Important message on plagiarism

The single most important point for you to realize before the beginning of your studies at ShanghaiTech is the meaning of “plagiarism”:

Plagiarism is the practice of taking someone else's work or ideas and passing them off as one's own. It is the misrepresentation of the work of another as your own. It is academic theft; a serious infraction of a University honor code, and the latter is your responsibility to uphold. Instances of plagiarism or any other cheating will be reported to the university leadership, and will have serious consequences. Avoiding any form of plagiarism is in your own interest. If you plagiarize and it is unveiled at a later stage only, it will not only reflect badly on the university, but also on your image/career opportunities.

Plagiarism is academic misconduct, and we take it very serious at ShanghaiTech. In the past we have had lots of problems related to plagiarism especially with newly arriving students, so it is important to get this right upfront:

You may…

• … discuss with your peers about course material.
• … discuss generally about the programming language, some features, or abstract lines of code. As long as it is not directly related to any homework, but formulated in a general, abstract way, such discussion is acceptable.
• … share test cases with each other.
• … help each other with setting up the development environment etc.

You may not …

• … read, possess, copy or submit the solution code of anyone else (including people outside this course or university)!
• … receive direct help from someone else (i.e. a direct communication of some lines of code, no matter if it is visual, verbal, or written)!
• … give direct help to someone else. Helping one of your peers by letting him read your code or communicating even just part of the solution in written or in verbal form will have equal consequences.
• … gain access to another one’s account, no matter if with or without permission.
• … give your account access to another student. It is your responsibility to keep your account safe, always log out, and choose a safe password. Do not just share access to your computer with other students without prior lock--out and disabling of automatic login functionality. Do not just leave your computer on without a lock even if it is just for the sake of a 5--minute break.
• … work in teams. You may meet to discuss generally about the material, but any work on the homework is to be done individually and in privacy. Remember, you may not allow anyone to even just read your source code.

With the Internet, "paste", and "share" are easy operations. Don’t think that it is easy to hide and that we will not find you, we have just as easy to use, fully automatic and intelligent tools that will identify any potential cases of plagiarism. And do not think that being the original author will make any difference. Sharing an original solution with others is just as unethical as using someone else’s work.

CS100 Homework 6 (Fall, 2019)

This homework focuses on use of STL containers and class templates, virtual classes and polymorphism, as well as code structuring.

Percentage of this homework over the whole score: 11%

Submission deadline: 2019-12-1 23:59

General Note for Homework 6

In this homework, you may need to write large classes with lots of member variables or functions. In addition to getting the correct answer, an additional requirement for this homework is to make your codes well organized. It is achieved by putting different classes into different files, splitting the header files(*.h or *.hpp) and source files(*.cpp), and linking them together correctly using `#include` lines. Your header files should contain correct include guards. Please do not use `#pragma once`, as some compilers may not recognize it.

In other words, you may need to submit multiple files, each functioning as a part of your whole project. For each part of code, we will specify which file it should be put into. Violating the specification may result in point deduction or cause errors.

Problem 1: A Template Set

● Basic Concepts

A set is a collection of distinct objects. By “distinct”, we mean that there are no duplicate objects in a set. An example of a set of integers, similar to what you have all learned in mathematics, is represented as {3, -1, 2147483647, 0}. However, we want our set to be more versatile, and therefore we are making it a template class. In general, the declaration of the set class looks like:

```
template <typename T>
class CustomSet {};
```

The reason we name it CustomSet instead of Set is that “Set” may be identified as a keyword by some editors.

For the objects in your CustomSet, it is recommended to store them in an STL container, either a *`vector`* or a *`list`*, at your free choice.

We have provided the declarations and descriptions of all functions you need to implement. See below or see the skeleton code we provided:

```
template <typename T>
class CustomSet
```

```
{
public:
// Constructors and destructor:
            CustomSet();
            // Default constructor

            CustomSet(const CustomSet& other);
            // Copy constructor. Should construct a copy of "other".

            CustomSet(T arr[], int size);
            // Constructor using an array.
            // Note that if there are duplicates in the array, you should ignore them.
            // And then the size of your "CustomSet" will be different from the "size" given!

            ~CustomSet();
            // Destructor. Pay attention to memory leaks!

// Other member functions:
            int size();
            // Returns the size of the set.

            bool add(const T& item);
            // If "item" is already in the set, return false.
            // Otherwise, add it to the set and return true.

            T* find(const T& item);
            // If an object in the set equals "item", return a pointer to it.
            // If "item" is not found, return nullptr(NULL).

            bool erase(const T& item);
            // If "item" is not in the set, return false.
            // Otherwise, erase it from the set and return true.

            CustomSet intersection(const CustomSet& other);
            // This function returns the intersection of two sets (*this and other).
            // In other words, the set of all objects that is in both *this and other.
            // If there is no intersection, just return an empty set.

            void sortSet();
            // This function sorts the objects in the set in ascending order.
            // Directly using (std::)sort is enough, if you are using an STL container.

            void printSet();
            // This function prints the set, seperating elements by { , , , }.
```

```
        // For example, Assume you've added 2, 4, -2, and 800 to the set, in this order.
        // This function will print: "{2, 4, -2, 800}\n"
        // Note that there are spaces between a comma(,) and the next object.
        // Print a newline at the end. (indicated by the '\n' above)

// Operators:
        CustomSet operator+ (const T& item);
        CustomSet operator+ (const CustomSet& that);
        CustomSet& operator+= (const T& item);
        CustomSet& operator+= (const CustomSet& that);
        // The operator+ and operator += are overloaded.
        // They can be understood intuitively, or consider this:
        // A set "plus" an item means adding in the item.
        // A set "plus" a set means taking the union of two sets.
        // However, the difference between + and += is not written here.
        // Try to figure out on your own!

        CustomSet operator- (const T& item);
        CustomSet operator- (const CustomSet& that);
        CustomSet& operator-= (const T& item);
        CustomSet& operator-= (const CustomSet& that);
        // The operator- and operator -= are overloaded.
        // They can be understood intuitively, or consider this:
        // A set "minus" an item means erasing the item.
        // A set A "minus" a set B means subtracting B from A, or namely, A\B.
        // However, the difference between - and -= is not written here.
        // Try to figure out on your own!


        CustomSet<std::pair<T, T>> operator* (const CustomSet& that);
        // This function returns the Cartesian product of two sets (*this and that).
        // The Cartesian product of two sets is the set of all ordered pairs that satisfy:
        //         The first element of the ordered pair belongs to first set (*this).
        //         The second element of the ordered pair belongs the second set (that).
        // The (std::)pair (in header <utility>) helps representing the object in it.
        // If you have question with Cartesian products or std::pair, look up on Internet!
private:
};
```

The description of each function is in comments. Apart from the part given, feel free to add your own functions or member variables if you want to.

An additional (trivial) requirement: If there is any function that you think should return a const value (for example, `size()`), please make the return type `const`. It's OK to do so only on functions that should be const for certain. If you fail to do so throughout your codes, you may receive a slight point deduction.

Since separating the declarations and implementations of a template class can be a little tricky, it's not required to separate them into different files. All your work for Problem 1 should be written in a header file named "customset.h" or "customset.hpp".

Your project should also contain a file for `main()` function. You do not need to submit it for the homework, therefore there's no rule for its name. A suggested name may be "main.cpp". This will also apply for problem 2 and problem 3.

Example output

Once you have completed this task, your `CustomSet` should be ready to be used as a template class. You should test and debug on your own before submitting. Here is an example test case:

```
int main()
{
    int a[5] = { 4,6,32,-4,0 };
    CustomSet<int> sample1(a, 5);
    int b[5] = { 4,2,10,-2,0 };
    CustomSet<int> sample2(b, 5);
    sample1 -= sample2;
    sample1.sortSet();
    sample1.printSet(); // prints "{-4, 6, 32}\n"
}
```

Submission details

You are going to submit one file named "customset.h" or "customset.hpp".

As we are switching to Geekpie OJ, the address of OJ is not the same as before. More details on how to submit to OJ will be announced soon through piazza (1st source), please pay close attention to instructor notes.

Problem 2: Lottery Station

As Gezi Wang has cleared his girlfriend's Taobao "shopping cart" during the 11-11 net-shopping festival, he is low on money and thus decides to win a fortune with lottery.

The second task of this homework is based on the scenario above. In general, you are to write a class named `LotteryStation`, which sells two different types of lottery tickets, and supports operations like buying tickets and claiming prizes.

● Basic Concepts: Lotteries, Tickets, and Their Rules

Although many lotteries are much more complicated, in this task, we only consider the simplest ones: choosing a few numbers out of some numbers. The rule is:

> There are many (for example, 30) possible numbers.
> When buying a single ticket, you choose some (like 7) numbers out of them.
> For every round, 7 random numbers, called "prize numbers", will be generated.
> You will win a prize if your numbers match some prize numbers.
> For example, if you match 6 prize numbers, you may win the second prize.
> (If you match all 7 prize numbers, you will win the top prize (jackpot)!)

The concept of this lottery can be easily realized by using the "set" in problem 1. The numbers on a ticket and prize numbers can be considered as sets of integers. Then, finding how many numbers match turns into finding the size of their intersection.
You can #include the "customset.h" you wrote for problem 1.

Based on this, we can construct a class:

```
class LotteryTicket
{
public:
    LotteryTicket();
    LotteryTicket(const CustomSet<int>& numbers, int round);
    virtual ~LotteryTicket();

    int getRound();
    int getPrice();
    virtual int getTicketType() = 0;
    void setCost(const int& cost);
    virtual int claimPrize(const CustomSet<int>& prizeNumbers, int round) = 0;

private:
    CustomSet<int> m_numbers;
    int m_round; // The valid round of a ticket. An outdated ticket is invalid.
    int m_cost; // The amount of money spent on this ticket.
```

```
};
```

The basic information of ticket is the numbers on it and its valid round. The valid round identifies its validity when claiming for prize. (You cannot claim to the prize numbers of 5th round if you bought a lottery ticket for the 4th round.) It will be also useful if you store how much money you spent on buying a ticket.

As you can see, this class contains pure virtual functions like `claimPrize()`, and is therefore a virtual class. Two classes for two different lotteries are to be derived from this class. Their names and prize rules are as below:

The rule of "Lotto 7", represented by `class Lotto7Ticket`, is to choose 7 numbers from 1 to 30, inclusive. Each single ticket costs 2 Yuan. The prizes are listed below:

Level	Conditions	Prize
Top Prize (jackpot)	Matching all 7 prize numbers	¥2,000,000
Second Prize	Matching 6 prize numbers	¥4,500
Third Prize	Matching 5 prize numbers	¥75
Fourth Prize	Matching 4 prize numbers	¥5

The rule of "Lotto 6", represented by `class Lotto6Ticket`, is to choose 6 numbers from 1 to 43, inclusive. Each single ticket costs 2 Yuan. The prizes are listed below:

Level	Conditions	Prize
Top Prize (jackpot)	Matching all 6 prize numbers	¥2,000,000
Second Prize	Matching 5 prize numbers	¥4,500
Third Prize	Matching 4 prize numbers	¥75
Fourth Prize	Matching 3 prize numbers	¥5

*Based on "Lucky 7" from China Welfare Lottery, and "LOTO6" from Takarakuji in Japan. With variations.

A ticket wins only the highest prize it fulfills. (i.e. A ticket for Lotto 6 with 4 matching numbers wins only 75 Yuan, not 75+5 Yuan.)

*Multiple purchasing:

Multiple purchasing is a way for lottery players to easily buy multiple tickets. A multiple ticket chooses more numbers and costs more, but is counted as all single ticket that these numbers cover. For example:

A multiple ticket of Lotto 6 can be a ticket with 7 numbers:

{2, 7, 16, 21, 30, 37, 43}

This multiple ticket counts as 7 single tickets and therefore costs 14 Yuan:

(2,7,16,21,30,37) (2,7,16,21,30,43) (2,7,16,21,37,43) (2,7,16,30,37,43)
(2,7,21,30,37,43) (2,16,21,30,37,43) (7,16,21,30,37,43)

Similarly, a multiple ticket of Lotto 6 with 9 numbers would count for 84 single tickets, and would cost 168 Yuan.

However, if numbers in your multiple ticket match the prize numbers, you can win prizes for many single tickets! For example, if you bought the {2, 7, 16, 21, 30, 37, 43} above, and the prize number is {2, 7, 16, 21, 22, 23}, you will win 3 Third Prizes and four Fourth Prizes! (You can verify it.)

Steps for this problem

Step 1: Derive classes for tickets

The most of the work is to derive two classes from the given base class `LotteryTicket`. Their names should be `class Lotto7Ticket` and `class Lotto6Ticket`.

Important functions are their constructors and `claimPrize()`.

In constructors, it is guaranteed that the parameter set contains enough numbers. This ticket should be well initialized, and its price should be calculated. For example, a Lotto6 ticket with 7 numbers has a price of 14.

In `claimPrize()`, the parameter is a set of prize numbers and their round number. It returns how much this ticket wins. If the round number is wrong or the ticket does not win any prize, return 0. Otherwise, return the amount of money it wins. Pay special attention if it is a multiple ticket!

The declarations of `class LotteryTicket`, `class Lotto7Ticket`, and `class Lotto6Ticket` should all be put into a single header file named "tickets.h" or "tickets.hpp". All implementations should be in a source file named "tickets.cpp".

Step 2: A class for lottery station

Now you can put your lotteries into a playable station. An outlook of the class:

```
class LotteryStation
{
public:
    LotteryStation();
    ~LotteryStation();

    LotteryTicket* buy(CustomSet<int> numbers, int tickettype);
    // Represents buying a ticket of "tickettype" with "numbers" at round "m_round".
    // If succeeds, output a message:
    // "Bought a (??) ticket for (??) Yuan at round (??).\n"
    // Then return a pointer to it. See examples for more details.
    // If the numbers are not enough, or some numbers are out of bounds for the ticket,
    //   output "Failed.\n" and return nullptr(NULL).

    void claimPrize(LotteryTicket* ticket);
```

```
        // Claims prize for a given ticket.
        // You should use the claimPrize() function of the ticket.
        // However, you should check the type of the ticket first,
        //   as you need to pass the right prize numbers as parameter.
        // Output a message in the end:
        // "This ticket wins (??) Yuan.\n"
        // You should not claim a ticket that's already claimed or with wrong rounds.
        // In either case above, simply print: "This ticket wins 0 Yuan.\n"

        void newRound();
        // Begins a new round, generating new prize numbers for Lotto7 and Lotto6.
        // You may use the randInt() function given.

       bool setPrizeNumbers(CustomSet<int> numbers, int tickettype);
       // This function simply serves for the purpose of:
       // Making it easier for you to debug, and also easier for us to check your results.
       // It sets the prize numbers of the type given as the "numbers" given.
       // If "numbers" are invalid (not enough/out of bounds), do nothing and return false.
       // If succeeded, return true.
   private:
        int m_round; // Should be initialized to 0.
        // Other private variables...
   };
```

You can begin a round, buy several tickets of that round and claim prizes for them, and then begin another round. After a new round has begun, you cannot claim tickets of former rounds. See the example below for details.

An additional (trivial) requirement: If there is any function that you think should return a const value (for example, `size()`), please make the return type `const`. It's OK to do so only on functions that should be const for certain. If you fail to do so throughout your codes, you may receive a slight point deduction.

You should put declarations into a header file named "lotterystation.h" or "lotterystation.hpp", and implementations into a source file named "lotterystation.cpp".

Even though a function is so trivial that it contains only one line, it is still a good manner to put it into the source file rather than write in header file, for the sake of consistency.

● Example output

You should test and debug on your own before submitting. Here is an example test case:

```
int main()
```

```
{
    int a[6] = { 1,2,3,4,5,6 };
    CustomSet<int> foo(a, 6);
    LotteryStation sample;
    sample.newRound();
    sample.setPrizeNumbers(foo, LOTTO6); // LOTTO6 is predefined
    LotteryTicket* jackpot = sample.buy(foo, LOTTO6);
    // prints "Bought a Lotto 6 ticket for 2 Yuan at round 1.\n"
    sample.claimPrize(jackpot);
    // prints "This ticket wins 2000000 Yuan.\n"
}
```

- **Submission details**

You are going to submit five files, named

"customset.h" (or "customset.hpp"),
"tickets.h" (or "tickets.hpp"),
"tickets.cpp",
"lotterystation.h" (or "lotterystation.hpp"),
"lotterystation.cpp",

respectively.

As we are switching to Geekpie OJ, the address of OJ is not the same as before. More details on how to submit to OJ will be announced soon through piazza (1st source), please pay close attention to instructor notes.

Problem 3: Social Networking

After Gezi Wang's girlfriend broke up with him because he cannot help her with CS100 homework, he is hurt badly, and is going to "double-delete" his girlfriend in the popular social app "Gechat". ("Double-deleting" means that Gezi Wang will delete her girlfriend from his "friend list", and will also delete himself from his girlfriend's "friend list".)

The third task of this homework is based on the scenario above. In general, you are to write a class `GechatUser` to represent users of the "Gechat" app that supports features like adding and deleting friends.

● Basic Concepts

A username is all it's needed to create a `GechatUser`. All usernames should be distinct and are used for identification.

Friends are mutual in Gechat. Adding a friend also adds you as a friend of this person. Adding someone as a friend puts both of you into the friend lists of each other. You can check if someone is your friend, or check the number of friends you have.

Friends can be singly or doubly deleted through usernames. For example, there are two users, user1 and user2. Their usernames are also "user1" and "user2".

> Calling `user1.singleDelete("user2");` will remove user2 from user1's friend list. User2 is now not a friend of user1, but user1 is still a friend of user2.
>
> Calling `user1.doubleDelete("user2");` will remove user2 from user1's friend list, as well as remove user1 from user2's friend list. Now, neither of them is a friend to the other.

● Implementation

For this task, we would like you to practice the use of shared pointers, which is a kind of smart pointers that has been specially noted in class. Using shared pointers can free yourselves from caring about potential memory leaks when you delete friends.

One important thing to keep in mind is that you should never mix using raw pointers and smart pointers. Every time you create a raw pointer to an object that involves using smart pointers, convert it to a smart pointer!

It is recommended that you store the friend list in an STL container of *`shared_ptr`*, either a *`vector`* or a *`list`*, at your free choice.

In this task, a "Gechat Group" is not implemented in the form of a class, but rather represented by a `CustomSet` of `GechatUser`.

Note that in `CustomSet`, we have assumed many operators of `T`, its template parameter, to be well-defined:

`CustomSet::sortSet()` involves the use of the comparison operator `operator<`;
`CustomSet::printSet()` involves the use of `operator<<` of *`ostream`*.

These operators, however, are not overloaded in `GechatUser`. Letting `CustomSet` use the default operators will cause trouble and unexpected results. Therefore, you will need to overload them on your own, in order to safely use functions in `CustomSet`.

These functions should be implemented so that:

`CustomSet::sortSet()` sorts the set of `GechatUser` in lexicographical order of their usernames.

`CustomSet::printSet()` prints each `GechatUser` in the form of:

> "USERNAME (friends: ??)", where ?? is the number of friends of this user.
>
> (See example below)

If you are having trouble, especially if on `operator<<` of *`ostream`*, please refer to slides for usage of operators and the keyword `friend`.

Part of the declaration of `class GechatUser` is provided:

```
class GechatUser
{
public:
    // Assume using namespace std;
    GechatUser();
    GechatUser(string username);
    ~GechatUser();

    int numFriends();
    bool haveFriend(shared_ptr<GechatUser> user); // return true if user is a friend to *this.
    bool addFriend(shared_ptr<GechatUser> user);// return false if invalid or already a friend
    bool singleDelete(shared_ptr<GechatUser> user); // return false if invalid or not a friend
    bool singleDelete(string username); // return false only if username not found in friends.
    bool doubleDelete(shared_ptr<GechatUser> user); // return false if invalid or not a friend
    bool doubleDelete(string username); // return false only if username not found in friends.
    // Not complete...
private:
};
```

An additional (trivial) requirement: If there is any function that you think should return a const value (for example, `size()`), please make the return type `const`. It's OK to do so only on functions that should be const for certain. If you fail to do so throughout your codes, you may receive a slight point deduction.

You should put declarations into a header file named "gechatuser.h" or "gechatuser.hpp", and implementations into a source file named "gechatuser.cpp".

Example output

You should test and debug on your own before submitting. Here is an example test case:

```
int main() {
    shared_ptr<GechatUser> gzw(new GechatUser("GeziWang"));
    shared_ptr<GechatUser> gf(new GechatUser("Girlfriend"));
    gzw->addFriend(gf);
    gf->doubleDelete(gzw);
    CustomSet<GechatUser> sad_story;
    sad_story.add(*gzw);
    sad_story.add(*gf);
    sad_story.printSet();
    // prints "{GeziWang (friends: 0), Girlfriend (friends: 0)}\n"
}
```

Submission details

You are going to submit three files, named

"customset.h" (or "customset.hpp"),
"gechatuser.h" (or "gechatuser.hpp"),
"gechatuser.cpp",

respectively.

As we are switching to Geekpie OJ, the address of OJ is not the same as before. More details on how to submit to OJ will be announced soon through piazza (1st source), please pay close attention to instructor notes.