Фонд оценочных средств для промежуточной аттестации

по дисциплине

Программирование на C#

Направление подготовки
02.03.02 Фундаментальная информатика и информационные технологии

Направленность (профиль)
Компьютерные науки

Присваиваемая квалификация
бакалавр

Форма обучения
очная

Миасс 2026 г.

02.03.02 Фундаментальная информатика и информационные технологии, Компьютерные науки, Программирование на C#, 2026, очная

Фонд оценочных средств одобрен и рекомендован:
Проректор по учебной работе утверждено 27.02.26 А.А. Саламатов

Ученым советом Миасского филиала ФГБОУ ВО "ЧелГУ"

Протокол заседания № 8 от 24.02.2026

Председатель Ученого совета Миасского филиала ФГБОУ ВО "ЧелГУ" согласовано Т.В. Малькова

Заседанием кафедры прикладной математики

Протокол заседания № 6 от 30.01.2026

Заведующий кафедрой согласовано Е.В. Дутикова

Автор (составитель) К.А. Лихачев

Структура фонда оценочных средств соответствует приказу ректора ФГБОУ ВО «ЧелГУ» от «13» апреля 2021 г. № 247-1

1. ПАСПОРТ ФОНДА ОЦЕНОЧНЫХ СРЕДСТВ

Направление подготовки 02.03.02 «Фундаментальная информатика и информационные технологии»
Направленность (профиль): Компьютерные науки.
Дисциплина: Программирование на C#
Семестр (семестры) изучения: 6
Форма промежуточной аттестации: зачет
Система оценивания: оценивание результатов осуществляется по системе зачтено / не зачтено.

2. ПЕРЕЧЕНЬ ФОРМИРУЕМЫХ КОМПЕТЕНЦИЙ

2.1. Компетенции, закреплённые за дисциплиной

Изучение дисциплины направлено на формирование следующих компетенций:

Коды компетенции (по ФГОС)	Содержание компетенций согласно ФГОС	Индикаторы достижений	Перечень планируемых результатов обучения по дисциплине
1	2	3	4
ПК-2	Способен к разработке программного обеспечения, осуществлению интеграции программных модулей и компонент и проверке работоспособности программного обеспечения на основе международных и профессиональных стандартов в области информационных технологий	ПК-2.1 Демонстрирует знание основных принципов и технологий разработки программного обеспечения, методов и средств сборки модулей и компонент программного обеспечения; разработки процедур для развертывания программного обеспечения, методов и средств миграции и преобразования данных, методов создания пользовательских интерфейсов; средств программирования ПК-2.2 Демонстрирует умения разрабатывать программный код на языках программирования высокого и низкого уровня, осуществлять отладку программ, оформлять техническую доку-	Знать: классы, объекты и модули в языке C#; компоненты стандартных библиотек языка C#; основные понятия, связанные с системами программирования; состав и схемы работы систем объектно-ориентированного программирования; Уметь: применять на практике стандартные средства языка C# и концепции объектно-ориентированного программирования для разработки программного обеспечения Владеть: навыками написания и от-

		ментацию; использовать выбранную среду программирования для разработки процедур интеграции программных модулей, проводить оценку работоспособности программного обеспечения ПК-2.3 Имеет практический опыт разработки исходного кода, тестирования программного обеспечения, сборки модулей и компонент программного обеспечения, разработки процедур для развертывания программного обеспечения, миграции и преобразования данных, создания программных интерфейсов	ладки кода на языке C#; навыками использования средств объектно-ориентированного программирования в разработке приложений

3. СОДЕРЖАНИЕ ОЦЕНОЧНЫХ СРЕДСТВ ПО ДИСЦИПЛИНЕ

3.1 Виды оценочных средств

№ п/п	Контролируемые темы/разделы	Код компетенции/ планируемые результаты обучения	Наименование оценочного средства для текущего контроля	Наименование оценочного средства на промежуточной аттестации
1.	Введение. Базовые функции языка	ПК-2 Знает Понятие абстрактных данных, указателей, статических и динамических массивов, функций, процедур, модулей. Умеет Создавать функции, процедуры и модули. Применять их в решении задач Владеет Навыками применения итераторов и функторов, использования файлов для ввода-вывода данных	Лабораторная работа	Практическая работа Тест
2.	Сортировки	ПК-2 Знает Виды сортировок линейной сложности, сложности	Лабораторная работа	Практическая работа Тест

		$n \cdot \log n$ и n^2. Умеет Реализовывать основные алгоритмы сортировок Владеет навыками применения сортировок при разработке программ.		
3.	Динамические структуры данных	ПК-2 Знает Принципы выделения и освобождения памяти. Принципы работы деревьев, односвязных и двусвязных списков, ассоциативных массивов. Умеет Реализовывать основные структуры данных Владеет Навыками использования динамических структур данных при разработке приложений	Лабораторная работа	Практическая работа Тест
4.	Объектно-ориентированное программирование на языке C#	ПК-2 Знает Основные понятия объектно-ориентированного программирования. Умеет Создавать пользовательские типы данных, разделяя их интерфейс и реализацию. Применять принципы наследования для избегания дублирования кода Владеет Понятиями виртуальных методов класса, таблицы виртуальных функций. Навыками применения виртуальных функций для создания абстрактных классов.	Лабораторная работа	Практическая работа Тест
5.	Шаблоны	ПК-2 Знает Понятия шаблона, шаблонной функции, обобщенного программирования. Умеет	Лабораторная работа	Практическая работа Тест

		Реализовывать шаблонные функции и шаблонные пользовательские типы данных. Владеет Принципами обобщенного программирования, умеет использовать шаблоны при разработке пользователь-ских приложений		

Типовые задания, критерии и показатели оценивания в рамках текущего контроля представлены в рабочей программе по дисциплине. Полные комплекты оценочных средств и контрольно-измерительных материалов хранятся на кафедре и являются учебно-методическими материалами ограниченного (конфиденциального) пользования.

3.2 Содержание оценочных средств для текущей аттестации

3.2.1 Лабораторные работы

Лабораторная работа №1

Реализация UDT длинное целое.

Цель работы: реализовать класс LongNumbers, который реализует логику работы с длинными целыми числами.

Задачи:

1) Создать класс LongNumbers, реализующий представленный ниже интерфейс.
2) Создать перегрузки операторов +, -, *, /, % для класса LongNumbers, которые давали бы возможность работать с данным типом также, как и с типами int, double.

```
namespace LN {
    typedef vector<unsigned char> v_char;

    /*
        Класс для хранения длинных чисел.
        Хранит числа в виде вектора, каждая ячейка
которого - это одна цифра этого числа.
```

```
    */
    class LongNumbers
    {
        vector<unsigned char> data;
        char sign = '+';

        v_char addVectors(const v_char& left, const
v_char& right) const;
        v_char subVectors(const v_char& left, const
v_char& right) const;
        v_char multVectors(const v_char& left, const
v_char& right) const;
        v_char divVectors(const v_char& left, const
v_char& right) const;
        void normaliseVector(v_char& v) const;

    public:
        struct BadSymbols { string message = "Bad
symbols in string"; };
        LongNumbers() : data(v_char(1, 0)) {}
        LongNumbers(long long int a);
        LongNumbers(string s);
        LongNumbers(char sign, v_char d);
        LongNumbers(const LongNumbers& a);
        LongNumbers add(const LongNumbers& r) const;
        LongNumbers sub(const LongNumbers& r) const;
        LongNumbers mult(const LongNumbers& r) const;
        LongNumbers div(const LongNumbers& r) const;
        LongNumbers res(const LongNumbers& r) const;
        operator string() const;
        char getSign() const { return sign; }
        v_char getElements() const { return data; }
        bool moreThan(const LongNumbers& right) const;
        bool equalTo(const LongNumbers& right) const;
        bool lessThan(const LongNumbers& right) const;
    };

    ostream& operator<<(ostream& os, const
```

```
LongNumbers& a);
    bool isEqual(const v_char& l, const v_char& r);
    bool leftIsMore(const v_char& l, const v_char& r);
    bool rightIsMore(const v_char& l, const v_char& r);

    LongNumbers operator + (const LongNumbers& l, const
LongNumbers& r);
    void operator += (LongNumbers& l, const
LongNumbers& r);

    LongNumbers operator - (const LongNumbers& l, const
LongNumbers& r);
    void operator -= (LongNumbers& l, const
LongNumbers& r);

    LongNumbers operator * (const LongNumbers& l, const
LongNumbers& r);
    void operator *= (LongNumbers& l, const
LongNumbers& r);

    LongNumbers operator / (const LongNumbers& l, const
LongNumbers& r);
    void operator /= (LongNumbers& l, const
LongNumbers& r);

    LongNumbers operator % (const LongNumbers& l, const
LongNumbers& r);
    void operator %= (LongNumbers& l, const
LongNumbers& r);

    bool operator < (const LongNumbers& l, const
LongNumbers& r);
    bool operator <= (const LongNumbers& l, const
LongNumbers& r);
    bool operator == (const LongNumbers& l, const
LongNumbers& r);
    bool operator > (const LongNumbers& l, const
LongNumbers& r);
```

```
    bool operator >= (const LongNumbers& l, const
LongNumbers& r);
}
```

Рассмотрим более детально методы класса:

```
v_char addVectors(const v_char& left, const v_char&
right) const;
v_char subVectors(const v_char& left, const v_char&
right) const;
v_char multVectors(const v_char& left, const v_char&
right) const;
v_char divVectors(const v_char& left, const v_char&
right) const;
```

Данные методы выполняют сложение, вычитание, умножение и целочисленное деление левого вектора left на правый вектор right. Данные методы являются основой для выполнения сложения, вычитания, умножения и деления длинных чисел.

```
void normaliseVector(v_char& v) const;
```

Метод выполняет нормализацию вектора. В теории, при выполнении представленных выше операций над векторами или при создании объекта длинного числа в векторе могут содержаться незначащие нули. Данный метод должен удалять из вектора незначащие нули. Если при этом вектор целиком состоит из нулей, один нуль должен оставаться.

```
struct BadSymbols { string message = "Bad symbols in
string"; };
```

Структура для генерации исключения при неправильном символе в строке, на основании которой создаётся длинное число. То есть если в строке символов или в векторе содержатся значения, которые выходят за отрезок $[0;10]$, должно генерироваться исключение.

```
LongNumbers() : data(v_char(1, 0)) {}
```

Конструктор по умолчанию. При создании объекта LongNumbers без каких-либо аргументов будет вызываться данный конструктор. По умолчанию число равно нулю.

```
LongNumbers(long long int a);
```
Создание объекта LongNumbers на основании типа long long int. Данный конструктор должен разбивать число на цифры, раскладывать их в вектор data, знак числа определяется на основании знака переменной a.

```
LongNumbers(string s);
```
Создание объекта LongNumbers на основании строки. Предусмотреть, что в начале строки может быть знак + или -. Если в начале строки знака нет, то число считается положительным. Число нуль будем считать положительным. Предусмотрите сбрасывание исключения, если в строке содержатся символы, выходящие за пределы отрезка $[0;10]$.

```
LongNumbers(char sign, v_char d);
```
Создание объекта LongNumbers на основании знака sign, то есть символа ‘+’ или ‘-‘, а также вектора данных d. Предусмотрите сбрасывание исключения, если символ Предусмотрите сбрасывание исключения, если символ sign принимает значение отличное от ‘+’ или ‘-‘, а также, если вектор d содержит символы, выходящие за пределы отрезка $[0;10]$.

```
LongNumbers(const LongNumbers& a);
```
Создание объекта LongNumbers на основании другого объекта LongNumbers. То есть данный конструктор представляет собой конструктор копирования. В данном случае выполняется копирование данных из объекта a в текущий объект.

```
LongNumbers add(const LongNumbers& r) const;
```
Выполняет сложение двух чисел. Возвращает новое число, не изменяет текущее число и число r.

```
LongNumbers sub(const LongNumbers& r) const;
```
Выполняет вычитание двух чисел. Возвращает новое число, не изменяет текущее число и число r.

```
LongNumbers mult(const LongNumbers& r) const;
```
Выполняет умножение двух чисел. Возвращает новое число, не изменяет текущее число и число r.

```
LongNumbers div(const LongNumbers& r) const;
```
Выполняет целочисленное деление двух чисел. Возвращает новое число, не изменяет текущее число и число r.

```
LongNumbers res(const LongNumbers& r) const;
```
Выполняет остаток от деления двух чисел. Возвращает новое число, не изменяет текущее число и число r.

```
operator string() const;
```
Оператор выполняет приведение типа LongNumbers в тип string. Если определен данный оператор, то будет возможно осуществлять вызовы типа string(a), где a – это переменная типа LongNumbers.

```
char getSign() const { return sign; }
```
Возвращает знак числа.

```
v_char getElements() const { return data; }
```
возвращает вектор, в котором хранятся цифры числа.

```
bool moreThan(const LongNumbers& right) const;
```
Возвращает true, если текущее число больше, чем число right.

```
bool equalTo(const LongNumbers& right) const;
```
Возвращает true, если текущее число равно числу right.

```
bool lessThan(const LongNumbers& right) const;
```
Возвращает true, если текущее число меньше числа right.

```
ostream& operator<<(ostream& os, const LongNumbers&
a);
```
Оператор реализует логику вывода числа a в поток вывода os. Позволяет, например, выполнять операции вида cout << a << endl; Где a – это число типа LongNumbers.

```
bool isEqual(const v_char& l, const v_char& r);
```
Вспомогательная функция, которая сравнивает два вектора. Если их длины и значения элементов равны, возвращается true, иначе – false.

```
bool leftIsMore(const v_char& l, const v_char& r);
```
Вспомогательная функция. Функция возвращает true, если число, представленное вектором l больше, чем число, представленное вектором r. Знак чисел при сравнении не учитывается. Можно считать, что эта функция выполняет сравнение двух длинных чисел по модулю.

```
bool rightIsMore(const v_char& l, const v_char& r);
```
Вспомогательная функция. Возвращает true, если число, представленное вектором r больше, чем число, представленное вектором l, иначе возвращает false. Эту функцию удобно реализовать с помощью функций leftIsMore и isEqual.

```
LongNumbers operator + (const LongNumbers& l, const
LongNumbers& r);
void operator += (LongNumbers& l, const LongNumbers&
r);

LongNumbers operator - (const LongNumbers& l, const
LongNumbers& r);
void operator -= (LongNumbers& l, const LongNumbers&
r);

LongNumbers operator * (const LongNumbers& l, const
LongNumbers& r);
void operator *= (LongNumbers& l, const LongNumbers&
r);

LongNumbers operator / (const LongNumbers& l, const
LongNumbers& r);
void operator /= (LongNumbers& l, const LongNumbers&
r);

LongNumbers operator % (const LongNumbers& l, const
LongNumbers& r);
void operator %= (LongNumbers& l, const LongNumbers&
r);
```

```
bool operator < (const LongNumbers& l, const
LongNumbers& r);
bool operator <= (const LongNumbers& l, const
LongNumbers& r);
bool operator == (const LongNumbers& l, const
LongNumbers& r);
bool operator > (const LongNumbers& l, const
LongNumbers& r);
bool operator >= (const LongNumbers& l, const
LongNumbers& r);
```

Представленные выше перегрузки операторов предназначены для того, чтобы классом LongNumbers можно было бы оперировать так же, как обычными числами int или double. Способ использования этих типов после определения представленных выше операторов будет совпадать. Реализовывать эти операторы лучше в последнюю очередь, потому что для их реализации вам понадобятся указанные выше методы класса.

Тестовые данные:

A + B		Результат
A	B	
-100000245001236	-11510130001000091	-11610130246001327
-100000245001236	11510130001000091	11410129755998855
1111100000245001236	11510130001000091	1122610130246001327

A * B		Результат
A	B	
2589614578	28745131513210	744388116131358155753 80
-12456	115113154861	-1433849456948616
-1245001236	-1151131000091	1433159517911211112476

A / B		Результат
A	B	
-2030102	-200000	10
-2000030102	200000	-10000

1541	151	10
203141	151	1345

Лабораторная работа №2

Реализация UDT рациональное число.

Цель работы: реализовать класс Rational, который позволяет выполнять вычисления при помощи рациональных чисел.

Задачи:

1) Создать класс Rational, который реализует представленный ниже интерфейс.
2) Реализовать правила грамматики для вычисления арифметических выражений, содержащих основные знаки операций: +, -, *, /, возведение в степень, при помощи класса Rational.

```
class Rational
{
    int numerator; // Числитель дроби
    int denominator; // Знаменатель дроби

public:
    struct ZeroDenominator { string message =
"Denominator cannot be equal to zero"; };
    Rational(int num, int den);
    Rational(int num) : numerator(num), denominator(1)
{}
    Rational() : numerator(0), denominator(1) {}
    Rational add(const Rational& right) const;
    Rational sub(const Rational& right) const;
    Rational mult(const Rational& right) const;
    Rational div(const Rational& right) const;
    bool equalTo(const Rational& right) const;
    bool moreThan(const Rational& right) const;
    bool lessThan(const Rational& right) const;
    operator string() const;
    operator double() const;
};
```

```
istream& operator >> (istream&, Rational&);
ostream& operator << (ostream&, const Rational&);

Rational operator + (const Rational& left, const
Rational& right);
void operator += (Rational& left, const Rational&
right);

Rational operator - (const Rational& left, const
Rational& right);
void operator -= (Rational& left, const Rational&
right);

Rational operator * (const Rational& left, const
Rational& right);
void operator *= (Rational& left, const Rational&
right);

Rational operator / (const Rational& left, const
Rational& right);
void operator /= (Rational& left, const Rational&
right);

bool operator == (const Rational& left, const Rational&
right);
bool operator > (const Rational& left, const Rational&
right);
bool operator < (const Rational& left, const Rational&
right);
bool operator >= (const Rational& left, const Rational&
right);
bool operator <= (const Rational& left, const Rational&
right);
```

Рациональное число – это число вида $\frac{m}{n}$, где m – это целое число, а n – натуральное. Следует учитывать это при реализации класса. Например, при вызо-

ве конструктора с аргументами 2 и -5 следует проконтролировать, чтобы дробь была представлена в виде $\frac{-2}{5}$, а не $\frac{2}{-5}$.

Рассмотрим более подробно структуру класса:

1) `struct ZeroDenominator` – структура, которая используется для перехвата исключений при нулевом знаменателе.
2) `Rational(int num, int den);` - конструктор класса, принимающий значения числителя и знаменателя дроби соответственно.
3) `Rational(int num)` – конструктор дроби, который принимает значение числителя. Знаменатель принимается равный единице.
4) `Rational()` – конструктор по умолчанию. Будем считать, что по умолчанию в дроби будет число вида $\frac{0}{1}$.
5) Методы add, sub, mult, div предназначены для сложения, вычитания, умножения и деления дробей. Обратите внимание на модификатор const. Данные методы являются чистыми, то есть они не изменяют своих аргументов. При их вызове не изменяются аргументы методов и объект, для которого они вызываются. В ходе вычислений должен создаваться новый объект типа Rational и возвращаться в качестве результата работы методов.
6) Методы equalTo, moreThan, lessThan предназначены для реализации сравнения чисел следующим образом:

$a.equalTo(b) \leftrightarrow a == b$
$a.moreThan(b) \leftrightarrow a > b$
$a.lessThan(b) \leftrightarrow a < b$

Данные методы также являются чистыми. Число a не изменяется при вызове этих методов.

7) `operator string() const;` - назначение оператора – конвертация объектов типа Rational в объекты типа std::string.

Также для класса нужно реализовать ряд операторов для более удобной работы с ним:

1) `istream& operator >> (istream&, const Rational&);` и `ostream& operator << (ostream&, const Rational&);` предназначены для работы с потоками ввода и вывода. Необходимо реализовать возможность считывания данных из потока ввода в виде простого числа, так, например, при считывании числа 5 следует превратить

его в дробь вида $\frac{5}{1}$, а при считывании выражения 5/3 нужно создать дробь $\frac{5}{3}$. Используйте соответствующие конструкторы. Вывод дроби происходит аналогичным образом: если знаменатель дроби равен 1, то выводится только числитель, если знаменатель не равен 1, то выводится дробь в формате числитель/знаменатель.

2) Операторы +, -, *, / предназначены для удобства использования класса. Они представляют собой оболочку над методами класса add, sub, mult и div. Используйте эти методы для реализации перегрузки этих операторов.
3) Операторы +=, -=, *=, /= выполняют те же функции, что операторы в предыдущем пункте. Обратите внимание, что представленные в предыдущем пункте операторы являются чистыми, то есть при вызове конструкции a + b объекты a и b не изменяют своего внутреннего состояния. Однако оператор += и другие не будут являться чистыми. При вызове a += b должно быть изменено состояние объекта a, поэтому в данном случае объект Rational& left передаётся не по константной ссылке.
4) Операторы сравнения ==, >, <, >=, <= выполняют сравнение двух объектов типа Rational. Для реализации этих операторов используйте функции equalTo, moreThan и lessThan.

Лабораторная работа №3

Абстрактные типы данных: стек, очередь.

Цель работы: изучить принципы работы абстрактных типов данных стек и очередь, список, научиться применять эти типы данных в решении задач.

Задание 1. Простой стек

Реализуйте структуру данных "стек", реализовав все указанные здесь методы. Напишите программу (функцию main), содержащую описание стека и моделирующую работу стека. Функция main считывает последовательность команд и в зависимости от команды выполняет ту или иную операцию. После выполнения одной команды программа должна вывести одну строчку. Возможные команды для программы:

push n

Добавить в стек число n (значение n задается после команды). Программа

должна вывести ok.

pop

Удалить из стека последний элемент. Программа должна вывести его значение.

back

Программа должна вывести значение последнего элемента, не удаляя его из стека.

size

Программа должна вывести количество элементов в стеке.

clear

Программа должна очистить стек и вывести ok.

exit

Программа должна вывести bye и завершить работу.

Гарантируется, что набор входных команд удовлетворяет следующим требованиям: максимальное количество элементов в стеке в любой момент не превосходит 100, все команды pop_back и back корректны, то есть при их исполнении в стеке содержится хотя бы один элемент.

Пример протокола работы программы

Ввод	**Вывод**
push 2	ok
push 3	ok
push 5	ok
back	5
size	3
pop	5
size	2
push 7	ok
pop	7
clear	ok
size	0
exit	bye

Задание 2. Стек с обработкой ошибок

Аналогично предыдущему заданию, только снимается ограничение на корректность вызовов методов back и pop. Данные операции должны перед исполнением проверять, содержится ли в стеке хотя бы один элемент. Если во входных данных встречается операция back или pop, при этом стек пуст, то программа должна вместо числового значения вывести строку error.

При этом должна быть реализована двойная защита: вызов методов forward и pop для пустого стека не должен приводить к обращению к несуществующим элементам массива m_elems, а функция main должна выводить сообщение error, при считывании некорректной операции.
Пример протокола работы программы

Ввод	**Вывод**
push 2	ok
back	2
pop	2
size	0
pop	error
push 1	ok
size	1
exit	bye

Задание 3. Стек без ограничения на размер

Реализуйте стек динамического размера, то есть ограниченный только объемом свободной оперативной памяти. Для этого используйте указатели и динамически распределяемую память. Если для полностью заполненного стека вызывается метод push размер динамического массива, отведенного для хранения стека, должен увеличиваться.

Задание 4. В файле хранится строка в постфиксной записи, содержащая арифметическое выражение из целых чисел, а также знаков +, -, *, /, % без скобок. Вычислить значение выражения. Операции / и % определяют целую часть и остаток от целочисленного деления одного числа на другое.

Задание 5. Правильная скобочная последовательность

Входной файл содержит несколько строк, каждая из которых содержит последовательность символов (,), [и]. Выясните, является ли она правильной скобочной последовательность с двумя типами скобок.

input.txt	output.txt
()() ([]) ([)]	YES YES NO

((]])(	NO NO

Задание 6. Очередь на списке

В этой задаче обязательно использовать самостоятельно связный список.
Реализуйте работу очереди. Для каждой операции изъятия элемента выведите её результат.
На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда – это либо + N, либо -. Команда + N означает добавление в очередь числа N, по модулю не превышающего 10^9. Команда - означает изъятие элемента из очереди.

Формат входного файла.

В первой строке входного файла содержится количество команд – M ($1 \le M \le 10^6$). Каждая последующая строка исходного файла содержит ровно одну команду.

Форма выходного файла.

Выведите числа, которые удаляются из очереди, по одному в каждой строке. Гарантируется, что изъятий из пустой очереди не производится.

input.txt	**output.txt**
4 + 1 + 10 - -	1 10

Задание 7. Очередь с минимумом

В этой задаче обязательно использовать самостоятельно написанный вектор.
Реализуйте работу очереди. Необходимо отвечать на запрос о минимальном элементе, который сейчас находится в очереди. Для каждой операции изъятия элемента или запроса минимального элемента выведите её результат.
На вход программе подаются строки, содержащие команды. Каждая строка содержит одну команду. Команда – это либо + N, либо -, либо ?. Первая команда добавляет в очередь новый элемент. Вторая – убирает из очереди элемент. Третья – означает поиск минимального элемента в очереди.

Формат входного файла.

В первой строке содержится количество команд – M ($1 \le M \le 500000$). В последующих строках содержатся команды, по одной в каждой строке.

Формат выходного файла

Для каждой операции поиска минимума в очереди выведите её результат. Гарантируется, что операции извлечения или поиска минимума для пустой очереди не производятся.

input.txt	output.txt
7 + 1 ? + 10 ? - ? -	1 1 10

Лабораторная работа №4

Сортировки

Цель работы: изучить принципы алгоритмы работы основных сортировок сложности $O(n^2)$, $O(n \cdot \log(n))$, $O(n)$: сортировка вставками, сортировка слиянием, быстрая сортировка, цифровая сортировка, карманная сортировка.

Сортировка вставками

Простейший алгоритм сортировки сложности n^2. Идея крайне простая.
Пусть нумерация элементов начинается с 0 до n - 1. Находимся в позиции i. Элементы в позиции от 0 до i - 1 уже отсортированы. j изменяет значение от i - 1 до 0. На каждом шаге смотрим элементы j и j + 1. Если предыдущий больше, чем последующий, то меняем их местами. В результате получим отсортированный массив.

Пример реализации

```
for (int i = 1; i < n; i++) {
        for (int j = i - 1; j >= 0 && arr[j] > arr[j + 1]; j--) {
                swap(arr[j], arr[j+1]);
        }
}
```

Или по-другому

```
        for (int i = 1; i < n; i++) {
            int j = i - 1;
            while (j >=0 && arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j+1]);
                j--;
            }
        }
```

Задание 8. Сортировка вставками

Дан массив целых чисел. Ваша задача — отсортировать его в порядке неубывания с помощью сортировки вставками.

Сортировка вставками проходится по всем элементам массива от меньших индексов к большим («слева направо») для каждого элемента определяет его место в предшествующей ему отсортированной части массива и переносит его на это место (возможно, сдвигая некоторые элементы на один индекс вправо). Чтобы проконтролировать, что Вы используете именно сортировку вставками, мы попросим Вас для каждого элемента массива, после того, как он будет обработан, выводить его новый индекс.

Формат входного файла

В первой строке входного файла содержится число n ($1 \le n \le 1000$) — число элементов в массиве. Во второй строке находятся различных целых чисел, по модулю не превосходящих 10^9.

Формат выходного файла

В первой строке выходного файла выведите чисел. При этом i-ое число равно индексу, на который, в момент обработки его сортировкой вставками, был перемещен i-ый элемент исходного массива. Индексы нумеруются, начиная с единицы. Между любыми двумя числами должен стоять ровно один пробел.

Во второй строке выходного файла выведите отсортированный массив. Между любыми двумя числами должен стоять ровно один пробел.

Пример

input.txt	output.txt
10 1 8 4 2 3 7 5 6 9 0	1 2 2 2 3 5 5 6 9 1 0 1 2 3 4 5 6 7 8 9

Комментарий к примеру

В примере сортировка вставками работает следующим образом:

1. Первый элемент остается на своем месте, поэтому первое число в ответе — единица. Отсортированная часть массива: [1]

2. Второй элемент больше первого, поэтому он тоже остается на своем месте, и второе число в ответе — двойка. [1 8]
3. Четверка меньше восьмерки, поэтому занимает второе место. [1 4 8]
4. Двойка занимает второе место. [1 2 4 8]
5. Тройка занимает третье место. [1 2 3 4 8]
6. Семерка занимает пятое место. [1 2 3 4 7 8]
7. Пятерка занимает пятое место. [1 2 3 4 5 7 8]
8. Шестерка занимает шестое место. [1 2 3 4 5 6 7 8]
9. Девятка занимает девятое место. [1 2 3 4 5 6 7 8 9]
10. Ноль занимает первое место. [0 1 2 3 4 5 6 7 8 9]

Задание 9. Знакомство с жителями Сортлэнда

Владелец графства Сортлэнд, граф Бабблсортер, решил познакомиться со своими подданными. Число жителей в графстве нечетно и составляет *n*, где *n* может быть достаточно велико, поэтому граф решил ограничиться знакомством с тремя представителями народонаселения: с самым бедным жителем, с жителем, обладающим средним достатком, и с самым богатым жителем.

Согласно традициям Сортлэнда, считается, что житель обладает средним достатком, если при сортировке жителей по сумме денежных сбережений он оказывается ровно посередине. Известно, что каждый житель графства имеет уникальный идентификационный номер, значение которого расположено в границах от единицы до *n*. Информация о размере денежных накоплений жителей хранится в массиве *M* таким образом, что сумма денежных накоплений жителя, обладающего идентификационным номером *i*, содержится в ячейке *M*[*i*]. Помогите секретарю графа мистеру Свопу вычислить идентификационные номера жителей, которые будут приглашены на встречу с графом.

Формат входного файла

Первая строка входного файла содержит число жителей *n* ($3 \le n \le 9999$, n - нечетно). Вторая строка содержит описание массива *M*, состоящее из *n* положительных вещественных чисел, разделенных пробелами. Гарантируется, что все элементы массива *M* различны, а их значения имеют точность не более двух знаков после запятой и не превышают 10^6.

Формат выходного файла

В выходной файл выведите три целых положительных числа, разделенных пробелами — идентификационные номера беднейшего, среднего и самого богатого жителей Сортлэнда.

Пример

input.txt	**output.txt**
5 10.00 8.70 0.01 5.00 3.00	3 4 1

Комментарий к примеру
Если отсортировать жителей по их достатку, получится следующий массив:
[0.01, 3] [3.00, 5] [5.00, 4] [8.70, 2] [10.00, 1]
Здесь каждый житель указан в квадратных скобках, первое число — его достаток, второе число — его идентификационный номер. Таким образом, самый бедный житель имеет номер 3, самый богатый — номер 1, а средний — номер 4.

Задание 10. Секретарь Своп

Уже знакомый нам из предыдущей задачи граф Бабблсортер поручил своему секретарю, мистеру Свопу, оформлять приглашения беднейшему, богатейшему и среднему по достатку жителю своих владений. Однако кто же, в отсутствие мистера Свопа, будет заниматься самым важным делом — сортировкой массивов чисел? Видимо, это придется сделать Вам!
Дан массив, состоящий из n целых чисел. Вам необходимо его отсортировать по неубыванию. Но делать это нужно так же, как это делает мистер Своп — то есть, каждое действие должно быть взаимной перестановкой пары элементов. Вам также придется записать все, что Вы делали, в файл, чтобы мистер Своп смог проверить Вашу работу.
Формат входного файла
В первой строке входного файла содержится число n ($1 \le n \le 5000$) — число элементов в массиве. Во второй строке находятся целых чисел, по модулю не превосходящих 10^9. Числа могут совпадать друг с другом.
Формат выходного файла
В первых нескольких строках выведите осуществленные Вами операции перестановки элементов. Каждая строка должна иметь следующий формат:
*Swap elements at indices **X** and **Y**.*
где X и Y — различные индексы массива, элементы на которых нужно переставить ($1 \le X, Y \le n$). Мистер Своп любит порядок, поэтому сделайте так, чтобы $X < Y$.
После того, как все нужные перестановки выведены, выведите следующую фразу:
No more swaps needed.

Во последней строке выходного файла выведите отсортированный массив, чтобы мистер Своп не переделывал работу за Вас. Между любыми двумя числами должен стоять ровно один пробел.

Пример

input.txt	**output.txt**
5 3 1 4 2 2	Swap elements at indices 1 and 2. Swap elements at indices 2 and 4. Swap elements at indices 3 and 5. No more swaps needed. 1 2 2 3 4

Послесловие и предостережение

Семья секретаря Свопа занималась сортировками массивов, и именно с помощью перестановок пар элементов, как минимум с XII века, поэтому все Своппы владеют этим искусством в совершенстве. Мы не просим Вас произвести минимальную последовательность перестановок, приводящую к правильному ответу. Однако учтите, что для вывода слишком длинной последовательности у Вашего алгоритма может не хватить времени (или памяти — если выводимые строки хранятся в памяти перед выводом). Подумайте, что с этим можно сделать. Решение существует!

Сортировка слиянием или сортировка фон Неймана

Сортировка работает по следующему принципу: пусть даны два отсортированных массива. Эта сортировка сливает два массива в один. Например: *A*: [4, 6, 6] и *B* [1, 4, 5, 7]. После слияния будет образован новый массив *C* [1, 4, 4, 5, 6, 6, 7]. Код, который реализует такое слияние, может быть записан следующим образом:

```
vector<int> merge(vector<int>& a, vector<int>& b) {
        int i = 0, j = 0;
        int n = a.size(), m = b.size();
        vector<int> c;

        while (i < n || j < m) {
                if (j == m || (i < n && a[i] <= b[j])) {
                        c.push_back(a[i]);
                        i++;
                } else {
                        c.push_back(b[j]);
                        j++;
```

```
                }
        }

        return c;
}
```

Условие `j == m` проверяет дошел ли код до конца массива `B` или нет, если это так, то первая часть условия будет истинной, а вторая часть в этом случае не будет проверяться, и будут браться элементы из массива `A`, который еще не пуст. Если первая часть условия ложна, то будет проверяться вторая часть. `i < n` проверяет, что мы еще не дошли до конца массива `A`. И проверка `a[i] <= b[j]` выясняет в каком из массивов сейчас лежит меньший элемент, который должен быть помещен в массив `C`.

Сама процедура сортировки будет выглядеть следующим образом:

```
vector<int> merge_sort(vector <int>& a) {
        int n = a.size();
        if (n == 1) {
                return a;
        }

        vector<int> left, right;
        copy(a.begin(), a.begin() + n / 2, back_inserter(left));
        copy(a.begin() + n / 2, a.end(), back_inserter(right));

        left = merge_sort(left);
        right = merge_sort(right);

        return merge(left, right);
}
```

И окончательно полный код:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>

using namespace std;

vector<int> merge(vector<int>& a, vector<int>& b) {
        int i = 0, j = 0;
```

```
        int n = a.size(), m = b.size();
        vector<int> c;

        while (i < n || j < m) {
                if (j == m || (i < n && a[i] <= b[j])) {
                        c.push_back(a[i]);
                        i++;
                } else {
                        c.push_back(b[j]);
                        j++;
                }
        }

        return c;
}

vector<int> merge_sort(vector <int>& a) {
        int n = a.size();
        if (n == 1) {
                return a;
        }

        vector<int> left, right;
        copy(a.begin(), a.begin() + n / 2, back_inserter(left));
        copy(a.begin() + n / 2, a.end(), back_inserter(right));

        left = merge_sort(left);
        right = merge_sort(right);

        return merge(left, right);
}

int main()
{
        vector<int> a = { 1, 2, 4, 0, 2, 4, 4, 5, 7 };

        vector<int> b = merge_sort(a);
```

```
        return 0;
}
```

Быстрая сортировка или сортировка Хоара

Пусть дан массив `A`. Выберем в нём какой-нибудь элемент `x` и разделим массив `A` на две части: в первой части элементы, которые меньше `x`, а во второй части будут элементы, которые больше или равны `x`. Например, для массива `A[5, 2, 6, 8, 1, 4, 7]` выберем `x = 4`. Тогда разделение будет выглядеть следующим образом: `B[2, 1]` и `C [5, 6, 8, 4, 7]`, так как эта сортировка будет использовать O(1) дополнительной памяти, значит, хранить эти части нужно в исходном массиве `A`. То есть происходит переупорядочение элементов массива `A[2, 1, 5, 6, 8, 4, 7]`.

Пример реализации такой сортировки:

```
void quick_sort(vector<int>& a, int left, int right) {
        int middle = a[(left + right) / 2];
        int i = left;
        int j = right;

        while (i < j) {
                while (a[i] < middle) {
                        i++;
                }
                while (a[j] > middle) {
                        j--;
                }

                if (i <= j) {
                        swap(a[i], a[j]);
                        i++;
                        j--;
                }
        }

        if (left < j) {
                quick_sort(a, left, j);
        }
        if (right > i) {
                quick_sort(a, i, right);
```

```
        }
}
```

Задание 11. Сортировка слиянием

Дан массив целых чисел. Ваша задача — отсортировать его в порядке неубывания с помощью сортировки слиянием.

Чтобы убедиться, что Вы действительно используете сортировку слиянием, мы просим Вас, после каждого осуществленного слияния (то есть, когда соответствующий подмассив уже отсортирован!), выводить индексы граничных элементов и их значения.

Формат входного файла

В первой строке входного файла содержится число n ($1 \le n \le 10^5$) — число элементов в массиве. Во второй строке находятся n целых чисел, по модулю не превосходящих 10^9.

Формат выходного файла

Выходной файл состоит из нескольких строк.

В **последней строке** выходного файла требуется вывести отсортированный в порядке неубывания массив, данный на входе. Между любыми двумя числами должен стоять ровно один пробел.

Все предшествующие строки описывают осуществленные слияния, по одному на каждой строке. Каждая такая строка должна содержать по четыре числа: I_f, I_l, V_f , V_l, где I_f — индекс начала области слияния, I_l — индекс конца области слияния, V_f — значение первого элемента области слияния, V_l — значение последнего элемента области слияния.

Все индексы начинаются с единицы (то есть, $1 \le I_f, I_l \le n$). **Индексы области слияния должны описывать положение области слияния в исходном массиве!** Допускается не выводить информацию о слиянии для подмассива длиной 1, так как он отсортирован по определению.

Приведем небольшой пример: отсортируем массив [9, 7, 5, 8]. Рекурсивная часть сортировки слиянием (процедура SORT(A, L, R), где A — сортируемый массив, L — индекс начала области слияния, R — индекс конца области слияния) будет вызвана с A = [9, 7, 5, 8], L = 1, R = 4 и выполнит следующие действия:

- разделит область слияния [1; 4] на две части, [1; 2] и [3; 4];
- выполнит вызов SORT(A, L = 1, R = 2):
  - разделит область слияния [1; 2] на две части, [1; 1] и [2; 2];
  - получившиеся части имеют единичный размер, рекурсивные вызовы можно не делать;

- осуществит слияние, после чего A станет равным [7; 9; 5; 8];
- выведет описание слияния: $I_f = L =$ 1, $I_l = R =$ 2, $V_f = A_L =$ 7, $V_l = A_R = 9$.

- выполнит вызов SORT(A, $L = 3$, $R = 4$):
  - разделит область слияния [3; 4] на две части, [3; 3] и [4; 4];
  - получившиеся части имеют единичный размер, рекурсивные вызовы можно не делать;
  - осуществит слияние, после чего A станет равным [7; 9; 5; 8];
  - выведет описание слияния: $I_f = L =$ 3, $I_l = R =$ 4, $V_f = A_L =$ 5, $V_l = A_R = 8$.
- осуществит слияние, после чего A станет равным [5, 7, 8, 9];
- выведет описание слияния: $I_f = L = 1$, $I_l = R = 4$, $V_f = A_L = 5$, $V_l = A_R = 9$.

Описания слияний могут идти в произвольном порядке, необязательно совпадающем с порядком их выполнения. Однако, с целью повышения производительности, рекомендуем выводить эти описания сразу, не храня их в памяти. Именно по этой причине отсортированный массив выводится в самом конце.

Корректность выданного Вами сценария сортировки слиянием проверяется специальной программой. Любая корректная сортировка слиянием, делящая подмассивы на две части (необязательно равных!), будет зачтена, если успеет завершиться, уложившись в ограничения.

Примеры

input.txt	**output.txt**
10 1 8 2 1 4 7 3 2 3 6	1 2 1 8 3 4 1 2 1 4 1 8 5 6 4 7 1 6 1 8 7 8 2 3 9 10 3 6 7 10 2 6 1 10 1 8 1 1 2 2 3 3 4 6 7 8

Задание 12. Число инверсий

Инверсией в последовательности чисел A называется такая ситуация, когда $i < j$, а $A_i > A_j$.

Дан массив целых чисел. Ваша задача — подсчитать число инверсий в нем.

Подсказка: чтобы сделать это быстрее, можно воспользоваться модификацией сортировки слиянием.

Формат входного файла

В первой строке входного файла содержится число n ($1 \le n \le 10^5$) — число элементов в массиве. Во второй строке находятся целых чисел, по модулю не превосходящих 10^9.

Формат выходного файла

В выходной файл надо вывести число инверсий в массиве.

Примеры

input.txt	output.txt
10 1 8 2 1 4 7 3 2 3 6	17

Задание 13. Анти-quick sort

Для сортировки последовательности чисел широко используется быстрая сортировка — QuickSort. Далее приведена программа, которая сортирует массив a, используя этот алгоритм.

```
var a : array [1..N] of integer;

procedure QSort(left, right : integer);
var i, j, key, buf : integer;
begin
   key := a[(left + right) div 2];
   i := left;
   j := right;
   repeat
      while a[i] < key do
         inc(i);
      while key < a[j] do
         dec(j);
      if i <= j then begin
         buf := a[i];
         a[i] := a[j];
```

```
            a[j] := buf;
            inc(i);
            dec(j);
        end;
    until i > j;
    if left < j then QSort(left, j);
    if i < right then QSort(i, right);
end;
begin
    ...
    QSort(1, N);
end.
```

Хотя QuickSort является очень быстрой сортировкой в среднем, существуют тесты, на которых она работает очень долго. Оценивать время работы алгоритма будем числом сравнений с элементами массива (то есть, суммарным числом сравнений в первом и втором while). Требуется написать программу, генерирующую тест, на котором быстрая сортировка сделает наибольшее число таких сравнений.

Формат входного файла

В первой строке находится единственное число n ($1 \le n \le 10^6$)

Формат выходного файла

Вывести перестановку чисел от 1 до n, на которой быстрая сортировка выполнит максимальное число сравнений. Если таких перестановок несколько, вывести любую из них.

Примеры

input.txt	**output.txt**
3	1 3 2

Примечание

На этой странице можно ввести ответ, выводимый Вашей программой, и страница посчитает число сравнений, выполняемых указанным выше алгоритмом Quicksort. Вычисления будут производиться в Вашем браузере. Очень большие массивы могут обрабатываться долго.

Задание 14. K-ая порядковая статистика

Дан массив из n элементов. Какие числа являются k_1-ым, (k_1+1)-ым, …, k_2-ым в порядке неубывания в этом массиве?

Формат входного файла

В первой строке входного файла содержатся три числа: n — размер массива, а также границы интервала k_1 и k_2, при этом $2 \le n \le 4 \cdot 10^7$, $1 \le k_1 \le k_2 \le n$, $k_2 - k_1 < 200$.

Во второй строке находятся числа A, B, C, a_1, a_1, по модулю не превосходящие 10^9. Вы должны получить элементы массива, начиная с третьего, по формуле: $a_i = A \cdot a_{i-2} + A \cdot a_{i-1} + C$. Все вычисления должны производится в 32-битном знаковом типе, переполнения должны игнорироваться.

Обращаем Ваше внимание, что массив из $4 \cdot 10^7$ 32-битных целых чисел занимает в памяти **160 мегабайт**! Будьте аккуратны!

Подсказка: эту задачу лучше всего решать модификацией быстрой сортировки. Однако сортировка массива целиком по времени, скорее всего, не пройдет, поэтому нужно подумать, как модифицировать быструю сортировку, чтобы не сортировать те части массива, которые не нужно сортировать.

Формат выходного файла

В первой и единственной строке выходного файла выведите k_1-ое, (k_1+1)-ое, …, k_2-ое в порядке неубывания числа в массиве a. Числа разделяйте одним пробелом.

Примеры

input.txt	output.txt
5 3 4 2 3 5 1 2	13 48
5 3 4 200000 300000 5 1 2	2 800005

Примечание

Во втором примере элементы массива a равны: [1, 2, 800005, −516268571, 1331571109].

Задание 15. Сортировка пугалом

«Сортировка пугалом» — это давно забытая народная потешка, которую восстановили по летописям специалисты платформы «Открытое образование» специально для этого курса.

Участнику под верхнюю одежду продевают деревянную палку, так что у него оказываются растопырены руки, как у огородного пугала. Перед ним ставятся n матрёшек в ряд. Из-за палки единственное, что он может сделать — это взять в руки две матрешки на расстоянии k друг от друга (то есть i-ую и (i + k)-ую), развернуться и поставить их обратно в ряд, таким образом поменяв их местами.

Задача участника — расположить матрёшки по неубыванию размера. Может ли он это сделать?

Формат входного файла

В первой строчке содержатся числа n и k ($1 \le n, k \le 10^5$) — число матрёшек и размах рук.

Во второй строчке содержится n целых чисел, которые по модулю не превосходят 10^9 — размеры матрёшек.

Формат выходного файла

Выведите «YES», если возможно отсортировать матрёшки по неубыванию размера, и «NO» в противном случае.

Примеры

input.txt	output.txt
3 2 2 1 3	NO
5 3 1 5 3 4 1	YES

Сортировка подсчетом

Является устойчивой сортировкой с линейным временем выполнения. Условие использования - заранее известный диапазон значений элементов массива `A`. Если это так и, например, элементы массива `A` лежат в интервале от `0` до `k`, можно создать дополнительный массив `B[0..k]`, в который подсчитать количества элементов массива со значениями из отрезка `[0..k]`.

После этого можно модифицировать массив `B` таким образом, чтобы в ячейках находились бы не количества элементов, а номер ячейки, в которой должен располагаться соответствующий элемент массива `A`. После этого остаётся только переписать массив `A`, указав новый порядок элементов.

Пример реализации алгоритма:

```
#include <iostream>
#include <vector>

using namespace std;

vector<int> countSort(vector<int>& a, int k) {
	vector<int> b;
	vector<int> sortedArray;
	for (int i = 0; i <= k; i++) {
		b.push_back(0);
```

```
        }

        for (int i = 0; i < a.size(); i++) {
                int number = a[i];
                b[number]++;
                sortedArray.push_back(0);
        }

        for (int i = 1; i <= k; i++) {
                b[i] += b[i - 1];
        }


        for (int i = a.size() - 1; i >= 0; i--) {
                int number = a[i];
                int position = b[number] - 1;
                sortedArray[position] = number;
                b[number]--;
        }

        return sortedArray;
}

int main()
{
        vector<int> a = { 1, 0, 3, 4, 2, 0, 0, 0, 1, 3 };
        a = countSort(a, 4);
        return 0;
}
```

Цифровая сортировка

Предназначена для сортировки массивов, состоящих из последовательностей одинаковой длины. Эти последовательности состоят в свою очередь из элементов, для которых задано отношение линейного порядка.

По аналогии с разрядами чисел можно называть элементы, из которых состоят сортируемые последовательности, разрядами.

Алгоритм состоит в сортировке заданных последовательностей какой-либо устойчивой сортировкой по каждому разряду, в порядке от младшего к старшему.

Карманная сортировка

Предназначена для сортировки данных, равномерно распределенных в некотором интервале. Идея сортировки - разбить этот интервал на n одинаковых интервалов - карманов - и распределить по ним n сортируемых чисел. далее в каждом кармане производится сортировка чисел, а затем последовательно перечисляется содержимого каждого из карманов. Предполагается, что в каждом кармане окажется не очень много элементов, так как сортируемые числа равномерно распределены на рассматриваемом интервале.

Задание 16. Сортировка целых чисел

В этой задаче Вам нужно будет отсортировать много неотрицательных целых чисел.

Вам даны два массива, A и B, содержащие соответственно n и m элементов. Числа, которые нужно будет отсортировать, имеют вид $A_i \cdot B_i$, где $1 \le i \le n$ и $1 \le j \le m$. Иными словами, каждый элемент первого массива нужно умножить на каждый элемент второго массива.

Пусть из этих чисел получится отсортированная последовательность C длиной $n \cdot m$. Выведите сумму каждого десятого элемента этой последовательности (то есть, $C_1 + C_{11} + C_{21} + ...$).

Формат входного файла

В первой строке содержатся числа n и m ($1 \le n, m \le 6000$) — размеры массивов. Во второй строке содержится n чисел — элементы массива A. Аналогично, в третьей строке содержится m чисел — элементы массива B. Элементы массива неотрицательны и не превосходят 40000.

Формат выходного файла

Выведите одно число — сумму каждого десятого элемента последовательности, полученной сортировкой попарных произведенй элементов массивов A и B.

Примеры

input.txt	output.txt
4 4 7 1 4 9 2 7 8 11	51

Пояснение к примеру

Неотсортированная последовательность C выглядит следующим образом:
[14, 2, 8, 18, 49, 7, 28, 63, 56, 8, 32, 72, 77, 11, 44, 99].
Отсортировав ее, получим:
[**2**, 7, 8, 8, 11, 14, 18, 28, 32, 44, **49**, 56, 63, 72, 77, 99].

Жирным выделены первый и одиннадцатый элементы последовательности, при этом двадцать первого элемента в ней нет. Их сумма — 51 — и будет ответом.

Задание 17. Цифровая сортировка

Дано *n* строк, выведите их порядок после *k* фаз цифровой сортировки.

Формат входного файла

В первой строке входного файла содержатся числа *n* — число строк, *m* — их длина и *k* – число фаз цифровой сортировки ($1 \le n \le 10^6$, $1 \le k \le 10^6$, $n \cdot m \le 5 \cdot 10^7$). Далее находится описание строк, **но в нетривиальном формате**. Так, *i*-ая строка ($1 \le i \le n$) записана в *i*-ых символах второй, …, (*m*+1)-ой строк входного файла. Иными словами, строки написаны по вертикали. **Это сделано специально, чтобы сортировка занимала меньше времени.**

Строки состоят из строчных латинских букв: от символа "a" до символа "z" включительно. В таблице символов ASCII все эти буквы располагаются подряд и в алфавитном порядке, код буквы "a" равен 97, код буквы "z" равен 122.

Формат выходного файла

Выведите номера строк в том порядке, в котором они будут после *k* фаз цифровой сортировки.

Примеры

input.txt	output.txt
3 3 1 bab bba baa	2 3 1
3 3 2 bab bba baa	3 2 1
3 3 3 bab bba baa	2 3 1

Примечание 1

Во всех примерах входных данных даны следующие строки:

- «bbb», имеющая индекс 1;
- «aba», имеющая индекс 2;
- «baa», имеющая индекс 3. Разберем первый пример. Первая фаза цифровой сортировки отсортирует строки по последнему символу, таким

образом, первой строкой окажется «aba» (индекс 2), затем «baa» (индекс 3), затем «bbb» (индекс 1). Таким образом, ответ равен «2 3 1».

Лабораторная работа №5

UDT Polynomial, грамматика и перегрузка операторов.

Цель работы: использовать средства языка для разработки UDT Polynomial, позволяющего выполнять операции над многочленами. Реализовать правила грамматики для корректной интерпретации многочлена, заданного в строковом формате.

Задачи:

1) Сформулировать правила грамматического разбора для парсинга входных данных.
2) Реализовать класс Polynomial, реализующего абстракцию над понятием многочлена. Класс должен содержать следующие методы и перегрузки операторов:

```
class PolynomialClass
{
        vector<Rational> items;
public:
        struct ZeroDivision { string message = "You cannot divide on polynomial that is equal zero."; };
        struct NonePositivePow { string message = "You cannot pow with n <= 0."; };

        PolynomialClass() : items(vector<Rational>(1, Rational())) {}
        explicit PolynomialClass(const vector<Rational> elements);

        int power() const;
        operator string() const;

        Rational operator[] (int index) const;
        PolynomialClass add(const PolynomialClass& right) const;
        PolynomialClass sub(const PolynomialClass& right) const;
        PolynomialClass mult(const PolynomialClass& right) const;
        PolynomialClass div(const PolynomialClass& right) const;
        PolynomialClass res(const PolynomialClass& right) const;
```

```
        PolynomialClass pow(int n) const;
        PolynomialClass d_dx(int n) const;
        Rational valueAt(Rational x) const;
        bool isRoot(Rational x) const;
};

ostream& operator << (ostream& stream, const PolynomialClass& p);
istream& operator >> (istream& stream, PolynomialClass& p);

PolynomialClass operator + (const PolynomialClass& left, const PolynomialClass& right);
void operator += (PolynomialClass& left, const PolynomialClass& right);

PolynomialClass operator - (const PolynomialClass& left, const PolynomialClass& right);
void operator -= (PolynomialClass& left, const PolynomialClass& right);

PolynomialClass operator * (const PolynomialClass& left, const PolynomialClass& right);
void operator *= (PolynomialClass& left, const PolynomialClass& right);

PolynomialClass operator / (const PolynomialClass& left, const PolynomialClass& right);
void operator /= (PolynomialClass& left, const PolynomialClass& right);

PolynomialClass operator % (const PolynomialClass& left, const PolynomialClass& right);
void operator %= (PolynomialClass& left, const PolynomialClass& right);

PolynomialClass NOD(const PolynomialClass& a, const PolynomialClass& b);
```

Рассмотрим более детально содержимое класса:

```
struct ZeroDivision { string message = "You cannot divide on polynomial that is equal zero."; };
struct NonePositivePow { string message = "You cannot pow with n <= 0."; };
```

Структуры предназначены для вывода сообщений об ошибках.

```
PolynomialClass() : items(vector<Rational>(1, Rational())) {}
```

```
explicit PolynomialClass(const vector<Rational> elements);
```
Конструктор по умолчанию и конструктор, создающий объект на основании вектора коэффициентов многочлена.

```
int power() const;
```
Возвращает степень многочлена.

```
operator string() const;
```
Оператор преобразования многочлена в строку.

```
Rational operator[] (int index) const;
```
Оператор, позволяющий получить значение коэффициента многочлена по его индексу.

```
PolynomialClass add(const PolynomialClass& right) const;
PolynomialClass sub(const PolynomialClass& right) const;
PolynomialClass mult(const PolynomialClass& right) const;
PolynomialClass div(const PolynomialClass& right) const;
PolynomialClass res(const PolynomialClass& right) const;
PolynomialClass pow(int n) const;
PolynomialClass d_dx(int n) const;
```
Методы многочлена, реализующие основные операции над многочленами: сложение, вычитание, умножение, целую часть от деления, остаток от деления, возведение в степень и вычисление производной порядка n.

```
Rational valueAt(Rational x) const;
```
Метод, возвращающий значение многочлена в точке x.

```
bool isRoot(Rational x) const;
```
Метод, который выполняет проверку, является ли число x корнем многочлена.

```
ostream& operator << (ostream& stream, const PolynomialClass& p);
```
Оператор выгрузки многочлена в поток вывода.

```
istream& operator >> (istream& stream, PolynomialClass& p);
```
Оператор считывания многочлена из потока ввода. Операции считывания вынести в отдельный класс, реализующего правила грамматики над

многочленами.

```
PolynomialClass operator + (const PolynomialClass& left, const PolynomialClass& right);
void operator += (PolynomialClass& left, const PolynomialClass& right);
```
Перегрузка операторов сложения многочленов. Используют метод add для своей работы.

```
PolynomialClass operator - (const PolynomialClass& left, const PolynomialClass& right);
void operator -= (PolynomialClass& left, const PolynomialClass& right);
```
Перегрузка операторов вычитания для многочленов. Используют метод sub для своей работы.

```
PolynomialClass operator * (const PolynomialClass& left, const PolynomialClass& right);
void operator *= (PolynomialClass& left, const PolynomialClass& right);
```
перегрузка операторов умножения для многочленов. Используют метод mult для своей работы.

```
PolynomialClass operator / (const PolynomialClass& left, const PolynomialClass& right);
void operator /= (PolynomialClass& left, const PolynomialClass& right);
```
Перегрузка операторов для получения целой части от деления многочленов. Используют метод div для своей работы.

```
PolynomialClass operator % (const PolynomialClass& left, const PolynomialClass& right);
void operator %= (PolynomialClass& left, const PolynomialClass& right);
```
Перегрузка операторов для получения остатка от деления многочленов. Используют метод res для своей работы.

```
PolynomialClass NOD(const PolynomialClass& a, const PolynomialClass& b);
```
Находит наибольший общий делитель двух многочленов.

Класс для реализации грамматики будет иметь следующий интерфейс:
```
class PolynomialParser {
        class Token {
```

```
        PolynomialClass value;
        char kind;
    public:
        Token(char k) : kind(k) {}
        Token(PolynomialClass v) : kind('8'), value(v) {}
        PolynomialClass getValue() { return value; }
        char getKind() { return kind; }
    };

    istream& is;
    Token buffer = 0;
    bool full = false;

    Rational getNumber();
    PolynomialClass createPolynomial(Rational coef, int power);
    void putback(Token t);

    Token getToken();

    PolynomialClass primary();
    PolynomialClass term();
    PolynomialClass expression();
public:
    struct ParseNumberError { string message = "Number parse error."; };
    struct WrongToken { string message = "Wrong token."; };
    struct PutbackOnFullBuffer { string message = "Putback in full buffer"; };
    explicit PolynomialParser(istream& stream) : is(stream) {}

    PolynomialClass parse();
};
```

3) Реализовать класс PolynomialRoots, который исследует расположение действительных корней многочлена на заданном интервале на основании метода Штурма.

Подробнее смотрите здесь.

3.3. Критерии оценивания по видам оценочных средств

Критерии оценивания лабораторной работы

Оценка	Неудовлетворительно	Удовлетворительно	Хорошо	Отлично
Критерии	Студент не выполнил задание: не представил алгоритм решения задачи, не составил исходный код программной реализации, программное решение некорректно работает для некоторых значений входных параметров студент не моет объяснить предложенное решение	Студент правильно выполнил задание к лабораторной работе. Составил отчет в установленной форме. Привел исходный код решения, программное решение работоспособно. Студент может объяснить предложенное решение	Студент правильно выполнил задание к лабораторной работе. Составил отчет в установленной форме. Привел исходный код решения, программное решение работоспособно. Студент может объяснить предложенное решение. Студент способен ответить на теоретические вопросы, испытывая небольшие затруднения.	Студент правильно выполнил задание к лабораторной работе. Составил отчет в установленной форме. Привел исходный код решения, программное решение работоспособно. Студент может объяснить предложенное решение. Студент способен ответить на теоретические вопросы.

Тест

Во время прохождения теста студент отвечает на 20 тестовых вопросов. Продолжительность теста – 35 минут.

Пример заданий для тестирования:

Ознакомьтесь с кодом:

```
interface A { }
interface B : A { }

class X : A {}
class Y : X, B {}
class Z : X {}
```

Какие из этих следующих операторов не выбросят исключение?

- Y
- Y as B
- X
- X as B
- Z

4. ПОРЯДОК ПРОВЕДЕНИЯ И КРИТЕРИИ ОЦЕНИВАНИЯ ПРОМЕЖУТОЧНОЙ АТТЕСТАЦИИ

4.1 Порядок проведения и содержание оценочных средств для промежуточной аттестации

Пример заданий для тестирования:

Ознакомьтесь с кодом:

```
namespace MyNamespace
{
   internal class ClassA { }

   public class ClassB
   {
      public ClassA Method1() { return null; }
      private ClassB Method2() { return null; }
   }
}
```

Перечислите все способы, которыми можно избавиться от ошибок компиляции в этом коде

- Сменить модификатор доступа Method1 с public на internal
- Сменить модификатор доступа ClassB с public на internal
- Сменить модификатор доступа Method2 с private на public
- Перенести этот код в другую сборку
- Сменить модификатор доступа ClassA с internal на public
- Сменить модификатор доступа Method1 с public на private

Пример задания для практических работ:

Часто делегаты можно использовать для тонкой настройки алгоритмов, что позволит использовать один и тот же код для решения несколько разных задач. Скачайте проект проекте Delegates.TreeTraversal

Перед вами три задачи:

- Дано дерево категорий продуктов, в каждой категории могут быть другие категории и собственно продукты. Вам нужно вывести список продуктов.
- Дано дерево задача, каждая задача может содержать подзадачи. Вам нужно вывести список таких задач, у которых нет подзадач.
- Дано бинарное дерево, у которого каждый лист содержит величину, а каждый не-лист не содержит величины. Вам нужно вывести все величины, содержащиеся в этом дереве.

Вам нужно написать один алгоритм обхода дерева, который бы принимал в качестве аргументов делегаты, объясняющие алгоритму, как обходить дерево и какие величины выводить. Слишком сложные делегаты могут затруднять чтение кода, поэтому из всего многообразия решения выберите решение, максимально понятное неподготовленному читателю. После этого вам нужно написать реализации методов, указанных в тестах, так, чтобы тесты заработали.

4.2 Критерии оценивания компетенций в ходе промежуточной аттестации

Код компетенции	Планируемые результаты	Критерии оценивания	
		зачтено	Не зачтено
ПК-2	Знает классы, объекты и модули в языке C#; компоненты стандартных библиотек языка C#; основные понятия, связанные с системами программирования; состав и схемы работы систем объектно-ориентированного программирования;	Знает классы, объекты и модули в языке C#; компоненты стандартных библиотек языка C#; основные понятия, связанные с системами программирования; состав и схемы работы систем объектно-ориентированного программирования;	Знает классы, объекты и модули в языке C#; компоненты стандартных библиотек языка C#; основные понятия, связанные с системами программирования; состав и схемы работы систем объектно-ориентированного программирования;
	Умеет применять на практике стандартные средства языка C# и концепции объектно-ориентированного программирования для разработки программного обеспечения	Умеет применять на практике стандартные средства языка C# и концепции объектно-ориентированного программирования для разработки программного обеспечения	Умеет применять на практике стандартные средства языка C# и концепции объектно-ориентированного программирования для разработки программного обеспечения

Владеет навыками написания и отладки кода на языке C#; навыками использования средств объектно-ориентированного программирования в разработке приложений	Владеет навыками написания и отладки кода на языке C#; навыками использования средств объектно-ориентированного программирования в разработке приложений	Владеет навыками написания и отладки кода на языке C#; навыками использования средств объектно-ориентированного программирования в разработке приложений

4.3 Критерии оценивания зачета

Для получения зачета, обучающийся демонстрирует исходный код решения, свободно ориентируется в нём и может ответить на дополнительные вопросы. Каждое задание оценивается в различное количество баллов, в зависимости от его сложности и объема.

Исходный код задания должен пройти автоматическую проверку на корректность с помощью технологии unit-тестирования, а затем код-ревью преподавателя. При проверке заданий преподавателем, упор делается на:

- соответствие поставленным в условии требованиям
- соответствие общепринятым конвенциям стиля C#
- использование практик и принципов качественной архитектуры и организации кода
- уникальность решения.

Максимальный балл составляет—1135.

Для получения «зачтено» обучающийся должен набрать суммарное количество баллов за практические работы и тест: 300 баллов.

«Не зачтено» может быть поставлено обучающемуся в том случае, если он не набирает требуемого количества баллов или не может защитить свои решения в процессе код-ревью.

4.4. Результаты промежуточной аттестации и уровни сформированности компетенций

Уровень освоения компетенций	Оценка
Продвинутый	зачтено
Базовый	зачтено
Пороговый	зачтено
компетенции не сформированы	не зачтено

Уровни формирования компетенций:

1. Пороговый уровень:
   - предполагает формирование компетенций на начальном уровне: знание основ языка программирования C#;
   - студент способен давать ответы на теоретические вопросы дисциплины на удовлетворительном уровне.
2. Базовый уровень:
   - предполагает формирование компетенций на более высоком уровне: формируется комплексное знание особенностей и применения методов языка программирования C#;
   - студент способен давать развернутые ответы на теоретические вопросы дисциплины; способен решать практические задания.
3. Продвинутый уровень:
   - предполагает формирование компетенций на высоком уровне, использует полученные знания и умения при изучении смежных дисциплин, обнаруживает готовность к самостоятельной профессиональной деятельности;
   - студент способен аргументировать собственную точку зрения, формулировать собственные выводы на основе применения усвоенных компетенций.