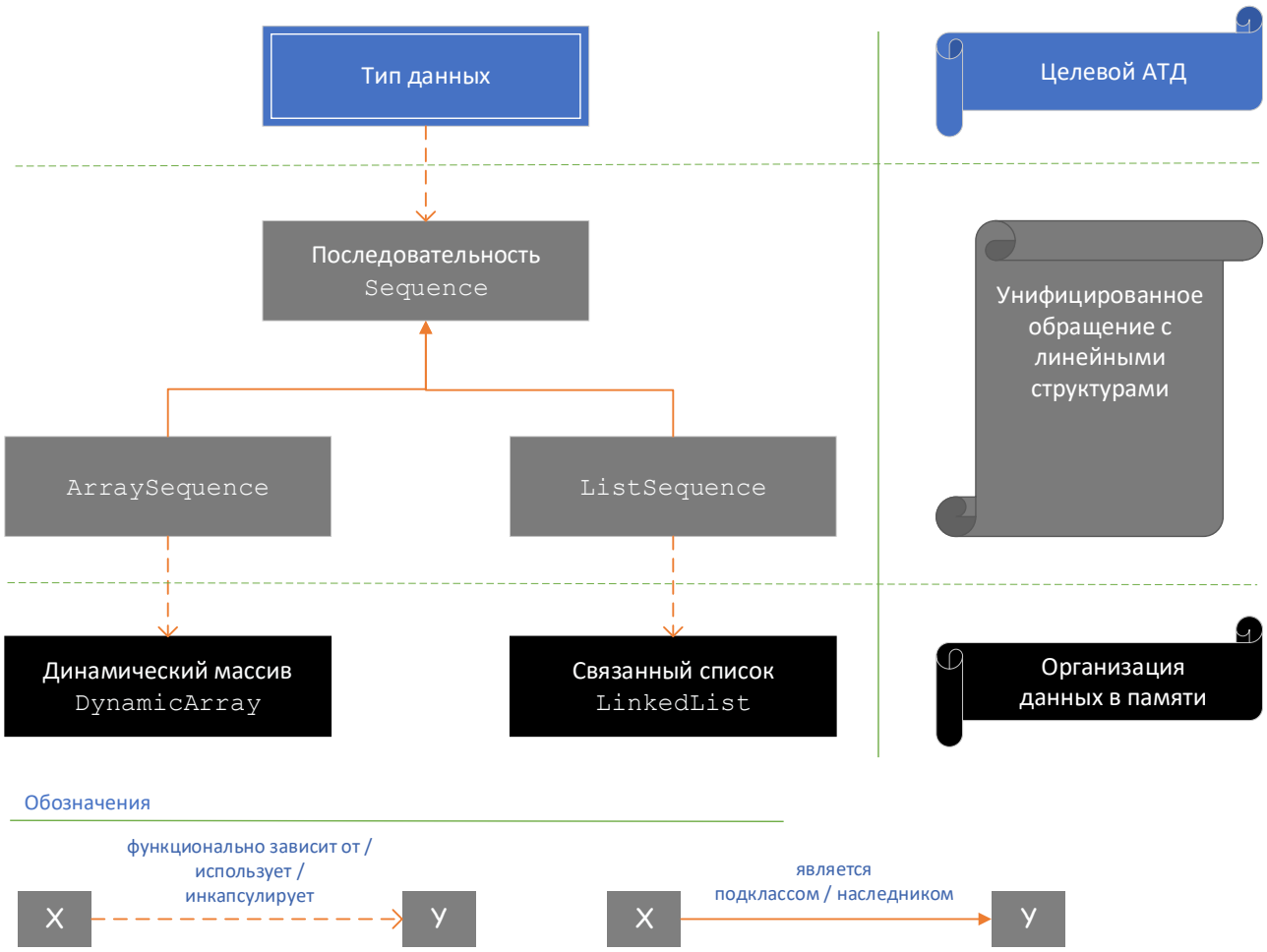
Лабораторная работа №3

по курсу информатики, 2 семестр

Варианты заданий

1. Постановка задачи

Написать на языке C++ реализацию некоторого целевого полиморфного абстрактного типа данных на базе АТД Последовательность (см. схему ниже).



Следует предусмотреть средства для оценки производительности полученной реализации.

Минимальные требования к программе. В составе лабораторной работы должны быть реализованы:

- АТД динамический массив,
- АТД линейный связанный список,
- АТД последовательность,
- целевой АТД, указанный в варианте задания,
- оболочка с UI для тестирования полученной системы объектов.

Для реализации необходимо использовать возможности ООП и шаблонов C++ (templates) – классов и функций. Во всех реализованных функциях необходимо обрабатывать случаи некорректных значений входных параметров – как правило, в таких случаях следует выбрасывать исключения.

Все реализованные классы и основные алгоритмы необходимо покрыть (модульными) тестами. Реализацию следует оснастить пользовательским интерфейсом (консольным) для проверки корректности реализации.

2. Содержание вариантов

Таблица 1. Список целевых АТД

№ варианта	Тип коллекции	Типы хранимых элементов	Дополнительные операции
1.	Очередь	Элементы: –Целые числа –Вещественные числа –Комплексные числа –Строки/символы –Функции[2)] –Студенты[3)] –Преподаватели[3)]	–map, where, reduce[1)] –Конкатенация –Извлечение подпоследовательности (по заданным индексам) –Поиск на вхождение подпоследовательности –Сцепление –Разделение на 2 очереди (по заданному признаку)
2.	Очередь с приоритетами		
3.	Стек		–map, where, reduce –Конкатенация –Извлечение подпоследовательности (по заданным индексам) –Поиск на вхождение подпоследовательности
4.	Дек		–Сортировка –map, where, reduce –Конкатенация –Извлечение подпоследовательности (по заданным индексам) –Поиск на вхождение подпоследовательности –Слияние
5.	Дек с сегментированным буфером		
6.	Вектор	Коэффициенты: –Целые числа –Вещественные числа –Комплексные числа	Сложение, умножение на скаляр, вычисление нормы, скалярное произведение
7.	Квадратная матрица		Сложение, умножение на скаляр, вычисление нормы, элементарные преобразования строк/столбцов

8.	Прямоугольная матрица		Сложение, умножение на скаляр, вычисление нормы, элементарные преобразования строк/столбцов
9.	Треугольная матрица		Сложение, умножение на скаляр, вычисление нормы
10.	Диагональная матрица[4)]		
11.	Разреженная матрица		Сложение, умножение, умножение на скаляр, вычисление нормы
12.	Многочлен[5)]	Коэффициенты: –Целые числа –Вещественные числа –Комплексные числа –Квадратные матрицы	Сложение, умножение, умножение на скаляр, вычисление значения для заданного значения аргумента, композиция
13.	«Линейная форма»[6)]	Коэффициенты: –Целые числа –Вещественные числа –Комплексные числа	Сложение (и вычитание), умножение на скаляр, вычисление значения при заданных значениях аргументов
14.	Поток (stream) данных	Элементы: –Целые числа –Вещественные числа –Комплексные числа –Строки/символы –Функции –Студенты –Преподаватели –Пары объектов	–map, where, reduce –Извлечение подпоследовательности (по заданным индексам) –Поиск на вхождение подпоследовательности –Слияние –Разделение (по заданному признаку)
15.	Множество	Элементы: –Целые числа –Вещественные числа –Комплексные числа –Строки/символы –Функции –Студенты –Преподаватели –Пары объектов	–map, where –объединение –пересечение –вычитание –проверка на включение подмножества –проверка на вхождение элемента –сравнение (равенство) двух множеств
16.	Кусочная функция[7)]	Коэффициенты: –Целые числа –Вещественные числа –Комплексные числа	–Доопределение или переопределение на отрезке –Проверка на монотонность –Проверка на непрерывность –Вычисление значения в

			точке
17.	«Функциональный список»	–	–

1) Если $l = [a_1, .., a_n]$ – некоторый список элементов типа T, а $f: T \to T$, то:

$$\text{map}(f, l) \mapsto [f(a_1), .., f(a_n)]$$

Если, при тех же соглашениях, $h: T \to \text{Bool}$ – некоторая функция, возвращающая булево значение, то результатом $\text{where}(h, l)$ будет новый список l', такой что: $a'_i \in l' \Leftrightarrow h(a'_i) = \text{true}$. Т.е. where фильтрует значения из списка l с помощью функции-фильтра h.

Функция reduce работает несколько иначе: «сворачивает» список в одно значение по заданному правилу $f: T \times T \to T$:

$$\text{reduce}(f, l, c) \mapsto f\left(a_n, \left(f\left(a_{n-1}, \left(\dots f\left(a_2, \left(f(a_1 c)\right)\right)\right)\right)\right)\right)$$

где c – константа, «стартовое» значение. Например, $l = [1,2,3]$, $f(x_1, x_2) = 2x_1 + 3x_2$, тогда:

$$reduce(f, [1,2,3], 4) = f\left(3, f\left(2, f(1,4)\right)\right) =$$
$$= 2 \cdot 3 + 3\left(2 \cdot 2 + 3(2 \cdot 1 + 3 \cdot 4)\right) =$$
$$= 2 \cdot 3 + 3(2 \cdot 2 + 3 \cdot 14) = 2 \cdot 3 + 3 \cdot 42 = 132$$

Пара функций zip и unzip позволяет «сцеплять вдоль» несколько списков и разъединять их.

$$zip: List\langle T_1 \rangle \times List\langle T_2 \rangle \to List\langle Tuple\langle T_1, T_2 \rangle \rangle$$

$$zip3: List\langle T_1 \rangle \times List\langle T_2 \rangle \times List\langle T_3 \rangle \to List\langle Tuple\langle T_1, T_2, T_3 \rangle \rangle$$

$$zipN: List\langle List\langle T \rangle \rangle \to List\langle List\langle T \rangle \rangle$$

Примеры:

$$zip([1,2,3], ["a", "b", "c", "d",]) = [(1, "a"), (2, "b"), (3, "c")]$$

$$zipN\big([[1,2,3], [1,2,3], [1,2,3]] = [[1,1], [2,2], [3,3]]\big)$$

Функция unzip является обратной к zip: «распаковывает» список кортежей в кортеж списков.

2) Точнее, указатели на функции. Ниже – минимальный пример, как создать «список функций»:

```
const int array_length = 3;
int(**f)(int) = malloc(array_length * sizeof(int(*)(int)));
f[0] = &inc1;
f[1] = &inc2;
f[2] = &inc3;
for (int index = 0; index < length; index++)
      printf("%i ", f[index](0));
// Вывод: 1 2 3
```

3) Точнее, описывающие их структуры. Персона характеризуется набором атрибутов, таких ФИО, дата рождения, некоторый идентификатор (в роли которого может выступать: номер в

некотором списке, номер зачетки/табельный номер, номер паспорта, и др.). Пример структуры, описывающей персону:

```
class Person {
private:
	PersonID id;
	char* firstName;
	char* middleName;
	char* lastName;
	time_t 5erson5te;
public:
	PersonID GetID();
	char* GetFirstName();
	…
}
```

Тип PersonID предназначен для идентификации персоны и может быть объявлен различным образом, в зависимости от выбранного способа идентификации человека. Если для этих целей используется, скажем, номер паспорта, можно предложить, по крайней мере, два различных определения:

первое:

```
#typedef Person_ID char* // null-terminated string[1] вида “0982 123243”
```

второе:

```
#typedef Person_ID struct { // можно и в виде класса
	int series;		// как вариант, char*
	int number;		// как вариант, char*
}
```

Для получения значения атрибутов предусматривают соответствующие методы, например:

```
char* name = person->getName(); //  = “Иван”
char* fullName = person->getFullName(); // = “Иван Иванович Иванов”, вычислимый атрибут
```

4) Диагональной называется матрица, у которого лишь элементы на главной диагонали отличны от 0. Также рассматривают трехдиагональные матрицы, которых все отличные от 0 элементы расположены на главной диагонали, а также на двух смежных с ней: верхней и нижней:

$$\begin{pmatrix} * & * & 0 & \dots & 0 \\ * & * & * & \dots & 0 \\ 0 & * & \ddots & & \vdots \\ \vdots & & \ddots & * & * \\ 0 & \dots & 0 & * & * \end{pmatrix}$$

Аналогично можно рассматривать 5-диагональные, $2k+1$-диагональные матрицы, и т.д.

[1] См. например: https://en.wikipedia.org/wiki/Null-terminated_string. Идея такая, что конец строки определяется по наличию символа с кодом 0.

5) Многочлен степени n записывается в виде: $P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ и может быть однозначно задан списком своих коэффициентов $a_0, \ldots, , a_n$. Многочлен является функцией, на множестве функций определена ассоциативная операция – композиция $\circ$: $(f \circ g)(x) = f\big(g(x)\big)$.

6) Подразумевается многочлен первой степени от n переменных: $F_n(x_1, \ldots, x_n) = a_0 + a_1 x_1 + \cdots + a_n x_n$.

7) Пусть есть полное упорядочение[2] $(<, R)$. Тогда на отрезке $[a, b] \subseteq R$ может быть определена функция $f: R \to D$ (D – область значений, произвольное множество, желательно, хотя бы частично упорядоченное). Пусть имеется $\xi_0 (= a) < \xi_1 < \cdots < \xi_n < \xi_{n+1} (= b) \in R$ – разбиение отрезка $[a, b]$; тогда кусочно-заданная функция – это система функций: $f_i: [\xi_{i-1}, \xi_i] \to D$, причем неоднозначность в точках ξ_i устраняется за счет, что берется значение из $(i+1)$-го сегмента (кроме последней точки): $f(\xi_i) = f_{i+1}(\xi_i), i \in \overline{1, n-1}$ и $f(\xi_n) = f_n(\xi_n)$.

(Задание должно включать пункты общей «стоимостью» не менее 9 (11 для гр. 511) баллов.)

Таблица 2. Содержание вариантов заданий

	Задача	**Пояснения**	**Баллы**
С-1.	Очередь	–	3
С-2.	Очередь с приоритетами	–	4
С-3.	Стек	–	3
С-4.	Дек	–	3
С-5.	Дек с сегментированным буфером	–	10
С-6.	Вектор	–	5
С-7.	Матрица		
С-7.1.	Хранение элементов	–	
С-7.1.1.	«Обычная» матрица	–	5
С-7.1.2.	Разреженная	–	5
С-7.1.3.	С автоматическим переключением	–	7
С-7.2.	Разновидность	–	
С-7.2.1.	Квадратная матрица		5
С-7.2.2.	Прямоугольная матрица		5
С-7.2.3.	Треугольная матрица		5
С-7.2.4.	Диагональная матрица		4
С-7.2.5.	Ортогональная матрица		
С-8.	Многочлен	–	6
С-9.	«Линейная форма»	–	5
С-10.	Поток (stream) данных	–	
С-10.1.	Специальная реализация для работы с файлами	–	5
С-10.2.	Обобщенный механизм	–	8
С-11.	Множество	–	6
С-12.	Кусочная функция	–	10

[2] Total ordering

C-13.	Сплайн-интерполяция (на базе косучно-полиномиальной функции)	1. http://statistica.ru/branches-maths/interpolyatsiya-splaynami-teor-osnovy/ 2. http://www.machinelearning.ru/wiki/index.php?title=%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%D0%BA%D1%83%D0%B1%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%BC%D0%B8_%D1%81%D0%BF%D0%BB%D0%B0%D0%B9%D0%BD%D0%B0%D0%BC%D0%B8 3.	10
C-14.	«Функциональный» список[9)]	Список, реализованный так, как он обычно реализуется в функциональных языках программирования	8
C-15.	Многомерный массив	–	
M-1.	ICollection<T>	Основные методы – T Get(size_t) –size_t GetCount() –копирующий конструктор	2
M-2.	Map-Reduce	–	
M-2.1.	Базовые операции	Map, reduce,	1
M-2.2.	Дополнительные	Where	2
M-2.3.		Zip/unzip	
M-2.4.		Split, Slice	
M-3.	Энумератор (IEnumerable + IEnumerator)	–	3
M-4.	Mutable/Immutable	–	3
M-5.	Использовать IGroup/IRing	–	4
M-6.	Перегрузка операторов[8)]		2
M-7.		–	
M-8.		–	
M-9.		–	
A-1.	Определение числа инверсий в последовательности	См. например Таблица инверсий — Викиконспекты (ifmo.ru). При этом необязательно, что последовательность – числовая, это могут быть элементы любого множества с тотальным упорядочением (total ordering, не путать с complete partial ordering – полное частичное упорядочение).	

A-1.1.	с помощью map-reduce	Потребуется вариант $map: \big((T_1 \times Int \to T_2) \times List\langle T_1\rangle\big) \to List\langle T_2\rangle$	1
A-1.2.	с помощью циклов (в несколько проходов)		1
A-1.3.	в цикле за один проход (с помощью вспомогательного списка)		2
A-2.	Реализация линейных рекуррентных последовательностей на основе рекуррентной порождающей функции и запоминающего списка	В общем случае, $x_n = \alpha_1 x_{n-1} + \cdots + \alpha_k x_{n-k}$ $(\alpha_k \neq 0)$. При этом первые k элементов задаются каким-либо образом заранее (в т.ч., возможно, с помощью другого рекуррентного закона). Число k называется порядком последовательности.	
A-2.1.	1-го порядка (на пример, факториал)	–	1
A-2.2.	2-го порядка (числа Фибоначчи)	–	1
A-2.3.	порядок заранее не фиксируется, передается список коэффициентов	–	1.5
A-3.	Генерирование и фильтрование ограниченного ряда натуральных чисел по тому или иному закону	1. Реализовать функцию Range, которая по двум числам $l \leq h$ генерирует последовательность целых чисел $[l, l+1, \dots, h-1, h]$ 2. Воспользоваться функцией where, чтобы отфильтровать из порожденной послед-ти требуемые элементы Варианты функций-фильтров перечислены ниже	1
A-3.1.	разложение заданного числа на множители		1
A-3.2.	определение простых чисел		1
A-3.3.	разложение данного числа на простые множители 😊		2
A-3.4.	решето Эратосфена	Например, так Решето Эратосфена — Википедия (wikipedia.org)	2
A-4.	Построение транзитивных замыканий на примере отношения частичного порядка	На вход подается диаграмма Хассе в виде множества пар элементов. Реализовать функцию IsLessOrEqual в составе АТД PartialOrdering<T>, которая для двух данных элементов x_1 и x_2типа T возвращает true, если и только если $x_1 \leq x_2$. Исходные данные для построения транзитивного замыкания хранятся	4

		в экземплярах класса PartialOrdering<T>. Далее возможны два варианта:	
А-4.1.	материализовать все опущенные в диаграмме Хассе дуги	Генерируется список всех пар, представляющих рефлексивные и транзитивные дуги (которые, как известно, опускаются в диаграмме Хассе)	
А-4.2.	искать наличие между двумя данными парами элементов в диаграмме Хассе, рассматриваемой в таком случае как ориентированный граф		
А-5.	Реализация декартова произведения для ЧУМ[3]	Реализовать вариант АТД PartialOrdering<T>, содержащий функцию Product, имеющей тип: $PO\langle T_1\rangle \times PO\langle T_2\rangle \to PO\langle T_1 \times T_2\rangle$.	2
А-5.1.	элементы задаются типом и функцией сравнения $\leq$		
А-5.2.	элементы явно перечисляются и для них задается диаграмма Хассе	реализация по аналогии с А-4	3
А-6.	Генерирование почти-равномерных случайных перестановок		8
А-7.	Построение булеана данного множества	Множество представляется последовательностью. Требуется, по сути, получить всевозможные сочетания элементов, в количестве от 0 до n (длина исходной последовательности), без учета порядка. Результат должен иметь тип Sequence<Sequence<T>>	3
А-8.	Построение множества всех возможных подпоследовательностей	Подпоследовательностью считается в данном случае набор элементов, смежных в исходной последовательности. Так, $[a_k, a_{k+1}, a_{k+2}, a_{k+3}]$ – подпослед-ть, в то время как $[a_k, a_{k+1}, a_{k+3}, a_{k+2}]$ и $[a_k, a_{k+1}, a_{k+3}]$ – нет. Очевидно, подпоследовательности могут иметь длину от 0 до n. Результат должен иметь тип Sequence<Sequence<T>>	3
А-9.	Расчет траектории полета («пристрелка»)	Пусть имеется материальная точка, в вакууме, в однородном,	

[3] Частично Упорядоченное Множество

		постоянном поле гравитации. Требуется подобрать начальный вектор скорости (т.е. угол наклона к горизонту α и величину начальной скорости v_0) так, чтобы попасть в заданную область (т.е. отрезок $[x_1, x_2]$ на оси X). Для поиска решения следует использовать метод дихотомии, а аналитические расчеты – для самопроверки. В качестве дополнительного ограничения – величину начальной скорости можно менять не произвольно, а лишь дискретно.	
Задания повышенной сложности			
П-1.	Словарь слов (список)	Построение «именного указателя»: для заданной строки определить список входящих в нее слов (возможно, за исключением некоторых, перечисленных в заданном словаре), и для каждого такого найденного слова указать список позиций в исходной строке, в которых оно встречается.	
П-2.	«Ханойская ба шня»	Написать программу, решающую задачу о «ханойской башне»[4]. Стержни моделировать стеками, в роли колец могут выступать произвольные предметы, характеризующиеся формой и цветом. Параметрами задачи являются список предметов и номер стержня, на котором предметы размещены изначально.	
П-3.	Интерпретатор регулярных выражений для полиморфных последовательностей	Исходными являются: символьная строка и последовательность элементов. Строка – регулярное выражение[5]. Определить, соответствует ли ему данная последовательность элементов.	
П-4.	«Сортировочная станция»	На станцию пришел поезд, составленный из пронумерованных (например, по возрастанию) вагонов различных типов. Требуется пересобрать состав таким образом, чтобы вагоны одного типа шли строго последовательно и с сохранением исходной нумерации. Вагоны можно отцеплять от состава только с переднего края, а прицеплять – только в конец. Станция оснащена несколькими тупиковыми путями. Решить задачи с использованием возможно меньшего количества тупиков. Пример. Пусть t_1, t_2, t_3 – типы вагонов; каждый вагон	

[4] См. https://ru.wikipedia.org/wiki/ханойская_башня или https://habrahabr.ru/post/200758
[5] https://ru.wikipedia.org/wiki/регулярное_выражение

		представлен парой «тип-номер» (t, n). Тогда «исходный» состав из 8-ми вагонов может быть представлен списком: $[(t_2, 1); (t_3, 2); (t_3, 3); (t_1, 4); (t_2, 5); (t_3, 6); (t_2, 7); (t_1, 8)]$. В результате работы алгоритма должен получиться состав: $[(t_2, 1); (t_2, 5); (t_2, 7); (t_3, 2); (t_3, 3); (t_3, 6); (t_1, 4); (t_1, 8)]$.
П-5.	Строка	Реализовать класс строк CString, включая различные операции: конкатенацию, получение подстроки, поиск вхождений подстроки, разбиение на строки, замена подстроки на другую строку, и др. С помощью наследования и переопределения методов, представить два варианта реализации: на основе связанных списков и на основе динамических массивов. (10-12)
П-6.	Калькулятор полиномов	
П-7.	Матричный калькулятор	
П-8.	Точная арифметика	
П-9.	Генерирование случайной двухмерной карты	
П-10.	Расчет параметров раекты и траектории ее полета	

[8] Для каждого типа данных свой перечень релевантных операторов. Например, для алгебраических типов – это операторы, представляющие операции (сложение, умножение и т.п.). Коллекции в некоторой степени тоже можно отнести к алгебраическим типам (коллекции, по крайней мере, являются моноидами относительно конкатенации[6]).

[9] Пусть [ ] – пустой список. Тогда список из n элементов $[a_1, \dots, a_n]$ представляется как:

$$a_1 :: a_2 :: \cdots a_n :: [\,] = \left[a_1, \left[a_2, \left[\dots, \left[a_n, [\,]\right]\right]\right]\right]$$

Т.е. список представляется системой вложенных упорядоченных пар («выровненных», в данном случае, вправо).

[10] Пусть имеется некоторая последовательность $a_1, \dots, a_n$. Равномерной ее перестановкой считается последовательность $a_{p_1}, \dots, a_{p_n}$, такая что любой из исходных элементов a_i с одинаковой вероятностью может оказаться в любой из позиций p_j. В случае с почти равномерной перестановкой эти вероятности могут несколько различаться.

Для решения задачи предположим, что элементы содержаться в массиве (линейной изменяемой структуре с производным доступом). Разделим это размещение на сегменты, например, для простоты можно взять 4 сегмента равной длины. Можно брать и большее число сегментов, особенно для больших коллекций, а также необязательно, чтобы они были равно длины, но есть смысл, чтобы они располагались симметрично относительно середины массива. Итак:

6

1. Для примера возьмем 4 равных по длине сегмента: $[1 - {}^{1}/_{4}\, n], [{}^{1}/_{4}\, n - {}^{1}/_{2}\, n], [{}^{1}/_{2}\, n - {}^{3}/_{4}\, n],$ и $[{}^{3}/_{4}\, n - n]$.
2. Выполним случайные различные и не пересекающиеся перестановки между двумя внутренними сегментами, в количестве примерно ${}^{1}/_{2}$ от количества элементов в сегменте.
3. Далее аналогично выполним перестановки двумя внешними сегментами.
4. Наконец, аналогично предыдущему, выполним перестановки между 1-м и 2-м, а также между 3-м и 4-м сегментами.

В результате получим неплохо перемешанную последовательность элементов.

При большом количестве элементов и, соответственно, большом количестве сегментов можно применять следующую схему:

1. Пусть имеется k сегментов $s_1, .., s_k$.
2. Для каждого $p \in \left\{ {}^{1}/_{m}\, k | m = \overline{2, \dots, {}^{k}/_{2}} \right\}$:
    a. Выполнять перестановки между сегментами на расстоянии p друг от друга. Так, на первом шаге, при $p = {}^{1}/_{2}\, k$, перестановки выполняются между s_1 и $s_{1_{/_{2}k+1}}$, s_2 и $s_{1_{/_{2}k+2}}$,…, $s_{1_{/_{2}k}}$ и s_k, т.е. между сегментами, отстоящими друг от друга на половину длины массива. На втором шаге – на треть длины, и так далее, вплоть до перестановок между соседними сегментами.

3. Критерии оценки

1.	Качество программного кода:	– стиль (в т.ч.: имена, отступы и проч.) (0-2) – структурированность (напр. декомпозиция сложных функций на более простые) (0-2) – качество основных и второстепенных алгоритмов (напр. обработка граничных случаев и некорректных исходных данных и т.п.) (0-3)	0-6 баллов
2.	Качество пользовательского интерфейса:	– предоставляемые им возможности (0-2) – наличие ручного/автоматического ввода исходных данных (0-2) – настройка параметров для автоматического режима – отображение исходных данных и промежуточных и конечных результатов и др. (0-2)	0-6 баллов
3.	Качество тестов	– степень покрытия – читаемость – качество проверки (граничные и	0-5 баллов

		некорректные значения, и др.) – полнота и качество представления результатов тестирования	
4.	Полнота выполнения задания и качество ТЗ	Оценивается качество подготовки ТЗ, функциональная полнота реализации,	0-3 баллов
5.	Владение теорией	знание алгоритмов, области их применимости, умение сравнивать с аналогами, оценить сложность, корректность реализации	0-5 баллов
6.	Оригинальность реализации	оцениваются отличительные особенности конкретной реализации – например, общность структур данных, наличие продвинутых графических средств, средств ввода-вывода, интеграции с внешними системами и др.	0-5 баллов
		Итого	0-30 баллов
7.	Объем выбранного задания	дополнительная работа, выполненная сверх установленного минимума, согласованность выбранных составляющих (например, нескольких взаимосвязанных задач оценивается выше, чем реализация набора независимых задач)	

Для получения зачета за выполнение лабораторной работы необходимо соблюдение всех перечисленных условий:

- оценка за п. 1 должна быть не менее 3 баллов
- оценка за п. 4 должна быть не менее 3 баллов
- оценка за п. 5 должна быть больше 0
- ~~суммарная оценка за работу без учета п. 6 должна быть не менее 17 баллов~~