МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра информационных систем

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Студентка гр. 0324 _____ Косенко А.Р.

Преподаватель _____ Глущенко А.Г.

Санкт-Петербург
2020

АННОТАЦИЯ

Темы, которые содержит курсовой проект: типы данных и их внутреннее представление в памяти, одномерные статические массивы, двумерные статические массивы, текстовые строки как массивы символов.

Код содержит программы, сделанные по темам перечисленным выше.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Необходимо объединить все 4 лабораторные работы в единый проект. Нужно добавить инфраструктуру переключения между заданиями (интерактивное меню).

Переключение между четырьмя практическими работами происходит при помощи инструкции множественного выбора.

1. Типы данных и их внутреннее представление в памяти

1.1. Основные теоретические положения.

Внутреннее представление величин целого типа – целое число в двоичном коде. При использовании спецификатора signed старший бит числа интерпретируется как знаковый (0 – положительное число, 1 – отрицательное). Для кодирования целых чисел со знаком применяется прямой, обратный и дополнительный коды.

Представление положительных и отрицательных чисел в прямом, обратном и дополнительном кодах отличается. В прямом коде в знаковый разряд помещается цифра 1, а в разряды цифровой части числа – двоичный код его абсолютной величины. Прямой код числа −3 (для 16- разрядного процессора)

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Знак числа

Обратный код получается инвертированием всех цифр двоичного кода абсолютной величины, включая разряд знака: нули заменяются единицами, единицы – нулями. Прямой код можно преобразовать в обратный, инвертировав все значения всех битов (кроме знакового). Обратный код числа −3:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

Знак числа

Дополнительный код получается образованием обратного кода с последующим прибавлением единицы к его младшему разряду. Дополнительный код числа −3:

16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1

Знак числа

Увидеть, каким образом тип данных представляется на компьютере, можно при помощи логических операций: побитового сдвига (<<) и поразрядной конъюнкции (&).

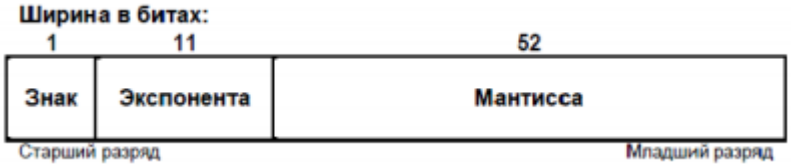
```
putchar(value & mask ? '1' : '0'); // если 1, то возвращается 1, иначе 0
value <<= 1; // побитовый сдвиг влево на 1 бит
```

Putchar возвращает один символ в консоль. Альтернатива - cout. В представленном способе, маска - то, с чем сравнивается значение. И побитовый сдвиг применяется для value. Таким образом 1 бит будет сравниваться с каждым битом числа. Альтернатива - побитовый сдвиг вправо, но при этом нужно проводить данную операцию не над значением(единицей), а над маской (исходым числом, битовое представление которого нужно получить).

При сдвиге вправо для чисел без знака позиции битов, освобожденные при операции сдвига, заполняются нулями. Для чисел со знаком бит знака используется для заполнения освобожденных позиций битов. Другими словами, если число 25 является положительным, используется 0, если число является отрицательным, используется 1. При сдвиге влево позиции битов, освобожденных при операции сдвига, заполняются нулями. Сдвиг влево является логическим сдвигом (биты, сдвигаемые с конца, отбрасываются, включая бит знака).

Вещественные типы данных хранятся в памяти компьютера иначе, чем целочисленные. Внутреннее представление вещественного числа состоит из двух частей – мантиссы и порядка.

Для 32-разрядного процессора для float под мантиссу отводится 23 бита, под экспоненту – 8, под знак – 1. Для double под мантиссу отводится 52 бита, под экспоненту – 11, под знак – 1:



Увидеть, каким образом вещественные типы данных представляются в компьютере немного сложнее. Логические операции, которые использовались с int, для вещественных типов данных не подходят. Но это ограничение можно легко обойти, использовав объединения.

Объединения – это две или более переменных расположенных по одному адресу (они разделяют одну и ту же память). Объединения определяются с использованием ключевого слова union. Объединения не могут хранить одновременно несколько различных значений, они позволяют интерпретировать несколькими различными способами содержимое одной и той же области памяти.

> С объединениями нужно быть острожным. Вся работа с памятью требует грамотного подхода. Более подробно с объединениями можно будет ознакомиться при изучении структур. Пока что объедения будут служить инструментом для работы с float и double.

```
#include <iostream>
using namespace std;
```

```
int main()
{
union
{ int tool; float numb_f = 3.14; };
cout << tool << endl; // 1078523331
cout << numb_f << endl; // 3.14
tool = tool >> 1; // побитовый сдвиг вправо
cout << tool << endl; // 5392261665
cout << numb_f; // 1.3932e-19
return 0; }
```

Алгоритма представления double немного отличается. Под вещественное число с двойной точностью отводиться 8 байт, в то время как под int всего 4 байта. Но и это ограничение можно легко обойти. Так как данные любой линейной структуры в память записываются последовательно (друг за другом), можно использовать массив из двух int, под который будет отведено 8 байт.

1.2. Постановка задачи и ее выполнение выполнение.

Постановка задачи.

Вывести, сколько памяти (в байтах) на вашем компьютере отводится под различные типы данных со спецификаторами и без: int, short int, long int, float, double, long double, char и bool.

Вывести на экран двоичное представление в памяти (все разряды) целого числа. При выводе необходимо визуально обозначить знаковый разряд и значащие разряды отступами или цветом.

Вывести на экран двоичное представление в памяти (все разряды) типа float. При выводе необходимо визуально обозначить знаковый разряд мантиссы, знаковый разряд порядка (если есть), мантиссу и порядок.

Вывести на экран двоичное представление в памяти (все разряды) типа double. При выводе необходимо визуально обозначить знаковый разряд мантиссы, знаковый разряд порядка (если есть), мантиссу и порядок.

Выполнение работы.

1.Чтобы узнать, сколько отводиться под тот или иной тип данных или объект памяти, нужно использовать операцию sizeof. Операция sizeof вычисляет размер в байтах.

2.Вывести то, каким образом тип данных представляется на компьютере, можно при помощи логических операций: побитового сдвига (<<) и поразрядной конъюнкции (&), сравнивая маску с двоичным представлением числа.

Размер типа int зависит от компьютера и компилятора. Для 16-разрядного процессора под величины этого типа отводится 2 байта. Для 32-разрядного – 4 байта. Знаковый бит отделяется пробелом.

3.Так как float нельзя использовать с побитовыми операциями, нужно воспользоваться объединением (union). Объединения позволяют интерпретировать несколькими различными способами содержимое одной и той же области памяти. Под int и float в памяти выделено одинаковое количество бит, поэтому float в памяти можно прочитать через Int.

Для 32-разрядного процессора для float под мантиссу отводится 23 бита, под экспоненту – 8, под знак – 1. Для double под мантиссу отводится 52 бита, под экспоненту – 11, под знак – 1, все это так же выделяется пробелами.

4. double занимает в 2 раза больше битов в памяти, чем int. Под вещественное число с двойной точностью отводиться 8 байт. Для того, чтобы посмотреть значение double нужно создать int массив из двух элементов и воспользоваться теми же приемами, как и в задании 3.

2. Одномерные статические массивы

2.1. Основные теоретические положения.

Понятие массива.

При использовании простых переменных каждой области памяти для хранения данных соответствует свое имя. Если с группой величин одинакового типа требуется выполнить однообразные действия, им дают одно имя, а различают по порядковому номеру (индексу). Это дает возможность компактно записать множество операций с использованием циклов. Массив представляет собой

индексированную последовательность однотипных элементов с заранее определенным количеством элементов. Наглядно одномерный массив можно представить, как набор пронумерованных ячеек, в каждой из которых содержится определенное значение. Все массивы можно разделить на две группы: одномерные и многомерные. Описание массива в программе отличается от объявления обычной переменной наличием размерности массива, которая задается в квадратных скобках после имени. В листинге 3.1 представлен пример работы с одномерным массивом. Элементы массива нумеруются с нуля. При описании массива используются те же модификаторы (класс памяти, const и инициализатор), что и для простых переменных. Инициализировать массив можно и другим более простым способом: инициализирующие значения записываются в фигурных скобках. Значения элементам присваиваются по порядку. Если элементов в массиве больше, чем инициализаторов, элементы, для которых значения не указаны, обнуляются:

```
int arr[4] = {3, 2, 1}; // arr[0] = 3, arr[1] = 2, arr[2] = 1, arr[3] = 0
```

Размерность массива вместе с типом его элементов определяет объем памяти, необходимый для размещения массива, которое выполняется на этапе компиляции, поэтому размерность должна быть задана целой положительной константой или константным выражением.

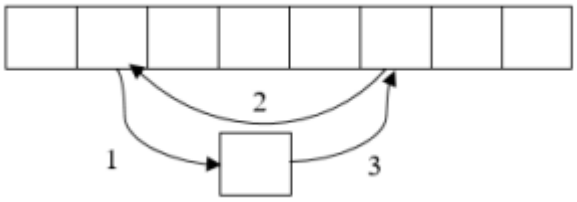
Пример заполнения одномерного массива

```
/* arr.cpp: Этот файл содержит функцию "main". Здесь начинается и
заканчивается выполнение программы */
#include
#include
using namespace std;
int main()
{ setlocale(LC_ALL, "Russian");
const int N = 10; // Задаём размерность массива
srand(time(0));
int arr[N]; // Объявляем массив arr размерности N
cout << "Исходный массив данных: ";
for (int i = 0; i < N; i++) // Цикл, заполняющий массив случайными числами
{ arr[i] = rand() % 9; // Заполняем массив случайными числами от 0 до 9
cout << arr[i] << " "; // Выводим элемент массива на экран
}
return 0;
}
```

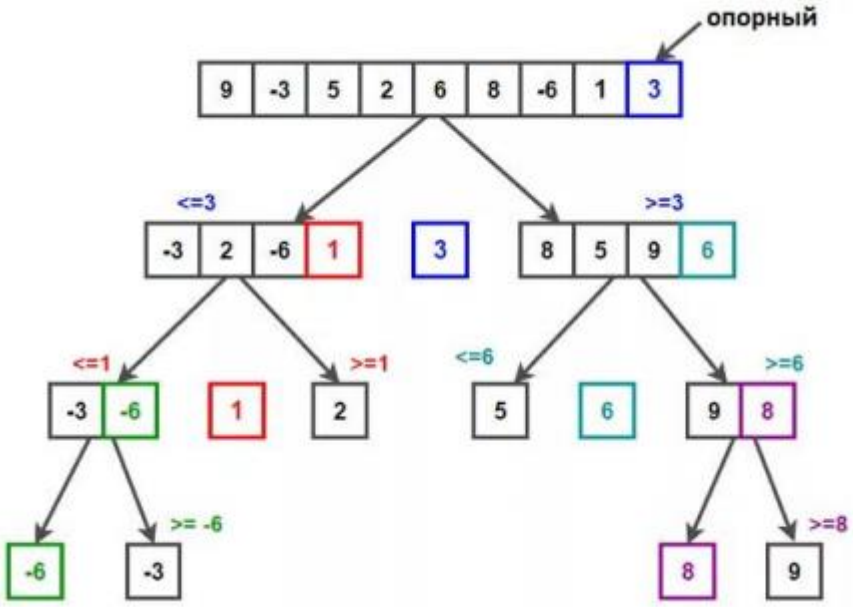
Если при описании массива не указана размерность, массив обязательно должен быть инициализирован. Компилятор сам определит размерность массива по количеству элементов:

```
int arr[] = {7, 6, 5, 4, 3, 2, 1}; // Всего 7 элементов
```

Для доступа к элементу массива после его имени указывается номер элемента (индекс) в квадратных скобках. Важно помнить, что при обращении к элементам массива автоматический контроль выхода индекса за границу массива не производится, это может привести к ошибкам. Обмен элементов массива осуществляется через буферную переменную либо через функцию swap(a, b)
Обмен местами элементов массива:



Определить, сколько памяти выделено под массив, можно с помощью операции sizeof. Необходимо определить, сколько выделяется памяти на 1 элемент массива и умножить на количество элементов массива. Принцип действия схож с задачей про переливание воды из одного сосуда в другой. Чтобы поменять жидкость сосудов местами, нужно использовать еще один сосуд, который будет временно содержать жидкость одного из сосудов.
Быстрая сортировка (quick sort)– одна из самых быстрых сортировок. Эта сортировка по сути является существенно улучшенной версией алгоритма пузырьковой сортировки. Общая идея алгоритма состоит в том, что сначала выбирается из массива элемент, который называется опорным. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность. Затем необходимо сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: меньше опорного, раны опорному и больше опорного. Для меньших и больших значений необходимо выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.



На практике массив обычно делят на две части: «меньше опорного» и «равные и большие» или «меньше опорного или равные» и «большие». Такой поход в общем случае эффективнее, ведь упрощается алгоритм разделения. При том, что это один из самых быстродействующих из алгоритмов, данные алгоритм сортировки неустойчив, а прямая реализация в виде функции с двумя рекурсивными вызовами может привести к ошибке переполнения стека.

Алгоритм бинарного поиска

Алгоритм бинарного поиска – классический алгоритм поиска в отсортированном массиве, который использует дробление массива на половины. Если элемент, который необходимо найти, присутствует в списке, то бинарный поиск возвращает ту позицию, в которой он был найден. Рассмотрим простой пример: имеется массив из 100 элементов, упорядоченных по возрастанию от 1 до 100. Было загадано какое-то число, необходимо его назвать. Компьютер имеет три ответа на ваше предположение: верно, число больше, число меньше. Сколько попыток нужно, чтобы ответить правильно? Обычный перебор – наихудшая стратегия. Можно назвать правильный ответ лишь с 100-ой попытки. Но если начать спрашивать с середины, то ситуация кардинально меняется. Если число больше 50, то необходимо делить правую половину, и следующее предположение – 75, если меньше – 25. Так необходимо продолжать до тех пор, пока не будет названо правильное число. Наибольшее число предположений равняется:

$$k = \lceil \log_2 100 \rceil = 7.$$

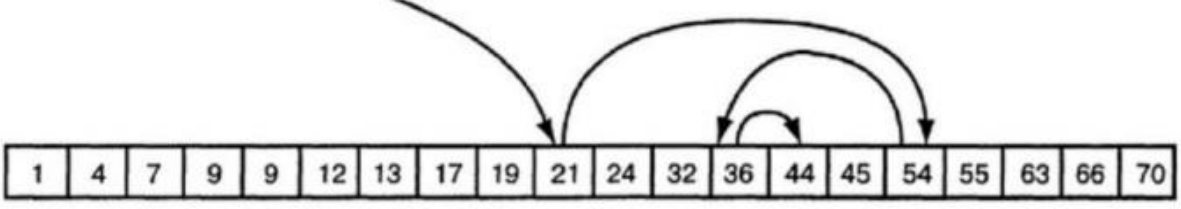
Этот результат явно лучше простого перебора. Схожий принцип работы имеет алгоритм бинарного поиска. Бинарный поиск работает только в топ случае, если список отсортирован. Например, если бы искомое минимальное значение стояло не на своем положенном месте, а на месте максимального элемента, то мы бы откинули его на первой же итерации. Сам алгоритм имеет вид:

1) Определение значения в середине массива (или иной структуры данных). Полученное значение сравнивается с ключом (значением, которое необходимо найти).

2) Если ключ меньше значения середины, то необходимо осуществлять поиск в первой половине элементов, иначе – во второй.

3) Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.

4) Процесс продолжается до тех пор, пока не будет определен элемент, равный значению ключа или не станет пустым интервал для поиска.



Бинарный поиск числа 44

Чтобы уменьшить количество шагов поиска, можно сразу смещать границы поиска на элемент, следующий за серединой отрезка.

2.2. Постановка задачи и ее выполнение выполнение.

Постановка задачи.

1) Создает целочисленный массив размерности $N = 100$. Элементы массивы должны принимать случайное значение в диапазоне от -99 до 99.

2) Отсортировать заданный в пункте 1 элементы массива […] сортировкой (от меньшего к большему). Определить время, затраченное на сортировку, используя библиотеку chrono.

3) Найти максимальный и минимальный элемент массива. Подсчитайте время поиска этих элементов в отсортированном массиве и неотсортированном, используя библиотеку chrono.

4) Выводит среднее значение (если необходимо, число нужно округлить) максимального и минимального значения. Выводит индексы всех элементов, которые равны этому значению, и их количество.

5) Выводит количество элементов в отсортированном массиве, которые меньше числа *a*, которое инициализируется пользователем.

6) Выводит количество элементов в отсортированном массиве, которые больше числа *b*, которое инициализируется пользователем.

7) Выводит информацию о том, есть ли введенное пользователем число в отсортированном массиве. Реализуйте алгоритм бинарного поиска. Сравните скорость его работы с обычным перебором. (*)

8) Меняет местами элементы массива, индексы которых вводит пользователь. Выведите скорость обмена, используя библиотеку chrono.

Выполнение работы.

1. Чтобы заполнить массив случайными числами, использую rand() – генератор случайных чисел

2. Заданный в пункте 1 массив сортирую при помощи быстрой сортировки (quick sort)

3. Сначала нахожу в неотсортированном массиве максимум и минимум, замеряю время поиска, затем в отсортированном массиве, также замеряя время поиска

4. Через цикл for нахожу среднее значение, которое было подсчитано ранее.
5. Нахожу в массиве число, которое меньше введенного, записываю его индекс в новую переменную. Через цикл for вывожу элементы от найденного индекса в обратном порядке
6. Нахожу в массиве число, которое больше введенного, записываю его индекс в новую переменную. Через цикл for вывожу элементы от найденного индекса.
7. Алгоритм бинарного поиска:

1) Определение значения в середине массива (или иной структуры данных). Полученное значение сравнивается с ключом (значением, которое необходимо найти).

2) Если ключ меньше значения середины, то необходимо осуществлять поиск в первой половине элементов, иначе – во второй.

3) Поиск сводится к тому, что вновь определяется значение серединного элемента в выбранной половине и сравнивается с ключом.

4) Процесс продолжается до тех пор, пока не будет определен элемент, равный значению ключа или не станет пустым интервал для поиска.

Замеряю затраченное время. После записываю перебор, также замеряю время.

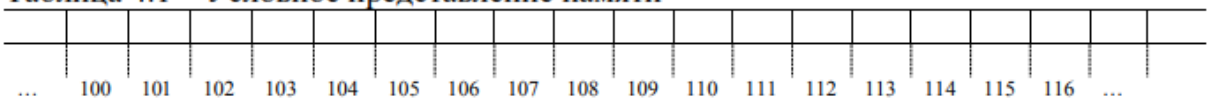
8. При помощи буферной переменной меняю элементы массива, после массив выводится на экран.

3. Указатели и многомерные статические массивы

3.1. Основные теоретические положения.

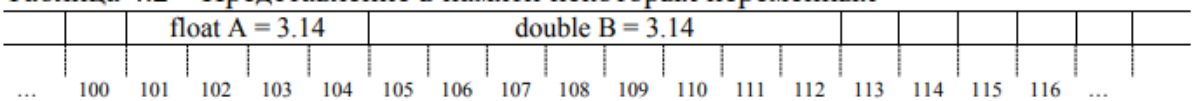
Понятие указателя Компилятор, обрабатывая оператор определения переменной, выделяет память в соответствии с типом переменной и инициализирует её указанным значением. Все обращения по имени переменной заменяются компилятором на адрес области памяти, в которой хранится значение переменной. Возможно создание собственных переменных, которые будут хранить какой-то адрес памяти. Такие переменные называются указателями.

Таблица 4.1 – Условное представление памяти



Указатели предназначены для хранения адресов областей памяти (табл. 4.1.). В C++ существует три вида указателей: 1) Указатели на объект, который содержит адрес области памяти, хранящей данные определенного типа. 2) Указатели на функцию. Указатель на функцию содержит адрес сегмента кода, по которому располагается исполняемый код функции. Указатели на функции используются для косвенного вызова функции (через обращение к переменной, хранящей её адрес), а также для передачи имени функции в другую функцию в качестве параметра. Указатель функции должен иметь тип «указатель функции, возвращающей значение заданного типа и имеющей аргументы заданного типа. 3) Указатели на void. Такой указатель применяется в тех случаях, когда тип объекта, адрес которого нужно хранить, не определен. Указателю типа void 46 можно присвоить значение любого типа, но перед выполнением каких-либо действий его нужно явным образом преобразовать к этому типу. Для получения адреса какого-либо программного объекта используют оператор &. Предположим, что у нас имеется две переменные: float A = 3.14 и double = 3.14 (табл. 4.2). Видно, что обе переменные занимают по 4 байта памяти. Они имеют адреса 101 и 105 соответственно.

Таблица 4.2 – Представление в памяти некоторых переменных



Если использовать оператор &, то он вернет адреса этих переменных. Правда, обычно значение адреса памяти выводится в шестнадцатеричном коде. Указатели на объекты определяются в программе следующим образом: *; Тип данных – базовый тип указателя. Имя переменной – идентификатор переменной-указателя. Признаком того, что переменная – указатель, является символ ‘*’, располагающийся перед именем переменной. Определим две переменных-указателя: float *p1; int *p2; Указатель $p1$ предназначен для хранения адресов участков памяти, размер которого соответствует типу int, а

переменная *p*2 - для хранения адресов участков памяти, размер которых соответствует типу double. Указатели представляют собой обычные целые числа значения типа int и занимают в памяти 4 байта не зависимо от базового типа указателя. Значения указателей при их выводе на экран представляются как целые значения в шестнадцатеричном формате.

Инициализация указателей.

Листинг 4.1 – Пример использования указателя

```
/* pointer.cpp: Этот файл содержит функцию "main". Здесь начинается и заканчивается
выполнение программы */

#include <iostream>

using namespace std;

int main()
{
	setlocale(LC_ALL, "Russian");
	float pi = 3.14, *p;
	p = &pi;
	cout << "Значение переменной pi: " << pi << endl;
	cout << "Значение переменной pi: " << *p << endl;
	cout << "Адрес переменной pi: " << p << endl;
	return 0;
}
```

Результат выполнения этой программы:

```
Значение переменной pi: 3.14
Значение переменной pi: 3.14
Адрес переменной pi: 012FF77C
```

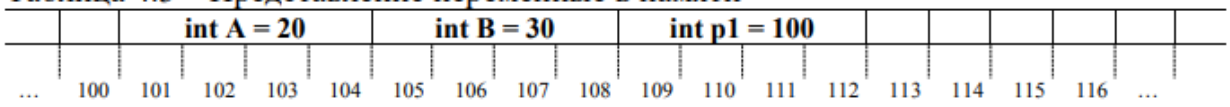
Присвоить указателю адрес некоторой переменной можно инструкцией присваивания и операции &. Пример инициализации указателя представлен в 47 листинге 4.1. Получить значение объекта, на который ссылается некоторый указатель можно с помощью операции * (разыменовывание указателя).

Из памяти по адресу, который хранится в указатели, берется столько байт памяти, сколько требуется базовому типу указателя. Далее с этими байтами работают, как со значением базового типа указателя. С помощью указателей можно не только получать значения, расположенные по адресам, хранящимся в указателях, но и записывать нужные значения по этим адресам: *p = 1.618; Указатели могут использоваться в различных выражениях наравне с обычными переменными и константами. При использовании указателей в выражениях важно помнить, что операция * имеет наивысший приоритет по отношению к другим операциям (за исключением операции унарный минус). Указателю можно присвоить значение другого указателю, если совпадают их базовые типы. Хотя указатели представляют собой целые значения, присваивать им произвольные целые значения нельзя. Единственным исключением является присвоение указателю нулевого значения. Нулевое значение указателя означает то, что указатель не на что не ссылается.

Арифметика указателей

К указателям можно применять некоторые арифметические операции, одни из них +, −, ++, − −. Результаты выполнения этих операций по отношению к указателям существенно отличаются от результатов соответствующих арифметических операций, выполняющихся с обычными числовыми данными.

Таблица 4.3 – Представление переменные в памяти



Указатель p1 содержит адрес переменной A. Если выполнить операцию p1 = p1 + 1; (или p1++;) Значение указателя изменится, и станет равно 104. Таким образом, добавление или вычитание 1 из указателя приводит к изменению его значения на размер базового типа указателя. При этом добавлять или вычитать из указателей можно только целые значения. Не стоит забывать о приоритете операций: *p1 + 1; и *(p1 + 1); имеют совершенно разный смысл.

Указатели и массивы

Между массивами и указателями имеется очень тесная связь. Когда в программе определяется некоторый массив. Имя переменной без индексов представляет собой указатель на первый элемент массива: int Arr[5]; 49 Если вывести на экран значение переменной Arr, то будет отображено некоторое целое значение в шестнадцатеричном формате, которое соответствует адресу первого элемента этого массива. Именно из-за этого операция присвоения сразу всех значений одного массива в другому запрещена в C++. Попытка присвоения Arr1 = Arr2; привела бы к тому, что переменная Arr1, стала бы указывать на ту же область памяти, что и переменная Arr2. Адрес, который ранее хранился в переменной Arr1, был бы утерян, что привело бы к утечке памяти. Более того, по этой причине запрещены любые изменения значения переменной массива. Указателю, который имеет такой же базовый тип, как и элемент массива, можно присвоить указатель на массив. Но обратное присвоение выполнить невозможно, так как переменная массива – это константа, изменение которой запрещено. Так как переменная массива является указателем на первый элемент массива, появляются дополнительные возможности по работе с массивами на основе использования арифметики указателей. Например, Arr[4] эквивалентно *(Arr + 4). Первое выражение – это пример обычной индексации элементов массива. Во втором выражении использовалась арифметика указателей, и с помощи операции + был получен адрес пятого элемента массива. Здесь нельзя забывать о приоритете операций. Использование арифметики указателей при работе с массивами приводит обычно к уменьшению объема генерируемого кода программы и к уменьшению времени ее выполнения, то есть к увеличению быстродействия. Поскольку указатель и имя массива взаимосвязаны, указатели можно индексировать, как обычные массивы, а также создавать массивы указателей.
Объявление многомерных массивов Многомерные массивы определяются аналогично одномерным массивам. Количество элементов по каждому

измерению указывается отдельно в квадратичных 50 скобках. Элементы многомерных массивов располагаются друг за другом в непрерывном участке памяти. При инициализации многомерного массива он представляется либо как массив из массивов, при этом каждый массив заключается в свои фигурные скобки (в этом случае левую размерность при описании можно не указывать), либо задается общий список элементов в том порядке, в котором элементы располагаются в памяти:

```
int arr[][] = { {0,0}, {1,1}, {1,0} };
int arr[3][3] = { 0, 0, 0, 1, 1, 1, 2, 2, 2 };
```

Определить, сколько памяти выделено под многомерный массив, можно аналогично одномерному массиву с помощью операции sizeof. Необходимо определить, сколько выделяется памяти на 1 элемент массива и умножить на количество элементов массива. В листинге 4.2 представлен пример заполнения многомерного массива.

Листинг 4.2 – Пример инициализации двумерного массива

```
/* arr[NxM].cpp: Этот файл содержит функцию "main". Здесь начинается и заканчивается
выполнение программы */

#include <iostream>
#include <ctime>

using namespace std;

int main()
{
	setlocale(LC_ALL, "Russian");
	const int N = 10, M = 10; // Количество строк матрицы N = 10, столбцов - M = 10
	srand(time(0));
	int arr[N][M]; // Обьявляем массив arr размерности NxM
	cout << "Исходный массив данных: ";
	for (int i = 0; i < N; i++)
	{
		for (int j = 0; j < M; j++)
		{
			arr[i][j] = rand() % 9;
			cout << arr[i][j] << " ";
		}
		cout << endl;
	}
	return 0;
}
```

В приведенном выше примере сначала объявляется двумерный массив, затем при помощи двух циклов (один из которых вложенный) происходит заполнение массива случайными числами. Первый цикл нужен, чтобы пройти все строчки 51 матрицы, а второй цикл – пройти все строчки, заполнить их значениями и вывести на экран.

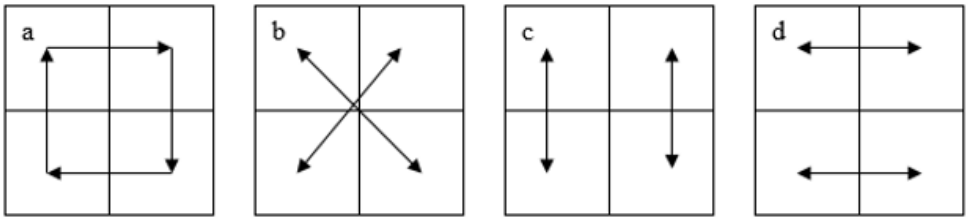
3.2. Постановка задачи и ее выполнение выполнение.

Постановка задачи.

1) Используя арифметику указателей, заполняет квадратичную целочисленную матрицу порядка *N* (6,8,10) случайными числами от 1 до N*N согласно схемам, приведенным на рисунках. Пользователь должен видеть процесс заполнения квадратичной матрицы.

а *б*

2) Получает новую матрицу, из матрицы п. 1, переставляя ее блоки в соответствии со схемами:



3) Используя арифметику указателей, сортирует элементы любой сортировкой.

4) Уменьшает, увеличивает, умножает или делит все элементы матрицы на введенное пользователем число.

Выполнение работы.

1) Для выполнения этого задания я использовала High-Level Input and Output Functions

2) Чтобы поменять блоки местами, создала 4 массива n/2 * n/2 для хранения всех блоков массива n*n. С помощью цикла for присваивала нужной части большого массива значения маленького массива.

3) Сортировка массива bubble sort при помощи указателей.

4) Через цикл for, при помощи арифметики указателей меняла (действие зависит от выбора пользователя) каждый элемент массива на введенное число.

4. Типы данных и их внутреннее представление в памяти

4.1. Основные теоретические положения.

Работа с файлами

Для работы с файлами в языке C++ используют потоки трех видов:

- поток ввода (класс ifstream);
- поток вывода (класс ofstream);
- поток ввода-вывода (класс fstream);

Класс ifstream используется для выполнения чтения данных из файлов. Поток ofstream – для записи данных в файл. Поток fstream – для чтения и записи данных в файл.

Открыть файл можно двумя способами:

1) сначала создать поток, а потом связать его с файлом:

```
ofstream File;
File.open("D:\\test.txt");
```

2) создать поток и связать поток с файлом:

```
ofstream File ("D:\\test.txt");
```

После того, как поток был связан с файлом, необходимо обязательно проверить открылся ли файл. Если файл открыть не удалось, то переменная потока (File) принимает значение false, если файл открыт – true.
Проверку успешного открытия файла можно выполнить с помощью условного оператора:

```
If (!File) // другой вариант
if (!File.is_open()) cout << "Ошибка при открытии файла!";
```

Функция потока is_open() возвращает логическое значение в зависимости от результата операции открытия файла. Файл закрывается с помощью функции потока close(): File.close(); 5 Каждый поток использует свой вариант функции open. Первый параметр определяет имя открываемого файла (представляет собой массив символов). Второй параметр определяет режим открытия файла. Этот параметр имеет значение по умолчанию, то есть является необязательным. Возможные значения этого параметра:

- ios::app – при открытии файла на запись (поток ofstream) обеспечивает добавление всех выводимых в файл данных в конец файла;
- ios::ate – обеспечивает начало поиска данных в файле начиная с конца файла;
- ios::in – файл открывается для чтения из него данных;

• ios::out – файл открывается для записи данных в файл;
• ios::binary – открытие файла в двоичном режиме (по умолчанию все файлы открываются в текстовом режиме);
• ios::trunc – содержимое открываемого файла уничтожается (его длина становится равной 0). Эти флаги можно комбинировать с помощью побитовой операции ИЛИ (|). Если файл открывается без использования функции open, эти флаги тоже можно использовать:

```
fstream File ("E:\\test.txt", ios :: out | ios :: app);
```

Файл открывается на вывод с добавлением записываемых данных в конец файла. Запись и чтение данных в текстовых файлах ничем не отличается от способов ввода-вывода данных с помощью потоков cin и cout. Методы форматирования вывода и вводы данных остаются такими же (флаги форматирования, манипуляторы, функции потоков). Необходимо помнить, что при использовании операции >> для чтения данных из текстовых файлов, процесс чтения останавливается при достижении пробельного символа. Поэтому для чтения строки текста, содержащей несколько слов, необходимо использовать, например, функцию getline().

Определение текстовой строки

Текстовые строки представляются с помощью одномерных массивов символов. В языке C++ текстовая строка представляет собой набор символов, обязательно заканчивающийся нулевым символом (‘\0’). Поэтому, если вы хотите создать текстовый массив для хранения 10 (n) символов, нужно выделить память под 11(n + 1) символов. Объявленный таким образом массив может использоваться для хранения текстовых строк, содержащих не более 10 символов. Нулевой символ позволяет определить границу между содержащимся в строке текстом и неиспользованной частью строки. При определении строковых переменных их можно инициализировать конкретными значениями с помощью строковых литералов:

```
char S1[15] = “This is text”;
char S2[] = “Пример текста”;
```

Последние два элемента переменной $S1$ просто не используются, а строка $S2$ автоматически подстраивается под длину инициализирующего текста. При работе со строками можно обращаться к отдельным символам строки как в обычном одномерном массиве с помощью индексов:

```
cout << S1[0]; // На экране будет выведен символ ‘T’
```

Если строка формируется при помощи цикла (или иного способа), то необходимо ее конец обязательно записать нулевой символ '\0'.

Ввод текстовых строк с клавиатуры

При выводе строк можно использовать форматирование (манипуляторы или функции потока вывода). Вывод текстовых строк на экран крайне простая задача:

```
char Str[21] = “Это пример текста”;
cout << Str << endl;
cout << “Это текстовый литерал.” << endl;
```

Ввод текста с клавиатуры можно осуществлять разными способами, каждый из которых имеет определенные особенности.
Непосредственное чтение текстовых строк из потока вывода осуществляется до первого знака пробела. Такой способ чтения обеспечивает ввод символов до первого пробельного символа (не до конца строки). Остальные символы введенного с клавиатуры остаются в потоке ввода и могут быть прочитаны из него следующими операторами >>. Для того чтобы прочесть всю строку полностью, можно воспользоваться одной из функций gets или gets_s (для этого в программу должен быть включен заголовочный файл). Функция gets имеет один параметр, соответствующий массиву символов, в который осуществляется чтение. Вторая функция (gets_s) имеет второй параметр, задающий максимальную длину массива символов Str.

Ввод текста, длина которого (вместе с нулевым символом) превышает значение второго параметра (то есть длины символьного массива Str), приводит к возникновению ошибки при выполнении программы Предпочтительно использование функции потока ввода cin.getline:

```
const int N = 21; char Str [N];
cin.getline (Str, N); // Пусть введена строка “Это пример текста”
cout << Str << endl; // На экран будет выведено “ Это пример текста”
```

Если длина введенного с клавиатуры текста превышает максимальную длину массива Str, в него будет записано (в нашем примере) 20 символов вводимого текста и нулевой символ. Остальные символы введенного текста остаются во входном потоке и могут быть взяты из него следующими инструкциями ввода. Функция cin.getline может иметь третий параметр, задающий символ, при встрече которого чтение строки из потока прекращается:

```
cin.getline (Str, N, ‘.’);
```

Иногда чтение из потока невозможно (например, попытка считать слишком длинный текст). Для того чтобы продолжить чтение из потока, необходимо восстановить его нормальное состояние. Этого можно достигнуть с помощью функции потока cin.clear(), которая сбрасывает состояние потока в нормальное. Если забирать остатки данных из потока ввода не надо, то следует очистить его с помощью функции cin.sync().
Класс string
Класс string предназначен для работы со строками типа char, которые представляют собой строчку с завершающим нулем (символ ‘\0’). Класс string был введен как альтернативный вариант для работы со строками типа char.

Чтобы использовать возможности класса string, нужно подключить библиотеку и пространство имен std. Объявление же переменной типа string осуществляется схоже с обычной переменной:

```
string S1; // Переменная с именем s1 типа string
string S2 = “Пример”; // объявление с инициализацией
```

Создание нового типа string было обусловлено недостатками работы с строками символов, который показывал тип char. В сравнении с этим типом string имеет ряд основных преимуществ:

- возможность использования для обработки строк стандартные операторы C++ (=, +, , +=, ! =, <=, >=,[])Использование типа char приводило требовало написание чрезмерного программного кода;
- обеспечение лучшей надежности программного кода;
- обеспечение строки, как самостоятельного типа данных.

Класс string обладает широким функционалом:

- функция compare() сравнивает одну часть строки с другой;
- функция length() определяет длину строки;
- функции find() и rfind() служат для поиска подстроки в строке (отличаются функции лишь направлением поиска);
- функция erase() служит для удаления символов;
- функция replace() выполняет замену символов;
- функция insert() необходима, чтобы вставить одну строку в заданную позицию другой строки;
- Аналогом использования оператора + является функция append();
- Аналогом использования оператора = является функция assign();

Но весь функционал string накладывает и свой негативный отпечаток. Основным недостатком string в сравнении с типом char является замедленная скорость обработки данных.

Поиск подстроки в строке

При работе со строками часто будет возникать потребность в поиске набора символа или слов (поиска подстроки в строке). При условии, что текст может быть крайне большим, хочется, чтобы алгоритм поиска подстроки работал быстро.

Самый простой способ подстроки в строке – Линейный поиск – циклическое сравнение всех символов строки с подстрокой. Действительно, этот способ первый приходит в голову, но очевидно, что он будет самым долгим.

1-й шаг цикла	**H**	E	L	L	O		W	O	R	L	D
	L	O									
2-й шаг цикла	H	**E**	L	L	O		W	O	R	L	D
		L	O								
3-й шаг цикла	H	E	**L**	**L**	O		W	O	R	L	D
			L	**O**							
4-й шаг цикла	H	E	L	**L**	**O**		W	O	R	L	D
				L	**O**						

Рисунок 1.1 – Поиск подстроки “LO” в строке “HELLO WORLD”

На первых двух итерациях цикла сравниваемые буквы не будут совпадать. На третьей же итерации, совпал символ ‘L’, это означает, что теперь нужно сравнивать следующий символ подстроки со следующим символом строки. Видно, что символы отличаются, поэтому алгоритм продолжает свою работу. На четвертой же итерации подстрока была найдена.

Если представить, что исходная строка непорядок больше и подстрока находится в конце строки (или вовсе отсутствует), то сразу видны минусы данного алгоритма.

Одним из самых популярных алгоритмов, который работает быстрее, чем приведенный выше алгоритм, является алгоритм Кнута-Морриса-Пратта (КМП). Идея заключается в том, что не нужно проходить и сравнивать абсолютно все символы строки, если известны символы, которые есть и в строке, и в подстроке.

Суть алгоритма: дана подстрока s и строка t. Требуется определить индекс, начиная с которого образец s содержится в строке t. Если s не содержится в t, необходимо вернуть индекс, который не может быть интерпретирован как позиция в строке.

ТАБЛИЦА ВЛЮЧЕНИЙ

0	1	**2**	**3**	4	5	6	7	8	**9**	10
H	E	L	L	O		W	O	R	L	D

	0	1	**2**	**3**	4	5	6	7	8	**9**	10
1-й шаг цикла	H	E	**L**	**L**	O		W	O	R	L	D
			L	**O**							
2-й шаг цикла	H	E	L	**L**	**O**		W	O	R	L	D
				L	**O**						

Рисунок 1.2 – пример работы алгоритма КМП

Хоть алгоритм и работает быстрее, по-прежнему необходимо сначала пройти всю строку, чтобы определить префиксы или суффиксы (вхождение (индексы) символов).

Алгоритм Бойера-Мура в отличие от КМП полностью не зависим и не требует заранее проходить по строке. Этот алгоритм считается наиболее быстрым среди

алгоритмов общего назначения, предназначенных для поиска подстроки в строке.
Преимущество этого алгоритма в том, что ценной некоторого количества предварительных вычислений над подстрокой (но не над исходной строкой, в которой ведётся поиск), подстрока сравнивается с исходным текстом не во всех позициях (пропускаются позиции, которые точно не дадут положительный результат).
Поиск подстроки ускоряется благодаря созданию таблиц сдвигов. Сравнение подстроки со строки начинается с последнего символа подстроки, а затем происходит прыжок, длина которого определяется по таблице сдвигов. Таблица сдвигов строится по подстроке так чтобы перепрыгнуть максимальное количество символов строки и не пропустить вхождение подстроки в строку.

Правила построения таблицы сдвигов:

1) Значение элемента таблицы равно удаленности соответствующего символа от конца шаблона (подстроки).
2) Если символ встречается более одного раза, то применятся значение, соответствующее символу, наиболее близкому к концу шаблона.
3) Если символ в конце шаблона встречается 1 раз, ему соответствует значение, равное длине образа; если более одного раза – значение, соответствующее символу, наиболее близкому к концу образа.
4) Для символов, отсутствующих в образе, применяется значение, равное длине шаблона.

	0	1	**2**	**3**	4	5	6	7	8	**9**	10
1-й шаг цикла	H	E	L	**L**	O		W	O	R	L	D
	E	L	L	**O**							
2-й шаг цикла	H	**E**	**L**	**L**	**O**		W	O	R	L	D
		E	**L**	**L**	**O**						

Таблица отступов

3	1	1	4
E	L	L	O

Длина подстроки: 4

Рисунок 1.3 – Пример работы алгоритма Бойра-Мура

Сначала была построена таблица отступов и подсчитана длина подстроки. Затем начинается алгоритм поиска подстроки в строке. Сравнивает символ ‘L’ строки и ‘O’ подстроки. Элементы не совпадают, поэтому необходимо определить длину отступа. Символ ‘L’ присутствует в таблице отступа, длина отступа равняется 1. Подстрока смещается на 1 символ вперед. На следующей итерации подстрока найдена.

4.2. Постановка задачи и ее выполнение выполнение.

Постановка задачи.

1) С клавиатуры или с файла (*) (пользователь сам может выбрать способ ввода) вводится последовательность, содержащая от 1 до 50 слов, в каждом из которых от 1 до 10 строчных латинских букв и цифр. Между соседними словами произвольное количество пробелов. За последним символом стоит точка.

2) Необходимо отредактировать входной текст:

· удалить лишние пробелы;

· удалить лишние знаки препинания (под «лишними» подразумевается несколько подряд идущих знаков (обратите внимание, что «…» - корректное использование знака) в тексте);

· исправить регистр букв, если это требуется (пример некорректного использования регистра букв: пРиМЕр);

3) Вывести на экран только те слова последовательности, в которых первая буква слова встречается в этом слове еще раз.
4) Вывести на экран ту же последовательность, удалив из всех слов заданный набор букв и (или) цифр.
5) Необходимо найти подстроку, которую введёт пользователь в имеющейся строке. Реализуйте два алгоритма: первый алгоритма – Линейный поиск, а второй алгоритм согласно вашему номеру в списке. Четные номера должны реализовать алгоритм КНМ, а нечетные – Бойера-Мура. (*)

Выполнение работы.

1) Использовала поток вывода (класс ofstream), создала поток и связала поток с файлом. С клавиатуры строка вводится через getline()

2)Редактировала текст, проходя по строке через цикл: пока встречались одинаковые знаки препинания, удаляла при помощи функции .erase.

3) Как только цикл дошел до пробела, в пустую строку вписывается слово(для этого я использовала функцию substr), полсле оно удаляется из главной строки. В слове ищется первая буква если совпадений больше чем одно, выводится на экран.

4) Пользователь вводит слово, которое хочет удалить, при помощи функции find нахожу индекс первого вхождения. Используя функцию erase, удаляю последовательность с позиции первого вхождения до позиции (первого вхождения + длинна последовательности).

5) Линейный поиск – циклично сравниваю все символы строки с подстрокой. КМП. Префикс-функция для i-ой позиции — это длина максимального префикса строки, который короче i и который совпадает с суффиксом префикса длины i. Берем каждый возможный префикс строки и смотрим самое длинное совпадение начала с концом префикса.

При каждом несовпадении двух символов текста и образа образ сдвигается на все пройденное расстояние, так как меньшие сдвиги не могут привести к полному совпадению.

ЗАКЛЮЧЕНИЕ

Было изучено представление значений различных типов данных в двоичном коде, а также способ применения побитового сдвига и сравнение двоичных значений двух переменных при помощи поразрядной конъюнкции.

Была создана программа, решающая несколько заданий для одномерного массива на 100 элементов. Узнала о работе с временем при помощи chrono.

Была создана программа , решающая задания для двумерного массива. Узнала о работе с массивом при помощи арифметики указателей.

Изучены способы обработки текстовых данных. Получены навыки работы с файлами. Изучен алгоритм поиска подстроки в строке.

ПРИЛОЖЕНИЕ А
РАБОТА ПРОГРАММЫ

4 лаба.

```
!!!далее вводите слова на латинице!!!
Введите строку: 123 timp tomp  !!!! nIna
Выберите задание:

1.Отредактируй текст
2.3 задание 4 вариант
3.4 задание 3 вариант
4. Выход
>>> 1
Неотредактированный текст: 123 timp tomp  !!!! nIna
Текст после редактирвоания: 123 timp tomp ! nina
```

```
1.Отредактируй текст
2.3 задание 4 вариант
3.4 задание 3 вариант
4. Выход
>>> 2
Слова, в которых повторяются буквы: nina

Выберите задание:
```

```
Какую полседоватльность нужно удалить: mp

Теперь последовательность выглядит так: 123 tri tro ! nina
Выберите задание:

1.Отредактируй текст
2.3 задание 4 вариант
3.4 задание 3 вариант
```

3 лаба.

1-3) Вывод спиралью

3 – 6) Переставление блоков

7) Сортировка

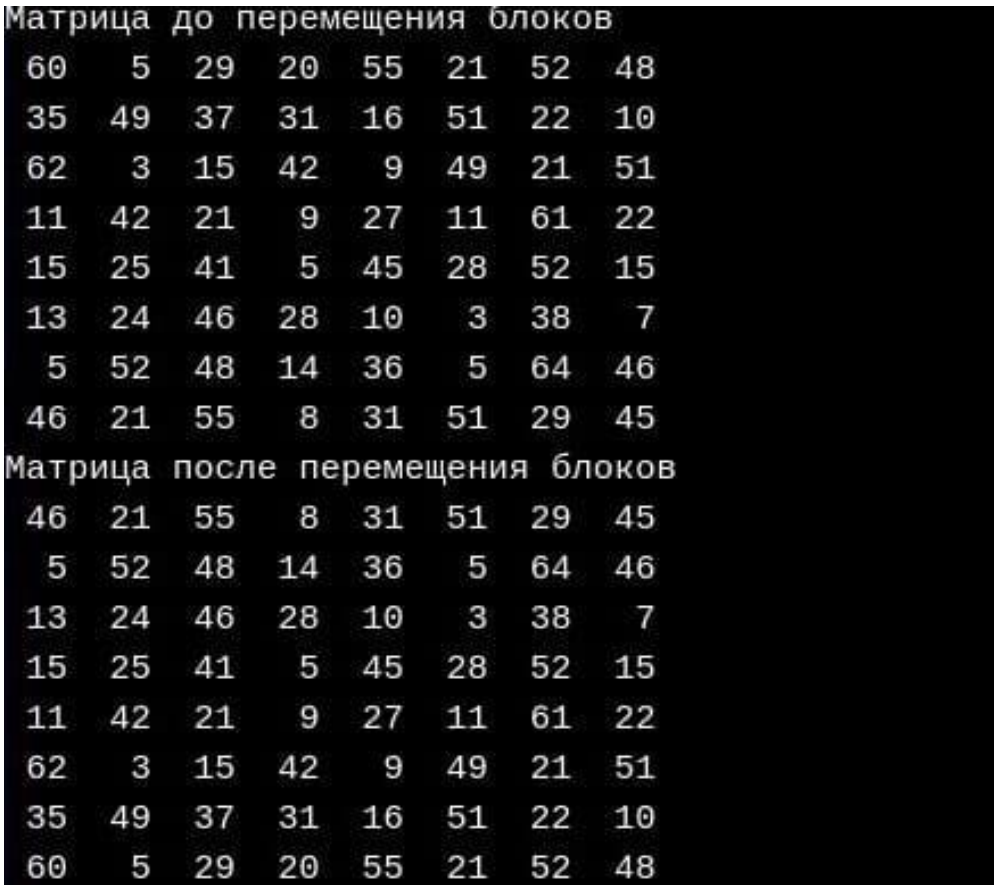
8 – 11) Изменение массива на заданное число

```
Матрица до заполнения
 20  35  10   8   6   8
 11  13  34   2  15  32
 15   8  24  23  25  19
 17  29  36   9  16  34
 27  27  27  32  20  20
  6  19  19  31   6   4
Так выглядит массив размерности 6*6 в пункте а:
  1  2  3  4  5  6
 20 21 22 23 24  7
 19 32 33 34 25  8
 18 31 36 35 26  9
 17 30 29 28 27 10
 16 15 14 13 12 11
```

```
Матрица до заполнения
 50   25   36   27   38   30    6   24
 25   42   31   21   44   51   14    7
 28   53   21   18   15    3   53    2
 34   62   29    8   49   42   63   34
  2   34   61   40   63    2   63   24
 43   29   44   23   16   57   29   43
 45   49   60   60   51   48   61   21
 45   25   28   29    3   27   63    4
Так выглядит массив размерности 8*8 в пункте а:
  1    2    3    4    5    6    7    8
 28   29   30   31   32   33   34    9
 27   48   49   50   51   52   35   10
 26   47   60   61   62   53   36   11
 25   46   59   64   63   54   37   12
 24   45   58   57   56   55   38   13
 23   44   43   42   41   40   39   14
 22   21   20   19   18   17   16   15
Выберите размер матрицы:
```

```
Матрица до заполнения
 96  71  35  79  68   2  98   3  18  93
 53  57   2  81  87  42  66  90  45  20
 41  30  32  18  98  72  82  76  10  28
 68  57  98  54  87  66   7  84  20  25
 29  72  33  30   4  20  71  69   9  16
 41  50  97  24  19  46  47  52  22  56
 80  89  65  29  42  51  94   1  35  65
 25  15  88  57  44  92  28  66  60  37
 33  52  38  29  76   8  75  22  59  96
 30  38  36  94  19  29  44  12  29  30
Так выглядит массив размерности 10*10 в пункте а:
  1    2    3    4    5    6    7    8    9   10
 36   37   38   39   40   41   42   43   44   11
 35   64   65   66   67   68   69   70   45   12
 34   63   84   85   86   87   88   71   46   13
 33   62   83   96   97   98   89   72   47   14
 32   61   82   95  100   99   90   73   48   15
 31   60   81   94   93   92   91   74   49   16
 30   59   80   79   78   77   76   75   50   17
 29   58   57   56   55   54   53   52   51   18
 28   27   26   25   24   23   22   21   20   19
Выберите размер матрицы:
```

```
Матрица до перемещения блоков
 21   29    8    8   18    3
 15   17   17   23   13    1
 34   10   30    1   21    4
  7   12   27   24   29   30
 29    7   30   12    3   35
 22    3    7    9   26    4
Матрица после перемещения блоков
 22   3   7   9  26   4
 29   7  30  12   3  35
  7  12  27  24  29  30
 34  10  30   1  21   4
 15  17  17  23  13   1
 21  29   8   8  18   3
```


Матрица до перемещения блоков
60 5 29 20 55 21 52 48
35 49 37 31 16 51 22 10
62 3 15 42 9 49 21 51
11 42 21 9 27 11 61 22
15 25 41 5 45 28 52 15
13 24 46 28 10 3 38 7
5 52 48 14 36 5 64 46
46 21 55 8 31 51 29 45
Матрица после перемещения блоков
46 21 55 8 31 51 29 45
5 52 48 14 36 5 64 46
13 24 46 28 10 3 38 7
15 25 41 5 45 28 52 15
11 42 21 9 27 11 61 22
62 3 15 42 9 49 21 51
35 49 37 31 16 51 22 10
60 5 29 20 55 21 52 48

```
Матрица до перемещения блоков
 91   85   77   43   37    8   46   57   80   19
 88   13   49   73   60   10   37   11   43   88
  7    2   14   73   22   56   20  100   22    5
 40   12   41   68    6   29   28   51   85   59
 21   25   23   70   97   82   31   85   93   73
 73   51   26   86   23  100   41   43   99   14
 99   91   25   91   10   82   20   37   33   56
 95    5   80   70   74   77   51   56   61   43
 80   85   94    6   22   68    5   14   62   55
 27   60   45    3    3    7   85   22   43   69
Матрица после перемещения блоков
 27  60  45   3   3   7  85  22  43  69
 80  85  94   6  22  68   5  14  62  55
 95   5  80  70  74  77  51  56  61  43
 99  91  25  91  10  82  20  37  33  56
 73  51  26  86  23 100  41  43  99  14
 21  25  23  70  97  82  31  85  93  73
 40  12  41  68   6  29  28  51  85  59
  7   2  14  73  22  56  20 100  22   5
 88  13  49  73  60  10  37  11  43  88
 91  85  77  43  37   8  46  57  80  19
```

```
>>> 3
Введите порядок матрицы: 5
Введите максимальное число для генерации рандомных чисел: 25
Матрица до сортировки:
  9  12   3  16  19
 11  12  18  25  22
 13   3  16  10  14
  2  16   2  23  12
 12  19  18   5   8
Матрица после сортировки:
  2   2   3   3   5
  8   9  10  11  12
 12  12  12  13  14
 16  16  16  18  18
 19  19  22  23  25
```

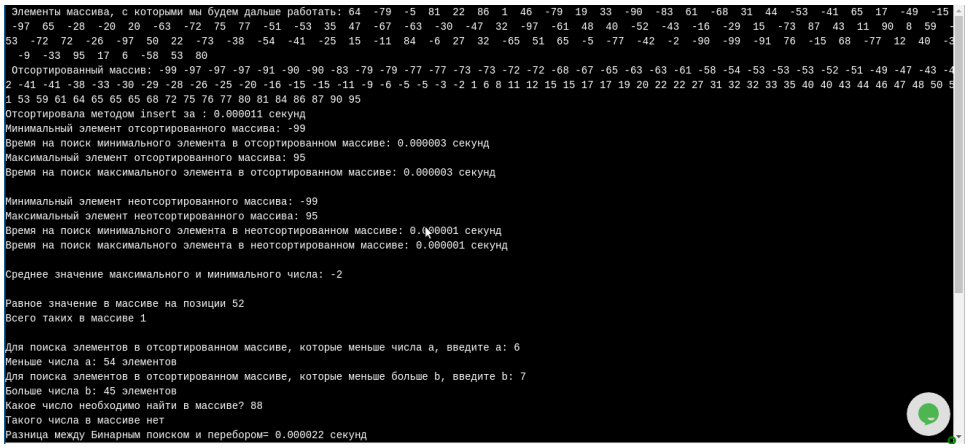
```
Введите порядок матрицы: 5
Введите максимальное число для генерации рандомных чисел: 25
Матрица до вычитания:
  6  13  24  18  11
  5   3  23   9  20
 18  19   7  12  18
  5  24  22  20  10
 13  24  25  16  21
Введите вычитаемое: 2
  4  11  22  16   9
  3   1  21   7  18
 16  17   5  10  16
  3  22  20  18   8
 11  22  23  14  19
```

```
Введите порядок матрицы: 6
Введите максимальное число для генерации рандомных чисел: 64
Матрица до сложения:
 14   7  28  53  21  18
 15   3  53   2  34  62
 29   8  49  42  63  34
  2  34  61  40  63   2
 63  24  43  29  44  23
 16  57  29  43  45  49
Введите слагаемое: 3
 17  10  31  56  24  21
 18   6  56   5  37  65
 32  11  52  45  66  37
  5  37  64  43  66   5
 66  27  46  32  47  26
 19  60  32  46  48  52
```

```
>>> 6
Введите порядок матрицы: 5
Введите максимальное число для генерации рандомных чисел: 25
Матрица до умножения:
  4   2   5  25   8
 11   2  19  15  13
  2  12  20  15  21
 21  10   4  18   2
 23   3  18  18   3
Введите множетель: 2
  8   4  10  50  16
 22   4  38  30  26
  4  24  40  30  42
 42  20   8  36   4
 46   6  36  36   6
Выберите задание:
```

```
Введите порядок матрицы: 5
Введите максимальное число для генерации рандомных чисел: 25
Матрица до деления:
  7   2   6  12  17
 16  15  20  20  16
  5   7  18  23  22
  7   1  10   3  18
  7  23   4  12  16
Введите делитель: 2
  3.5     1     3     6   8.5
    8   7.5    10    10     8
  2.5   3.5     9  11.5    11
  3.5   0.5     5   1.5     9
  3.5  11.5     2     6     8
Выберите задание:
```

2 лаба



1 лаба

```
int: 4
short int: 2
long int: 4
float: 4
double: 8
long double: 8
char: 1
bool: 1
Вывод на экран двоичного представления в памяти целого числа , Введите число: 128
 Результат: 0 00000000000000000000000010000000
Вывод на экран двоичного представления в памяти числа типа float , Введите число: 3.14
 Результат: 0 10000000 10010001111010111000011
Вывод на экран двоичного представления в памяти числа типа double , Введите число: 3.14
 Результат: 0 10000000000 1001000111101011100001010001111010111000010100011111
Для продолжения нажмите любую клавишу . . .
```