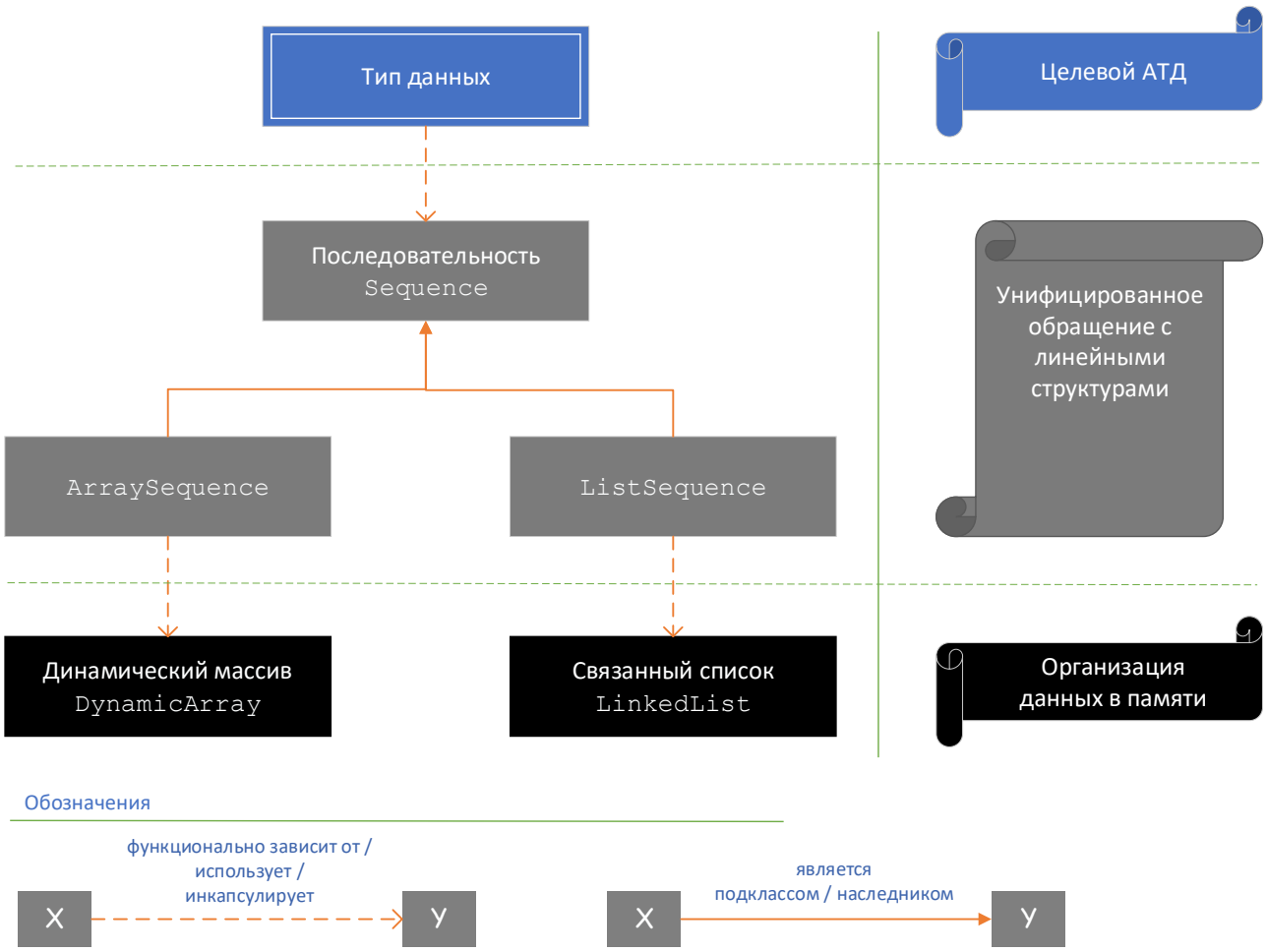
Лабораторная работа №2

по курсу информатики, 2 семестр

Варианты заданий

1. Постановка задачи

Написать на языке C++ реализацию полиморфного абстрактного типа данных – с помощью нескольких уровней абстракции. На нижем уровне реализуются структуры для организации данных в памяти – динамический массив и связанный список. Уровнем выше располагается абстракция (АТД[1] последовательность), обеспечивающая возможности унифицированной работы со всеми структурами нижнего уровня. Целевой АТД, реализация которого и является конечной целью задания, располагается на самом верхнем уровне и использует АТД последовательность для реализации большинства операций, см. схему на рис. ниже.



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

[1] Абстрактный Тип Данных

- АТД динамический массив,
- АТД линейный связанный список,
- АТД последовательность,
- целевой АТД, указанный в варианте задания.

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.

Дополнительные требования к программе (на повышение оценки).

- Перегрузка операторов (например, оператора [] для обращения/задания значения элемента по индексу).
- Для векторов, матриц и многочленов – реализовать основные операции с помощью map-reduce (и zip-unzip).
- Реализовать функцию Split, которая данную последовательность элементов разобьет на отдельные фрагменты. Границы между фрагментами – это элементы, удовлетворяющие условию, которое передается как параметр.

2. Требования к структурам данных

2.1. Класс DynamicArray

template <class T> class DynamicArray	
Создание объекта	
DynamicArray(T* items, int count);	Копировать элементы из переданного массива
DynamicArray(int size);	Создать массив заданной длины
DynamicArray(DynamicArray<T> & dynamicArray const);	Копирующий конструктор
Декомпозиция	
T Get(int index); Может выбрасывать исключения: – IndexOutOfRange (если индекс отрицательный, больше/равен числу элементов или указывает на не заданный элемент)	Получить элемент по индексу.
int GetSize();	Получить размер массива
Операции	
void Set(int index, T value);	Задать элемент по индексу

	Может выбросить IndexOutOfRange
void Resize(int newSize);	Изменить размер массива. Если размер увеличивается, все элементы копируются в начало новой памяти. Если уменьшается – элементы, которые не помещаются, отбрасываются.

2.2. Класс LinkedList

template <class T> class LinkedList	
Создание объекта	
LinkedList (T* items, int count);	Копировать элементы из переданного массива
LinkedList ();	Создать пустой список
LinkedList (LinkedList <T> & list const);	Копирующий конструктор
Декомпозиция	
T GetFirst(); Может выбрасывать исключения: – IndexOutOfRange (если список пуст)	Получить первый элемент в списке
T GetLast(); Может выбрасывать исключения: – IndexOutOfRange (если список пуст)	Получить последний элемент в списке
T Get(int index); Может выбрасывать исключения: – IndexOutOfRange (если индекс отрицательный или больше/равен числу элементов)	Получить элемент по индексу.
LinkedList<T>* GetSubList(int startIndex, int endIndex); Может выбрасывать исключения: – IndexOutOfRange (если хотя бы один из индексов отрицательный или больше/равен числу элементов)	Получить список из всех элементов, начиная с startIndex и заканчивая endIndex.
int GetLength();	Получить длину списка

Операции	
void Append(T item);	Добавляет элемент в начало списка
void Prepend(T item);	Добавляет элемент в конец списка
void InsertAt(T item, int index); Может выбрасывать исключения: – IndexOutOfRange (если индекс отрицательный или больше/равен числу элементов)	Вставляет элемент в заданную позицию
LinkedList<T>* Concat(LinkedList<T> *list);	Сцепляет два списка

2.3. Класс Sequence

template <class T> class Sequence template <class T> class ArraySequence : Sequence<T> template <class T> class LinkedListSequence : Sequence<T>	
Создание объекта	
ArraySequence (T* items, int count); LinkedListSequence (T* items, int count);	Копировать элементы из переданного массива
ArraySequence (); LinkedListSequence ();	Создать пустой список
ArraySequence (LinkedList <T> & list const); LinkedListSequence (LinkedList <T> & list const);	Копирующий конструктор
Декомпозиция	
T GetFirst(); Может выбрасывать исключения: – IndexOutOfRange (если список пуст)	Получить первый элемент в списке
T GetLast(); Может выбрасывать исключения: – IndexOutOfRange (если список пуст)	Получить последний элемент в списке
T Get(int index);	Получить элемент по индексу.

Может выбрасывать исключения: – IndexOutOfRange (если индекс отрицательный или больше/равен числу элементов)	
Sequence<T>* GetSubsequence(int startIndex, int endIndex); Может выбрасывать исключения: – IndexOutOfRange (если хотя бы один из индексов отрицательный или больше/равен числу элементов)	Получить список из всех элементов, начиная с startIndex и заканчивая endIndex.
int GetLength();	Получить длину списка
Операции	
void Append(T item);	Добавляет элемент в начало списка
void Prepend(T item);	Добавляет элемент в конец списка
void InsertAt(T item, int index); Может выбрасывать исключения: – IndexOutOfRange (если индекс отрицательный или больше/равен числу элементов)	Вставляет элемент в заданную позицию
Sequence <T>* Concat(Sequence <T> *list);	Сцепляет два списка

Кроме того, здесь же оптимально реализовывать операции из серии map-reduce: map, reduce, zip, unzip, where (и др.).

Примерная схема реализации – на примере класса ArraySequence:

```
      // Упрощенная реализация, без «умного» управления буфером.
template <class T>
class ArraySequence : Sequence<T>
{
   private:
      DynamicArray<T>* items;
      //…
   public:
      //…
      int GetLength()
      {
            return this->items->GetSize();
      }
      //…
```

```
	void Append(T item)
	{
		this->items->Resize(this->items->GetSize()+1);
		this->items->Set(this->items->GetSize()-1, item);
	}
};
```

Т.е. для реализации используется инкапсуляция DynamicArray и делегирование ему большей части работы.

3. Содержание вариантов

Таблица 1. Список целевых АТД			
№ варианта	**Тип коллекции**	**Типы хранимых элементов**	**Операции**
1.	Очередь	–Целые числа –Вещественные числа –Комплексные числа –Строки –Функции[2)] –Студенты[3)] –Преподаватели[3)]	–map, where, reduce[1)] –Конкатенация –Извлечение подпоследовательности (по заданным индексам) –Поиск на вхождение подпоследовательности ~~–Слияние~~ ~~–Разделение (по заданному признаку)~~
2.	Стек		–map, where, reduce –Конкатенация –Извлечение подпоследовательности (по заданным индексам) –Поиск на вхождение подпоследовательности ~~–Слияние~~ ~~–Разделение (по заданному признаку)~~
3.	Дек		–Сортировка –map, where, reduce –Конкатенация –Извлечение подпоследовательности (по заданным индексам) –Поиск на вхождение подпоследовательности ~~–Слияние~~ ~~–Разделение (по заданному признаку)~~

Производные типы данных:			
4.	Вектор	Коэффициенты: −Целые числа −Вещественные числа −Комплексные числа	Сложение, умножение на скаляр, вычисление нормы, скалярное произведение
5.	Квадратная матрица		Сложение, умножение на скаляр, вычисление нормы, элементарные преобразования строк/столбцов
6.	Прямоугольная матрица		Сложение, умножение на скаляр, вычисление нормы, элементарные преобразования строк/столбцов
7.	Треугольная матрица		Сложение, умножение на скаляр, вычисление нормы
8.	Диагональная матрица[4)]		
9.	Многочлен[5)]	Коэффициенты: −Целые числа −Вещественные числа −Комплексные числа	Сложение, умножение, умножение на скаляр, вычисление значения для заданного значения аргумента, композиция
10.	«Линейная форма»[6)]	Коэффициенты: −Целые числа −Вещественные числа −Комплексные числа	Сложение (и вычитание), умножение на скаляр, вычисление значения при заданных значениях аргументов
11.	Очередь с приоритетами	−Целые числа −Вещественные числа −Комплексные числа −Строки/символы −Функции −Студенты −Преподаватели	−map, where, reduce −Конкатенация −Извлечение подпоследовательности (по заданным индексам) −Поиск на вхождение подпоследовательности −Слияние −Разделение (по заданному признаку)
12.	Поток (stream) данных	−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)
13.	Множество	−Целые числа −Вещественные числа	−map, where −объединение

		–Комплексные числа –Строки/символы –Функции –Студенты –Преподаватели –Пары объектов	–пересечение –вычитание –проверка на включение подмножества –проверка на вхождение элемента –сравнение (равенство) двух множеств

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:

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

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 birthDate;
    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[2] 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 = oerson->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.

5) Многочлен степени n записывается в вде: $P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \dots + a_1 x + a_0$ и может быть однозначно задан списком своих коэффициентов $a_0, \dots, , a_n$. 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: $F_n(x_1, \dots, x_n) = a_0 + a_1 x_1 + \dots + a_n x_n$.

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

Таблица 2. Содержание вариантов заданий			
№ варианта	**Тип коллекции**	**Типы хранимых элементов**	**Примечания**[*)]
1.	Очередь	–Целые числа –Вещественные числа –Строки	
2.	Стек	–Строки –Функции	
3.	Дек	–Вещественные числа –Комплексные числа –Строки	Без сортировки
4.	Вектор	–Целые числа –Вещественные числа	
5.	Квадратная матрица	–Целые числа –Вещественные числа	
6.	Прямоугольная матрица	–Вещественные числа –Комплексные числа	Без элементарных преобразований
7.	Треугольная матрица	–Целые числа –Вещественные числа	
8.	Диагональная матрица[4)]	–Целые числа –Вещественные числа	
9.	Многочлен[5)]	–Вещественные числа –Комплексные числа	
10.	«Линейная форма»[6)]	–Комплексные числа –Вещественные числа	
11.	Очередь с приоритетами	–Целые числа –Студенты –Преподаватели	Без поиска подпоследовательности
12.	Поток (stream) данных	–Целые числа –Вещественные числа –Комплексные числа	
13.	Множество	–Функции –Студенты –Преподаватели	
14.	Очередь	–Целые числа –Комплексные числа –Строки	
15.	Стек	–Функции –Студенты –Преподаватели	Без поиска подпоследовательности
16.	Дек	–Вещественные числа –Комплексные числа –Строки	Без поиска подпоследовательности
17.	Вектор	–Вещественные числа –Комплексные числа	
18.	Квадратная матрица	–Вещественные числа	Без элементарных преобразований

		–Комплексные числа	
19.	Прямоугольная матрица	–Целые числа –Вещественные числа	
20.	Треугольная матрица	–Целые числа –Комплексные числа	
21.	Диагональная матрица[4)]	–Целые числа –Комплексные числа	
22.	Многочлен[5)]	–Вещественные числа –Комплексные числа	
23.	«Линейная форма»[6)]	–Вещественные числа –Комплексные числа	
24.	Очередь с приоритетами	–Студенты –Преподаватели	
25.	Поток (stream) данных	–Целые числа –Функции –Студенты	Без поиска подпоследовательности
26.	Множество	–Функции –Студенты –Преподаватели	
27.	Очередь	–Комплексные числа –Студенты –Преподаватели	Без поиска подпоследовательности
28.	Стек	–Строки –Студенты –Преподаватели	Без поиска подпоследовательности
29.	Дек	–Функции –Студенты –Преподаватели	Без поиска подпоследовательности
30.	Вектор	–Вещественные числа –Комплексные числа	
Задания повышенной сложности			
31.	Словарь слов (список)	–Символы –Строки	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.

32.	«Ханойская башня»	Написать программу, решающую задачу о «ханойской башне»[3]. 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.	
33.	Интерпретатор регулярных выражений для полиморфных последовательностей	Исходными являются две строки, представленные список символов. Первая строка – регулярное выражение[4]. Определить, соответствует ли ему вторая строка.	
34.	«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 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: $[(t_2,1)(t_2,5)(t_2,7)(t_3,2)(t_3,3)(t_3,6)(t_1,4)(t_1,8)]$.	
35.	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.	
36.			

*) Если ничего не указано, то реализуется весь список операций, указанных в таблице 1.

4. Критерии оценки

1.	Качество программного кода:	– стиль (в т.ч.: имена, отступы и проч.) (0-2) – структурированность (напр. декомпозиция сложных функций	0-7 баллов

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

		на более простые) (0-2) – качество основных и второстепенных алгоритмов (напр. обработка граничных случаев и некорректных исходных данных и т.п.) (0-3)	
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 некорректные значения, и др.)	0-5 баллов
4.	Полнота выполнения задания и качество ТЗ	Оценивается качество подготовки ТЗ, полнота выполнений минимальных требований	0-5 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	0-33 баллов

Для получения зачета за выполнения 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