**实验名称：Operator Overloading and Class Template 重修刷分**

**班级: 软件工程2021级中外1班 学号：** 222021321062002 **姓名：** 雷艺湧

|  |  |  |  |
| --- | --- | --- | --- |
| **分析设计** | **代码实现** | **分析总结** | **总评** |
|  |  |  |  |

**实验类型：设计型**

**实验目的：**

1. Master the use of operator overloading.
2. Be able to define and use class inheritance relationships, and define derived classes.
3. Be able to define and use class template.
4. Be able to use STL.

**实验要求：**

1. Define a class called StringSet that will be used to store a set of STL strings. Use an array or a vector to store the strings. Create a constructor that takes as an input parameter an array of strings for the initial values in the set. Then write member functions to add a string to the set, remove a string from the set, clear the entire set, return the number of strings in the set, and output all strings in the set. Overload the + operator so that it returns the union of two StringSet objects. Also overload the \* operator so that it returns the intersection of two StringSet objects. Write a program to test all member functions and overloaded operators in your class.
2. Reliability and Maintainability Simulation involves a large number of random variables. The use of dynamic arrays will greatly improve the efficiency of simulation and the scale of problem solving. Please design Vector.

This problem requires the implementation of a vector class template, which can realize the storage and access of data.

(1) [] operator can only access the existing elements.

(2) The add method can automatically expand the internal storage space when accessing.

Note that the behavior of this vector is different from that of std:: vector.

Function interface definition:

template <class T>

class Vector {

...

}

Example of referee test procedure:

#include <iostream>

using namespace std;

/\* Todo: write down your code here! \*/

int main()

{

Vector<int> vint;

int n,m;

cin >> n >> m;

for ( int i=0; i<n; i++ ) {

// add() can inflate the vector automatically

vint.add(i);

}

// get\_size() returns the number of elements stored in the vector

cout << vint.get\_size() << endl;

cout << vint[m] << endl;

// remove() removes the element at the index which begins from zero

vint.remove(m);

cout << vint.add(-1) << endl;

cout << vint[m] << endl;

Vector<int> vv = vint;

cout << vv[vv.get\_size()-1] << endl;

vv.add(m);

cout << vint.get\_size() << endl;

}

Input example:

100 50

Output example:

100

50

99

51

-1

100

**实验内容与设计：**（学生作答区）

说明：根据每一实验要求，给出：（1）分析与设计（画出类图、算法流程图等）；（2）程序代码（注意代码风格和必要注释）；（3）测试数据和执行结果（用截图展现输入和输出）

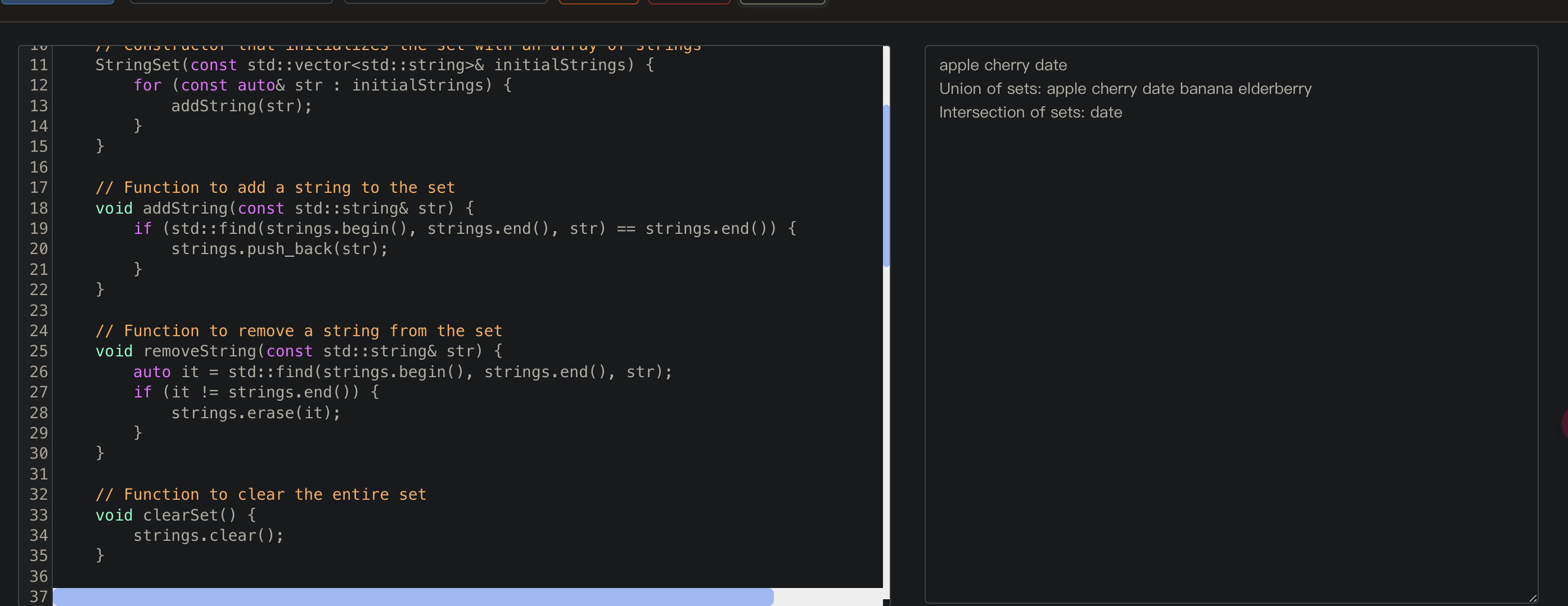
第一题

要求完我们创建一个StringSet类，包含以下函数：1.构造函数接受一个字符串并初始化到集合中。2.addString讲不存在集合中的字符串加入到集合中。3.removeString把存在集合中的函数从集合中删除。4.clearSet清空集合。5.size输出集合大小。6.output输出集合。同时还需要重载加号+和乘号\*，实现集合的并集和交集操作。

我的代码如下

|  |
| --- |
| #include <iostream>  #include <vector>  #include <algorithm> // for std::find  class StringSet {  private:  std::vector<std::string> strings;  public:  // Constructor that initializes the set with an array of strings  StringSet(const std::vector<std::string>& initialStrings) {  for (const auto& str : initialStrings) {  addString(str);  }  }  // Function to add a string to the set  void addString(const std::string& str) {  if (std::find(strings.begin(), strings.end(), str) == strings.end()) {  strings.push\_back(str);  }  }  // Function to remove a string from the set  void removeString(const std::string& str) {  auto it = std::find(strings.begin(), strings.end(), str);  if (it != strings.end()) {  strings.erase(it);  }  }  // Function to clear the entire set  void clearSet() {  strings.clear();  }  // Function to return the number of strings in the set  size\_t size() const {  return strings.size();  }  // Function to output all strings in the set  void output() const {  for (const auto& str : strings) {  std::cout << str << " ";  }  std::cout << std::endl;  }  // Overload + operator to return the union of two StringSet objects  StringSet operator+(const StringSet& other) const {  StringSet result = \*this;  for (const auto& str : other.strings) {  result.addString(str);  }  return result;  }  // Overload \* operator to return the intersection of two StringSet objects  StringSet operator\*(const StringSet& other) const {  StringSet result({});  for (const auto& str : strings) {  if (std::find(other.strings.begin(), other.strings.end(), str) != other.strings.end()) {  result.addString(str);  }  }  return result;  }  };  int main() {  std::vector<std::string> initialStrings = {"apple", "banana", "cherry"};  StringSet set1(initialStrings);  set1.addString("date");  set1.removeString("banana");  set1.output(); // Output: apple cherry date  StringSet set2({"banana", "date", "elderberry"});  StringSet unionSet = set1 + set2;  StringSet intersectionSet = set1 \* set2;  std::cout << "Union of sets: ";  unionSet.output(); // Output: apple cherry date elderberry  std::cout << "Intersection of sets: ";  intersectionSet.output(); // Output: date  return 0;  } |

代码运行结果如下：



问题二：

要求我们设计一个Vector类，用来存储和访问数据，且使用[]运算符只能访问现有的元素，而且使用add方法时要自动扩展内部空间，

我的代码如下：

|  |
| --- |
| #include <iostream>  #include <stdexcept> // for std::out\_of\_range  using namespace std;  template <class T>  class Vector {  private:  T\* data;  size\_t capacity;  size\_t size;  void expand() {  capacity \*= 2;  T\* newData = new T[capacity];  for (size\_t i = 0; i < size; ++i) {  newData[i] = data[i];  }  delete[] data;  data = newData;  }  public:  // Constructor  Vector() : capacity(10), size(0) {  data = new T[capacity];  }  // Destructor  ~Vector() {  delete[] data;  }  // Add method  void add(const T& value) {  if (size == capacity) {  expand();  }  data[size++] = value;  }  // Get size method  size\_t get\_size() const {  return size;  }  // Remove method  void remove(size\_t index) {  if (index >= size) {  throw std::out\_of\_range("Index out of range");  }  for (size\_t i = index; i < size - 1; ++i) {  data[i] = data[i + 1];  }  --size;  }  // Overload [] operator  T& operator[](size\_t index) {  if (index >= size) {  throw std::out\_of\_range("Index out of range");  }  return data[index];  }  const T& operator[](size\_t index) const {  if (index >= size) {  throw std::out\_of\_range("Index out of range");  }  return data[index];  }  // Copy constructor  Vector(const Vector& other) : capacity(other.capacity), size(other.size) {  data = new T[capacity];  for (size\_t i = 0; i < size; ++i) {  data[i] = other.data[i];  }  }  // Assignment operator  Vector& operator=(const Vector& other) {  if (this != &other) {  delete[] data;  capacity = other.capacity;  size = other.size;  data = new T[capacity];  for (size\_t i = 0; i < size; ++i) {  data[i] = other.data[i];  }  }  return \*this;  }  };  // Test program  int main() {  Vector<int> vint;  int n, m;  cin >> n >> m;  for (int i = 0; i < n; i++) {  // add() can inflate the vector automatically  vint.add(i);  }  // get\_size() returns the number of elements stored in the vector  cout << vint.get\_size() << endl;  cout << vint[m] << endl;  // remove() removes the element at the index which begins from zero  vint.remove(m);  cout << vint.add(-1) << endl;  cout << vint[m] << endl;  Vector<int> vv = vint;  cout << vv[vv.get\_size() - 1] << endl;  vv.add(m);  cout << vint.get\_size() << endl;  return 0;  } |

我的运行结果如下：

|  |
| --- |
|  |

**分析总结：**（学生作答区）

说明：遇到的bug和排错、多种解决方法、实验要求之外更多的尝试、心得体会等。

问题一的难点在于两个符号的重载。

加号重载，要实现的是两个集合的并集，我的思路是新建一个集合去装另外两个结合的元素。

|  |
| --- |
| StringSet operator+(const StringSet& other) const {  StringSet result = \*this; // 创建一个当前对象的副本  for (const auto& str : other.strings) { // 遍历另一个集合的元素  result.addString(str); // 添加到结果集合中  }  return result; // 返回结果集合  } |

这里中要排出二者重复的元素，使用的是类中的addString方法。这个方法会遍历集合，看看传入的string是否在集合中，在就不添加，不在才进行添加。

|  |
| --- |
| // Function to add a string to the set  void addString(const std::string& str) {  if (std::find(strings.begin(), strings.end(), str) == strings.end()) {  strings.push\_back(str);  }  } |

乘号的重载要求实现两个集合的交集，我的思路是把存在于两个集合中的元素存储到一个新集合中。

|  |
| --- |
| StringSet operator\*(const StringSet& other) const {  StringSet result({}); // 创建一个空的结果集合  for (const auto& str : strings) { // 遍历当前集合的元素  if (std::find(other.strings.begin(), other.strings.end(), str) != other.strings.end()) {  result.addString(str); // 如果元素在另一个集合中，则添加到结果集合中  }  }  return result; // 返回结果集合  } |

问题二的难点在与如何去自动扩容。我的思路就是在添加新元素时，先去做一个判断，如果超过了容量，就要去进行扩容。扩容的流程是：新建一个两倍大小的容器，把原来容器中的元素赋值进去，在把指向原来容器的指针指向新容器，设置容器大小为原来两倍，在删除原来容器。

|  |
| --- |
| void expand() {  capacity \*= 2;  T\* newData = new T[capacity];  for (size\_t i = 0; i < size; ++i) {  newData[i] = data[i];  }  delete[] data;  data = newData;  } |