Лабораторная работа №1

по курсу информатики, 2 семестр

Варианты заданий

Постановка задачи

Написать на языке C реализацию абстрактного типа данных – полиморфной коллекции на основе динамического массива. Написать программу-оболочку для тестирования этой реализации. Под полиморфным, в данном случае, подразумевается такой массив, который может хранить (и обрабатывать) значения различных типов. Что конкретно понимается под обработкой – зависит от конкретного варианта задания.

Минимальные требования к программе. В программе, в зависимости от варианта, требуется реализовать одну из следующих структур данных: вектор, квадратная матрица, прямоугольная матрица, список (или один из производных от них). Для реализации необходимо использовать концепцию динамических массивов. Структура данных должна поддерживать работу с элементами различных типов: вообще говоря, произвольных, если они удовлетворяют некоторым условиям, которые зависят от специфики задачи. Т.е. система должна быть «открытой»: должно быть можно использовать любой тип, для которого предоставлены реализации необходимых операций.

Основные алгоритмы необходимо покрыть (модульными) тестами. Реализацию следует оснастить пользовательским интерфейсом (консольным) для проверки корректности реализации.

Содержание вариантов

Типы-контейнеры:

№	Тип коллекции	Примеры типов хранимых элементов	Операции
1.	Вектор	–Целые числа –Вещественные числа –Комплексные числа	–Векторное сложение, скалярное произведение
2.	Квадратная матрица	–Целые числа –Вещественные числа –Комплексные числа –…	–Матричное сложение и умножение, умножение на скаляр, прибавление к строке линейной комбинации других строк[1)]
3.	Прямоугольная матрица	–Целые числа –Вещественные числа –Комплексные числа	–Матричное сложение и умножение, транспонирование, прибавление к строке линейной комбинации других строк
4.	Динамический массив	–Целые числа –Вещественные числа	–Сортировка –map, where, reduce[2)]

		–Комплексные числа –Строки –Функции[3)] –Студенты[4)] –Преподаватели[4)]	–Конкатенация
Производные типы данных:			
5.	Многочлен[5)]	Коэффициенты: –Целые числа –Вещественные числа –Комплексные числа	Сложение, умножение, умножение на скаляр, вычисление значения для заданного значения аргумента, композиция
6.	Строка	Символы	Конкатенация, получение подстроки (с i-го символа по j-й), поиск подстроки (сравнение может быть чувствительным к регистру, так и нет), перекодирование строки[6)]
7.	«Линейная форма»[6)]	Коэффициенты: –Целые числа –Вещественные числа –Комплексные числа	Сложение (и вычитание), умножение на скаляр, вычисление значения при заданных значениях аргументов

1) Т.е. функцию, которая в качестве аргумента принимает номер i строки, к которой требуется прибавить линейную комбинацию, и список коэффициентов $\beta_1, \dots, \beta_m$ для остальных строк. Если матрица $A = (a_{ij}) =$ имеет размерность $m \times n$, то результатом действия функции будет новая матрица, у которой i-я строка имеет вид: $\overline{a}_i + \sum_k \beta_k \overline{a}_k$. Пример сигнатуры функции:

```
Matrix* AddLinearCombination(Matrix* matrix, int rowIndex, void* alhpas)
```

Тип `void*` у списка коэффициентов `alphas` обусловлен тем, что, в общем случае, эти коэффициенты могут быть любого числового типа (целое, вещественное, комплексное – в зависимости от того, что указано в варианте задания).

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

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

$$\text{map}\left(\left(f \to f(1)\right), [f_1, \dots, f_n]\right) \mapsto [f_1(1), \dots, f_n(1)]$$

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.

$$\text{where } h\ l \mapsto l', \text{где } a \in l' \Leftrightarrow a \in l \wedge h(a) = \text{true}$$

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:

$$reduce(f, [1,2,3], 4) = f\left(3, f\left(2, f(1,4)\right)\right) =$$
$$= 2 \cdot 3 + 3\left(2 \cdot 2 + 3(2 \cdot 1 + 3 \cdot 4)\right) =$$
$$= 2 \cdot 3 + 3(2 \cdot 2 + 3 \cdot 14) = 2 \cdot 3 + 3 \cdot 46 = 144$$

3) 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
```

4) Точнее, описывающие их структуры. Персона характеризуется набором атрибутов, таких как 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:

```
struct Person {
      Person_ID id;
      char* firstName;
      char* middleName;
      char* lastName;
      time_t birthDate;
}
```

Тип Person_ID 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 {
      int series;       // как вариант, char*
      int number;       // как вариант, char*
}
```

Для получения значения атрибутов предусматривают соответствующие функции, например:

```
char* name = getName(person); //  = “Иван”
```

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

```
char* fullName = getFullName(person); // = "Иван Иванович Иванов", вычислимый атрибут
```

5) Многочлен степени n записывается в вде: $P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ 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) Перекодирование состоит в замене каждого символа на другой, получаемый с помощью функции кодирования, которая передается в качестве аргумента.

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

№ варианта	Тип коллекции	Типы хранимых элементов	Операции
1.	Вектор	–Целые числа –Вещественные числа	Векторное сложение, скалярное произведение
2.	Динамический массив	–Целые числа –Вещественные числа	–Сортировка –map, where –Конкатенация
3.	Квадратная матрица	–Целые числа –Вещественные числа	Матричное сложение и умножение, умножение на скаляр, прибавление к строке линейной комбинации других строк
4.	Прямоугольная матрица	–Целые числа –Вещественные числа	Матричное сложение и умножение, транспонирование, прибавление к строке линейной комбинации других строк
5.	Динамический массив	–Целые числа –Комплексные числа	–map, where –Конкатенация
6.	Многочлен	Коэффициенты –Целые числа –Вещественные числа	Сложение, умножение, умножение на скаляр, вычисление значения для заданного значения аргумента, композиция
7.	Строка	Символы	Конкатенация, получение подстроки (с i-го символа по j-й), поиск подстроки (реализовать два варианта: чувствительное к регистру сравнение, и нечувствительное)
8.	Динамический массив	–Вещественные числа –Комплексные числа	–Сортировка –map, where –Конкатенация

№ варианта	Тип коллекции	Типы хранимых элементов	Операции
9.	«Линейная форма»	Коэффициенты: –Целые числа –Вещественные числа	Сложение (и вычитание), умножение на скаляр, вычисление значения при заданных значениях аргументов
10.	3-мерный вектор	–Вещественные числа –Комплексные числа	Векторное сложение, скалярное произведение, векторное произведение
11.	Динамический массив	–Целые числа –Строки	–Сортировка –map, where –Конкатенация
12.	Квадратная матрица	–Вещественные числа –Комплексные числа	Матричное сложение и умножение, умножение на скаляр
13.	Прямоугольная матрица	–Вещественные числа –Комплексные числа	Матричное сложение и умножение, транспонирование
14.	Динамический массив	–Вещественные числа –Функции	–map, where –Конкатенация
15.	Многочлен	Коэффициенты: –Целые числа –Комплексные числа	Сложение, умножение, умножение на скаляр, вычисление значения для заданного значения аргумента
16.	Строка	Символы	Конкатенация, получение подстроки (с i-го символа по j-й), перекодирование строки
17.	Динамический массив	–Студенты –Преподаватели	–map, where –Конкатенация
18.	«Линейная форма»	Коэффициенты: –Вещественные числа –Комплексные числа	Сложение (и вычитание), умножение на скаляр, вычисление значения при заданных значениях аргументов
19.	Вектор	–Вещественные числа –Комплексные числа	Векторное сложение, скалярное произведение
20.	Динамический массив	–Строки –Функции	–map, where –Конкатенация
21.	Квадратная матрица	–Целые числа –Комплексные числа	Матричное сложение и умножение, умножение на скаляр
22.	Прямоугольная матрица	–Целые числа –Комплексные числа	Матричное сложение и умножение, транспонирование
23.	Динамический массив	–Строки –Функции	–map, where, reduce –Конкатенация

№ варианта	Тип коллекции	Типы хранимых элементов	Операции
24.	Многочлен	Коэффициенты: –Вещественные числа –Комплексные числа	Сложение, умножение, умножение на скаляр, вычисление значения для заданного значения аргумента
25.	Строка	–Символы –Строки	Конкатенация, получение элементов с i-го по j-й (подмассив), разбиение строки на слова
26.	Динамический массив	–Вещественные числа –Строки	–Сортировка –map, where –Конкатенация
27.	«Линейная форма»	Коэффициенты: –Вещественные числа –Комплексные числа	Сложение (и вычитание), умножение на скаляр, вычисление значения при заданных значениях аргументов
28.	Вектор	–Целые числа –Комплексные числа	Векторное сложение, скалярное произведение
29.	Строка	Символы: строка содержит запись какого-либо из следующих чисел: целого вещественного или комплексного (char)	Конкатенация, получение подстроки (с i-го символа по j-й), преобразование в/из числа.
30.	Квадратная матрица (трехленточная[2])	–Вещественные числа –Комплексные числа	Матричное сложение и умножение, умножение на скаляр
Задания повышенной сложности			
31.	Динамический массив, строка	–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.			

[2] Т.е. от нуля отличны только элементы, стоящие на главной диагонали, а также на диагоналях непосредственно над и под ней.

Методические указания к реализации АТД на С

Обеспечение согласованности типов элементов, содержащихся в коллекции, т.е. как-то гарантировать, чтобы нельзя было "смешивать" коллекции с элементами разных типов, например, сложить Vector<T1> с Vector<T2>. В случае использования структуры типа FieldInfo, это реализовать можно следующим образом.

В каком-нибудь "модуле" (модулем условно называем пару файлов <name>.h и <name>.c) создаем (желательно, скрытую) переменную, например, INT_FIELD_INFO (для ее нужно определить в .c-файле и сделать ее типом FiedlInfo*). Далее, в .h декларируем функцию GetIntFieldInfo. Реализуем эту функцию в .h, в этой функции сначала проверяем, что переменная INT_FIELD_INFO == NULL. Если это так, то создаем FieldInfo для (в данном случае) целых и присваиваем в INT_FIELD_INFO. Возвращаем результат. Если INT_FIELD_INFO != NULL, то просто возвращаем результат. Это, кстати, называется ленивой инициализией с мемоизацией.

В результате получается, что для каждого типа T, которым мы будем пользоваться, у нас единственный описывающий его объект FieldInfo (это кстати называется объект-как-глобальная-константа). И тогда при сложении двух векторов мы можем просто проверять равенство соответствующих указателей на FieldInfo.

Альтернативно, вместо объекта-глобальной константы, можно реализовать функцию сравнения для FieldInfo, которая будет сравнивать значения всех полей (т.е. всех указателей на функции, а также размер элемента).

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.) (0-2)	0-6 баллов
2.	Качество тестов	– степень покрытия – читаемость – качество проверки (граничные и некорректные значения, и др.)	0-5 баллов
3.	Полнота выполнения задания	Оценивается в целом соответствие предложенной реализации требованиям, самостоятельность выполнения задания.	0-3 баллов
4.	Владение теорией	– понимание алгоритмов работы с динамическими массивами – понимание арифметики указателей – владение методикой работы с void*	0-3 баллов
5.	Обработка ошибок	– Простота идентификации ошибки	0-3

		– Простота локализации ошибки – Простота и полнота расшифровки содержания/причины ошибки	баллов
		Итого	0-20 баллов
Бонусные задачи			
6.	Качество пользовательского интерфейса:	– предоставляемые им возможности – наличие ручного/автоматического ввода исходных данных – настройка параметров для автоматического режима – отображение исходных данных и промежуточных и конечных результатов и др.	0-5 баллов
7.	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

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

- основные структуры данных и алгоритмы должны быть реализованы
- для основных алгоритмов должны быть написаны тесты
- должна быть реализована хотя бы минимальная обработка ошибок