Контрольная работа

Тема: Последовательные контейнеры STL и модульное тестирование

Цель: Сформировать практические навыки разработки абстракций данных на основе контейнеров STL и модульного тестирования средствами Visual Studio.

Задание

Реализовать обработку данных пользовательского типа (объектов класса) с помощью контейнера в соответствии с вариантом задания и со следующей спецификацией:

- приложение заполняет контейнер данными, которые вводятся пользователем с консоли;
- выводит содержимое контейнера на консоль для контроля ввода;
- выполняет сортировку контейнера в порядке возрастания значений объектов с помощью алгоритма или метода контейнера;
- выводит содержимое контейнера на консоль для контроля операции;
- выполняет сортировку контейнера в порядке убывания значений объектов с помощью алгоритма или метода контейнера;
- выводит содержимое контейнера на консоль для контроля операции;
- вычисляет сумму значений объектов с помощью алгоритма и выводит значение на консоль.

Протестировать его, используя средства модульного тестирования Visual Studio. Тестовые наборы необходимо построить на основе критериев тестирования C_0,C_1,C_2 в зависимости от варианта задания.

Выбора варианта:

Номер варианта контрольной работы соответствует двум последним цифрам вашего пароля, если эти две цифры образуют число меньшее или равное 18. Если это число больше 18, то номер вашего варианта вы получите вычитанием из него числа 18.

Например, две последние цифры пароля 08, тогда номер вашего варианта будет 8.

Например, две последние цифры пароля 24, тогда номер вашего варианта будет 24 – 18 = 6.

Варианты задания

Варианты заданий контрольной работы представлены в таблице ниже. Вариант определяет тип контейнера используемого для обработки данных, тип значений помещаемых в контейнер, критерии тестирования разработанного приложения (таблица 1).

Таблица 1 - Варианты заданий контрольной работы

№ Варианта	Тип контейнера	Класс объектов	Критерий тестирования
1	deque	Простая дробь	C_0,C_1
2	deque	Комплексное число	C_0,C_1
3	deque	P-ичное число	C_0,C_1
4	deque	Простая дробь	C_1,C_2
5	**deque**	**Комплексное число**	**C_1,C_2**
6	deque	P-ичное число	C_1,C_2
7	vector	Простая дробь	C_0,C_1
8	vector	Комплексное число	C_0,C_1
9	vector	P-ичное число	C_0,C_1
10	vector	Простая дробь	C_1,C_2
11	vector	Комплексное число	C_1,C_2
12	vector	P-ичное число	C_1,C_2
13	list	Простая дробь	C_0,C_1
14	list	Комплексное число	C_0,C_1
15	list	P-ичное число	C_0,C_1
16	list	Простая дробь	C_1,C_2
17	list	Комплексное число	C_1,C_2
18	list	P-ичное число	C_1,C_2

Рекомендации к выполнению

1. Заданную обработку данных реализуйте как консольное приложение, используя классы языка программирования и библиотеку шаблонов STL.
2. Добавьте в класс объектов (в соответствии с вариантом задания), помещаемых в контейнер перегруженные операторы < и > для выполнения сортировки. В контрольной работе используются классы, разработанные вами в лабораторных работах.
3. Выполняйте сортировку контейнера с помощью алгоритма sort или метода контейнера в зависимости от варианта.
4. Вычисление суммы значений объектов контейнера выполняйте с помощью алгоритма accumulate и функционального объекта plus<A>(), здесь A – класс ваших объектов;

5. Для выполнения описанных в задании операций по обработке данных, разработайте класс со следующим описанием:

```
class InOutDo
{
public:
	static void Input(…)
	{
		//Вводим данные с клавиатуры и заносим в
контейнер.
	}
	static void Output(…)
	{
	//Выводим содержимое контейнера на монитор.
	}
	static A Sum(…)
	{
		//Находим сумму
	}
	static void SortUp(…)
	{
		//Сортируем вектор по возрастанию
	}
	static void SortDown(…)
	{
		//Сортируем вектор по убыванию
	}
};
```

Тогда функция main примет примерно такой вид:

```
int _tmain(int argc, _TCHAR* argv[])
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	//Описание вашего Контейнера с объектами вашего
класса A.
	//Контейнер<A> m;
	InOutDo::Input(m);
	InOutDo::Output(m);
	InOutDo::SortUp(m);
	InOutDo::Output(m);
	InOutDo::SortDown(m);
```

```
	InOutDo::Output(m);
	cout << InOutDo::Sum(m).get() << endl;
	system("PAUSE");
	return 0;
}
```

6. Для тестирования разработанного класса используйте средства модульного тестирования Visual Studio.
7. Примеры программ обработки данных с помощью деки, вектора и списка приведены ниже.

Пример 1. Обработка данных с декой.

```
// ConsoleDeque.cpp: определяет точку входа для
консольного приложения.
//

#include "stdafx.h"
#include <iostream>
#include <string>
#include <sstream>
#include <deque>
#include <algorithm>
#include <numeric>//Определяет шаблонны функций
контейнера, которые выполняют алгоритмы числовой
обработки.
#include <functional>
#include "windows.h"

using namespace std;

class A
{
	int n, d;
public:
	A(int n = 0, int d = 1) : n(n), d(d){};
	A operator+(const A b)const
	{
		return A((n*b.d + b.n*d), d*b.d);
	};
	string get()
	{
		string a;
```

```
            ostringstream os;
            os << n << "/" << d;
            return os.str();
      };
      bool operator>(const A b)const
      {
            return n*b.d > d*b.n;
      };
      bool operator<(const A b)const
      {
            return n*b.d < d*b.n;
      };
};

int _tmain(int argc, _TCHAR* argv[])
{
      SetConsoleCP(1251);
      SetConsoleOutputCP(1251);
      //Дека простых дробей
      deque<A> m;
      //Итератор для деки
      deque<A>::iterator iter;
      //Проталкиваем дроби в деку
      m.push_back(A(2));
      m.push_back(A(3));
      m.push_back(A(1));
      m.push_back(A(5));
      m.push_back(A(9));
      m.push_back(A(7));
      //Сортируем деку по возрастанию
      sort(m.begin(), m.end(), less<A>());
      for (iter = m.begin(); iter != m.end(); iter++)
            cout << iter->get() << " ";
      cout << endl;
      //Сортируем деку по убыванию
      sort(m.begin(), m.end(), greater<A>());
      for (int j = 0; j != m.size(); j++)
            cout << m[j].get() << " ";
      cout << endl;
      //Находим сумму
      A sum = accumulate(m.begin(), m.end(), A(),
plus<A>());
```

```
     cout << sum.get() << endl;
     system("PAUSE");
     return 0;
     return 0;
}
```

Пример 2. Обработка данных с вектором.

```
// ConsoleVector.cpp: определяет точку входа для
консольного приложения.
//

#include "stdafx.h"
#include <iostream>
#include <string>
#include <sstream>
#include <vector>
#include <algorithm>
#include <numeric>//Определяет шаблонны функций
контейнера, которые выполняют алгоритмы числовой
обработки.
#include <functional>
#include "windows.h"

using namespace std;
class A
{
     int n, d;
public:
     A(int n = 0, int d = 1) : n(n), d(d){};
     A operator+(const A b)const
     {
          return A((n*b.d + b.n*d), d*b.d);
     };
     string get()
     {
          string a;
          ostringstream os;
          os << n << "/" << d;
          return os.str();
     };
     bool operator>(const A b)const
     {
```

```
            return n*b.d > d*b.n;
        };
        bool operator<(const A b)const
        {
            return n*b.d < d*b.n;
        };
};

int _tmain(int argc, _TCHAR* argv[])
{
        SetConsoleCP(1251);
        SetConsoleOutputCP(1251);
        //Вектор простых дробей
        vector<A> m;
        //Итератор для вектора
        vector<A>::iterator iter;
        //Проталкиваем дроби в вектор
        m.push_back(A(2));
        m.push_back(A(3));
        m.push_back(A(1));
        m.push_back(A(5));
        m.push_back(A(9));
        m.push_back(A(7));
        //Сортируем вектор по возрастанию
        sort(m.begin(), m.end(), less<A>());
        for (iter = m.begin(); iter != m.end(); iter++)
            cout << iter->get() << " ";
        cout << endl;
        //Сортируем вектор по убыванию
        sort(m.begin(), m.end(), greater<A>());
        for (int j = 0; j != m.size(); j++)
            cout << m[j].get() << " ";
        cout << endl;
        //Находим сумму
        A sum = accumulate(m.begin(), m.end(), A(),
plus<A>());
        cout << sum.get() << endl;
        system("PAUSE");
        return 0;
}
```

Пример 3. Обработка данных со списком.

```
// ConsoleList.cpp: определяет точку входа для
консольного приложения.
//
#include "stdafx.h"
#include <iostream>
#include <string>
#include <sstream>
#include <list>
#include <algorithm>
#include <numeric>//Определяет шаблонны функций
контейнера, которые выполняют алгоритмы числовой
обработки.
#include <functional>
#include "windows.h"

using namespace std;
class A
{
     int n, d;
public:
     A(int n = 0, int d = 1) : n(n), d(d){};
     A operator+(const A b)const
     {
          return A((n*b.d + b.n*d), d*b.d);
     };
     string get()
     {
          string a;
          ostringstream os;
          os << n << "/" << d;
          return os.str();
     };
     bool operator>(const A b)const
     {
          return n*b.d > d*b.n;
     };
     bool operator<(const A b)const
     {
          return n*b.d < d*b.n;
     };
};
```

```
int _tmain(int argc, _TCHAR* argv[])
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);
	//Список простых дробей
	list<A> m;
	//Итератор для списка
	list<A>::iterator iter;
	//Проталкиваем дроби в вектор
	m.push_back(A(2));
	m.push_back(A(3));
	m.push_back(A(1));
	m.push_back(A(5));
	m.push_back(A(9));
	m.push_back(A(7));
	//Сортируем список по возрастанию
	//sort(m.begin(), m.end(), );
	m.sort(less<A>());
	for (iter = m.begin(); iter != m.end(); iter++)
		cout << iter->get() << " ";
	cout << endl;
	//Сортируем список по убыванию
	//sort(m.begin(), m.end(), );
	m.sort(greater<A>());
	for (iter = m.begin(); iter != m.end(); iter++)
		cout << iter->get() << " ";
	cout << endl;
	//Находим сумму
	A sum = accumulate(m.begin(), m.end(), A(),
plus<A>());
	cout << sum.get() << endl;
	system("PAUSE");
	return 0;
	return 0;
}
```

Пример модульного теста.

```
#include "stdafx.h"
#include "CppUnitTest.h"
#include "..\ConsoleDeque\A.h"
#include "..\ConsoleDeque\InOutDo.h"
```

```
using namespace
Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest1
{
	TEST_CLASS(UnitTest1)
	{
	public:

		TEST_METHOD(TestMethod1)
		{
			A d = A(0, 1);
			deque<A> m;
			InOutDo::Input(m);
			d = InOutDo::Sum(m);
			string s = "27/1";
			Assert::AreEqual(s, d.get());
		}

	};
}
```

Порядок выполнения

Реализуйте заданную абстракцию данных в режиме консольного приложения:

1. Создайте консольное приложение и сохраните его под именем CJob_1.
2. Добавьте к исходному тексту консольного приложения описание вашего класса в соответствии с вариантом задания.
3. Добавьте в описание класса перегруженные операторы отношения (<,>) и, если необходимо, недостающие методы.
4. Добавьте к исходному тексту консольного приложения описание класса InOutDo в соответствии с заданием.
5. Разработайте тестовый набор данных для тестирования операций класса InOutDo и протестируйте их.
6. Разработайте тестовый набор данных для тестирования операций, заданных на множестве. Тестовый набор поместите в таблицу следующего вида:

Таблица 1 - Тестовый набор для тестирования шаблона классов «множество»

Тестовый набор для тестирования операции Сложить множества целых чисел				
Номер теста	Исходные данные		Ожидаемый результат	
	Вход	контейнер	Возвращаемое значение	контейнер
1	()	()	()	()
2	(0)	()	(0)	()
3	(1)	(0)	(1 0)	(0)
4	(1 0)	(1 0)	(1 0)	(1 0)
5	(1 2 3)	(3 4 5)	(1 2 3 4 5)	(3 4 5)

7. Протестируйте разработанную абстракцию данных с помощью средств модульного тестирования Visual Studio.

Контрольные вопросы

1. В чём состоит сущность критерия C_0?
2. В чём состоит сущность критерия C_1?
3. В чём состоит сущность критерия C_2?
4. Что такое УПГ?
5. Что такое путь в УПГ?
6. Что такое ветвь УПГ?
7. В каком файле описан последовательный контейнер deque?
8. В каком файле описан последовательный контейнер vector?
9. В каком файле описан последовательный контейнер list?
10. Что означает имя iterator в области видимости последовательного контейнера?
11. Что такое функциональный объект?
12. Назначение и параметры алгоритма sort() для последовательных контейнеров?
13. Назначение и параметры алгоритма accumulate () для последовательных контейнеров?
14. Назначение метода size() последовательного контейнера?
15. В чём особенности статических методов?
16. В чём особенности последовательных контейнеров?

Содержание отчета

1. Задание.
2. Текст программы.
3. Скриншот.

Литература

1. Написание модульных тестов для C/C++ в Visual Studio [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/visualstudio/test/writing-unit-tests-for-c-cpp (дата обращения 21.03.18).
2. Руководство по программированию на C# [Электронный ресурс] URL: https://metanit.com/cpp/tutorial/1.1.php (дата обращения 20.03.18).