**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

**отчет**

**по курсовой работе №1**

**по дисциплине «Программирование»**

**Тема: «Программирование на языке С++»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 0324 |  | Костюхина Э.В. |
| Преподаватель |  | Глущенко А.Г |

Санкт-Петербург

2020

**Работа №1**

**Цель работы.**

Знакомство с внутренним представлением различных типов данных, используемых компьютером при их обработке.

**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):

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

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

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

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 (исходным 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.

Подобные манипуляции возможны благодаря тому, что int и float занимают 4 байта. Проводя манипуляции над tool, мы изменяем значение numb\_f. Таким образом, алгоритм, который использовался для представления в памяти int может использоваться и для float.

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.

**Постановка задачи.**

Необходимо разработать алгоритм и написать программу, которая позволяет:

1) 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.

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

3) 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.

4) 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.**

Для выполнения поставленной задачи была написана программа на языке C++, которая помогла выполнить поставленную задачу. Итоговый код программы представлен ниже.

Для выполнения первого пункта задания достаточно использовать оператор sizeof, который возвращает размер в байтах объекта или типа данных. Далее вывести результат.

Для выполнения второго пункта задания я создаю маску и с помощью побитового сдвига и операции побитовой конъюнкции получаю двоичное представление в памяти целого числа.

Для выполнения третьего пункта я воспользовалась объединением union (группирует переменные, которые разделяют одну и ту же область памяти). И далее действовала по такой же логике, как и в первом пункте. Отличием является то, что мы проводим все необходимые манипуляции над переменной типа int. Благодаря использованию объединения изменяется так же значение и переменной типа float, потому что они занимают одинаковое количество байтов в памяти.

В четвертом пункте мы действуем аналогично третьему. Только из-за того, что тип double занимает в памяти 8 байт, надо создать массив из двух элементов int, так как он тоже будет занимать 8 байт. Далее мы по очереди работаем с элементами массива и маской.

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

setlocale(0, "");

//1) 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.

cout << setw(12) << "int " << sizeof(int) << " байта" <<endl;

cout << setw(12) << "short int " << sizeof(short int) << " байта" << endl;

cout << setw(12) << "long int " << sizeof(long int) << " байта" << endl;

cout << setw(12) << "float " << sizeof(float) << " байта" << endl;

cout << setw(12) << "double " << sizeof(double) << " байт" << endl;

cout << setw(12) << "long double " << sizeof(long double) << " байт" << endl;

cout << setw(12) << "char " << sizeof(char) << " байт" << endl;

cout << setw(12) << "bool " << sizeof(bool) << " байт" << endl;

//2) Вывести на экран двоичное представление в памяти (все разряды) целого числа. При выводе необходимо визуально обозначить знаковый разряд и значащие разряды отступами или цветом.

int intNum;

unsigned intMask = 1 << 15;

cout << "Введите число типа int: ";

cin >> intNum;

if (intNum < 0) {

cout << "1 ";

intNum \*= -1;

} else cout << "0 ";

intNum <<= 1;

for (int i = 0; i < 15; i++) {

putchar(intNum & intMask ? '1' : '0');

intNum <<= 1;

}

cout << endl;

//3) 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.

union {

float floatNum;

int floatTool;

};

unsigned floatMask = 1 << 31;

cout << "Введите число типа float: ";

cin >> floatNum;

if (floatNum < 0) {

cout << "1 ";

floatNum \*= -1;

}

else cout << "0 ";

floatTool <<= 1;

for (int i = 0; i < 31; i++) {

if (i == 8) cout << " ";

putchar(floatTool & floatMask ? '1' : '0');

floatTool <<= 1;

}

cout << endl;

//4) 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.

union {

double doubleNum;

int doubleTool[2];

};

unsigned doubleMask = 1 << 31;

cout << "Введите число типа double: ";

cin >> doubleNum;

if (doubleNum < 0) {

cout << "1 ";

doubleNum \*= -1;

}

else cout << "0 ";

doubleTool[1] <<= 1;

for (int i = 0; i < 31; i++) {

if (i == 11) cout << " ";

putchar(doubleTool[1] & doubleMask ? '1' : '0');

doubleTool[1] <<= 1;

}

for (int i = 0; i < 32; i++) {

putchar(doubleTool[0] & doubleMask ? '1' : '0');

doubleTool[0] <<= 1;

}

return 0;

}

**Результаты работа программы.**

**Выводы.**

Проверить результаты работы программы можно с помощью онлайн конвертора:

Результаты работы программы совпадают с вышеприведенными.

**Работа №2**

**Цель работы.**

Знакомство с одномерными статическими массивами, способами сортировки и бинарным поиском.

**Основные теоретические положения.**

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.

Сортировка – процесс размещения элементов заданного множества объектов в определенном порядке. Когда элементы отсортированы, их проще найти, производить с ними различные операции. Сортировка напрямую влияет на скорость алгоритма, в котором нужно обратиться к определенному элементу массива. Большинство алгоритмов по обработке массивов реализуются с помощью циклов. Ввод и вывод массивов не являются исключением.

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.

**Постановка задачи.**

Необходимо написать программу, которая:

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.

Должна присутствовать возможность запуска каждого пункта многократно.

**Выполнение работы.**

Для выполнения поставленной задачи была написана программа на языке C++, которая помогла выполнить поставленную задачу. Итоговый код программы представлен ниже.

Для упрощения программы я воспользовалась функциями, написав отдельную функцию для каждого пункта задания.

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

Для выполнения первого пункта задания я создала функцию returтMas(), которая выводит массив и функцию randomMas(), которая заполняет массив случайными значениями (arr[i] = rand() % 199 - 99;).

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

Для выполнения второго пункта задания я создала функцию sort(), которая сортирует заданный ранее массив. Для этого я использовала самую стандартную сортировку “bubble sort”.

Так же функция timeSort() выводит время, которое было затрачено на сортировку в микросекундах. (используется библиотека chrono)

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

Для выполнения третьего пункта я создала функцию minMaxRandom(), которая находит максимальные и минимальный элемент в неотсортированном массиве, а также выводит время, затраченное на поиск.

И функцию minMaxSort(), которая делает все то же самое в отсортированном массиве.

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

Для выполнения четвертого пункта была создана функция average(), которая выводит среднее значение максимального и минимального значения и индексы элементов, которые равны этому значению.

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

Функция numA() выводит количество элементов, которые меньше числа, заданного пользователем.

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

Функция numB() выводит количество элементов, которые больше числа, заданного 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. (\*)

В первой части программы binSearch() реализуется алгоритм бинарного поиска элемента и подсчитывается время, затраченное ан поиск.

Во второй части программы реализуется поиск элемента при помощи обычного перебора, подсчитывается время и сравнивается со временем работы бинарного поиска. Если элемент был не найден в массиве, то программа говорит об этом.

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

Функция change() меняет местами элементы с индексами, которые задает пользователь и подсчитывает время замены.

**Результаты работа программы.**

В начале в консоль выводится список заданий, которые программа может выполнить.

Далее нужно выбрать номер задания.

Пример работы первого задания:

Пример работы второго задания:

Пример работы третьего задания:

Пример работы четвертого задания:

Пример работы пятого задания:

Пример работы шестого задания:

Пример работы седьмого задания:

Пример работы восьмого задания:

**Выводы.**

С одномерными статическими массивами работать легче, когда они отсортированы. Так же важно знать разные виды сортировок, которые могут пригодиться в разных ситуациях и заданиях и уметь пользоваться бинарным поиском.

**Работа №3**

**Цель работы.**

Знакомство с указателями. Научиться применять арифметику указателий при работе с матрицами.

**Основные теоретические положения.**

Указатели и ссылки являются одними из самых важных и достаточно сложных для понимания и использования средств языка программирования. Они ориентированы на прямую работу с памятью компьютера. С помощью этих средств реализуется работа с динамической памятью и динамическими объектами, возвращение из функций измененных данных и многое другое.

Указатели – это тоже обычные переменные, но они служат для хранения адресов памяти.

Указатели поддерживают ряд операций: присваивание, получение адреса указателя, получение значения по указателю, некоторые арифметические операции и операции сравнения. Указателю можно присвоить либо адрес объекта того же типа, либо значение другого указателя. Когда указателю присваивается другой указатель, то фактически первый указатель начинает также указывать на тот же адрес, на который указывает второй указатель. Указатель хранит адрес переменной, и по этому адресу мы можем получить значение этой переменной. Но кроме того, указатель, как и любая переменная, сам имеет адрес, по которому он располагается в памяти.

К указателям можно применять некоторые арифметические операции. К таким операциям относятся: +, -, ++, --. Результаты выполнения этих операций по отношению к указателям существенно отличаются от результатов соответствующих арифметических операций, выполняющихся с обычными числовыми данными. Добавлять к указателям или вычитать из указателей можно только целые значения.

Основная и наиболее часто встречающаяся ошибка при работе с указателями связана с использованием неинициализированных указателей. Вторая группа ошибок может быть связана с некорректным использованием арифметики указателей. При некорректном выполнении наращивания или уменьшения указателей с помощью операций + или - можно выйти за пределы предполагаемого объекта (например, массива) и в результате получить неверные данные или модифицировать не те значения.

**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.

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.**

Для выполнения поставленной задачи была написана программа на языке C++. Итоговый код программы представлен ниже.

В программе я использовала динамические двумерные массивы, чтобы в памяти выделялось место именно под ту размерность матрицы, которую вводит пользователь. Так же программа разбита на функции для упрощения.

Сначала я создаю три динамических массива, с которыми в последствии буду работать (с помощью функции sozdanie(int size), аргументом является размер матрицы, который вводит пользователь).

Далее для выполнения первого пункта я получаю положение каретки и вызываю функцию spiral(int\*\* a, int size, int num, int strStart, int strEnd, int colStart, int colEnd, int x, int y). Аргументами являются: динамический двумерный массив, его размер, четыре счетчика, которые нам нужны для выполнения функции и координаты, с которых будет начинаться рисунок. Функция состоит из четырех циклов, которые по очереди заполняют двумерный массив по определенной схеме, а пользователь может увидеть процесс заполнения, благодаря перемещению каретки по заданным координатам. Так как функция является рекурсивной, она подходит под любые четные размерности матрицы.

Чтобы заполнить матрицу по схеме б), была создана другая рекурсивная функция: snake(int\*\* a, int size, int str, int col, int x, int y). Аргументами так же являются динамический двумерный массив, размер массива, два счетчика и координаты. Функция состоит из двух циклов, которые по очереди заполняют двумерный массив.

Для выполнения второго пункта используется функция zadanie\_2(int\*\* a, int\*\* C, int size, int X, int Y). Функция переставляет блоки матрицы в соответствии с указанными схемами.

Для выполнения третьего пункта была написана функция сортировки для двумерного массива sort(int\*\* a, int size). Сначала значения упорядочиваются по строкам и столбцам, далее если последний элемент в стоке больше первого элемента в следующей строке, то они меняются местами, и значения опять упорядочиваются по строкам и столбцам, и так продолжается пока массив не будет полностью отсортирован.

Для выполнения четвертого пункта используется функция actions(int\*\* A, int\*\* C, int N, int znach, int size). Которая принимает на два вход массива, число, введенное пользователем и размер массива. Функция копирует массив A в массив С и далее работает с массивом С.

Как вы могли заметить в программе присутствует функция copy(int\*\* A, int\*\* C, int size). Она создана для того, чтобы работать с двумя массивами, заполненными в первом пункте, посредством создания их копий.

**Результаты работа программы.**

**Выводы.**

Программа работает правильно. Все полученные результаты соответствуют теоретическим данным.

**Работа №4**

**Цель работы.**

Знакомство с текстовыми строками как массивами символов.

**Основные теоретические положения.**

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

**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 слова последовательности, не содержащие цифр.

4) Вывести на экран количество символов в каждом слове исходной последовательности.

**Выполнение работы.**

Для выполнения поставленной задачи была написана программа на языке C++.

Программа разбита на функции:

Sites() – дает пользователю возможность выбрать способ ввода последовательности;

StrLen(string text) – рассчитывает длину строки text;

Whitespace(string text, int length) – убирает лишние пробелы в строке text ,длина которой равно length;

Punctuation(string text, int length) – убирает лишние знаки препинания в строке text ,длина которой равно length;

Lettet\_case(string text, int length) – меняет регистр букв в строке text ,длина которой равно length;

Chisla(string word) – проверяет содержит ли слово word числа.

Zadanie\_3(tring text, int length) – выводит слова без чисел в строке text, длина которой равно length;

Zadanie\_4(string text) – выводит строку text с количеством букв в каждом слове;

Lin\_saerch(string text, int length) – линейный поиск в строке text подстроки, которую вводит пользователь;

**Результаты работа программы.**