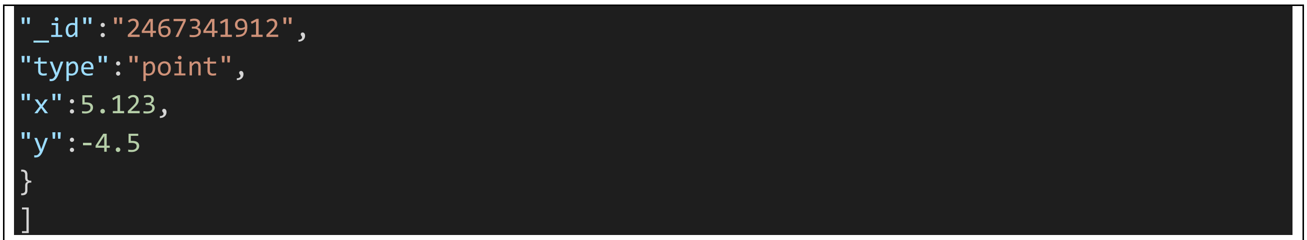
```
int main()
{
    BasicJson json;
    if(json.Parse("Test.json"))
    {
        cout<<json[0]["type"]<<endl;
        cout<<json[0]["x"]<<endl;
        cout<<json[1]["_id"]<<endl;
        json[1]["y"]=20;
        cout<<json.TotalArea()<<endl;
        json.Delete(“129123345”);
        cout<<json.TotalArea()<<endl;
        json.write(“out.json”);
    }
}
```

Given json file (Test.json):

```
[
    {
        "_id":"123456789",
        "type":"point",
        "x":5,
        "y":0
    },
    {
        "-id":"129123345",
        "type":"point",
        "x":0,
        "y":0
    },
    {
        "_id":"246532179",
        "type":"point",
        "x":0,
        "y":5
    },
    {
        "_id":"2467341912",
        "type":"point",
        "x":5.123,
        "y":-4.5
    },
    {
        "_id":"222221341",
        "type":"triangle",
        "id1":"123456789",
        "id2":"129123345",
        "id3":"246532179"
    }
]
```

Output file (output.json):

```
[
{
"_id":"123456789",
"type":"point",
"x":5,
"y":0
},
{
"_id":"246532179",
"type":"point",
"x":0,
"y":5
},
{
```



```
"_id":"2467341912",
"type":"point",
"x":5.123,
"y":-4.5
}
]
```

Heron's formula

$$A=\sqrt{s(s-a)(s-b)(s-c)}\text{，其中}s=\frac{a+b+c}{2}$$

a, b, c are triangle edge length