**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра «Информационные Системы»**

отчет

**по практической работе №1**

**по дисциплине «Программирование»**

Тема: «Типы данных и их внутреннее представление в памяти»

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Сомова П. Д. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2020

**Цель работы.**

Знакомство с внутренним представление различных типов данных, используемых компьютером при их обработке. Научиться работать с побитовыми операциями.

**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) показан на рис. 1.



Рисунок 1 – Прямой код числа 3

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 показан на рис. 2.



Рисунок 2 – Обратный код числа -3

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



Рисунок 3 – Дополнительный код числа -3

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 (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 (см. рис. 4).



Рисунок 4

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, используя 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.

#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;

}

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.

Алгоритм 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.

3) Вывести на экран двоичное представление в памяти (все разряды) типа float.

4) Вывести на экран двоичное представление в памяти (все разряды) типа double.

**Выполнение работы.**

Для знакомства с внутренним представление различных типов данных, используемых компьютером при их обработке, и побитовым сдвигом отлично подойдет язык программирования C++. Поэтому была написана программа на C++, которая помогла выполнить поставленную задачу. Итоговый код программы представлен в приложении А. Скриншоты работы программы представлены в приложении B.

Для удобства работы над задачами были созданы функции. Часть функций отвечает за номер поставленных задач, а часть - вспомогательные функции для упрощения работы алгоритмов.

Функция «void SizeOf ()» отвечает за количество памяти (в байтах) выделяемое под различные типы. Для этого была использована встроенная функция sizeof(тип).

Функция «void BinaryInt ()» отвечает за двоичное представление в памяти целого (int) числа. Для удобства реализации алгоритма для данной задачи была создана функция «int RankValueInt(int)», определяющая количество значащих разрядов, а также использован левый побитовый сдвиг.

Функции «void BinaryFloat ()» и «void BinaryDouble ()» отвечают за двоичное представление в памяти вещественных типов float и double соответственно. Были введены объединения ( union {}) и для double массив из двух int. Это было сделано в связи своей «специфичности» данных типов. А также использован левый побитовый сдвиг.

Для удобства оформления программы и вывода данных были введены еще 3 функции. Функция «short int MenuTasks ()» перечисляет задачи, которые может выполнить программа. Функция «void DoTasks (int)» зависит от предыдущей функции и выполняет выбранную пользователем задачу. И функция «void DoLabWork1 ()» отвечает за выполнение всей практической работы. Данная функция была введена для того, чтобы функцию «int main()» не загромождать и при ее просмотре было сразу понятно, что делает программа. К тому же в функции «int main()» произведена русификация с помощью функции «setlocale( LC\_ALL, "Russian")»

Также введена глобальная константа «const int BIT = 8», которая обозначает «1 байт = 8 бит». Данная константа была введена для того, чтобы алгоритм сам подсчитывал сколько бит отводится в памяти на определенный тип данных, что позволяет сделать код независимым от системы.

**Выводы.**

На основании проделанной работы, можно сделать вывод, что побитовые операции позволяют через двоичное представление типа работать с битами, и убедиться, что каждый тип данных имеет определенное количество памяти и оно зависит от конкретной системы.

Приложение А

Полный код программы

#include <iostream>  
  
using namespace std**;**const int BIT = **8;** // константа: 1 байт = 8 бит  
  
void SizeOf ()**;** // Прототип метода: Количество памяти (в байтах) выделяемое под различные типы  
void BinaryInt ()**;** // Прототип метода: Двоичное представление в памяти целого (int) числа  
int RankValueInt (int)**;** // Прототип функции: Количесво значащих разрядов в int числе  
void BinaryFloat ()**;** // Прототип метода: Двоичное представление в памяти типа float  
void BinaryDouble ()**;** // Прототип метода: Двоичное представление в памяти тип double  
short int MenuTasks ()**;** // Прототип функции: Меню (перечисление задач)  
void DoTasks (int)**;** // Прототип метода: Выполнение задач по выбору  
void DoLabWork1 ()**;** // Прототип метода: Выполнение лабораторной работы номер 1  
  
int main() {  
  
 setlocale( LC\_ALL**,** "Russian")**;** DoLabWork1()**;** return **0;**}  
  
/\*  
 \* Метод: Определяет количество памяти (в байтах) выделяемое под различные типы  
 \*/  
void SizeOf () {  
  
 cout << "int: " << sizeof(int) << "\n"**;** cout << "short int: " << sizeof(short int) << "\n"**;** cout << "long int: " << sizeof(long int) << "\n"**;** cout << "float: " << sizeof(float) << "\n"**;** cout << "double: " << sizeof(double) << "\n"**;** cout << "long double: " << sizeof(long double) << "\n"**;** cout << "char: " << sizeof(char) << "\n"**;** cout << "bool: " << sizeof(bool) << "\n"**;**}  
  
/\*  
 \* Метод: Двоичное представление в памяти int числа  
 \* 1. Ввод int числа с клавиатуры  
 \* 2. Подсчет значащих разрядов, разрядность маски и самой маски  
 \* 3. С помощью сдвига влево произсводится двоичное представление в памяти int числа  
 \*/  
void BinaryInt () {  
  
 int valueInt**;** cout << "Введите целое число: "**;** cin >> valueInt**;** int rank = RankValueInt(valueInt)**;** unsigned int order = sizeof(int)\*BIT**;** //разрядность для маски  
 unsigned int mask = **1** << order - **1;** for (int i = **1;** i <= order**;** i++) {  
 putchar(valueInt & mask ? '1' : '0')**;** valueInt <<= **1;** if ( (i == **1**) || (i == order - rank) ) {  
 putchar(' ')**;** }  
 }  
  
}  
  
/\*  
 \* Функция: Подсчет количества разрядов в int числе  
 \* Назначение: В двоичном представении int числа отделить от значащих разрядов  
 \*/  
int RankValueInt(int rValueInt) {  
  
 int rankCopy = **0;** int valueIntCopy = rValueInt**;** while (valueIntCopy) {  
 rankCopy++**;** valueIntCopy=valueIntCopy/**2;** }  
 return rankCopy**;**}  
  
/\*  
 \* Метод: Двоичное представление в памяти float числа  
 \* 1. Ввод float числа с клавиатуры  
 \* 2. Подсчет разрядность маски и самой маски  
 \* 3. Двоичное представление в памяти float числа  
 \*/  
void BinaryFloat () {  
  
 union {  
 float valueFloat**;** int tool**;** }**;** cout << "Введите float число: "**;** cin >> valueFloat**;** unsigned int orderFloat = sizeof(float)\*BIT**;** //разрядность для маски  
 unsigned int maskFloat = **1** << orderFloat - **1;** for (int i = **1;** i <= orderFloat**;** i++) {  
  
 putchar( tool & maskFloat ? '1' : '0')**;** if ( (i == **1**) || (i == **9**) ) {  
 putchar(' ')**;** }  
  
 tool <<= **1;** }  
  
}  
  
/\*  
 \* Метод: Двоичное представление в памяти double числа  
 \* 1. Ввод double числа с клавиатуры  
 \* 2. Подсчет разрядность маски и самой маски  
 \* 3. Двоичное представление в памяти double числа через массив из 2 int  
 \*/  
void BinaryDouble () {  
  
 union {  
 double valueDouble**;** int toolDouble[**2**]**;** }**;** cout << "Введите double число: "**;** cin >> valueDouble**;** unsigned int orderDouble = sizeof(double) \* BIT**;** //разрядность для маски  
 unsigned int orderDoubleHalf = orderDouble / **2;** unsigned int maskDouble = **1** << (orderDoubleHalf - **1**)**;** for (int i = **1;** i <= orderDoubleHalf**;** i++) {  
  
 putchar(toolDouble[**1**] & maskDouble ? '1' : '0')**;** if ((i == **1**) || (i == **12**)) {  
 putchar(' ')**;** }  
  
 toolDouble[**1**] <<= **1;** }  
  
 for (int i = orderDoubleHalf + **1;** i <= orderDouble**;** i++) {  
  
 putchar(toolDouble[**0**] & maskDouble ? '1' : '0')**;** toolDouble[**0**] <<= **1;** }  
  
}  
  
/\*  
 \* Функция: Меню. Перечисление задач  
 \* Назначение: Выбор необходимой задачи  
 \*/  
short int MenuTasks () {  
  
 short int numberTask**;** cout << "\n\n"**;** cout << "\*\*\*\n"**;** cout << "Введите номер задачи: \n"**;** cout << "1) Вывести количество памяти (в байтах) выделяемое под различные её типы \n"**;** cout << "2) Вывести двоичное представление в памяти целого (int) числа \n"**;** cout << "3) Вывести двоичное представление в памяти типа float \n"**;** cout << "4) Вывести двоичное представление в памяти типа double \n"**;** while (**1**) {  
  
 cin >> numberTask**;** if ( (numberTask > **0**) && (numberTask < **5**) )  
 break**;** else  
 cout << "Error: Неверно введены данные \n"**;** }  
 cout << "\n"**;** return numberTask**;**}  
  
/\*  
 \* Выполнение необходимой задачи  
 \*/  
void DoTasks (int option) {  
  
 switch (option) {  
 case **1**: {  
 SizeOf()**;** break**;** }  
 case **2**: {  
 BinaryInt()**;** break**;** }  
 case **3**: {  
 BinaryFloat()**;** break**;** }  
 case **4**: {  
 BinaryDouble()**;** break**;** }  
 }  
}  
  
/\*  
 \* Выполнение лабораторной работы номер 1  
 \*/  
void DoLabWork1 () {  
  
 while (**1**) {  
 DoTasks(MenuTasks())**;** }  
}

Приложение Б

скриншоты работы программы

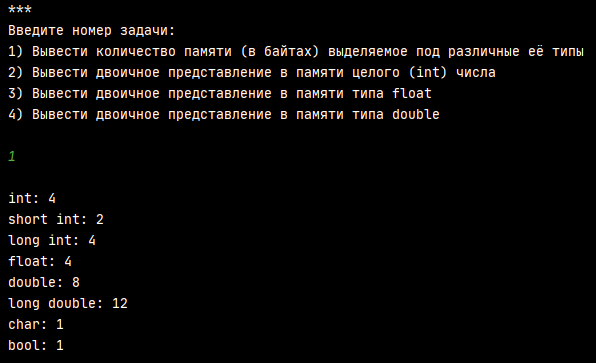


Рисунок 5 – Вывод сколько памяти (в байтах) на вашем компьютере отводится под различные типы данных со спецификаторами и без: int, short int, long int, float, double, long double, char и bool.

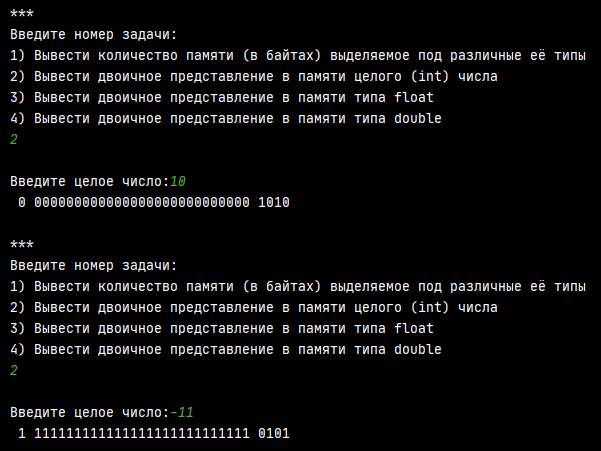


Рисунок 6 – Вывод на экране двоичное представление в памяти (все разряды) целого числа.

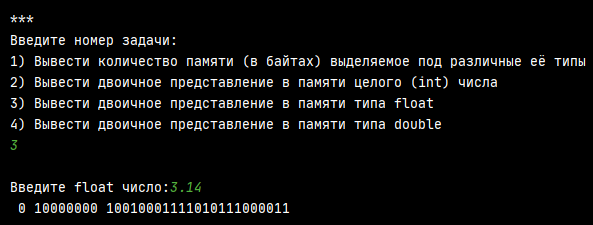


Рисунок 7 – Вывод на экране двоичное представление в памяти (все разряды) типа float.

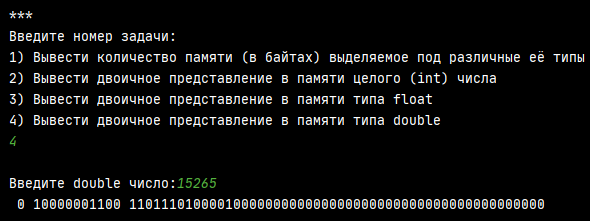


Рисунок 8 – Вывод на экране двоичное представление в памяти (все разряды) типа double.