**Лабораторная работа № 2**

УКАЗАТЕЛИ, ФУНКЦИИ

1. Цель работы

Освоить правила написания и использования функций в языке СИ. Научиться использовать указатели при обработке массивов данных.

3. Методические указания.

В ходе выполнения лабораторной работы потребуется выполнить сортировку массива строк. Для ускорения этой операции обычно используется дополнительный массив указателей. В этом случае вместо перестановки двух строк с помощью функции ***strcpy()*** используется перестановка указателей обычным присваиванием.

Формирование массива указателей можно совместить с вводом строк.

4. Порядок выполнения работы

4.1. Написать программу сортировки массива строк по вариантам (табл. 2.2.). Ввод данных, сортировку и вывод результатов оформить в виде функций. Входные и выходные параметры функции сортировки указаны в таблице. Входные и выходные параметры функций для ввода-вывода приведены в таблице 2.1.

4.2. Модифицировать программу п.1., применив в функциях передачу параметров и возврат результатов по ссылке (с использованием указателей). Сравнить результаты.

Таблица 2.1

|  |
| --- |
| Прототип функции для ввода строк |
| ***length = inp\_str(char\* string, int maxlen);***  ***// length*** - длина строки  ***// string*** - введенная строка  ***// maxlen*** - максимально возможная длина строки (размерность массива ***string***) |
| Прототип функции для вывода строк |
| ***void out\_str(char\* string, int length, int number);***  ***// string -*** выводимая строка  ***// length -*** длина строки  ***// number*** - номер строки |

Таблица 2.2

| Вари ант | Задание | Входные  параметры | Выходные  Параметры |
| --- | --- | --- | --- |
| 1 | Расположить строки по возрастанию длины | 1. Массив 2. Размерность массива | 1. Количество перестановок 2. Длина меньшей строки |
| 2 | Расположить строки по убыванию длины | 1. Массив 2. Размерность массива | 1. Количество перестановок 2. Длина большей строки |
| 3 | Расположить строки в алфавитном порядке | 1. Массив 2. Размерность массива | 1. Количество перестановок  2. Первая буква первой строки |
| 4 | Расположить строки в обратном алфавитном порядке | 1. Массив  2. Размерность массива | 1. Количество перестановок 2. Длина первой строки |
| 5 | Расположить строки по возрастанию количества слов | 1. Массив 2. Размерность массива | 1. Количество перестановок  2. Первый символ последней строки |
| 6 | Расположить строки по убыванию количества слов | 1. Массив  2. Размерность массива | 1. Количество перестановок 2. Максимальное количество слов |
| 7 | Расположить строки по возрастанию количества цифр | 1. Массив  2. Размерность массива. | 1. Количество цифр  2. Вторая цифра строки |
| 8 | Расположить строки по убыванию количества цифр | 1. Массив  2. Размерность массива | 1. Количество перестановок  2. Количество цифр |
| 9 | Расположить строки по возрастанию длины первого слова | 1. Массив  2. Размерность массива. | 1. Максимальная длина слова  2. Количество перестановок |
| 10 | Расположить строки по убыванию длины первого слова | 1. Массив  2. Размерность массива. | 1. Минимальная длина слова  2. Последняя буква первого слова |
| 11 | Расположить строки в алфавитном порядке по последней букве строки | 1. Массив  2. Размерность массива | 1. Количество перестановок  2. Последняя буква первой строки |
| 12 | Расположить строки в обратном алфавитном порядке по последней букве строки | 1. Массив  2. Размерность массива | 1. Количество перестановок  2. Последняя буква последней строки |
| 13 | Расположить строки по возрастанию количества пробелов в строке | 1. Массив  2. Размерность массива | 1. Количество перестановок  2. Максимальное количество пробелов в строке |
| 14 | Расположить строки по убыванию количества пробелов в строке | 1. Массив  2. Размерность массива | 1. Количество перестановок  2. Максимальное количество пробелов подряд в строке |
| 15 | Расположить строки по возрастанию длины последнего слова | 1. Массив  2. Размерность массива | 1. Количество перестановок  2. Максимальная длина последнего слова |
| 16 | Расположить строки по убыванию длины последнего слова | 1. Массив  2. Размерность массива | 1. Количество перестановок  2. Минимальная длина последнего слова |
| 17 | Расположить строки по возрастанию длины самого длинного слова | 1. Массив  2. Размерность массива | 1. Количество перестановок  2. Максимальная длина самого длинного слова |
| 18 | Расположить строки по убыванию длины самого длинного слова | 1. Массив  2. Размерность массива | 1. Количество перестановок  2. Минимальная длина самого длинного слова |

**5. Содержание отчета**

5.1. Титульный лист

5.2. Цель работы.

5.3. Задание.

5.4. Текст(ы) программ.

5.5. Полученные результаты.

5.6. Выводы.

**6. Контрольные вопросы**

**6.1. Правила описания указателей.**

Хз че он ожидает в ответе на этот вопрос если честно поэтому навалю базы.

*int zaebalsya = 100;*

Объявление указателя (указатель – адрес на какое-то значение):

*int \*ya = &zaebalsya;*

Амперсанд – взятие адреса.

Чтобы получить значение по указателю – нужно его разыменовать:

*int realno = \*ya; // realno == zaebalsya*

Указатель может указывать на указатель на указатель на указатель...

Неинициализированный указатель может указывать на случайный адрес!

**6.2. Как связаны указатели и массивы?**

Имя массива без индекса является указателем на его первый элемент (под нулевым индексом).

В связи с этим возможен такой сатанизм, как адресная арифметика:

*int arr[5] = {1, 2, 3, 4, 5};*

*int \*ptr = arr;*

*printf("%d\n", \*ptr); // 1 (первый элемент массива)*

*printf("%d\n", \*(ptr + 1)); // 2 (второй элемент массива)*

Их можно и вычитать но это хуйня бесполезная по-моему

*#include <stddef.h>*

*int arr[5] = {1, 2, 3, 4, 5};*

*int \*ptr1 = &arr[1];*

*int \*ptr2 = &arr[3];*

*ptrdiff\_t diff = ptr2 - ptr1;*

*printf("%td\n", diff); // 2 (разница между индексами элементов на ктр указывают ptr1 и ptr2)*

Массивы передаются в функцию по указателю на первый элемент.

**6.3. Назначение прототипа функции.**

Начиная со стандарта языка Си 1999 года неявное определение функции запрещено. Это означает, что перед тем, как вызвать функцию, необходимо ее объявить.

Конкретно в коде нашей лабораторной работы прототипы необходимы для того, чтобы мы могли определить необходимые функции после main, чтобы повысить читаемость кода.

Также прототипы пригодятся тогда, когда наш проект будет состоять из нескольких файлов, чтобы разделить объявление функции и ее реализацию.

**6.4. Структура описания функции.**

тип\_возвращаемого имя\_функции(тип\_аргумента имя\_аргумента);

*int get\_spaces\_count(char \*string);*

**6.5. Какие операции допустимы с адресами?**

Взятие: переменная (см. 6.1)

Разыменование: (см. 6.1)

Арифметика (см. 6.2)

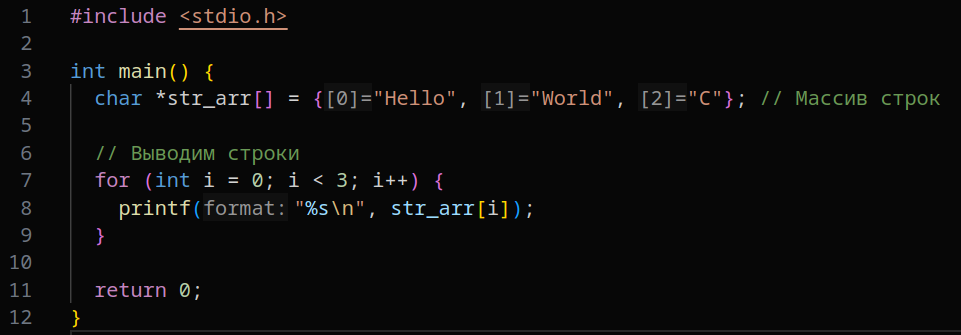
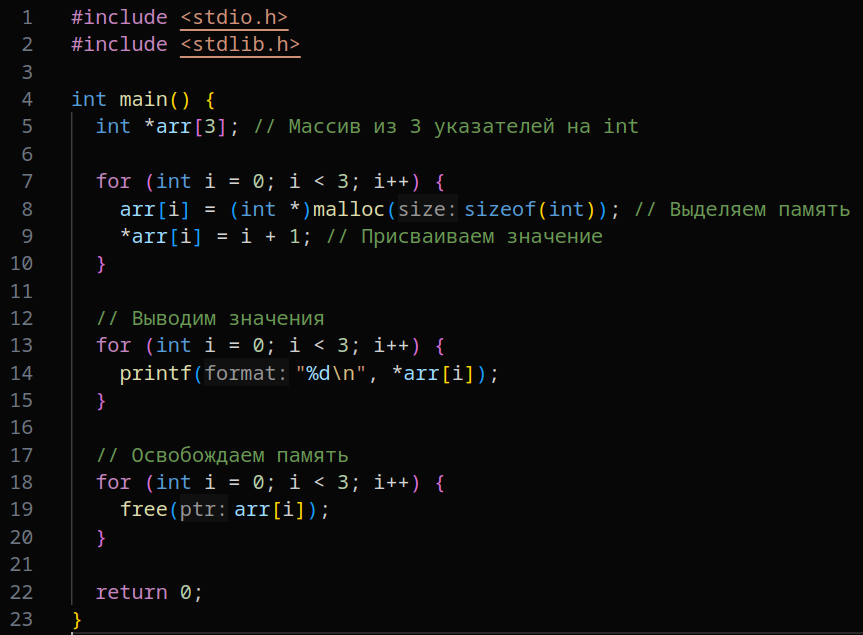
Присваивание (логично)

**6.6. Массивы указателей – описание и использование.**

**— Нахуя?**

1. Работаем с объедками напрямую по их адресу
2. Экономим память (передаем в функцию, не копируя данные, а лишь указывая на них, например)

**— Как?**

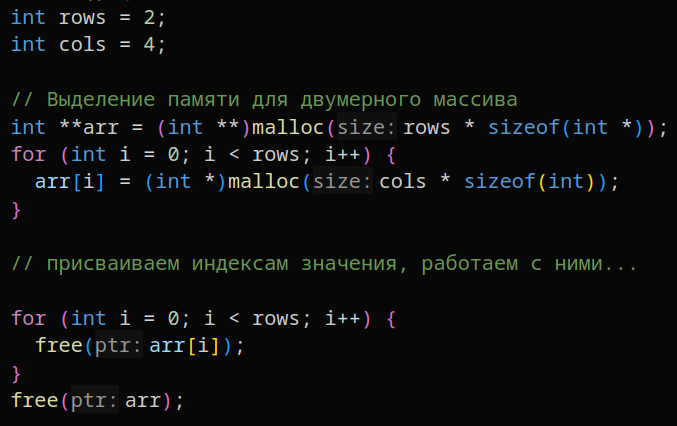
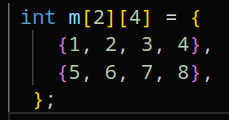


**— Че важно помнить?**

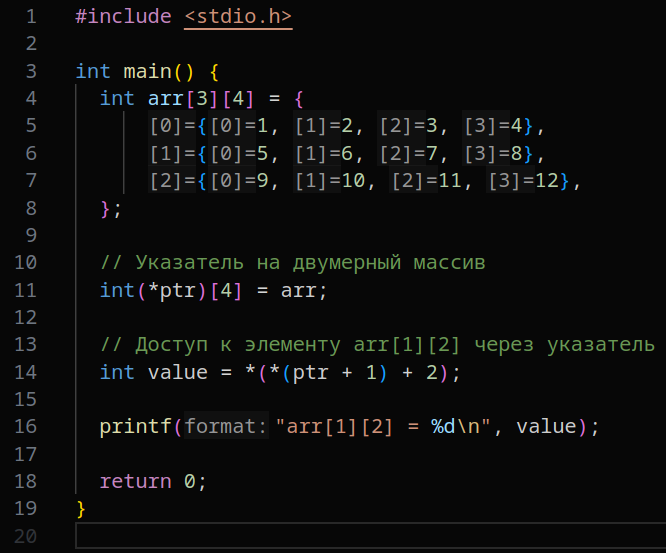
Занял денег — верни. Выделил память — освободи.

Каждый указатель нужно сначала инициализировать, а потом юзать.

**6.7. Привести пример описания двумерного массива.**



**6.8. Привести пример доступа к элементу двумерного массива через указатель на него.**



**6.9. Какие классы памяти существуют в языке СИ?**

Классы памяти определяют срок жизни и область видимости переменной или функции.

**Auto** — класс памяти по-умолчанию для локальных переменных. Располагаются на стеке, область видимости ограничивается своим блоком. При выходе из блока auto-переменные удаляются. При инициализации сначала в переменной лежит мусор (зависит от того, что лежало в этом куске оперативки ранее).

int auto x = 10; // auto можно опустить

**Register** — класс памяти для переменных, которые мы хотим расположить на регистрах процессора, а не в оперативной памяти. Не имеют адреса, попытку взять его вызовет ошибку компиляции.

register int x = 20;

**Static** — время жизни совпадает со временем жизни приложения. Область видимости таких переменных ограничена блоком, в котором они объявлены. Время жизни совпадает со временем жизни программы. При инициализации сразу проставляется нулевое значение.

static int counter = 0;

**External** — внешние (глобальные) переменные. Срок жизни такой же, как у всей программы в целом, я так понимаю. Прямым текстом это нигде не написано, но это было бы логично.

