**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.

|  |  |  |
| --- | --- | --- |
| Студентка гр. 2373 |  | Басова К. В. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2022

**Цель работы.**

Изучение различных типов данных; применение ввода и вывода типов данных; работа с условной инструкцией, инструкцией множественного выбора и циклами.

**Основные теоретические положения.**

Тип данных для каждого программного объекта, представляющего данные, определяет: характер данных (число, целое или с дробной частью, одиночный символ или текст и т.д.); объем памяти, который занимают в памяти данные; диапазон возможных значений; правила обработки данных (например, допустимые операции). Типы данных делят на 2 группы: **простые** и **структурированные**. Простые типы данных представляют неразделимые данные, не имеющие внутренней структуры (числа, символы и т.д.). Структурированные типы строятся на основе простых типов данных. Другой уровень классификации разделяет все типы данных на **предопределенные** (встроенные в язык программирования) и **пользовательские** (определяемые программистом).

Существует 4 спецификатора типа, уточняющих внутреннее представление и диапазон значений стандартных типов: **short, long**, **signed**, **unsigned**.

Переменная — это именованная область памяти, в которой хранятся данные определенного типа. Именем переменной является идентификатор и служит для обращения к области памяти, в которой хранится значение этой переменной. Перед использованием любая переменная должна быть описана.

Общее правил определения переменной можно сформулировать так:

[класс памяти]  <тип данных>  <идентификатор - имя> [инициализатор];

Пример описания переменных:

unsigned short int d; // переменная d – короткое целое без знака

int i, j, k; // однотипные переменные можно определять в одной строке

Во время выполнения программы значение переменной можно изменять. Константа – это величина, значение которой в процессе работы программы не изменяется. Они бывают двух видов: константы – литералы и именованные константы. Константы – литералы представляют собой сами значения. Пример: 123    -245    0.003   -12.45   ’R’     ”Текст”

Тип данных, которому принадлежит константа – литера, определяется компилятором автоматически по виду самого значения. Именованные константы задаются с помощью ключевого слова const: const double Pi = 3.14;

**Область видимости** определяет, где можно использовать переменную. **Продолжительность жизни**определяет, где переменная создается и где уничтожается. Переменные, определенные внутри **блока**, называются **локальными переменными**. Они имеют **автоматическую продолжительность жизни**: они создаются в точке определения и уничтожаются при выходе из блока. Локальные переменные имеют **локальную область видимости**, они входят в область видимости с точки объявления и выходят в самом конце блока, в котором определены.

#include <iostream>

int main(){

int numb(15); // переменная numb создается и инициализируется здесь

double pi(3.14); // переменная pi создается и инициализируется здесь

return 0;

} // numb и pi выходят из области видимости и уничтожаются здесь

Ввод/вывод данных осуществляется с помощью функций и объектов, содержащихся в стандартных библиотеках. Способ ввода/вывода, характерный для C++, основан на использовании потоков ввода (**cin**) и вывода (**cout**).

#include <iostream>

using namespace std;

int main ()

{

     setlocale (0, "");

     int a;

     cout << "Введите целое число:\t";

     cin >> a;

     cout << "Вы ввели значение:\t" << a << "\n\n";

     system ("Pause");

     return 0;

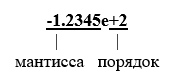
}.

Размер типа **int** зависит от компьютера и компилятора. Для 16-разрядного процессора под его величины отводится 2 байта – диапазон возможных значений -32 768 ... 32 767. Для 32-разрядного - 4 байта – диапазон значений -2 147 483 648 ... 2 147 483 647. Спецификатор **short** перед именем типа указывает компилятору, что под число требуется отвести 2 байта. Спецификатор **long** означает, что целая величина будет занимать 4 байта.

Внутреннее представление величины целого типа — целое число в двоичном коде. При использовании спецификатора **signed** старший бит числа интерпретируется как знаковый (0 — положительное число, 1 — отрицательное). Спецификатор **unsigned** позволяет представлять только положительные числа, старший разряд рассматривается как часть кода числа.

Стандарт C++ определяет 3 типа данных для хранения вещественных значений: **float**, **double** и **long double**. Они предназначены для представления отрицательных и положительных значений в разных диапазонах.

Внутреннее представление вещественного числа состоит из двух частей — мантиссы и порядка:



Тип **float** занимает 4 байта, из которых 1 двоичный разряд отводится под знак мантиссы, 8 разрядов под порядок и 23 под мантиссу. Для величин типа **double**, занимающих 8 байт, под порядок и мантиссу отводится 11 и 52 разряда соответственно. Длина мантиссы определяет точность числа, а длина порядка — его диапазон.

Величины логического типа могут принимать значения **true** и **false**. Внутренняя форма представления значения **false** – 0. Любое другое значение интерпретируется как **true**. При преобразовании к целому типу **true** имеет значение **1.** В памяти переменные этого типа занимают 1 байт.

Для обозначения символьного типа используется ключевое слово **char**. Под его величину отводится количество байт, достаточное для размещения любого символа из набора символов для данного компьютера. Как правило, это 1 байт. Тип **char** может быть со знаком или без. В величинах со знаком можно хранить значения от **-128** до **127**. При использовании спецификатора **unsigned** значения могут находиться в пределах от **0** до **255**.

Обработка данных выполняется с помощью операций. В зависимости от количества операндов в языке C++ имеются унарные (с одним операндом), бинарные (с двумя операндами) и тернарная (с тремя операндами) операция. Из знаков операций, операндов и круглых скобок строятся **выражения**. В качестве операндов могут использоваться константы, переменные, функции и другие выражения. Тип данных значения выражения зависит от выполняемой операции и типов данных операндов.

Условная инструкция if позволяет выбрать одно из двух направлений выполнения программы.

**Блок инструкций** представляет собой последовательность инструкций, каждая из которых заканчивается символом **;**. Блок можно рассматривать как одну инструкцию (составную инструкцию).

Условная инструкция if позволяет выбрать 1 из 2-х направлений выполнения программы в зависимости от значения выражения.

int K;

cin >> K;

if  (K  >=  0)

   cout << “Вы ввели положительное число.” << endl;

else

   cout << “Вы ввели отрицательное число.” << endl;

Здесь в качестве выражения использовано логическое выражение, значение которого равно **true** или **false** в зависимости от введенного с клавиатуры значения переменной **K**.

Инструкция множественного выбора (switch) служит для ветвления программы во многих направлениях.

     cin >> i;

     switch ( i )

     {

           case 0:

                 cout << "ноль\n";

                 break;

           case 1:

                 cout << "один\n ";

                 break;

           case 2:

                 cout << "два\n ";

                 break;

           default:

                 cout << "много\n ";

     }

Инструкция **break** осуществляет прерывание выполнения инструкции **switch**и управление передается следующему за **switch-**инструкцией оператору. Если значение выражения не совпадет ни с одной из констант, то будут выполнены инструкции ветви **default**. Значение выражения в инструкции **switch**обязательно должно быть либо целого, либо символьного типа **–**вещественные значения не допускаются.

Формат записи этой инструкции:

Выражение в инструкции цикла с предусловием (while)может быть любого типа. Если значение этого выражения истинно (**true** или не равно **0**), то тело цикла выполняется вновь, если же ложно, то цикл заканчивается и управление передается следующей за циклом инструкции.

**Пример.** Необходимо в виде строки вывести на экран цифры от 0 до 9.

int k = 0;  // На экран выведено k цифр

while (k <= 9)  // Логическое выражение

    {

         cout << k;

         ++k;

    }

**//**На экран выведено k = 10 цифр: 0123456789

При запуске итерационного цикла (for) однократно выполняется **инициализация** параметров цикла, после чего осуществляется проверка **условия**, определяющего необходимость выполнения тела цикла, и выполнение инструкций в цикле. На каждой итерации выполняется **модификация** параметров и снова проверяется **условие**. Так продолжается до тех пор, пока **условие** не станет ложным.

int k;

for  (k  =  0;  k  <=  9;  ++k)

   cout  <<  k;

**Постановка задачи.**

Необходимо разработать алгоритм и написать программу, которая позволяет:

1) Вывести, сколько памяти (в байтах) на компьютере пользователя 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.

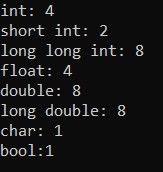
4) Вывести на экран двоичное представление в памяти (все разряды) типа double. При выводе необходимо визуально обозначить знаковый разряд мантиссы, знаковый разряд порядка (если есть), мантиссу и порядок.

**Выполнение работы.**

Код программы представлен в приложении А.

Описание кода и использованных алгоритмов.

1. При запуске программы появляется окно, в котором выводится количество памяти, которое отводится под различные типы данных на компьютере пользователя. Значения для каждого типа выводятся с новой строки:



2. Одновременно появляется строка, в которой пользователю предлагается ввести целое число. После ввода строится двоичное представление данного числа, и полученное значение выводится на экран.

Скриншот 04-10-2022 204954.jpg

3. После вывода двоичного представления целого числа, пользователю предлагается ввести вещественное число, относящееся к типу float. С ним проделывается та же операция. Дробная часть отделяется от целой части точкой.Скриншот 04-10-2022 205454.jpg

4. Последним вводится значение типа double, и выводится его двоичное представление.

Скриншот 06-10-2022 104747.jpg

**Выводы.**

Независимо от типа данных, можно вычислить количество памяти, которое отводится под них на компьютере, с помощью одной функции – sizeof().

Каждый числовой тип данных требует написания индивидуального кода, с помощью которого вычисляется его двоичное представление в памяти. Так, вещественные типы float и double предполагают использование функции union, которая позволяет работать с данными посредством объединения с типом int.

Приложение А

код Программы

#include <iostream>

using namespace std;

int main()

{

setlocale(0, "Rus");

cout <<"Кол-во памяти (в байтах), которое отводится под различные типы данных:\n" << "int: " << sizeof(int) << "\n" << "short int: " << sizeof(short) << "\n" << "long long int: " << sizeof(long long int) << "\n"

<< "float: " << sizeof(float) << "\n" << "double: " << sizeof(double) << "\n" << "long double: " << sizeof(long double) << "\n"

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

int number;

cout << "Введите целое число: ";

cin >> number;

cout << "Его двоичное представление в памяти: ";

int r = sizeof(number) \* 8; // кол-во разрядов (32)

int mask = 1 << r - 1; // маска - 1 в двоичном представ-и, сдвинутая до 32-го разряда

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

if (i == 1 || i % 8 == 0) { // выделение знакового и значащих разрядов

putchar(' ');

}

putchar(number & mask ? '1' : '0'); // побитовое сравнение с 1, "слепок" числа

number <<= 1;

}

union {

float value;

int v;

};

cout << "\n\n" << "Введите вещественное число (float): ";

cin >> value;

cout << "Его двоичное представление в памяти: ";

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

if (i == 1 || i == 9) { // выделение знакового разряда, мантиссы и порядка

putchar(' ');

}

putchar(v & mask ? '1' : '0'); // побитовое сравнение с 1, "слепок" числа

v <<= 1;

}

union {

double num;

int arr[2];

};

cout << "\n\n" << "Введите вещественное число (double): ";

cin >> num;

cout << "Его двоичное представление в памяти: ";

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

if (i == 1 || i == 12) { // выделение знакового разряда, мантиссы и порядка

putchar(' ');

}

putchar(arr[1] & mask ? '1' : '0');// побитовое сравнение с 1, "слепок" первой части числа

arr[1] <<= 1;

}

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

putchar(arr[0] & mask ? '1' : '0'); // побитовое сравнение с 1, "слепок" второй части числа

arr[0] <<= 1;

}

}