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

|  |  |  |
| --- | --- | --- |
| Студент гр. 0324 |  | Наквасин А.А. |
| Преподаватель |  | Глущенко А.Г |

Санкт-Петербург

2020

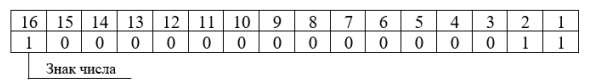
**Цель работы.**

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

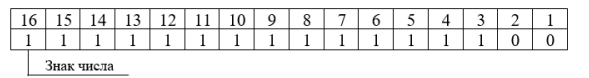
**Основные теоретические положения.**

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.

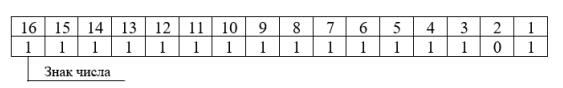
Представление положительных и отрицательных чисел в прямом, обратном и дополнительном кодах отличается. В прямом коде в знаковый разряд помещается цифра 1, а в разряды цифровой части числа – двоичный код его абсолютной величины. Прямой код числа −3 (для 16- разрядного процессора):



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 (&).

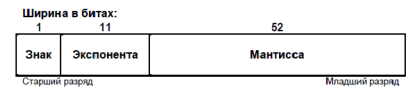


Putchar возвращает один символ в консоль. Альтернатива - cout. В представленном способе, маска - то, с чем сравнивается значение. И побитовый сдвиг применяется для value. Таким образом 1 бит будет сравниваться с каждым битом числа. Альтернатива - побитовый сдвиг вправо, но при этом нужно проводить данную 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.**

Нужно 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 решения поставленных была создана программа на языке программирования C++.

В первой задаче был использован оператор sizeof, который помогает вычислить размер типа данных в байтах и выводит результат в консоль.

Во второй задаче для двоичного представления целого числа была создана маска, а также использовался побитовый сдвиг с побитовой конъюнкцией, в результате чего получено введенное число в двоичной системе счисления.

В третьей задаче было использовано объединение union, благодаря чему мы получаем возможность вывода двоичного представления значения переменной типа float аналогично подобной операции над типом int.

В четвертой задаче также используется union. Однако в данном случае создан массив из двух элементов типа int, так как тип int занимает в памяти 4 байта, тогда как переменная типа double занимает в памяти 8 байт.

**Итоговая программа:**

#include <iostream>

#include <Windows.h>

/\*Black = 0,

Blue = 1,

Green = 2,

Cyan = 3,

Red = 4,

Magenta = 5,

Brown = 6,

LightGray = 7,

DarkGray = 8,

LightBlue = 9,

LightGreen = 10,

LightCyan = 11,

LightRed = 12,

LightMagenta = 13,

Yellow = 14,

White = 15\*/

void SetColor(int text, int background)

{

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, (WORD)((background << 4) | text));

}

int one()

{

std::cout << "\nКоличество байт отведённых в памяти под различные типы данных:\n\n";

std::cout << "int: " << sizeof(int) << "\n" << "short int: " << sizeof(short int) << "\n" << "long int: " << sizeof(long int) << "\n";

std::cout << "float: " << sizeof(float) << "\n" << "double: " << sizeof(double) << "\n" << "long double: " << sizeof(long double) << "\n";

std::cout << "char: " << sizeof(char) << "\n" << "bool: " << sizeof(bool) << "\n\n";

return 0;

}

int two()

{

int a;

int mask = 1 << 31;

std::cout << "\nВаше число: ";

std::cin >> a;

std::cout << "\nЕго двоичное представление: ";

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

{

if (i != 0)

{

SetColor(1,0);

putchar(a & mask ? '1' : '0');

a <<= 1;

SetColor(7,0);

}

else

{

SetColor(4, 0);

putchar(a & mask ? '1' : '0');

a <<= 1;

SetColor(7, 0);

}

}

std::cout << "\n";

return 0;

}

float three()

{

union

{

float a;

int b;

};

int mask = 1 << 31;

std::cout << "\nВаше число: ";

std::cin >> a;

std::cout << "\nЕго двоичное представление: ";

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

{

if (i > 8)

{

SetColor(1, 0);

putchar(b & mask ? '1' : '0');

b <<= 1;

SetColor(7, 0);

}

else if (i > 0 && i < 9)

{

SetColor(2, 0);

putchar(b & mask ? '1' : '0');

b <<= 1;

SetColor(7, 0);

}

else if (i == 0)

{

SetColor(4, 0);

putchar(b & mask ? '1' : '0');

b <<= 1;

SetColor(7, 0);

}

}

std::cout << "\n";

return 0;

}

double four()

{

union

{

double a;

int b[2];

};

int mask = 1 << 31;

std::cout << "\nВаше число: ";

std::cin >> a;

std::cout << "\nЕго двоичное представление: ";

for (int j = 1; j >= 0; j--)

{

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

{

if (j == 0 || i > 11)

{

SetColor(1, 0);

putchar(b[j] & mask ? '1' : '0');

b[j] <<= 1;

SetColor(7, 0);

}

else if (j == 1 && i == 0)

{

SetColor(4, 0);

putchar(b[j] & mask ? '1' : '0');

b[j] <<= 1;

SetColor(7, 0);

}

else if (j == 1 && i < 12)

{

SetColor(2, 0);

putchar(b[j] & mask ? '1' : '0');

b[j] <<= 1;

SetColor(7, 0);

}

}

}

std::cout << "\n";

return 0;

}

int main()

{

int i;

setlocale(0, "");

std::cout << "1 - Вывод количества байт для различных типов данных\n2 - Двоичное представление целого числа\n3 - Двоичное представление типа float\n4 - Двоичное представление типа double\n5 - Выход\n";

menu: std::cout << "\nВыберите пункт меню: ";

std::cin >> i;

switch (i)

{

case 1:

one();

goto menu;

case 2:

two();

goto menu;

case 3:

three();

goto menu;

case 4:

four();

goto menu;

case 5:

break;

default:

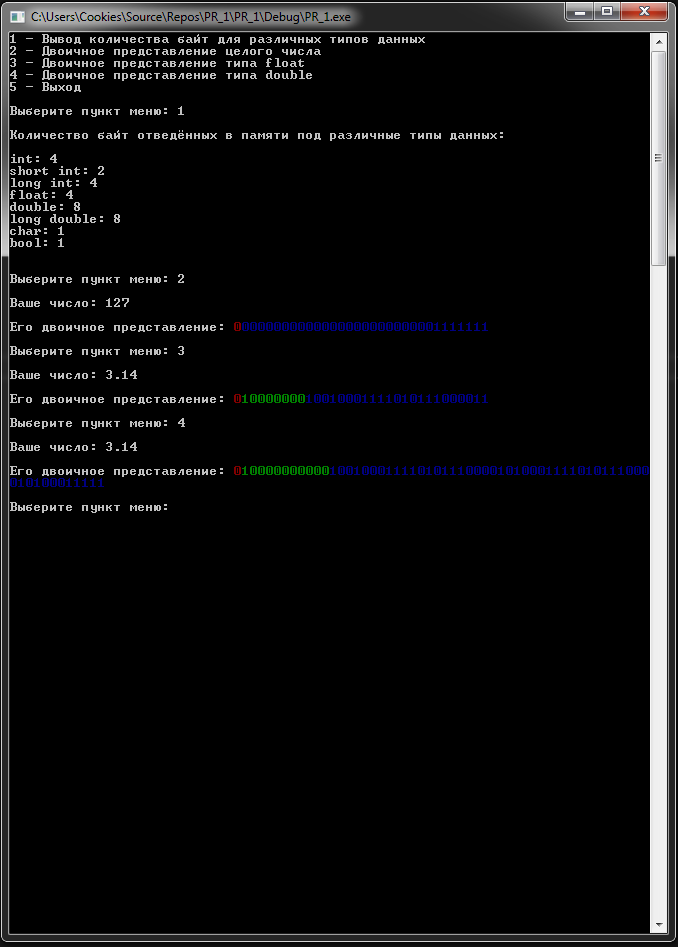
std::cout << "\nТакого пункта не существует\n";

goto menu;

}

}

**Результат работы программы.**

****

**Вывод.**

При разработке программы мною 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.