Лабораторная работа №1.
Типы данных языка С++.

Язык С++ является строго типизированным. Это означает, что любой программный объект должен быть определенного типа. Данные различных типов хранятся и обрабатываются по-разному. Тип данных определяет:

- внутреннее представление данных в памяти машины;
- множество значений, которые могут применять величины этого типа;
- операции (функции), которые можно применить к величинам этого типа.

От типа величины зависят машинные команды, которые будут использоваться для обработки данных.

Все типы языка С++ делятся на основные и составные. В языке С++ определено шесть основных типов для представления целых, вещественных, символьных и логических величин. На их основе программист может вводить описание составных типов. К ним относятся массивы, перечисления, структуры, указатели и ссылки, объединения и классы.

Основные типы данных.

Основные (стандартные) типы часто называют арифметическими, поскольку их можно использовать в арифметических операциях. Для описания основных типов используют следующие ключевые слова: int (целый), char (символьный), wchar_t (расширенный символьный), bool (логический), float (вещественный), double (вещественный с двойной точностью). Первые четыре типа называют целочисленными, т.к. они представляются в машине с помощью целых чисел. Два последних типа называют типами с плавающей точкой. Коды, формируемые компилятором для обработки целочисленных и вещественных данных, будут различными.

Существует четыре спецификатора типа, уточняющих внутреннее представление и диапазон значений стандартных типов: short (короткий), long (длинный), signed (знаковый), unsigned (беззнаковый).

Целый тип (int). Размер типа int стандартом языка не регламентирован, он зависит от реализации компилятора и разрядности процессора. Как правило, размер целого типа совпадает с разрядностью процессора. Так для 16-и разрядного процессора компилятор выделяет 2 байта (16 двоичных разрядов), а для 32-х разрядного – 4 байта.

Спецификатор short перед именем типа указывает компилятору, что под число требуется отвести 2 байта, независимо от разрядности процессора. Спецификатор long требует 4 байта. Так, для 32-х разрядного процессора данное тип int потребует 4 байта, short int – 2 байта, long int – 4 байта.

Использование спецификатора signed (по умолчанию) предполагает, что старший разряд числа интерпретируется как знаковый (0- для положительных чисел, 1- для отрицательных). Спецификатор unsigned позволяет представлять только положительный числа. В зависимости от спецификаторов перед типом int диапазон представления различен.

31	0
Знак	Значение числа

Рис. 1. Внутреннее представление типов int, signed int и long для 32-х разрядной платформы

31	0
Значение числа	

Рис. 2. Внутренне представление типа unsigned int для 32-х разрядной платформы

16	0
Знак	Значение числа

Рис. 3. Представление типа sort int

16	0
Значение числа	

Рис. 4. Представление типа unsigned short int

Возможны и более сложные комбинации типов и спецификаторов, например, unsigned long long int.

Символьный тип (char). Как правило под данные типа char выделяется 1 байт. Этого достаточно для хранения до 256 различных символов кода ASCII. Тип char, как и другие целые типы может быть со знаком или без знака. В величинах со знаком можно хранить величины от -128 до 127. Если используется спецификатор unsigned, то величины могут принимать значения от 0 до 255.

Величины типа char могут использоваться для хранения целых чисел в указанных диапазонах.

Расширенный символьный тип (wchar_t). Тип wchar_t предназначен для хранения символов, для которых 1 байта не достаточно, например, для символов в кодировке Unicode. Размер этого типа соответствует типу short (2 байта).

Логический тип (bool). Множество значений этого типа – это true (1) и false (0). Любое другое значение интерпретируется как true. При преобразовании к целому типу true имеет значение 1, а false – 0.

Для величин этого типа компилятор выделяет 1 байт.

Типы с плавающей точкой (float, double, long double). Типы данных с плавающей точкой хранятся в памяти компьютера иначе, чем целочисленные. Под тип float обычно выделяется 4 байта, один разряд из которых отводится под знак мантиссы, 8 разрядов под порядок и 23 – под мантиссу. Мантисса – это число, большее 1 и меньшее 2.

Под величины типа double выделяется 8 байт. Спецификатор long перед double гарантирует выделение 10 байт.

Константы с плавающей точкой имеют по умолчанию тип double.

Диапазоны изменений величин арифметических типов представлены в таблице 1.

Таблица 1.

Тип данных	Размер памяти, бит	Диапазон значений
`char` (символьный)	8	от–128 до 127
`signed char` (знаковый символьный)	8	от–128 до 127
`unsigned char` (беззнаковый символьный)	8	от 0 до 255
`short int` (короткое целое)	16	от–32768 до 32767
`unsigned int` (беззнаковое целое)	16	от 0 до 65535 (16-битная платформа) от 0 до 4294967295 (32-битная платформа)
`int` (целое)	16 32	от–32768 до 32767 (16-битная платформа) от –2147483648 до 2147483647 (32-битная платформа)
`long` (длинное целое)	32	от–2147483648 до 2147483647
`unsigned long` (длинное целое без знака)	32	от 0 до 4294967295
`long long int` (C99)	64	от–(2^{63}–1) до 2^{63}–1
`unsigned long long int` (C99)	64	от 0 до 2^{64}–1
`float` (вещественное)	32	от 3.4E–38 до 3.4E38
`double` (двойное вещественное)	64	от 1.7E–308 до 1.7E308
`long double` (длинное вещественное)	80	от 3.4E–4932 до 3.4E4932

bool (C++)	8	true(1), false(0)

Минимальные и максимальные допустимые значения для целых типов зависят от реализации и приведены в заголовочном файле <limits.h> (<climits>), значения вещественных типов – в файле <floath> (cfloat>), а также в шаблоне клсса numeric_limits.

Тип void. Кроме перечисленных стандартных типов, к основным типам относят тип void, множество значений которого пусто. Он используется для определения функций, не возвращающих никакого результата, а также для представления пустого списка параметров. Кроме того этот тип используется как базовый тип для указателей.

Составные типы данных.

Как уже было отмечено, к составным типам относят массивы, перечисления, структуры, указатели и ссылки, объединения и классы. Из перечисленных типов будут рассмотрены переименование типов, перечисления и частично структуры. Остальные типы будут подробно рассмотрены в следующих лабораторных работах.

Переименование типов(typedef). В некоторых случаях типу можно дать новое имя. Это позволяет сделать текст программы более ясным. Общий формат преобразования можно представить следующим образом:

typedef тип новое_имя [размерность];

В данном контексте квадратные скобки и содержимое между ними следует рассматривать как необязательный элемент конструкции.

Классический пример использования преобразования типа – сокращение длинных имен существующих типов, например:

typedef unsigned int UINT;

С момента объявления этого переименования слово UINT компилятором будет интерпретироваться как unsigned int.

Еще один пример, заменяющий объявление анонимной структуры на конкретное имя:

```
typedef  struct
 {
        char *Name;
        int Age;
        float Ball;
 } Students;
```

Введенные имена теперь можно использовать таким же образом, как и имена стандартных типов:

```
UINT  _x, _y;          // две переменные типа unsigned int
Students group[20];    // массив из 20 структур
```

Перечисления(enum). В некоторых случаях необходимо иметь конечное множество именованных констант, имеющих различные значения. Для этого удобно воспользоваться перечислимым типом данных. Формат перечислимого типа следующий:

enum [имя_типа] {список_констант};

Необязательное имя типа требуется в случае, если требуется определить переменные этого типа. Компилятор обеспечит, чтобы переменные принимали значения только из списка констант. Константы должны быть обязательно целочисленными и могут быть инициализированы обычным образом. При отсутствии инициализации первая константа обнуляется, а каждая последующая принимает значение большее на 1, чем предыдущая.

Например:

```
enum error{ERR_READ, ERR_WRITE, ERR_CONVERT};  // объявление типа error
error err;  // объявление переменной err типа уккщк
```

Константы ERR_READ, ERR_WRITE, ERR_CONVERT получают соответственно значения 0, 1, 2. Переменная err может принимать значения только из этого диапазона, т. е. 0, 1, 2.

Константам можно присвоить другие значения при объявлении:

```
enum error{ERR_READ = 3, ERR_WRITE = 38, ERR_CONVERT = 99};
```

Над переменными типа перечисления можно выполнять арифметические операции, при этом они автоматически преобразуются к целому типу.

Структуры (struct). В отличие от других типов данных, например массивов, представляющих величины одного типа, структуры могут содержать в себе элементы различных типов. В языке C++

структура является видом класса и обладает всеми его свойствами. Общий формат объявления структуры следующий:

```
struct [ имя_типа ]
 { тип_1 поле_1;
   тип_2 поле_2;
   ……
   тип_n поле_n;
  } [ список_описателей ];
```

Имя структуры (необязательное, при этом такая структура называется анонимной) представляет новый тип, определенный пользователем. Этот тип в дальнейшем может быть использован наравне со стандартными. При этом компилятор не дает никаких привилегий стандартным типам. Если имя структуры отсутствует, должен быть указан список описателей переменных, указателей или массивов.

Элементы структуры, именуемые полями, могут имеет любой тип, кроме типа определяемой структуры, но могут быть указателями на этот тип.

Описание структуры, представляющее собой блок (последовательность операторов, заключенного в фигурные скобки) обязательно должно заканчиваться точкой запятой.

Рассмотрим пример структуры.

```
struct Students
{
        char *Name;
        int Age;
      bool  Sex;
        float Ball;
};
```

Здесь представлен новый тип Students через объявление структуры. Каждый объект этого типа (переменная типа структуры) в своем составе будет иметь четыре поля типов char *, int, bool и float.

Для данной структуры не определено ни одного действия, кроме операции присваивания, при которой происходит копирование одного объекта в другой методом «поле за полем». Эта операция генерируется компилятором автоматически без участия программиста. Если необходимо специфическая форма копирования, то программист должен об этом позаботиться сам. Все остальные операции над объектами данного типа – обязанность программиста.

Обращение или доступ к полям структуры осуществляется с помощью операции выбора (‘.’ - символ точки) при обращение к полю через имя объекта или с помощью последовательности символов’-‘ и ‘>’, если обращение осуществляется через указатель. Например,

```
Student st, *ptr_st = &st;
st.Name = “ Ivan”;
st.Age = 20;
ptr_st->Sex = 1;
ptr_st->Ball = 4.75;
```

Структура может выступать в качестве типа аргумента функции, т. е. некоторой функции можно передать в качестве аргумента объект типа структура. Также структура может выступать в качестве типа возвращаемого результата некоторой функции.

«Операции» над типами данных.

Слово операции в названии заключено в скобки, поскольку операции определены над данными определенных типов, а не над самими типами. Язык С++, являясь объектно-ориентированным, не предполагает, что стандартный тип или тип объявленный пользователем в свою очередь является объектом (переменной) какого-нибудь типа.

Однако, для работы с типами С++ предоставляет следующие возможности:

- определить размер типа в байтах оператором sizeof, в качестве аргумента может быть имя типа, имя переменной или более сложное выражение;

- задать альтернативное имя (синоним) существующему типу оператором typedef;

- средства преобразования одних типов в другие;

- механизм определения типа объекта в реальном времени (RTTI);

Оператор sizeof.

Оператор sizeof- это унарный оператор, возвращающий длину в байтах переменной или типа, помещенных в скобки. Например:

```
float var_float =  3.14;
// ……
cout << " Количество байт для  вещественного типа = "
     << sizeof(var_float) << endl;
// ……..
cout << " Количество байт для целого типа = "
     << sizeof(int);
```

В качестве аргумента операции может выступать любой тип, как стандартный, так и определенный пользователем, а также объект или переменная любого типа.

Этот оператор может использоваться при выполнении программы на машинах с процессорами, имеющими различную разрядность, например, 16 и 32. Это позволяет создавать машинно-независимые программы.

С помощью оператора sizeof можно определить размер поля структуры или класса, например для структуры Students:

```
cout  << " Количество байт поля структуры = " << sizeof(st.Age) Б << endl;
```

Преобразование типов. В языке С++ возможно несколько способов преобразования данных одного типа в другой, если это допустимо. Один из способов состоит в указании необходимого типа, заключенного в скобки перед данным другого типа, в двух формах: тип(выражение) или тип(выражение). Например, оператор (float)var_int; гарантирует преобразование переменной var_int целого типа в вещественный формат. Явное преобразование следует использовать осторожно, т.к. оно является источником возможных ошибок. Понижающее преобразование может привести к потере точности, например, (int)var_float отсечет дробную часть вещественного числа.

В некоторых случаях компилятор автоматически переводит данные одного типа в другой, например, в выражении var_int+var_float, переменная целого типа var_int приводится к вещественному типу, т.е., компилятор выполняет следующие действия (float)var_int+var_float. Такие преобразования относятся к стандартным. К стандартным преобразованиям также относятся и преобразования указателей и ссылок.

Второй способ характерен для последних версий языка С++ и осуществляется с помощью специфических операторов const_cast, static_cast, dynamic_cast, reinterpret_cast, которые будут рассмотрены в следующих работах.

Механизм определения типа объекта в реальном времени (RTTI). Этот механизм позволяет определить, на какой тип в текущий момент времени ссылается указатель. Для доступа к RTTI в стандарт языка введена операция typeid и класс type_info. Класс содержится в заголовочном файле <typeinfo>.

Основная операция typeid, которая применима к основным и производным типам. Метод name возвращает указатель на строку, представляющую имя типа, а метод before выполняет побуквенное сравнение двух имен типов. Для сравнения используется следующая конструкция: tpeid(T1).before(typeid(T2)). Если имя типа T1 лексикографически предшествует имени T2, результат будет истинным.

Несколько примеров:

```
cout << " Имя типа = " << typeid(var_int).name() << endl;          // int
cout << " Имя типа = " << typeid(2+3.4).name() << endl;            // double
cout << typeid(Base).before(typeid(Derved)) << endl;               // true
```

Информация о типе бывает полезной в диагностических целях.

Задания для выполнения лабораторной работы.

Для выполнения работы необходимо открыть пустой проект в среде Visual Studio, набрать программу, содержащую объявление переменных основных типов, а также структуру, состоящую из 3-4 полей.

1. Определить число байт, необходимых для хранения основных типов для данной реализации компилятора и разрядности процессора. По возможности выполнить программу на машинах с разной разрядностью и оценить полученные результаты.
2. Осуществить преобразование отдельных типов с целью улучшения читаемости программы, а также сокращения длинных имен типов.
3. Объявить переменные перечислимого типа без инициализации констант и с их инициализацией. Выполнить допустимые для них операции. Определить число байт, требуемое для хранения таких переменных.
4. Объявить структуру. Определить число байт, требуемое для хранения всех полей структуры. Оценить полученные результаты и сопоставить с результатами по пункту 1.Определить число байт, необходимых для хранения отдельных полей.
5. Пользуясь преобразованием типов, осуществить преобразования объектов к другим типам. Оценить возможность или невозможность преобразований.
6. Пользуясь механизмом определения типа в реальном времени, определить тип фактических объектов и выражений.