Фонд оценочных средств  
по дисциплине  
«Информатика (информатика и программирование, архитектура вычислительных систем, проектирование человеко-машинного интерфейса)»

Приложение 1

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 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).

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

Содержание вариантов

Типы-контейнеры:

|  |  |  |  |
| --- | --- | --- | --- |
| **№ варианта** | **Тип коллекции** | **Типы хранимых элементов** | **Операции** |
|  | Вектор | * Целые числа * Вещественные числа * Комплексные числа | * Векторное сложение, скалярное произведение |
|  | Квадратная матрица | * Целые числа * Вещественные числа * Комплексные числа | * Матричное сложение и умножение, умножение на скаляр, прибавление к строке линейной комбинации других строк1) |
|  | Прямоугольная матрица | * Целые числа * Вещественные числа * Комплексные числа | * Матричное сложение и умножение, транспонирование, прибавление к строке линейной комбинации других строк |
|  | Динамический массив | * Целые числа * Вещественные числа * Комплексные числа * Строки * Функции3) * Студенты4) * Преподаватели4) | * Сортировка * map, where, reduce2) * Конкатенация |
| Производные типы данных: | | | |
|  | Многочлен5) | Коэффициенты:   * Целые числа * Вещественные числа * Комплексные числа | Сложение, умножение, умножение на скаляр, вычисление значения для заданного значения аргумента, композиция |
|  | Строка | Символы | Конкатенация, получение подстроки (с i-го символа по j-й), поиск подстроки (сравнение может быть чувствительным к регистру, так и нет), перекодирование строки6) |
|  | «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 строки, к которой требуется прибавить линейную комбинацию, и список коэффициентов для остальных строк. Если матрица имеет размерность , то результатом действия функции будет новая матрица, у которой -я строка имеет вид: . 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 любого числового типа (целое, вещественное, комплексное – в зависимости от того, что указано в варианте задания).

**2)** Если – некоторый список элементов типа , а , то:

Если, при тех же соглашениях, – некоторая функция, возвращающая булево значение, то результатом будет новый список , такой что: . Т.е. where фильтрует значения из списка с помощью функции-фильтра .

Функция reduce работает несколько иначе: «сворачивает» список в одно значение по заданному правилу :

где – константа, «стартовое» значение. Например, , , тогда:

**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 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]](#footnote-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”

char\* fullName = getFullName(person); // = “Иван Иванович Иванов”, вычислимый атрибут

**5)** Многочлен степени записывается в вде: и может быть однозначно задан списком своих коэффициентов . Многочлен является функцией, на множестве функций определена ассоциативная операция – композиция : .

**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 переменных: .

|  |  |  |  |
| --- | --- | --- | --- |
| **№ варианта** | **Тип коллекции** | **Типы хранимых элементов** | **Операции** |
|  | Вектор | * Целые числа * Вещественные числа | Векторное сложение, скалярное произведение |
|  | Динамический массив | * Целые числа * Вещественные числа | * Сортировка * map, where * Конкатенация |
|  | Квадратная матрица | * Целые числа * Вещественные числа | Матричное сложение и умножение, умножение на скаляр, прибавление к строке линейной комбинации других строк |
|  | Прямоугольная матрица | * Целые числа * Вещественные числа | Матричное сложение и умножение, транспонирование, прибавление к строке линейной комбинации других строк |
|  | Динамический массив | * Целые числа * Комплексные числа | * map, where * Конкатенация |
|  | Многочлен | Коэффициенты   * Целые числа * Вещественные числа | Сложение, умножение, умножение на скаляр, вычисление значения для заданного значения аргумента, композиция |
|  | Строка | Символы | Конкатенация, получение подстроки (с i-го символа по j-й), поиск подстроки (реализовать два варианта: чувствительное к регистру сравнение, и нечувствительное) |
|  | Динамический массив | * Вещественные числа * Комплексные числа | * Сортировка * map, where * Конкатенация |
|  | «Линейная форма» | Коэффициенты:   * Целые числа * Вещественные числа | Сложение (и вычитание), умножение на скаляр, вычисление значения при заданных значениях аргументов |
|  | 3-мерный вектор | * Вещественные числа * Комплексные числа | Векторное сложение, скалярное произведение, векторное произведение |
|  | Динамический массив | * Целые числа * Строки | * Сортировка * map, where * Конкатенация |
|  | Квадратная матрица | * Вещественные числа * Комплексные числа | Матричное сложение и умножение, умножение на скаляр |
|  | Прямоугольная матрица | * Вещественные числа * Комплексные числа | Матричное сложение и умножение, транспонирование |
|  | Динамический массив | * Вещественные числа * Функции | * map, where * Конкатенация |
|  | Многочлен | Коэффициенты:   * Целые числа * Комплексные числа | Сложение, умножение, умножение на скаляр, вычисление значения для заданного значения аргумента |
|  | Строка | Символы | Конкатенация, получение подстроки (с i-го символа по j-й), перекодирование строки |
|  | Динамический массив | * Студенты * Преподаватели | * map, where * Конкатенация |
|  | «Линейная форма» | Коэффициенты:   * Вещественные числа * Комплексные числа | Сложение (и вычитание), умножение на скаляр, вычисление значения при заданных значениях аргументов |
|  | Вектор | * Вещественные числа * Комплексные числа | Векторное сложение, скалярное произведение |
|  | Динамический массив | * Строки * Функции | * map, where * Конкатенация |
|  | Квадратная матрица | * Целые числа * Комплексные числа | Матричное сложение и умножение, умножение на скаляр |
|  | Прямоугольная матрица | * Целые числа * Комплексные числа | Матричное сложение и умножение, транспонирование |
|  | Динамический массив | * Строки * Функции | * map, where, reduce * Конкатенация |
|  | Многочлен | Коэффициенты:   * Вещественные числа * Комплексные числа | Сложение, умножение, умножение на скаляр, вычисление значения для заданного значения аргумента |
|  | Строка | * Символы * Строки | Конкатенация, получение элементов с i-го по j-й (подмассив), разбиение строки на слова |
|  | Динамический массив | * Вещественные числа * Строки | * Сортировка * map, where * Конкатенация |
|  | «Линейная форма» | Коэффициенты:   * Вещественные числа * Комплексные числа | Сложение (и вычитание), умножение на скаляр, вычисление значения при заданных значениях аргументов |
|  | Вектор | * Целые числа * Комплексные числа | Векторное сложение, скалярное произведение |
|  | Строка | Символы: строка содержит запись какого-либо из следующих чисел: целого вещественного или комплексного | Конкатенация, получение подстроки (с i-го символа по j-й), преобразование в/из числа. |
|  | Квадратная матрица (трехленточная[[2]](#footnote-2)) | * Вещественные числа * Комплексные числа | Матричное сложение и умножение, умножение на скаляр |
| Задания повышенной сложности | | | |
|  | 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. |
|  |  |  |  |

Критерии оценки

|  |  |  |  |
| --- | --- | --- | --- |
|  | 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) | 0-5  баллов |
|  | 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) | 0-5  баллов |
|  | Качество тестов | * степень покрытия * читаемость * качество проверки (граничные и некорректные значения, и др.) | 0-5  баллов |
|  | Полнота выполнения задания и качество ТЗ | Оценивается качество подготовки ТЗ, полнота выполнений минимальных требований | 0-5  баллов |
|  | Владение теорией | знание алгоритмов, области их применимости, умение сравнивать с аналогами, оценить сложность, корректность реализации | 0-3  баллов |
|  | 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 |

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

1. См. например: <https://en.wikipedia.org/wiki/Null-terminated_string>. Идея такая, что конец строки определяется по наличию символа с кодом 0. [↑](#footnote-ref-1)
2. Т.е. от нуля отличны только элементы, стоящие на главной диагонали, а также на диагоналях непосредственно над и под ней. [↑](#footnote-ref-2)