Лабораторная работа 5. Типы данных и операции C++

1. *ЦЕЛЬ РАБОТЫ*

Познакомится с основными числовыми типами данных языка C++, принципами объявления переменных, а также некоторыми операциями, используемыми в программах на C++.

2. *КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ*

Любая программа, так или иначе, обрабатывает *данные.* Рассмотрим, какие возможны варианты представления информации в C.

Литералы

Прежде всего, данные могут присутствовать непосредственно в тексте программы. В этом случае они представляются в виде *литеральных констант.* Литералы – это фиксированные значения, которые не могут быть изменены программой. Литералы могут быть *числовыми*, *символьными* и *строковыми*. В программе `Hello All` мы пользовались строковыми литералами. Это — последовательность символов, заключенная в двойные кавычки. Чтобы указать константу расширенного символьного типа, перед символьной или строковой константой следует использовать префикс L.

Символьный литерал служит для представления одиночного знака. Это символ, заключенный в одиночные кавычки (апострофы).

Можно дать литеральной константе некоторое имя, определив ее в качестве макроса препроцессора. После этого можно вместо литерала использовать имя. Это особенно удобно в том случае, когда одна и та же константа встречается в различных частях программы; используя имя вместо литералов, вы гарантированы от опечаток и, кроме того, гораздо проще вносить в код изменения, если значение константы нужно модифицировать. Макросы определяются директивой препроцессора #define:

```
#define PI 3.14159265
#define TRUE 1
#define FALSE 0
```

При обработке исходного кода препроцессором выполняется просто текстовая подстановка: каждое вхождение имени макроса заменяется соответствующим ему литералом. Макросы называют также символическими константами (не путайте с символьными).

Иногда удобно вместо десятичной системы счисления использовать для записи числовых литералов восьмеричную или шестнадцатеричную систему. Шестнадцатеричный литерал должен начинаться с префикса 0x (цифра «нуль» и буква «икс»), а восьмеричный с нуля. Примеры:

```
int hex = 0xFF;
into oct = 011;
```

Основные типы данных

Однако данные могут не только вписываться в текст программы, но и храниться в памяти во время ее выполнения. С физической точки зрения любая информация в памяти машины выглядит одинаково — это просто последовательности нулей и единиц, сгруппированных в байты. Поэтому наличные в памяти данные должны как-то интерпретироваться процессором; этой интерпретацией управляет, естественно, компилятор C++. Любая информация рассматривается компилятором как принадлежащая к некоторому *типу данных.* В языке C++ имеется набор *основных* типов (возможны и другие типы данных, например, определяемые пользователем). Основные типы перечислены в следующей таблице.

Таблица 1. Основные типы данных C++. Типичные размеры значений

Тип	Типичный размер в битах	Соответствующий диапазон	Примеры применения
`char`	8	-128 до 127	Самые малые числа и символы ASCII
`wchar_t`	16	0 – 65535	Двухбайтовые символы или целые положительные числа
`int` (32-разрядная среда)	32	-2 147 483 648 – 2 147 483 647	Целые числа со знаком
`float`	32	$\pm 3.4 \times 10^{-38}$ до $\pm 3.4 \times 10^{38}$	Научные расчеты (точность 7 разрядов)
`double`	64	$\pm 1.7 \times 10^{-308}$ до $\pm 1.7 \times 10^{308}$	Научные расчеты (точность 15 разрядов)
`bool`	8 (не для всех систем)	true / false	логические значения
`void`	-	без значения	Задание переменных типа `void` невозможно. Этот тип служит в основном для объявления функций, не возвращающих значения, или универсальных указателей на нетипизированные или произвольно типизированные данные.

Необходимо помнить, что размеры и диапазоны типов, используемые Вашим компилятором, могут отличаться от приведенных в таблице выше.

Модификаторы типов

Основные типы данных могут иметь различные модификаторы. Модификаторы используются изменения значения базового типа, чтобы он более точно соответствовал конкретной ситуации. Возможны следующие модификаторы:

`signed`

`unsigned`
`short`
`long`

Модификаторы `signed`, `unsigned`, `long` и `short` могут применяться к целочисленным типам. К символам можно применять `signed` и `unsigned`; `long` может применяться к типу `double`.

Таблица 2. Допустимые комбинации базовых типов и модификаторов для 32-разрядной среды

Тип	Длина в битах	Диапазон
`char`	8	от -128 до 127
`unsigned char`	8	от 0 до 255
`signed char`	8	от -128 до 127
`int`	32	от -2147483648 до 2147483647
`unsigned int`	32	от 0 до 4294967295
`signed int`	32	от -2147483648 до 2147483647
`short int`	16	от -32768 до 32767
`unsigned short int`	16	от 0 до 65535
`signed short int`	16	от -32768 до 32767
`long int`	32	от -2147483648 до 2147483647
`unsigned long int`	32	от 0 до 4294967295
`signed long int`	32	от -2147483648 до 2147483647
`long long int`	64	примерно от -10^{18} до $+10^{18}$
`float`	32	от 3.4e-38 до 3.4e+38
`double`	64	от 1.7e-308 до 1.7e+308
`long double`	80	от 3.4e-4932 до 1.1e+4932

Использование `signed` для целочисленных типов является избыточным (но допустимым), поскольку объявление целочисленных типов по умолчанию предполагает знаковое число.

Различие между знаковыми и беззнаковыми целочисленными типами заключается в способе интерпретации старшего бита. Если используется знаковый тип, то компилятор генерирует код, предполагающий, что старший бит используется как знак числа. Если знаковый бит равен 0, то число положительное, а если 1 - отрицательное.

Допускается сокращенная запись. Так, `unsigned short` — на самом деле сокращение для `unsigned short int`.

Переменные

Итак, отдельная единица данных должна обязательно иметь определенный тип. Для ее хранения во время работы программы мы должны, во-первых, отвести соответствующее место в памяти, а во-вторых, идентифицировать ее, присвоив

некоторое *имя.* Именованная единица памяти для хранения данных называется *переменной.*

Переменные создаются с помощью оператора *объявления переменных,* в котором указывается тип, имена переменных и (при необходимости) начальные значения, которыми переменные *инициализируются.* Вот несколько примеров:

```
short i;            // Объявление короткой целой переменной.
char quit = 'Q';    // Инициализация символьной переменной.
float f1, factor = 3.0, f2; // Три переменных типа float,
                            //одна из которых инициализируется.
```

Синтаксис оператора объявления можно описать примерно так:

```
тип имя_переменной [= инициализирующее_значение][, ...];
```

Как и любой другой оператор C++, он оканчивается точкой с запятой.

Имена в C++ могут состоять из букв латинского алфавита, цифр и символов подчеркивания, причем первый символ имени не может быть цифрой. Следует помнить, что компилятор C++ различает регистр (прописные и строчные буквы). Таким образом, имена aVariable и AVariable считаются различными.

Объявление переменной должно предшествовать ее использованию в программе. Обычно все объявления размещают в начале тела функции или блока, до всех исполняемых операторов.

Типизированные константы

Разновидностью переменных являются *типизированные константы.* Это переменные, значение которых (заданное при инициализации) нельзя изменить. Создание типизированной константы ничем не отличается от инициализации переменной, за исключением того, что перед оператором объявления ставится ключевое слово `const`:

```
const тип имя_константы = значение [, ...];
```

Например:

```
const double Pi = 3.14159265;
```

Ранее мы демонстрировали определение символической константы:

```
#define PI 3.14159265
```

Чем этот макрос отличается от показанной выше типизированной константы? Здесь следует иметь в виду два момента. Во-первых, типизированная константа по своему смыслу относится к конкретному типу данных, поэтому компилятор генерирует совершенно определенное представление для ее значения. Представление символической константы не определено.

Во-вторых, имя символической константы значимо только на этапе препроцессорной обработки исходного кода, поэтому компилятор не включает его в отладочную информацию объектного модуля. Вы не можете использовать это имя в выражениях при отладке. Напротив, типизированные константы являются по существу переменными, и их имена доступны отладчику. В силу этих причин предпочтительнее применять для представления постоянных величин типизированные константы, а не макросы `#define`.

Операции и выражения

В языке C++ следующим уровнем представления данных после одиночных переменных и констант являются своего рода формулы, называемые *выражениями.*

Операции характеризуются своим *приоритетом,* определяющим порядок, в котором производится оценка выражения, и правилом *ассоциации,* задающим направление последовательных оценок идущих друг за другом операций одного приоритета.

Как и в обычных формулах, для изменения порядка оценки выражения могут применяться круглые скобки. Знак равенства здесь также является *операцией присваивания,* которая сама (и, соответственно, все выражение в целом) возвращает значение. В этом отличие C от других языков, в частности Pascal, где присваивание является оператором, а не операцией. Оператором выражение станет, если поставить после него точку с запятой.

В следующей таблице дана сводка всех операций языка C в порядке убывания приоритета.

Таблица 3. Операции языка C++

Операция	Описание	Приоритет	Ассоциация
Первичные и постфиксные операции			
[]	индексация массива	16	слева направо
()	вызов функции	16	слева направо
.	элемент структуры	16	слева направо
->	элемент указателя	16	слева направо
++	постфиксный инкремент	15	слева направо
--	постфиксный декремент	15	слева направо
Одноместные операции			
++	префиксный инкремент	14	справа налево
--	префиксный декремент	14	справа налево
sizeof	размер в байтах	14	справа налево
(тип)	приведение типа	14	справа налево
~	поразрядное NOT	14	справа налево
!	логическое NOT	14	справа налево
-	унарный минус	14	справа налево
&	взятие адреса	14	справа налево
*	разыменование указателя	14	справа налево
Двухместные и трехместные операции			
Мультипликативные			
*	умножение	13	слева направо
/	деление	13	слева направо
%	взятие по модулю	13	слева направо
Аддитивные			
+	сложение	12	слева направо
-	вычитание	12	слева направо
Поразрядного сдвига			
<<	сдвиг влево	11	слева направо
>>	сдвиг вправо	11	слева направо
Отношения			
<	меньше	10	слева направо
<=	меньше или равно	10	слева направо
>	больше	10	слева направо
>=	больше или равно	10	слева направо

Операция	Описание	Приоритет	Ассоциация
==	равно	9	слева направо
! =	не равно	9	слева направо
Поразрядные			
&	поразрядное AND	8	слева направо
^	поразрядное XOR	7	слева направо
|	поразрядное OR	6	слева направо
Логические			
&&	логическое AND	5	слева направо
||	логическое OR	4	слева направо
Условные			
? :	условная операция	3	справа налево
Присваивания			
=	присваивание	2	справа налево
*=	присвоение произведения	2	справа налево
/=	присвоение частного	2	справа налево
%=	присвоение модуля	2	справа налево
+=	присвоение суммы	2	справа налево
-=	присвоение разности	2	справа налево
<<=	присвоение левого сдвига	2	справа налево
>>=	присвоение правого сдвига	2	справа налево
&=	присвоение AND	2	справа налево
^=	присвоение XOR	2	справа налево
|=	присвоение OR	2	справа налево
,	запятая	1	слева направо

Арифметические операции

К арифметическим относятся те операции, которые перечислены в таблице под рубриками “Мультипликативные” и “Аддитивные”. Нужно сказать, что только эти операции (да и то за исключением взятия по модулю) имеет смысл применять к вещественным операндам (типам `float`, `double` и `long double`). Для таких операндов все обстоит вполне понятным образом; это обычные умножение, деление, сложение и вычитание.

Операция взятия по модулю применяется только к целочисленным операндам (`char`, `short`, `int`, `long`) и дает остаток от деления первого операнда на второй. Специальной операции деления нацело в C++ нет — для него применяется обычная операция деления (/). Если оба операнда ее являются целыми, то результат этой

операции *также будет целым,* равным частному от деления с остатком первого операнда на второй.

Операции присваивания

Операция присваивания (=) не представляет особых трудностей. При ее выполнении значением переменной в левой части становится результат оценки выражения справа. Как уже говорилось, эта операция сама возвращает значение, что позволяет, например, написать:

```
a = b = c = someExpression;
```

После исполнения такого оператора все три переменных a, b, c получат значение, равное `someExpression`. Что касается остальных десяти операций присваивания, перечисленных в таблице, то они просто служат для сокращенной нотации присваивании определенного вида. Например,

`s += i;` эквивалентно `s = s + i;`

Другими словами, оператор вроде `x *= 10;` означает “присвоить переменной x ее текущее значение, умноженное на 10”.

Присваивание — единственная операция, меняющая содержимое одного из своих операндов (если не считать специальные операции инкремента и декремента).

Приведение типа

Если в операторе присваивания тип результата, полученного при оценке выражения в правой части, отличен от типа переменной слева, компилятор выполнит автоматическое *приведение типа* (по-английски typecast или просто cast) результата к типу переменной. Например, если оценка выражения дает вещественный результат, который присваивается целой переменной, то дробная часть результата будет отброшена, после чего будет выполнено присваивание.

Возможно и принудительное приведение типа, которое выполняется посредством *операции приведения* и может применяться к любому операнду в выражении, например:

```
p = pO + (int)(pReal + 0.5); // Округление pReal
```

Следует иметь в виду, что операция приведения типа может работать двояким образом. Во-первых, она может производить действительное преобразование данных, как это происходит при приведении целого типа к вещественному и наоборот. Получаются совершенно новые данные, физически отличные от исходных. Во-вторых, операция может никак не воздействовать на имеющиеся данные, а только изменять их интерпретацию. Например, если переменную типа short со значением -1 привести к типу unsigned short, то данные останутся теми же самыми, но будут интерпретироваться по-другому (как целое без знака), в результате чего будет получено значение 65535.

Логические операции и операции отношения

Операции отношения служат для сравнения (больше — меньше) или проверки на равенство двух числовых операндов. Операции возвращают “логическое” значение, т. е. ненулевое целое в случае, если условие отношения удовлетворяется, и нулевое в противном случае.

Поразрядные операции и сдвиги

Эти операции применяются к целочисленным данным. Последние рассматриваются просто как набор отдельных битов.

При *поразрядных операциях* каждый бит одного операнда комбинируется (в зависимости от операции) с одноименным битом другого, давая бит результата. При единственной одноместной поразрядной операции — отрицании (~) — биты результата являются инверсией соответствующих битов ее операнда.

При *сдвиге влево* биты первого операнда перемещаются влево (в сторону старших битов) на заданное вторым операндом число позиций. Старшие биты, оказавшиеся за пределами разрядной сетки, теряются; справа результат дополняется нулями.

Результат *сдвига вправо* зависит от того, является ли операнд знаковым или без знаковым. Биты операнда перемещаются вправо на заданное число позиций. Младшие биты теряются. Если операнд — целое со знаком, производится *расширение знакового бита* (старшего), т. е. освободившиеся позиции принимают значение 0 в случае положительного числа и 1 — в случае отрицательного. При беззнаковом операнде старшие биты заполняются нулями.

Сдвиг влево эквивалентен умножению на соответствующую степень двойки, сдвиг вправо — делению. Например,

```
aNumber = aNumber <<4;
```

умножает aNumber на **16.**

Пример программы 1. Составим программу, которая определяет и выводит экран значение третьего бита числа (биты нумеруются справа налево с нуля). Пусть имеется число 9 = 00001001_2. Чтобы выделить третий бит, умножим его на число, у которого все биты равны нулю, кроме третьего:

```
00001001 & 00001000 = 00001000.
```

Таким образом, если мы получаем ответ, равный нулю, то на искомой позиции находится ноль, иначе единица. Чтобы получить ответ (0 или 1), сдвинем результат предыдущей операции вправо на три позиции.

```
#include <iostream>
#include <conio.h>

using namespace std;

int main()
{unsigned int n, result;
 cout << "Enter a number: ";  cin >> n;
 result = n & 8;
 result = n >> 3;
 cout << "3 bit: " << result;
 getch(); return 0;
}
```

Инкремент и декремент

Операции инкремента (++) и декремента (--) соответственно увеличивают или уменьшают свой операнд (обязательно переменную) на единицу. *Они изменяют значение самой переменной,* т. е. являются скрытыми присваиваниями. Иногда эти операции применяют в качестве самостоятельного оператора:

```
i++; или ++i;
```

И то и другое эквивалентно `i = i + 1;`

Но эти операции могут использоваться и в выражениях:

```
sum = sum + x * --i;
```

Инкремент и декремент реализуются в двух формах: префиксной (++i) и постфиксной (i--). Префиксные операции выполняются *перед* тем, как будет производиться оценка *всего выражения.* Все постфиксные операции выполняются уже после оценки выражения, в которое они входят.

Использование математических функций

Как известно, в C++ нет встроенных функций. Однако в дополнительных библиотеках имеется большое количество разнообразных полезных функций.

В C++ определены в заголовочном файле `<cmath>` функции выполняющие некоторые часто используемые математические задачи. Например, нахождение корня, возведение в степень, `sin()`, `cos()` и многие другие. В таблице 1 показаны основные математические функций, прототипы которых содержатся в заголовочном файле `<cmath>`.

Таблица 4. Математические функции в C++

Функция	Описание	Пример
abs(a)	модуль или абсолютное значение от **a**	abs(-3.0)= 3.0 abs(5.0)= 5.0
sqrt(a)	корень квадратный из **a,** причём **a** не отрицательно	sqrt(9.0)=3.0
pow(a, b)	возведение **a** в степень **b**	pow(2,3)=8
ceil(a)	округление **a** до наименьшего целого, но не меньше чем **a**	ceil(2.3)=3.0 ceil(-2.3)=-2.0
floor(a)	округление **a** до наибольшего целого, но не больше чем **a**	floor(12.4)=12 floor(-2.9)=-3
fmod(a, b)	вычисление остатка от a/b	fmod(4.4, 7.5) = 4.4 fmod(7.5, 4.4) = 3.1
exp(a)	вычисление экспоненты $\mathbf{e}^{\mathbf{a}}$	exp(0)=1
sin(a)	**a** задаётся в радианах	
cos(a)	**a** задаётся в радианах	
log(a)	натуральный логарифм **a**(основанием является экспонента)	log(1.0)=0.0
log10(a)	десятичный логарифм **a**	Log10(10)=1

Таблица 4. Математические функции в C++		
Функция	**Описание**	**Пример**
asin(a)	арксинус **a**, где **-1.0 < a < 1.0**	asin(1)=1.5708

Необходимо помнить то, что операнды данных функций всегда должны быть вещественными, то есть a и b – числа с плавающей точкой.

Разработаем программу, которая будет использовать математические функции.

Пример программы 2

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

int main()
{    float a, x;
     a = 3.14159 / 2;      // 90 градусов, перевели в радианы
     x = sin (a);
     cout << "sin(90')    = " << x << endl; // sin 90 градусов
     system("pause");
     return 0;
}
```

3. *ЗАДАНИЕ НА РАБОТУ*

Разработать программы с использованием арифметических и битовых операций, а также операций сдвига.

4. *ОФОРМЛЕНИЕ ОТЧЕТА*

Отчет по работе оформляется в бумажном виде.

1) Название работы и цель,

2) Для всех созданных программ: текст созданной программы; результаты выполнения разработанной программы для трех разных наборов исходных данных; расчеты, выполненные вручную для тех же наборов исходных данных.

5. *ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ*

1. Изучить теоретические положения и конспект лекций по данной теме.

2. Проверить работу примеров программы (1 и 2), приведенных в тексте теоретических положений к ЛР.

3. Создать программы по своему варианту. Во всех заданиях предполагается линейный алгоритм, ввод данных – с клавиатуры, вывод – на экран.

4. Составить отчет по работе.

6. БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Шилдт Г. C++: базовый курс, 3-е издание. : Пер. с англ. – М.: «Издательский дом «Вильямс», 2005. – 624 с.
2. Язык программирования C++: Учебник для начинающих http://cppstudio.com/cat/274/

7. ПРИМЕРЫ ЗАДАНИЙ НА ЛР (ВАРИАНТЫ ЗАДАНИЙ ВЫДАЮТСЯ ПРЕПОДАВАТЕЛЕМ ВО ВРЕМЯ ЛР)

1. Дано целое число. Проинвертировать младшие четыре бита этого числа.
2. Дано целое число. Произвести циклический сдвиг этого числа вправо на четыре разряда.
3. Найти площадь треугольника, две стороны которого равны a и b, а угол между этими сторонами γ.
4. Найти (в радианах в градусах) все углы треугольника со сторонами a, b, c.
5. Найти сумму заданного четырехзначного числа.
6. На плоскости заданы два вектора с координатами (X1, Y1) и (X2, Y2). Определить угол между векторами.