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

ОТЧЕТ

по практической работе №1

по дисциплине «Программирование»

ТЕМА: РАЗРАБОТАТЬ АЛГОРИТМ И НАПИСАТЬ ПРОГРАММУ

СТУДЕНТ ГР.0324 _____ КОШЕЛЯЕВ А.С

ПРЕПОДАВАТЕЛЬ _____ ГЛУЩЕНКО А.Г

САНКТ-ПЕТЕРБУРГ

2020

Цель работы.

Разработать алгоритм и написать программу на языке C++, которая позволяет:

Вывести на экран двоичное представление разных типов данных.

Основные теоретические положения.

C++ — это строго типизированный язык, который также является статически типизированным; Каждый объект имеет тип, и этот тип никогда не изменяется (не следует путать с статическими объектами данных). При объявлении переменной в коде необходимо либо явно указать ее тип, либо использовать **auto** ключевое слово, чтобы указать компилятору вывести тип из инициализатора.

В отличие от некоторых других языков, в C++ нет универсального базового типа, от которого наследуются все остальные типы. Язык включает множество фундаментальных типов, известных также как встроенные типы.

Тип данных для каждого программного объекта, представляющего данные, определяет.

1. Характер данных (число, со знаком или без знака, целое или с дробной частью, одиночный символ или текст, представляющий последовательность символов и т.д.)
2. Объем памяти, который занимают в памяти эти данные
3. Диапазон или множество возможных значений
4. Правила обработки этих данных: например, допустимые операции

Каждая переменная имеет определенный тип. И этот тип определяет, какие значения может иметь переменная, какие операции с ней можно производить и сколько байт в памяти она будет занимать. В языке C++ определены следующие базовые типы данных:

- **bool:** логический тип. Может принимать одну из двух значений **true** (истина) и **false** (ложь). Размер занимаемой памяти для этого типа точно не определен. (по умолчанию 1 байт)
- **char:** представляет один символ в кодировке ASCII. Занимает в памяти 1 байт (8 бит). Может хранить любое значение из диапазона от -128 до 127, либо от 0 до 255
- **short:** представляет целое число в диапазоне от –32768 до 32767. Занимает в памяти 2 байта (16 бит). Данный тип также имеет синонимы **short int**, **signed short int**, **signed short**
- **int:** представляет целое число. В зависимости от архитектуры процессора может занимать 2 байта (16 бит) или 4 байта (32 бита). Диапазон предельных значений соответственно также может варьироваться от –32768 до 32767 (при 2 байтах) или от −2 147 483 648 до 2 147 483 647 (при 4 байтах). Но в любом случае размер должен быть больше или равен размеру типа **short** и меньше или равен размеру типа **long**
- **long:** представляет целое число в диапазоне от −2 147 483 648 до 2 147 483 647. Занимает в памяти 4 байта (32 бита). У данного типа также есть синонимы **long int**
- **float:** представляет вещественное число ординарной точности с плавающей точкой в диапазоне +/- 3.4E-38 до 3.4E+38. В памяти занимает 4 байта (32 бита)
- **double:** представляет вещественное число двойной точности с плавающей точкой в диапазоне +/- 1.7E-308 до 1.7E+308. В памяти занимает 8 байт (64 бита)
- **long double:** представляет вещественное число двойной точности с плавающей точкой не менее 8 байт (64 бит). В зависимости от размера занимаемой памяти может отличаться диапазон допустимых значений.

- И прочие…

На рис. 1 показаны относительные размеры встроенных типов в реализации Microsoft C++:

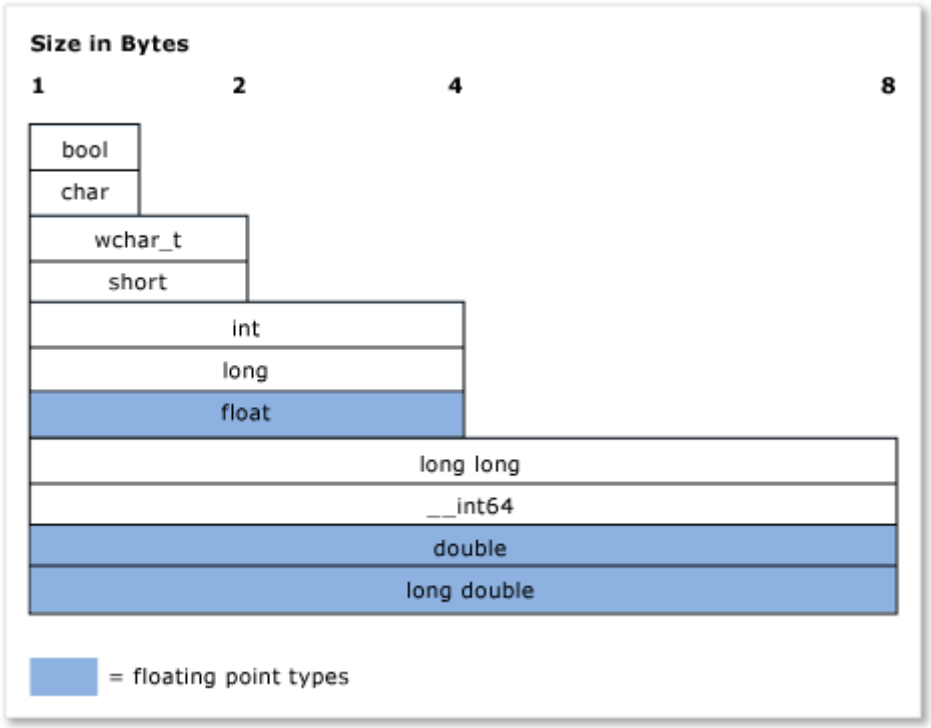

Рисунок 1 - относительные размеры встроенных типов в реализации Microsoft C++

Типы данных можно разделить на две группы: **простые и структурированные типы.**

Простые типы данных представляют неразделимые данные, не имеющие внутренней структуры это, например числа символы и т. д.

Структурированные типы данных имеют внутреннюю структуру (иногда достаточно сложную). Структурированные типы строятся на основе простых типов данных.

Другой уровень классификации разделяет все типы данных на **предопределенные** (изначально встроенные в язык программирования) и **пользовательские** (типы данных, определяемые программистом)

Классификация простых предопределённых типов данных.

Основные (предопределённые) типы данных часто называют арифметическими поскольку их можно использовать в арифметических операциях.

- **int (целый)**
- **float (вещественный)**
- **double (вещественный тип с двойной точностью)**
- **bool (логический)**
- **char (символьный)**

Типы **int**, **bool**, **char** относятся к группе целочисленных (целых) типов, а **float**, **double** к группе вещественных типов- типов с плавающей точкой. Код, который формирует компилятор для обработки целых величин отличается от кода для величин с плавающей точкой.

Существуют спецификаторы типа уточняющих внутренние представление и диапазон значений стандартных типов:

- **short (короткий)**
- **long (длинный)**

Для представления информации в памяти (как числовой, так и не числовой) используется двоичный способ кодирования. Байт является основной единицей измерения объема памяти.

С каждым байтом памяти связано понятие адреса, которые по сути являются номером байта в непрерывной последовательности байтов памяти компьютера. То есть каждый байт памяти имеет свой адрес. По этому адресу и осуществляется доступ к данным, хранящимся в памяти.

Адресное пространство программного режима 32 битного процессора (для 64 бит все по аналогии). Адресное пространство этого режима будет состоять из 2^32 ячеек памяти пронумерованных от 0 и до 2^32-1. (2^64 ячеек памяти пронумерованной от 0 до 2^64-1).Программист работает с этой

памятью, если ему нужно определить переменную, он просто говорит ячейка памяти с адресом таким-то будет содержать такой-то тип данных, при этом сам программист может и не знать какой номер у этой ячейки он просто напишет что-то вроде:

Int a=10;

Компьютер поймет это так: нужно взять какую-то ячейку с номером X и поместить в нее цело число 10. При том про адрес ячейки 18896 вы и не узнаете, он от вас будет скрыт. Все бы хорошо, но возникает вопрос, а как компьютер ищет эту ячейку памяти, ведь память у нас может быть разная: (3 уровень кэша, 2 уровень кэша, 1 уровень кэша, основная память, жесткий диск) Это все разные памяти, но компьютер легко находит в какой из них лежит наша переменная **int a**. Этот вопрос решается операционной системой совместно с процессором…

Внутреннее представление величин целого типа – целое число в двоичном коде. 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. Сразу отмечу, что положительные числа в двоичном коде вне зависимости от способа представления *(***прямой, обратный или дополнительный коды***)* имеют одинаковый вид.

Прямой код — способ представления двоичных чисел с фиксированной запятой. Главным образом используется для записи неотрицательных чисел. Прямой код используется в двух вариантах.
В первом (основной) — для записи только неотрицательных чисел табл.1

Таблица 1 Вид десятичного числа в прямом коде (в 32-битном представлении)

Десятичное число	Двоичное число в прямом коде (в 32-битном представлении)
0	0000 0000 0000 0000 0000 0000 0000 0000
255	0000 0000 0000 0000 0000 0000 1111 1111
65 535	0000 0000 0000 0000 1111 1111 1111 1111
16 777 215	0000 0000 1111 1111 1111 1111 1111 1111
4 294 967 295	1111 1111 1111 1111 1111 1111 1111 1111

В этом варианте (для 32-битного двоичного числа) мы можем записать максимальное число 4 294 967 295

Второй вариант – для записи как положительных, так и отрицательных чисел. В этом случае старший бит (в нашем случае – 32-й) объявляется знаковым разрядом (знаковым битом). При этом, если: знаковый разряд равен 0, то число положительное; если 1, то число отрицательное.

Рисунок 2 Вид десятичного числа в прямом коде (в 32-битном представлении со знаковым битом)

В этом случае диапазон целых чисел, которые можно записать в прямом коде составляет от −2 147 483 648 до 2 147 483 647. Прямой код используется главным образом для представления неотрицательных чисел. Использование прямого кода для представления отрицательных чисел

является неэффективным — очень сложно реализовать арифметические операции и, кроме того, в прямом коде два представления нуля — положительный ноль и отрицательный ноль (чего не бывает).

Обратный код - метод вычислительной математики, позволяющий вычесть одно число из другого, используя только операцию сложения над натуральными числами. Обратный n-разрядный двоичный код *положительного* целого числа состоит из одноразрядного кода знака (битового знака “двоичной цифры 0”) за которым следует n-1 разрядное двоичное число (обратный код положительного числа совпадает с прямым кодом).

Пример: Двоичное представление числа 526 есть 10 0000 1110. В 32-разрядный двоичный код числа +526 записывается как 0000 0000 0000 0000 0000 0010 0000 1110.

Обратный *n*-разрядный двоичный код *отрицательного* целого числа состоит из одноразрядного кода знака (двоичной цифры 1), за которым следует n-1 разрядное двоичное число, представляющее собой инвертированное n-1 разрядное двоичное число. Для отрицательных чисел обратный код получается из неотрицательного числа в прямом коде, путём инвертирования всех битов (1 меняем на 0, а 0 на 1). Следует отметить, что для изменения знака числа достаточно проинвертировать все его разряды, не обращая внимания, знаковый ли это разряд или информационный.

Пример: Двоичное представление числа 526 есть 10 0000 1110, его 32-разрядное двоичное представление 0000 0000 0000 0000 0000 0010 0000 1110. Обратный 32-разрядный двоичный код числа -526 есть 1111 1111 1111 1111 1111 1101 1111 0001

Для преобразования отрицательного числа в положительное тоже применяется операция инвертирования. Этим обратные коды удобны в

применении. В качестве недостатка следует отметить, что в обратных двоичных кодах имеются два кода числа 0 положительный ноль и отрицательный ноль (чего не бывает). Это приводит к некоторому усложнению операции суммирования.

Дополнительный код- наиболее распространенный способ представления отрицательных чисел. Он позволяет заменить операцию вычитания на операцию сложения и сделать операции сложения и вычитания одинаковыми для знаковых и беззнаковых чисел. (В англоязычной литературе обратный код называют *первым дополнением*, а *дополнительный код* называют *вторым дополнением*). В дополнительном коде (как и в прямом и обратном) старший разряд отводится для представления знака числа (знаковый бит). Дополнительный код для отрицательного числа можно получить инвертированием его двоичного модуля (первое дополнение) и прибавлением к инверсии единицы (второе дополнение), либо вычитанием числа из нуля.

Десятичное представление	Двоичное представление (8 бит)		
	прямой	обратный	дополнительный
127	0111 1111	0111 1111	0111 1111
1	0000 0001	0000 0001	0000 0001
0	0000 0000	0000 0000	0000 0000
-0	1000 0000	1111 1111	
-1	1000 0001	1111 1110	1111 1111
-2	1000 0010	1111 1101	1111 1110
-3	1000 0011	1111 1100	1111 1101
-4	1000 0100	1111 1011	1111 1100
-5	1000 0101	1111 1010	1111 1011
-6	1000 0110	1111 1001	1111 1010
-7	1000 0111	1111 1000	1111 1001
-8	1000 1000	1111 0111	1111 1000
-9	1000 1001	1111 0110	1111 0111
-10	1000 1010	1111 0101	1111 0110
-11	1000 1011	1111 0100	1111 0101
-127	1111 1111	1000 0000	1000 0001
-128	---	---	1000 0000

Рисунок 3 Представления десятичного целого числа в разных видах кода.

Получение дополнительного кода отрицательного числа

Десятичное число	Двоичное число в прямом коде	Инвертирование значения (обратный код)	Прибавляем к инверсии единицу	Двоичное число в дополнительном коде
127	0111 1111	Положительные значения не меняются		0111 1111
10	0000 1010			0000 1010
0	0000 0000			0000 0000
- 0	1000 0000	-----	-----	-----
- 5	1000 0101	1111 1010	+1	1111 1011
- 10	1000 1010	1111 0101	+1	1111 0110
- 127	1111 1111	1000 0000	+1	1000 0001
- 128	-----			1000 0000

Рисунок 4 Получение дополнительного кода отрицательного числа в дополнительном коде.

Вывод:

1. Для арифметических операций сложения и вычитания положительных двоичных чисел наиболее подходит применение прямого кода.
2. Для арифметических операций сложения и вычитания отрицательных двоичных чисел наиболее подходит применение дополнительного кода.

Вещественные типы данных хранятся в памяти компьютера иначе, чем целочисленные. Вещественные числа обычно представляются в виде чисел с плавающей запятой. Числа с плавающей запятой — один из возможных способов представления действительных чисел, который является компромиссом между точностью и диапазоном принимаемых значений, его можно считать аналогом экспоненциальной записи чисел, но только в памяти компьютера. При этом лишь некоторые из вещественных чисел могут быть представлены в памяти компьютера точным значением, в то время как

остальные числа представляются приближёнными значениями. Более простым вариантом представления вещественных чисел является вариант с фиксированной точкой, когда целая и вещественная части хранятся отдельно. Например, на целую часть отводится всегда *X* бит и на дробную отводится всегда *Y* бит. Такой способ в архитектурах процессоров не присутствует. Отдаётся предпочтение числам с плавающей запятой, как компромиссу между диапазоном допустимых значений и точностью.

Число с плавающей запятой состоит из набора отдельных двоичных разрядов, условно разделенных на так называемые **знак** (англ. *sign*), **порядок** (англ. *exponent*) и **мантиссу** (англ. *mantis*). В наиболее распространённом формате (стандарт IEEE 754) число с плавающей запятой представляется в виде набора битов, часть из которых кодирует собой мантиссу числа, другая часть — показатель степени, и ещё один бит используется для указания знака числа (0 — если число положительное, 1 — если число отрицательное). При этом порядок записывается как целое число в коде со сдвигом, а мантисса —в нормализованном виде, своей дробной частью в двоичной системе счисления. Вот пример такого числа из 16 двоичных разрядов:

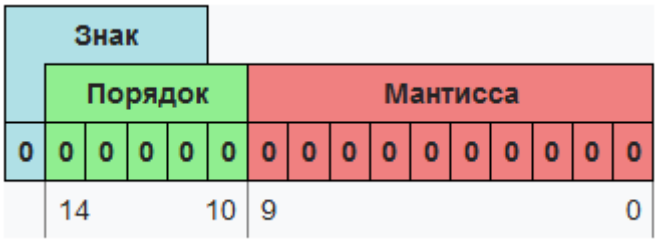

Рисунок 5 Представление вещественного числа в 16-разрядном двоичном представление.

Знак — один бит, указывающий знак всего числа с плавающей точкой.

Порядок также иногда называют **экспонентой** или просто **показателем степени**. Порядок и мантисса — целые числа, которые вместе со знаком дают представление числа с плавающей запятой.

Число одинарной точности — компьютерный формат представления чисел, занимающий в памяти одно машинное слово (в случае 32-битного компьютера — 32 бита или 4 байта). Используется для работы с вещественными числами везде, где не нужна очень высокая точность.

Рисунок 6 Число одинарной точности **float.**

Порядок записан со сдвигом −127.

Число двойной точности — компьютерный формат представления чисел, занимающий в памяти два машинных слова (в случае 32-битного компьютера — 64 бита или 8 байт). Часто используется благодаря своей неплохой точности, даже несмотря на двойной расход памяти и сетевого трафика относительно чисел одинарной точности.

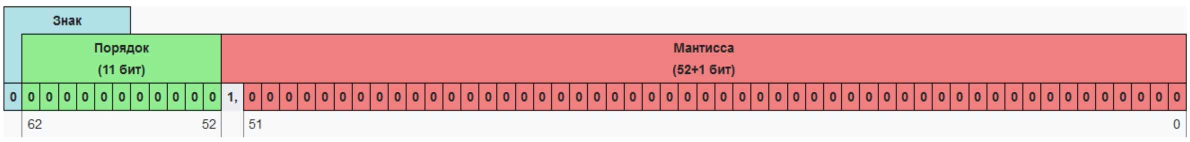

Рисунок 7 Число двойной точности **double.**

Порядок записан со сдвигом −1023.

Диапазон значений чисел с плавающей запятой, которые можно записать данным способом, зависит от количества бит, отведённых для представления мантиссы и показателя. Пара значений показателя (когда все разряды нули и когда все разряды единицы) зарезервирована для обеспечения возможности представления специальных чисел. К ним относятся ноль, значения NaN (Not a Number, "не число", получается, как результат операций типа деления нуля на ноль) и ±∞.

Постановка задачи.

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 поставленных задач подойдет язык программирования c++. Поэтому была написана программа на c++, которая помогла выполнить поставленную задачу. Итоговый код программы представлен в приложении А.

Для выполнения первой задачи нам подойдёт **Sizeof** — широко используемый оператор c++. Это унарный оператор времени компиляции, который может использоваться для вычисления размера его операнда. Результат sizeof имеет целочисленный тип без знака, который обычно обозначается size_t. sizeof может применяться к любому типу данных, включая примитивные типы, такие как целочисленные и с плавающей точкой, типы указателей или составные типы данных.

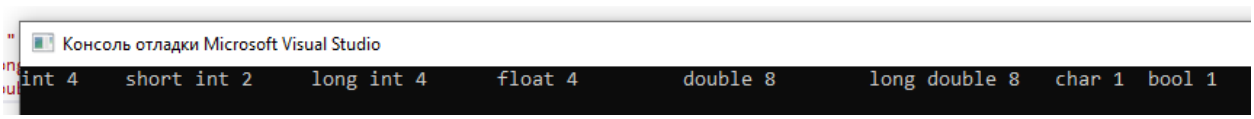

Рисунок 8 консольный вывод выполнения программы.

Выполняя вторую задачу тип данных должен быть целым числом так что используем **int** для задания переменной и побитовые операторы “Побитовое И **a & b,** побитовый сдвиг влево **<<,** побитовый сдвиг влево, совмещённый с присваиванием **<<= “** цикл **for** и знания, полученные при прохождении stepik.org составим несложный алгоритм результат работы которого представлен на рис.9

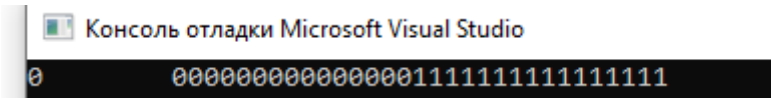

Рисунок 9 Консольный вывод числа 65535 в двоичном виде

Первое число в нашем случае 0 это знаковый бит говорящий о том, что число положительное.

Третья задача ненамного сложней благодаря подсказке. 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. Объединив типы float и int получилось вывести дробное десятичное число разделив его визуально на знаковую часть числа, порядок и мантиссу. Итог работы алгоритма представлен на рис.10

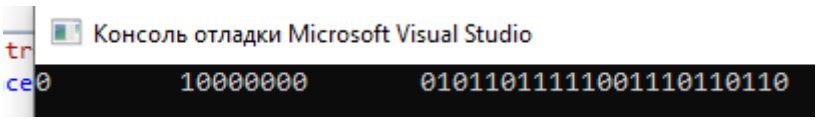

Рисунок 10 консольный вывод числа “e- математической константы” в двоичном виде.

Согласно задания был применён оператор float подробно о выделения памяти показано на рис.6

Выводы.

https://github.com/Aleksey-app/-.git

ПРИЛОЖЕНИЕ А

ПОЛНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
using namespace std;
int main()
{
	cout << "int " << sizeof(int) << "\t short int " << sizeof(short int)
	<< "\t long int " << sizeof(long int) << "\t float " << sizeof(float)
	<< "\t double " << sizeof(double) << "\t long double " << sizeof(long double)
<< "\t char " << sizeof(char) << "\t bool " << sizeof(bool) <<"\n";
	int a = 65535 ;
	int z = 1 << 31;
	for (int i = 0; i < 32; i++)
	{
		if (a&z)
			cout << 1;
		else
			cout << 0;
		a <<= 1;
		if (i == 0)
			cout << "\t ";
	}
		cout << "\n";
	union {
		float b = 2.718;
		int c;
	}x;
	for (int i = 0; i < 32; i++)
	{
		if (z&x.c)
			cout << 1;
		else
			cout << 0;
		x.c <<= 1;
		if (i == 0)
			cout << "\t ";
		if (i == 8)
			cout << "\t";
	}
	cout << "\n";
```

```
}
```