Лабораторная работа №5. Перегрузка операторов

Практические задания

На оценку «**удовлетворительно**» необходимо выполнить обязательное задание и набрать **не менее 60 баллов**.

На оценку «**хорошо**» необходимо выполнить дополнительное задание и (возможно, обязательное), набрав **не менее 400 баллов**.

На оценку «**отлично**» необходимо выполнить обязательное и дополнительное задания, набрав **не менее 600 баллов**.

Обязательные задания

Задание 1

Вариант 1. До 140 баллов

Разработайте класс CComplex, моделирующий работу с комплексными числами.

Каркас класса:

```
class CComplex
{
public:
	// инициализация комплексного числа значениями действительной и мнимой частей
	CComplex(double real = 0, double image = 0);

	// возвращает действительную часть комплексного числа
	double Re()const;

	// возвращает мнимую часть комплексного числа
	double Im()const;

	// возвращает модуль комплексного числа
	double GetMagnitude()const;

	// возвращает аргумент комплексного числа
```

```
        double GetArgument()const;
};
```

Реализуйте конструктор и методы класса, а также следующие операторы:

№	Оператор	Описание	Обязательно
1	Бинарный +	Реализовать две версии данного оператора: • Возвращает сумму двух комплексных чисел • Возвращает сумму действительного и комплексного числа Явная реализация суммы комплексного и действительного числа не потребуется, т.к. будет вызван оператор суммы двух комплексных чисел благодаря конструктору класса CComplex, который выполнит необходимое преобразование типа double в CComplex автоматически	Да
2	Бинарный -	Реализовать две версии данного оператора • Возвращает разность двух комплексных чисел • Возвращает разность действительного и комплексного числа Явная реализация разности комплексного и действительного числа не потребуется, т.к. будет вызван оператор разности двух комплексных чисел благодаря конструктору класса CComplex, который выполнит необходимое преобразование типа double в CComplex автоматически	Да
3	*	Реализовать две версии данного оператора • Возвращает произведение двух комплексных чисел • Возвращает произведение действительного и комплексного числа	Да
4	/	Реализовать две версии данного оператора • Возвращает частное двух комплексных чисел • Возвращает частное действительного и комплексного числа Убедиться в том, что комплексные числа	Да
5	Унарный + и Унарный -	Возвращают копию комплексного числа и противоположное комплексное число, соответственно.	Да
6	+=	Достаточно реализовать лишь следующую версию данного оператора complex += complex Благодаря конструктору класса CComplex она будет вызываться и для приращения на величину действительного числа	Да
7	-=	Аналогично +=	Да
8	*=	Аналогично -=	Да
9	/=	Аналогично /=	Да
10	==	Реализовать две версии данного оператора, выполняющие сравнение: • Комплексного числа с комплексным числом (она же будет вызываться при сравнении комплексного числа с действительным) • Действительного числа с комплексным числом Внимание, т.к. действительная и мнимая части комплексного числа хранятся в виде чисел с плавающей запятой для их сравнения следует использовать сравнение с заданной точностью: `bool numbersAreEqual = (fabs(number1 - number2) < DBL_MIN);` Примечание: константа DBL _MIN – объявлена в заголовочном файле float.h, а функция fabs – в math.h	Да
12	!=	Проверяет комплексные числа (а также действительное число и комплексное) на неравенство.	Да
13	<<	Оператор вывода в выходной поток std::ostream в формате **Re±Im**i,	Нет

		например: -3.5-4.8i 4+2i -3+2i 3-2i	
14	>>	Оператор ввода из входного потока std::istream в формате **Re±Im**i, например: -3.5-4.8i 4+2i -3+2i 3-2i	Нет

За реализацию каждой группы операторов из списка: 10 баллов

В процессе разработки класса использование подхода TDD обязательно. Без автоматических тестов работа будет принята с коэффициентом 0.5.

Вариант 2. До 120 баллов

Разработайте класс, моделирующий работу с трехмерными векторами. Каркас класса:

```
class CVector3D
{
public:
   // Конструирует нулевой вектор
   CVector3D();

   // Конструирует вектор с заданными координатами
   CVector3D(double x0, double y0, double z0);

   // Возвращает длину вектора
   double GetLength()const;

   // Нормализует вектор (приводит его к единичной длине)
   void Normalize();

   // Другие методы и операции класса

   // В данном случае данные можно сделать публичными
   double x, y, z;
};
```

Реализуйте следующие операторы над трехмерными векторами:

№	Операции	Описание	Обязательно
1	Унарный + и -	Унарный минус возвращает вектор с противоположным направлением: v1 = -v2 Унарный плюс возвращает тот же вектор (для полноты)	Да
2	Бинарный +	Возвращает результат сложения векторов	Да
3	Бинарный -	Возвращает результат вычитания векторов	Да
4	+=	Выполняют увеличение длины вектора на длину второго вектора: a += b;	Да
5	-=	Аналогично +=	Да
6	*	Умножает вектор на скаляр и скаляр на вектор.	Да
7	/	Выполняет деление вектора на скаляр.	Да
8	*=	Умножает вектор на скаляр: v *= 3.7;	Да
9	/=	Делит вектор на скаляр: v /= 17;	Да
10	== и !=	Выполняет проверку векторов на приблизительное равенство (или	Да

		неравенство). Вектора считаются равными, если абсолютная величина (модуль) разности их соответствующих координат не превышает некоторой величины (можно использовать DBL_EPSILON из заголовочного файла float.h)	
11	<<	Оператор вывода в выходной поток std::ostream в формате **x, y, z**, например: 3, -2.5, 7	Нет
12	>>	Оператор ввода из входного потока std::istream в формате **x, y, z**, например: 3, -2.5, 7	Нет

Реализуйте следующие функции для работы с трехмерными векторами:

№	Функция	Описание	Обязательно
13	DotProduct	Вычисляет результат скалярного произведения двух трехмерных векторов: `double DotProduct(CVector3D const& v1, CVector3D const& v2);`	Нет
14	CrossProduct	Вычисляет результат векторного произведения двух трехмерных векторов: `CVector3D CrossProduct(CVector3D const& v1, CVector3D const& v2);`	Нет
15	Normalize	Возвращает единичный вектор, имеющий то же направление, что и вектор, выступающий в качестве аргумента функции: `CVector3D Normalize(CVector3D const& v);`	Нет

За реализацию каждой группы операторов и функций из данных списков (помимо обязательных методов класса, указанных в каркасе), начисляется по 8 баллов.

В процессе разработки класса использование подхода TDD обязательно. Без автоматических тестов работа будет принята с коэффициентом 0.5.

Вариант 3. До 156 баллов

Разработайте класс CRational, моделирующий рациональное число и реализующий основные арифметические операций над рациональными числами.

Каркас класса CRational приведен ниже

```
class CRational
{
public:
   // Конструирует рациональное число, равное нулю (0/1)
   CRational();

   // Конструирует рациональное число, равное value (value/1)
   CRational(int value);

   // Конструирует рациональное число, равное numerator/denominator
   // Рациональное число должно храниться в нормализованном виде:
   // знаменатель положительный (числитель может быть отрицательным)
   // числитель и знаменатель не имеют общиз делителей (6/8 => 3/4 и т.п.)
   // Если знаменатель равен нулю, должно сконструироваться рациональное число, равное нулю
   CRational(int numerator, int denominator);

   // Возвращает числитель
   int GetNumerator()const;

   // Возвращает знаменатель (натуральное число)
   int GetDenominator()const;

   // Возвращает отношение числителя и знаменателя в виде числа double
   double ToDouble()const;
```

```
    // Прочие операторы согласно заданию
};
```

Реализуйте следующие операторы над рациональными числами. **В результате данных операций должны получаться нормализованные рациональные[1] числа**.

№	Операция	Описание	Обязательно
1	Унарный + и -	Унарный минус возвращает рациональное число с противоположным знаком: 3/5 => -3/5 Унарный плюс рациональное число, равное текущему.	Да
2	Бинарный +	Возвращает результат сложения двух рациональных чисел, рационального числа с целым, целого числа с рациональным[2]: $\frac{1}{2}+\frac{1}{6}=\frac{2}{3}$ $\frac{1}{2}+1=\frac{3}{2}$ $1+\frac{1}{2}=\frac{3}{2}$	Да
3	Бинарный -	Возвращает разность двух рациональных чисел, рационального числа и целого, либо целого числа и рационального[3]: $\frac{1}{2}-\frac{1}{6}=\frac{1}{3}$ $\frac{1}{2}-1=-\frac{1}{2}$ $1-\frac{1}{2}=\frac{1}{2}$	Да
4	+=	Выполняет увеличение рационального числа на величину второго рационального, либо целого числа[4]: $\frac{1}{2}+=\frac{1}{6}\rightarrow\frac{2}{3}$ $\frac{1}{2}+=1\rightarrow\frac{3}{2}$	Да
5	-=	Выполняет уменьшение рационального числа на величину второго рационального либо целого числа[5]: $\frac{1}{2}-=\frac{1}{6}\rightarrow\frac{1}{3}$ $\frac{1}{2}-=1\rightarrow-\frac{1}{2}$	Да
6	*	Возвращает результат произведения рациональных чисел, рационального и целого, либо целого и рационального[6]:	Да

[1] Нормализация необходима для того, чтобы в результате арифметических операций избежать чрезмерных значений числителя и знаменателя, способных выйти за пределы диапазона int. например, (49/100 + 1/100) * 9/30 в денормализованном виде равно 450/3000, а в нормализованном – 3/20).

[2] Подсказка: есть возможность реализовать все три типа сложения: CRational+int, CRational+CRational, int+CRational, написав оператор сложения всего один раз. Подумайте, каким образом этого можно добиться.

[3] См. пояснение насчет бинарного оператора сложения

[4] Здесь так же можно реализовать поддержку обоих случаев сложения, написав оператор += всего один раз. Объясните, почему?

[5] См. пояснение насчет оператора +=

[6] См. пояснение насчет бинарного оператора +

		$\frac{1}{2} * \frac{2}{3} = \frac{1}{3}$ $\frac{1}{2} * (-3) = -\frac{3}{2}$ $7 * \frac{2}{3} = \frac{14}{3}$	
7	/	Возвращает частное двух рациональных чисел, рационального и целого, целого и рационального[7]: $\frac{1}{2} \Big/ \frac{2}{3} = \frac{3}{4}$ $\frac{1}{2} \Big/ 5 = \frac{1}{10}$ $7 \Big/ \frac{2}{3} = \frac{21}{2}$	Да
8	*=	Умножает значение первого рационального числа на другое рациональное, либо целое[8]: $\frac{1}{2} *= \frac{2}{3} \to \frac{1}{3}$ $\frac{1}{2} *= 3 \to \frac{3}{2}$	Да
9	/=	Делит первое рациональное число на другое рациональное, либо целое[9]: $\frac{1}{2} /= \frac{2}{3} \to \frac{3}{4}$ $\frac{1}{2} /= 3 \to \frac{1}{6}$	Да
10	== и !=	Проверяют равенство (и неравенство) двух рациональных чисел, целого и рационального, рационального и целого[10]: $\frac{1}{2} == \frac{1}{2} \to true$ $\frac{1}{2} == \frac{2}{3} \to false$ $\frac{4}{1} == 4 \to true$ $\frac{1}{2} == 7 \to false$ $3 == \frac{3}{1} \to true$	Да

[7] См. пояснение насчет бинарного оператора +

[8] См. пояснение насчет оператора +=

[9] См. пояснение насчет оператора +=

[10] Есть возможность реализовать все три варианта сравнения в операторах == и !=, разработав по одной версии операторов == и !=. Подумайте, как?

		$3 == \frac{2}{3} \rightarrow false$ $\frac{1}{2} != \frac{1}{2} \rightarrow false$ $\frac{1}{2} != \frac{2}{3} \rightarrow true$ $\frac{4}{1} != 4 \rightarrow false$ $\frac{1}{2} != 7 \rightarrow true$ $3 != \frac{3}{1} \rightarrow false$ $3 != \frac{2}{3} \rightarrow true$	
11	<, <=, >, >=	Сравнивают два рациональных числа, рациональное с целым, целое с рациональным[11]: $\frac{1}{2} >= \frac{1}{3} \rightarrow true$ $\frac{1}{2} <= \frac{1}{3} \rightarrow false$ $\frac{3}{1} > 2 \rightarrow true$ $\frac{1}{2} < 7 \rightarrow true$ $3 <= \frac{7}{2} \rightarrow true$ $3 >= \frac{8}{2} \rightarrow false$	Да
12	<<	Оператор вывода рационального числа в выходной поток std::ostream в формате <числитель>/<знаменатель>, например: **7/15**	Нет
13	>>	Оператор ввода рационального числа из входного потока std::istream в формате <числитель>/<знаменатель>, например: **7/15**	Нет

За реализацию каждой группы операторов из данных списков (помимо обязательных методов класса, указанных в каркасе), начисляется по 12 баллов.

В процессе разработки класса использование подхода TDD обязательно. Без автоматических тестов работа будет принята с коэффициентом 0.5.

Бонус в 30 баллов за возможность получения смешанной дроби из рационального числа

Смешанная дробь – дробь, представленная в виде суммы целого числа и правильной дроби[12]:

[11] См. указания насчет операторов != и ==

[12] Правильной называется дробь, у которой модуль числителя меньше модуля знаменателя.

Например, рациональное число $\frac{9}{4}$ может быть представлено в виде смешанной дроби $2\frac{1}{4}$, а число $-\frac{9}{4}$ (минус девять четвертых) – в виде смешанной дроби: $-2+\left(-\frac{1}{4}\right)=-\left(2\frac{1}{4}\right)$.

```
class CRational
{
public:
   // Возвращает представление рационального числа в виде смешанной дроби
   std::pair<int, CRational> ToCompoundFraction()const;
};
```

Вариант 4. До 90 баллов

Реализуйте класс CTime, моделирующий время суток (количество часов, минут и секунд).

Каркас класса:

```
// моделирует время суток, задаваемое количеством часов (0-23), минут (0-59) и
// секунд (0-59)
class CTime
{
public:
	// инициализирует время заданным количеством часов, минут и секунд
	CTime(unsigned hours, unsigned minutes, unsigned seconds = 0);

	// инициализирует время количеством секунд после полуночи
	CTime(unsigned timeStamp = 0);

	// возвращает количество часов
	unsigned GetHours()const;

	// возвращает количество минут
	unsigned GetMinutes()const;

	// возвращает количество секунд
	unsigned GetSeconds()const;

	// возвращает информацию о корректности времени.
	// Например, после вызова конструктора CTime time(99, 32, 83);
	//    метод time.IsValid() должен возвращать false
	bool IsValid()const;
};
```

Реализуйте конструкторы и методы данного класса, а также следующие операторы.

№	Оператор	Описание	Обязательно
1	++ (префиксная и постфиксная формы)	Увеличивает время на 1 секунду	Да
2	-- (префиксная и постфиксная формы)	Увеличивает время на 1 секунду	Да
3	+	Выполняет сложение двух времен. Например: 14:30:25 + 03:18:44 = 17:49:09	Да
4	-	Выполняет вычитание двух времен. Например: 14:30:25 - 03:18:44 = 11:15:41	Да
5	+=		Да
6	-=		Да
7	*	Умножает время на целое число (и наоборот). Например: 03:05:15 * 3 = 3 * 03:05:15 = 09:15:45	Да

8	/	2 формы: • Деление времени на целое число. Например: 09:15:40 / 3 = 03:05:13 • Деление времени на время (нацело). Например: 09:15:40 / 03:05:13 = 3	Да
9	*=		Да
10	/=		Да
11	<<	Оператор вывода времени в поток вывода в формате ЧЧ:ММ:СС, либо INVALID, если время невалидно	Нет
12	>>	Оператор ввода времени из потока ввода в формате ЧЧ:ММ:СС, либо INVALID, если время невалидно	Нет
13	== и !=	Проверка двух значений времени на равенство и неравенство	Да
14	< и >	Проверка двух временных значений на строгое неравенство	Да
15	<= и >=	Проверка двух временных значений на нестрогое неравенство	Да

За реализацию каждой группы операторов из списка: 6 баллов.

При выходе результата после выполнения операций за пределы диапазона 00:00:00 – 23:59:59 приводить результат к этому диапазону. Например:

- 23:59:59 + 00:00:03 = 00:00:02
- 00:00:05 – 00:00:10 = 23:59:55

Подсказка: данный класс будет проще реализовать, если вместо трех приватных переменных (часы, минуты и секунды) вы будете использовать только одну единственную переменную для хранения секунд после полуночи и все операции производить над нею.

В процессе разработки класса использование подхода TDD обязательно. Без автоматических тестов работа будет принята с коэффициентом 0.5.

Вариант 5. До 110 баллов

Реализуйте класс CDate, моделирующий дату начиная с 1 января 1970 года. Каркас класса следующий:

```
// Месяц
enum Month
{
	JANUARY = 1, FEBRUARY, MARCH, APRIL,
	MAY, JUNE, JULY, AUGUST, SEPTEMBER,
	OCTOBER, NOVEMBER, DECEMBER
};

// День недели
enum WeekDay
{
	SUNDAY = 0, MONDAY, TUESDAY, WEDNESDAY,
	THURSDAY, FRIDAY, SATURDAY
};

// Дата в формате день-месяц-год. Год в диапазоне от 1970 до 9999
class CDate
{
public:
	// инициализируем дату заданными днем, месяцем и годом.
	// примечание: год >= 1970
	CDate(unsigned day, Month month, unsigned year);

	// инициализируем дату количеством дней, прошедших после 1 января 1970 года
```

```
        // например, 2 = 3 января 1970, 32 = 2 февраля 1970 года и т.д.
        CDate(unsigned timestamp = 0);

        // возвращает день месяца (от 1 до 31)
        unsigned GetDay()const;

        // возвращает месяц
        Month GetMonth()const;

        // возвращает год
        unsigned GetYear()const;

        // возвращает день недели
        WeekDay GetWeekDay()const;

        // возвращает информацию о корректности хранимой даты.
        // Например, после вызова CDate date(99, static_cast<Month>(99), 10983);
        // или после:
        //    CDate date(1, January, 1970); --date;
        // метод date.IsValid() должен вернуть false;
        bool IsValid()const;
};
```

Реализуйте конструкторы и методы данного класса, а также следующие операторы.

№	Оператор	Описание	Обязательно
1	++ (префиксная и постфиксная формы)	Переводит дату на следующий день	Да
2	-- (префиксная и постфиксная формы)	Переводит дату на предыдущий день	Да
3	+	Прибавляет к дате заданное целое количество дней. Например: 28/02/2010 + 3 = 03/03/2010	Да
4	-	Реализовать 2 версии данного оператора: • Вычитает из даты заданное количество дней. Например: 01/01/2010 - 2 = 30/12/2009 • Находит разность двух дат в днях. Например: 01/01/2010 - 30/12/2009 = 3 01/01/2010 - 03/01/2010 = -2	Да
5	+=	*<Дата> += <кол-во дней>*	Да
6	-=	*<Дата> -= <кол-во дней>*	Да
7	<<	Оператор вывода даты в поток вывода в формате ДД.ММ.ГГГГ, либо INVALID, если дата является недопустимой	Нет
8	>>	Оператор ввода времени из потока ввода в формате ДД.ММ.ГГГГ, либо INVALID, если дата является недопустимой	Нет
9	== и !=	Проверка двух дат на равенство и неравенство	Да
10	< и >	Проверка двух дат на строгое неравенство	Да
11	<= и >=	Проверка двух дат на нестрогое неравенство	Да

За реализацию каждой группы операторов из списка: 10 баллов

Результат применения данных операций к недопустимой дате не изменяет ее значения.

При выходе результата после выполнения операций за пределы диапазона 01:01:1970 – 31:12:9999 дата должна стать недопустимой.

Подсказка: данный класс будет проще реализовать, если вместо трех приватных переменных (день, месяц и год) вы будете использовать только одну единственную переменную для хранения количества дней после 1 января 1970 года и все операции производить над нею.

В процессе разработки класса использование подхода TDD обязательно. Без автоматических тестов работа будет принята с коэффициентом 0.5.

Дополнительные задания

Задание 2 – До 300 баллов

Реализовать и протестировать класс CMyString, моделирующий строку произвольной длины.

Внимание, **строка должна позволять хранить в середине символы с нулевым кодом** [13] . Проинициализировать такую строку можно при помощи конструктора, принимающего кроме адреса первого символа длину строки.

Каркас класса:

```
class CMyString
{
public:
	// конструктор по умолчанию
	CMyString();

	// конструктор, инициализирующий строку данными строки
	// с завершающим нулевым символом
	CMyString(const char * pString);

	// конструктор, инициализирующий строку данными из
	// символьного массива заданной длины
	CMyString(const char * pString, unsigned length);

	// конструктор копирования
	CMyString(CMyString const& other);

	// перемещающий конструктор (для компиляторов, совместимых с C++11)
	//  реализуется совместно с перемещающим оператором присваивания
	CMyString(CMyString && other);

	// конструктор, инициализирующий строку данными из
	// строки стандартной библиотеки C++
	CMyString(std::string const& stlString);

	// деструктор класса - освобождает память, занимаемую символами строки
	~CMyString();

	// возвращает длину строки (без учета завершающего нулевого символа)
	unsigned GetLength()const;

	// возвращает указатель на массив символов строки.
	// В конце массива обязательно должен быть завершающий нулевой символ
	// даже если строка пустая
	const char* GetStringData()const;

	// возвращает подстроку с заданной позиции длиной не больше length символов
```

[13] **Подсказка**: реализация класса строк должна помимо адреса первого элемента массива символов в динамической памяти хранить еще и длину строки, т.к. использование функций вроде strlen, strcpy и им подобных, воспринимающих символ с нулевым кодом как символ конца строки, не решает данную проблему.
Кроме того в массиве потребуется зарезервировать место под символ с нулевым кодом в конце строки, т.к. метод GetStringData(), объявленный в классе CMyString, согласно условиям задачи возвращает указатель на строку с завершающим нулевым символом.

```
        CMyString const SubString(unsigned start, unsigned length = UINT_MAX)const;

        // очистка строки (строка становится снова нулевой длины)
        void Clear();
};
```

Для хранения символов строки **не допускается** использовать классы вроде **std::string** и **std::vector**. Управление данными в динамической памяти должно быть реализовано целиком силами Вашего класса.

Внимание:

Реализуйте конструктор, деструктор и перечисленные в каркасе методы класса, а также следующие операторы:

№	Оператор	Описание	Обязательно
1	=	Присваивание CMyString (присваивание других типов, принимаемые конструктором класса будут реализованы автоматически) Корректно должна обрабатываться ситуации с самоприсваиванием, вроде: CMyString s(“SomeString”); s = s;	Да
2	+	Реализуйте следующие версии оператора конкатенации: • CMyString с CMyString • std::string с CMyString • const char* с CMyString	Да
3	+=	Конкатенация CMyString с CMyString с присваиванием	Да
4	==	Посимвольное сравнение содержимого двух строк	Да
5	!=	Проверка двух строк на неравенство	Да
6	<	Лексикографическое сравнение содержимого двух строк. Осуществляет проверку того, предшествует ли строка слева от знака «<» строке, находящейся справа, если сравнивать их содержимое в алфавитном порядке.	Нет
7	>	Лексикографическое сравнение содержимого двух строк. Аналогично оператору <	Нет
8	<= и >=	Лексикографическое сравнение содержимого двух строк. Аналогично оператор < и >	Нет
9	[]	Реализуйте две версии данного оператора: • Индексированный доступ к символам строки по целочисленному индексу для чтения • Индексированный доступ к символам строки по целочисленному индексу для записи	Да
10	<<	Оператор вывода в выходной поток	Нет
11	>>	Оператор ввода из входного потока	Нет
12	Перемещающий конструктор и оператор присваивания	Только для компиляторов, совместимых с C++11. Реализуется совместно с перемещающим конструктором.	Нет

За реализацию каждой группы операторов – 25 баллов

Бонус до 200 баллов за реализацию STL-совместимых итераторов,

Реализовать поддержку итераторов в STL-совместимой манере, позволяющих перебирать символы строки, использоваться совместно с основными алгоритмами стандартной библиотеки.

№	Функционал	Балл	Обязательно

1	Итерация по константным[14] и неконстантным строкам в прямом направлении и обратном направлении. Получение итератора, указывающего на начальный символ и на позицию, следующую за конечным символом строки (аналоги методов begin()/end() класса std::string) Разыменование итератора Нахождение разницы между двумя итераторами, сложение итератора с числом и числа с итератором	100	Да
2	Индексированный[15] доступ к элементам строки относительно итератора при помощи оператора []	20	Нет
3	Поддержка итерации по символам строки в обратном направлении (аналогично итерации при помощи методов std::string::rbegin(), std::string::rend()).	30	Нет
4	Проверка границ (при помощи assert) в отладочной конфигурации.	30	Нет
5	Поддержка итерации по элементам при помощи range-based версии оператора for.	20	Нет

В процессе разработки классов использование подхода TDD обязательно. Без автоматических тестов работа будет принята с коэффициентом 0.5.

[14] Для константных строк должен возвращаться итератор, предоставляющий доступ к содержимому строки только для чтения.

[15] Для константных строк доступ должен предоставляться только для чтения символов строки