Лабораторная работа №2

по дисциплине «Программирование на Си»

Обработка одномерных статических массивов

Кострицкий А. С., Ломовской И. В.

Москва — 2021 — TS2103081344

Содержание

1 Цель работы

Целью лабораторной работы является знакомство студентов со статическими одномерными массивами, адресной арифметикой и классическими алгоритмами программирования, такими как поиск минимума и максимума, накопление суммы, накопление произведения, вставка и удаление элементов в массиве, сортировка и др.

1. описывать одномерные статические массивы;
2. вводить и выводить одномерные массивы;
3. обрабатывать одномерные массивы;
4. передавать одномерные массивы в функции;
5. использовать адресную арифметику для обработки одномерных массивов;
6. измерять время выполнения отдельной функции;
7. выполнять профилирование программы.

2 Общее задание

В каждой задаче реализуйте программу, которая принимает у пользователя целочисленный статический массив, и выполняет его обработку в соответствии с вариантом. Максимальное количество элементов, которое может ввести пользователь, равно десяти.

2.1 Задача №1

Варианты

0. Найти и вывести на экран сумму чётных элементов массива.
1. Найти и вывести на экран произведение нечётных элементов массива.
2. Найти и вывести на экран среднее арифметическое отрицательных элементов массива.
3. Найти и вывести на экран среднее геометрическое положительных элементов массива.

2.2 Задача №2

Варианты

0. Сформировать и вывести на экран новый массив, в который скопировать элементы исходного массива, которые больше среднего арифметического его элементов.
1. Сформировать и вывести на экран новый массив, в который скопировать элементы исходного массива, которые являются простыми числами.
2. Сформировать и вывести на экран новый массив, в который скопировать элементы исходного массива, которые начинаются и заканчиваются на одну и ту же цифру.
3. Сформировать и вывести на экран новый массив, в который скопировать элементы исходного массива, которые являются числами Армстронга.

2.3 Задача №3

Варианты

0. Удалить из исходного массива все элементы, которые являются числами-палиндромами.
1. Вставить в исходный массив после каждого элемента, кратного трём, очередное число Фибоначчи. Рекурсивных функций не использовать, положить $\mathrm{Fib}_0 = 0$, $\mathrm{Fib}_1 = 1$.
2. Удалить из исходного массива все элементы, которые являются полными квадратами.
3. Вставить в исходный массив после каждого положительного элемента реверс этого же элемента.

2.4 Задача №4

В четвёртой задаче необходимо организовать ввод массива *по концевому признаку*. В качестве концевого признака выступает любая ошибка, которая возникает при очередном вводе значения с помощью функции `scanf`. Например, при вводе последовательности

```
1 2 3 a
```

должен быть сформирован массив из трёх элементов со значениями

$$A[0] = 1,\ A[1] = 2,\ A[2] = 3.$$

В случае, если массив уже заполнен, а концевой признак ещё не наступил, ввод элементов в массив прекращается. Такое «переполнение» не считается ошибкой: программа должна обработать полученный массив из 10 элементов, но функция `main` в такой ситуации должна вернуть специальный код ошибки, равный 100.

Варианты

0. Упорядочить массив по возрастанию с помощью сортировки *пузырьком* (англ. *Bubble Sort*) и вывести на экран.

1. Упорядочить массив по возрастанию с помощью сортировки *вставками* (англ. *Insertion Sort*) и вывести на экран.

2. Упорядочить массив по возрастанию с помощью сортировки *выбором* (англ. *Selection Sort*) и вывести на экран.

2.5 Задача №5

При решении пятой задачи в методических целях запрещено использовать выражения вида `a[i]` и вообще квадратные скобки. Вместо указанного выражения используется выражение `*pa`, где `pa` — указатель на элемент массива с индексом i (именно на i-ый элемент, а не выражение вида `*(pa + i)`). Также нельзя передавать как аргумент размер массива в элементах.

Вместо этого предлагается использовать пару указателей: на первый (нулевой) элемент массива и на элемент массива, расположенный за последним. Ситуация, когда эти указатели совпадают, означает пустоту обрабатываемого массива.

Варианты

0. Вычислить и вывести на экран значение

$$\max(A[0] + A[n-1],\ A[1] + A[n-2],\ A[2] + A[n-3],\ \ldots,\ A[(n-1)/2] + A[n/2]),$$

   где n — размер массива.

1. Вычислить и вывести на экран значение

$$X[0] \cdot Y[0] + X[1] \cdot Y[1] + \cdots + X[k] \cdot Y[k],$$

   где n — размер массива, X — отрицательные элементы массива в порядке следования, Y — положительные элементы массива в обратном порядке, $k = \min(p, q)$, p — количество положительных элементов, q — количество отрицательных элементов.

2. Вычислить и вывести на экран значение

$$A[0] + A[0] \cdot A[1] + A[0] \cdot A[1] \cdot A[2] + \cdots + (A[0] \cdot A[1] \cdot A[2] \cdot \cdots \cdot A[m]),$$

где n — размер массива, m — либо индекс первого отрицательного элемента, либо значение $n - 1$, если в массиве отрицательных элементов нет.

3. Вычислить и вывести на экран значение

$$\min\left(A[0] \cdot A[1],\ A[1] \cdot A[2],\ A[2] \cdot A[3],\ \ldots,\ A[n-3] \cdot A[n-2],\ A[n-2] \cdot A[n-1]\right),$$

где n — размер массива.

4. Найти и вывести на экран количество уникальных чисел в массиве.

2.6 Задача №6

На основе пятой задачи проведите сравнение производительности разных способов работы с элементами массива:

1. использование операции индексации `a[i]`;
2. формальная замена операции индексации на выражение `*(a + i)`;
3. использование указателей для работы с массивом.

Для этого:

1. реализуйте функцию, которая выполняет указанное в пятой задаче преобразование массива, используя операцию индексации и количество элементов массива (пусть эта функция называется `process_1`);
2. на основе функции `process_1` получить функцию `process_2`, выполнив замену `a[i]` на `*(a + i)`;
3. реализуйте программу для проведения измерений (её алгоритм приведён ниже);
4. проведите замеры времени.

Алгоритм выполнения измерений:

1. Сформировать случайный целочисленный массив `a` из `na` элементов.
2. `sum = 0`
3. в цикле от `1` до `N`
   (a) `b = a, nb = na` // скопируем массив a в массив b
   (b) `start = get_time` // засечем время начала интересующего действия
   (c) `process_i(b, nb, ...)` // засечем время окончания интересующего действия
   (d) `end = get_time`
   (e) `sum += (end - start)`
4. `time = sum / N`

Обычно из `sum` исключают минимальное и максимальное времена, делить в этом случае нужно на `(N - 2)`.

Для замера времени мы будем использовать функцию POSIX `gettimeofday`, которая возвращает число секунд и микросекунд с 1 января 1970 года. Пример использования этой функции приведён ниже.

```
#include <stdio.h>
#include <inttypes.h>
#include <sys/time.h>
...
int main(void)
{
...
    struct timeval tv_start, tv_stop;
    int64_t elapsed_time;
...
    gettimeofday(&tv_start, NULL);
    arr_sort(a, n);
    gettimeofday(&tv_stop, NULL);
    elapsed_time = (tv_stop.tv_sec - tv_start.tv_sec) * 1000000LL +
                   (tv_stop.tv_usec - tv_start.tv_usec);
    // время в микросекундах
    printf("%" PRId64 " us\n", elapsed_time);
...
}
```

По согласованию с преподавателем, проводящим практические занятия, для замеров времени может использоваться POSIX функция `times`.

В отчёте приведите таблицу со столбцами

1. Количество повторов (`N`)
2. Размер массива
3. Работа с `a[i]`
4. Работа с `*(a + i)`
5. Работа с указателями

Провариируйте количество повторов (десятки, сотни) и размер массива (десятки, сотни, тысячи).

В отчёте помимо результатов измерений приведите ответы на следующие вопросы:

1. На что влияет размер массива?
2. Почему приходится выполнять не один замер, а несколько?
3. Какой способ работы с элементами массива оказался самым производительным? Как Вы объясняете этот результат?

2.7 Задача №7

Прочитайте дополнительные материалы по профилированию (ЛР2_gprof.pdf). Проведите профилирование программы, написанной для решения четвёртой задачи.

Скорее всего, массив из 10 элементов будет обработан очень быстро и результаты профилирования ничего не покажут. Поэтому размер массива нужно увеличить хотя бы до 1000 элементов. Чтобы не вводить такое количество элементов вручную, предлагается использовать перенаправление ввода и вывода. Для этого нужно создать текстовый файл, в котором на первой строке указать размер массива, а на следующих строках — элементы этого массива. Создать такой файл можно, например, с помощью программы на Python. После чего запуск программы нужно выполнять следующим образом

```
app.exe < my_data.txt
```

Проанализируйте полученные данные профилирования. Часть этих данных приведите в отчёте.

В отчёте приведите ответы на следующие задания и вопросы:

1. Совпадают ли ваши представления о времени работы той или иной функции с тем, что вы получили на практике? Если нет, то для какой функции и почему?

2. Увеличьте количество элементов в массиве до 10000, выполните профилирование. Что изменилось? В ответе приведите как сами данные, так и объяснение полученных результатов.

3. Уменьшите количество элементов до 10, выполните профилирование. Что изменилось? В ответе приведите как сами данные, так и объяснение полученных результатов.

4. Изучите, что делают ключи `-O1`, `-O2`, `-O3`. Выполните профилирование вашей программы, увеличив размер массива до 10000 элементов, для каждого из этих ключей. Проанализируйте полученные результаты, сделайте выводы.

2.8 Примечания

1. *Статические массивы* следует отличать от *массивов переменной длины (англ. Variable Length Array, VLA)*. Во избежание случайного использования последних при компиляции программы необходимо указывать ключ `-Wvla`.

2. Для реализации каждой из задач этой лабораторной работы необходимо выделить несколько осмысленных функций. Необходимо предусмотреть обработку ошибочных ситуаций.

3. Под *вводом массива* подразумевается, если не указано иное, ввод количества вводимых элементов и самих элементов по порядку. Ввод неверного количества элементов следует считать исключительной ситуацией. Обратите внимание, что неверно можно указать как меньше, так и больше элементов, чем передаётся.

4. Под *выводом массива* подразумевается вывод его элементов без указания их общего количества.

5. Ситуации, когда решение задачи не может быть получено, следует считать исключительными. Например, если нужно подсчитать количество чётных элементов массива, а таких элементов в массиве нет.

Помните, что в случае возникновения ошибочной ситуации программа должна не только выдавать соответствующее сообщение, но и возвращать ненулевой код возврата.

3 Взаимодействие с системой тестирования

1. Исходный код лабораторной работы размещается студентом в ветви `lab_LL`, а решение каждой из задач — в отдельной папке с названием вида `lab_LL_PP_CC`, где LL — номер лабораторной, PP — номер задачи, CC — вариант студента. Если дана общая задача без вариантов, решение следует сохранять в папке с названием вида `lab_LL_PP`.

   Пример: решения восьми задач седьмого варианта пятой лабораторной размещаются в папках `lab_05_01_07`, `lab_05_02_07`, `lab_05_03_07`, ..., `lab_05_08_07`.

2. Исходный код должен соответствовать оглашённым в начале семестра правилам оформления.

3. Если для решения задачи студентом создаётся отдельный проект в IDE, разрешается поместить под версионный контроль файлы проекта, добавив перед этим необходимые маски в список игнорирования. Старайтесь добавлять маски общего вида. Для каждого проекта должны быть созданы, как минимум, два варианта сборки: `Debug` — с отладочной информацией, и `Release` — без отладочной информации.

4. Для каждой программы ещё до реализации студентом заготавливаются и помещаются под версионный контроль в подпапку `func_tests` функциональные тесты, демонстрирующие её работоспособность.

   Позитивные входные данные следует располагать в файлах вида `pos_TT_in.txt`, выходные — в файлах вида `pos_TT_out.txt`, аргументы командной строки при наличии — в файлах вида `pos_TT_args.txt`, где TT — номер тестового случая.

   Негативные входные данные следует располагать в файлах вида `neg_TT_in.txt`, выходные — в файлах вида `neg_TT_out.txt`, аргументы командной строки при наличии — в файлах вида `neg_TT_args.txt`, где TT — номер тестового случая.

   Разрешается помещать под версионный контроль в подпапку `func_tests` сценарии автоматического прогона функциональных тестов. Если Вы используете при автоматическом прогоне функциональных тестов сравнение строк, не забудьте проверить используемые кодировки. Помните, что UTF-8 и UTF-8(BOM) — две разные кодировки.

   Под версионный контроль в подпапку `func_tests` также помещается файл `readme.md` с описанием в свободной форме содержимого каждого из тестов. Вёрстка файла на языке Markdown обязательной не является, достаточно обычного текста.

   Пример: восемь позитивных и шесть негативных функциональных тестов без дополнительных ключей командной строки должны размещаться в файлах `pos_01_in.txt`, `pos_01_out.txt`, ..., `neg_06_out.txt`. В файле `readme.md` при этом может содержаться следующая информация:

```
# Тесты для лабораторной работы №LL

## Входные данные
Целые a, b, c

## Выходные данные
Целые d, e

## Позитивные тесты:
- 01 - обычный тест;
- 02 - в качестве первого числа нуль;
...
- 08 - все три числа равны.

## Негативные тесты:
- 01 - вместо первого числа идёт буква;
- 02 - вместо второго числа идёт буква;
...
- 06 - вводятся слишком большие числа.
```

5. Если не указано обратное, успешность ввода должна контролироваться. При первом неверном вводе программа должна прекращать работу с ненулевым кодом возврата.

6. Вывод программы может содержать текстовые сообщения и числа. Если не указано обратное, тестовая система анализирует числа в потоке вывода, поэтому они могут быть использованы только для вывода результатов — использовать числа в информационных сообщениях запрещено.

   Пример: сообщение «Input point 1:» будет неверно воспринято тестовой системой, а сообщения «Input point A:» или «Input first point:» — правильно.

   Тестовая система вычленяет из потока вывода числа, обособленные пробельными символами.

   Пример: сообщения «a=1.043» и «a = 1.043.» будут неверно восприняты тестовой системой, а сообщения «a: 1.043» или «a = 1.043» — правильно.

7. Если не указано обратное, числа двойной точности следует выводить, округляя до шестого знака после точки.