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, определяемые пользователем. Структуры

Студент гр. *Ильбульдин Р.М., 3372*

Преподаватель _____ Глущенко А. Г.

Санкт-Петербург
2023

Цель работы.

Разработать программу, позволяющий отобразить внутреннее представление в памяти различных типов данных.

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) изображен на рисунке 1.

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

Рисунок 1 – Отображение прямого кода

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 изображен на рисунке 2.

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

Рисунок 2 – Отображение обратного кода

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

Рисунок 3 – Отображение дополнительного кода

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

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 (рисунок 4).

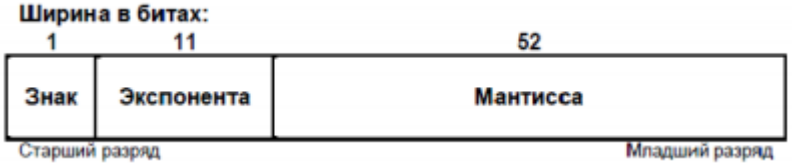

Рисунок 4 – Отображение битов в памяти типа double

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.

Постановка задачи.

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.

Код программы представлен в приложении А.

Выводы.

Данная практическая работа позволяет научится отображать количество памяти (в байтах) различные типы данных, двоичное представление в памяти целых (int), вещественных (float и double) чисел.

ПРИЛОЖЕНИЕ А
РАБОЧИЙ КОД

Листинг А.1 – Рабочий код программы

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

int main()
{
	bool isLive = true;
	while (isLive)
	{
		int selectNum;
		cout << "Please, Select a number to display memory size data types: \n";
		cout << "\t 1 - int\n";
		cout << "\t 2 - short int\n";
		cout << "\t 3 - long int\n";
		cout << "\t 4 - float\n";
		cout << "\t 5 - double\n";
		cout << "\t 6 - long double\n";
		cout << "\t 7 - char\n";
		cout << "\t 8 - bool\n";
		cout << "Number: ";
		cin >> selectNum;
		switch (selectNum)
		{
		case 1:
			cout << "size int: " << sizeof(int) << " bytes\n";
			break;
		case 2:
			cout << "size short int: " << sizeof(short int) << " bytes\n";
			break;
		case 3:
			cout << "size long int: " << sizeof(long  int) << " bytes\n";
			break;
		case 4:
			cout << "size float: " << sizeof(float) << " bytes\n";
			break;
		case 5:
			cout << "size double: " << sizeof(double) << " bytes\n";
			break;
		case 6:
```

```
			cout << "size long double: " << sizeof(long double) << " bytes\n";
			break;
		case 7:
			cout << "size char: " << sizeof(char) << " bytes\n";
			break;
		case 8:
			cout << "size bool: " << sizeof(bool) << " bytes\n";
			break;
		default:
			cout << "Error select number\n";
			break;
		}
		cout << "\n";
		int numberUserInt, tempNumber;
		cout << "Enter please value int number for set some value bit: ";
		cin >> numberUserInt;
		tempNumber = numberUserInt;
		int order = sizeof(numberUserInt) * 8 - 1;
		int mask = 1 << order;
		cout << "Number value: " << numberUserInt << "\n"
			<< "Number link: " << &numberUserInt << "\n"
			<< "Count rank: " << order << "\n"
			<< "Bitwise comparison mask: " << mask << "\n";
		for (int i = 0; i <= order; i++)
		{
			cout << ((numberUserInt & mask) ? 1 : 0);
			numberUserInt <<= 1;
			if (!i || (i + 1) % 8 == 0)
			{
				cout << " ";
			}
		}
		cout << "\n\n";
		numberUserInt = tempNumber;
		int positionUser;
		cout << "Enter please position bit starting from one: ";
		cin >> positionUser;
		int numberBit;
		cout << "Enter please number bit for change: ";
		cin >> numberBit;
		cout << "Enter please position bit = " << positionUser << " for set on " << numberBit
<< ".\n\n";
```

```
switch (numberBit)
{
case 0:
	numberUserInt = (numberUserInt ^ (1 << positionUser - 1));
	break;
case 1:
	numberUserInt = (numberUserInt | (1 << positionUser - 1));
	break;
default:
	cout << "You entered a bit different from zero or one";
	break;
}
cout << "Number value: " << numberUserInt << "\n"
		<< "Number link: " << &numberUserInt << "\n"
		<< "Count rank: " << order << "\n"
		<< "Bitwise comparison mask: " << mask << "\n";
for (int i = 0; i <= order; i++)
{
	cout << ((numberUserInt & mask) ? 1 : 0);
	numberUserInt <<= 1;
	if (!i || (i + 1) % 8 == 0)
	{
		cout << " ";
	}
}
cout << "\n\n";
float tempFloat;
union binaryRepresFloat {
	int unionInt;
	float unionFloat;
};
binaryRepresFloat brf;
cout << "Enter float number for display binary representation in memory: ";
cin >> brf.unionFloat;
tempFloat = brf.unionFloat;
for (int i = 0; i <= order; i++) {
	cout << ((brf.unionInt & mask) ? 1 : 0);
	brf.unionInt <<= 1;
	if (!i || (i + 1) % 8 == 0) {
		cout << " ";
	}
};
```

```
		cout << "\n\n";
		brf.unionFloat = tempFloat;
		cin.clear();
		cout << "Enter please position bit starting from one: ";
		cin >> positionUser;
		cout << "Enter please number bit for change: ";
		cin >> numberBit;
		cout << "Enter please position bit = " << positionUser << " for set on " << numberBit
<< ".\n\n";
		switch (numberBit)
		{
		case 0:
			brf.unionInt = (brf.unionInt ^ (1 << positionUser - 1));
			break;
		case 1:
			brf.unionInt = (brf.unionInt | (1 << positionUser - 1));
			break;
		default:
			cout << "You entered a bit different from zero or one";
			break;
		}
		for (int i = 0; i <= order; i++) {
			cout << ((brf.unionInt & mask) ? 1 : 0);
			brf.unionInt <<= 1;
			if (!i || (i + 1) % 8 == 0) {
				cout << " ";
			}
		};
		cout << "\n\n";
		double tempDouble;
		union binaryRepresDouble {
			int unionIntArr[2];
			double unionDouble;
		};
		binaryRepresDouble brd;
		cout << "Enter double number for display representation in memory: ";
		cin >> brd.unionDouble;
		tempDouble = brd.unionDouble;
		int orderArray = sizeof(brd.unionIntArr) * 4 - 1;
		int sizeIntegerArray = (sizeof(brd.unionIntArr) / sizeof(*brd.unionIntArr));
		mask = 1 << orderArray;
```

```
		swap(brd.unionIntArr[0], brd.unionIntArr[1]);

		for (int i = 0; i < sizeIntegerArray; i++) {
			for (int j = 0; j <= orderArray; j++) {
				cout << ((brd.unionIntArr[i] & mask) ? 1 : 0);
				brd.unionIntArr[i] <<= 1;
				if (!i && j == 11 || !i && !j || !i && j == 31) {
					cout << " ";
				}
			}
		}
		brd.unionDouble = tempDouble;
		cout << "\n\n";

		swap(brd.unionIntArr[0], brd.unionIntArr[1]);

		cin.clear();

		cout << "Enter please position bit starting from one: ";
		cin >> positionUser;

		cout << "Enter please number bit for change: ";
		cin >> numberBit;

		cout << "Enter please position bit = " << positionUser << " for set on " << numberBit
<< ".\n\n";
		switch (numberBit)
		{
		case 0:
			if (positionUser - 1 <= 32)
				brd.unionIntArr[1] = brd.unionIntArr[1] ^ (1 << positionUser - 1);
			else
				brd.unionIntArr[0] = brd.unionIntArr[0] ^ (1 << (positionUser - 1 -
32));

			break;
		case 1:
			if (positionUser - 1 <= 32)
				brd.unionIntArr[1] = brd.unionIntArr[1] | (1 << positionUser - 1);
			else
				brd.unionIntArr[0] = brd.unionIntArr[0] | (1 << (positionUser - 1 -
32));
```

```
                break;
        default:
                cout << "You entered a bit different from zero or one";
                break;
        }

        for (int i = 0; i < sizeIntegerArray; i++) {
                for (int j = 0; j <= orderArray; j++) {
                        cout << ((brd.unionIntArr[i] & mask) ? 1 : 0);
                        brd.unionIntArr[i] <<= 1;
                        if (!i && j == 11 || !i && !j || !i && j == 31) {
                                cout << " ";
                        }
                }
        }
        cout << "\n\n";

        int decisionUser;
        cout << "Do you want continue proccess working programm?\n";
        cout << "\t 1 - Yes\n";
        cout << "\t 2 - No\n";
        cin >> decisionUser;
        switch (decisionUser)
        {
        case 1:
                cout << "You select number 1. Let's start over!";
                system("cls");
                break;
        case 2:
                cout << "You select number 2. Finishing the proccess!";
                isLive = false;
                break;
        default:
                cout << "Error select number\n";
                break;
        }
    }
    return 0;
}
```