

Лабораторная работа №3

«Переменные, типы данных и операции над ними»

Переменные и типы данных

1 Понятие переменной

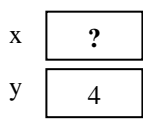
Переменная — это именованная ячейка памяти.

Каждая переменная имеет три неотъемлемых характеристики:

- имя;
- тип;
- адрес.

Графически переменные удобно изображать следующим образом:

Таблица 1. Графическое изображение переменных.

Код на C++	Иллюстрация
<pre>int x; int y = 4;</pre>	

В данном примере *объявляются*¹ две переменные *типа* `int` с именами `x` и `y`.

Различают *объявление с инициализацией* и *объявление без инициализации*. В первом случае переменная изначально получает некоторое, заданное программистом, значение. Во втором — (см. ячейку “`x`” на иллюстрации выше) переменная получает **неопределенное** значение. **Проверьте экспериментально, что произойдет, если программа попытается обратиться к неинициализированной переменной (например, вывести ее на экран). С помощью отладчика узнайте, какое значение хранит неинициализированная переменная.**

2 Фундаментальные² типы данных

Тип переменной определяет множество возможных значений, которые она может хранить. Тип переменной определяется программистом при ее объявлении и впоследствии изменен быть *не может*³. Например, переменная типа `int` хранит целое число в диапазоне $-2^{31} \div 2^{31}-1$. Ниже приведен список наиболее часто используемых фундаментальных типов:

Тип	Описание	Диапазон значений
<code>int</code>	32-битное целое число со знаком	$-2^{31} \div 2^{31}-1$
<code>unsigned int</code>	32-битное целое число без знака	$0 \div 2^{32}-1$
<code>char</code>	8-битное целое число со знаком. Используется для хранения символов («букв») ⁴	$-128 \div 127$
<code>unsigned char</code>	8-битное целое число без знака.	$0 \div 255$
<code>float</code>	32-битное число с плавающей точкой ⁵	7 значащих цифр
<code>double</code>	64-битное число с плавающей точкой	15 значащих цифр

Указанную в таблице *разрядность* базовых типов можно узнать при помощи оператора `sizeof`:

```
cout << sizeof(double);
```

— этот оператор возвращает количество байт, занимаемых переменной указанного типа.

¹ Каждая переменная *обязательно* должна быть объявлена до первого использования. *А зачем авторы языка придумали такое правило?

² На основе *фундаментальных* типов можно конструировать *производные* типы: массивы, указатели, ссылки — а также пользовательские типы: перечисления(`enum`), структуры и классы.

³ *Приведите примеры из других языков программирования, в которых изменения типа переменной во время выполнения программы допускается.

⁴ А как можно хранить *символы* в переменной, которая представляет собой «8-битное целое число со знаком»?

⁵ А в связи с чем эту точку называли «плавающей»?

Задание 1. При помощи оператора `sizeof` определите экспериментально размеры указанных ниже типов. Результаты занесите в таблицу:

Тип	Размер
int	
unsigned int	
char	
unsigned char	
float	
double	
short	
long	
long long	
long double	

Задание 2. Проверьте: что получится, если к максимальному значению некоторого целого типа прибавить единицу? А если вычесть единицу из минимального значения? Проведите эти эксперименты для одного типа со знаком и одного беззнакового типа.

3 О положении переменной в памяти

Переменная — это именованная ячейка памяти. А каждая ячейка памяти имеет *адрес*. Как правило, адрес ячейки — это просто целое число, обозначающее ее порядковый номер. Адрес любой переменной можно узнать программно при помощи операции *взятия адреса* (&):

```
int a;
double b;
cout << "Address of a = " << &a << "\n";
cout << "Address of b = " << &b << "\n";
```

В результате выполнения данного фрагмента кода мы увидим на экране адреса переменных `a` и `b`. Например, результат может получиться такой:

```
Address of a = 0012FF6C
Address of b = 0012FF70
```

Это можно изобразить графически следующим образом:

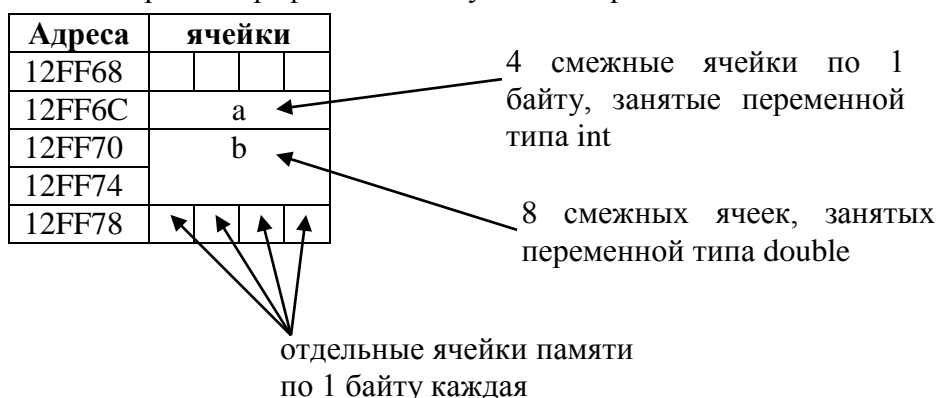


Рис. 1. Пример схемы расположения переменных в памяти

На рисунке видно, что переменная `a` занимает ячейки с адресами 12FF6C-12FF6F (включительно), а переменная `b` — адреса 12FF70-12FF77.

Задание 3. Объявите три или более переменные разного размера. Узнайте их адреса. Изобразите их на иллюстрации, подобной рис. 1 (нужна только таблица, стрелочек и пояснений не надо). Замечание: этот эксперимент нужно проводить с ехешкой, полученной в конфигурации проекта “Release” (меню Build->Configuration Manager).

4 Основные операции с переменными

Как таковая, любая переменная поддерживает **ровно две операции**:

1. **Получить значение переменной.**
2. **Изменить значение переменной.**

```
double pi;           // объявление без инициализации
pi = 3.14;           // изменение значения переменной
cout << pi;          // получение значения переменной и передача его
                     // оператору вывода (<<)
double k = pi;        // получение значения переменной pi и
                     // инициализация переменной k этим значением
pi = 3.1415926536;    // изменение значения переменной pi
```

5 Литералы

Непосредственные значения, помещенные прямо в текст программы, называются *литералами (literals)* и могут использоваться наравне с переменными — за тем очевидным исключением, что литерал не может стоять слева от знака "=". Литералы также имеют *тип*. В примере выше 3.14 и 3.1415926 — это литералы типа double. Тип литерала определяется компилятором автоматически. Чаще всего в программах встречаются литералы следующих типов:

Таблица 2. Примеры литералов.

Пример литерала	Его тип
25	int
25.0	double
'A'	char
"Hello"	char*

Литералы типа double также могут быть записаны в экспоненциальной форме; например: 1.6e-19 — что означает $1,6 \cdot 10^{-19}$.

Литералы целочисленных типов могут быть записаны в различных системах счисления. Например:

- **15** (десятичная система, число $10+5 = 15_{10}$);
- **015** (восьмеричная система (запись начинается с 0), число $8+5 = 13_{10}$);
- **0x15** (шестнадцатеричная система (запись начинается с 0x), число $16+5 = 21_{10}$).

В восьмеричной системе допустимы цифры от 0 до 7, в 16-ричной — от 0 до F, где A=10, B=11..F=15.

Подробнее о литералах см. раздел «Константы» главы 2 справочника по C++ под редакцией Шилдта (1_семестр\books\Полный справочник по C++.djvu, с. 54).

Задание 4. Определите, к какому типу относятся следующие литералы:

Литерал	Его тип	Литерал	Его тип
-2		2e+1	
'E'		'*'	
3.8		6.3e-1f	
"E"		18L	
-10.0		-2F	
"1979"		-5UL	
0x13		0XDEAL	

Выражения

Основной способ работы с переменными — это произведение с ними вычислений посредством построения *выражений* (*expressions*). Выражение — это комбинация имен переменных, литералов и вызовов функций, разделенных знаками операций. Результат *вычисления* (*evaluation*) выражения, как правило, записывается в какую-либо переменную посредством *операции присваивания* (=). Например:

```
h = v0*t + g*t*t / 2.0;
```

В этом примере сначала происходит извлечение из памяти значений всех переменных, стоящих справа от знака "=", затем на основе этих значений вычисляется значение выражения — и, наконец, это значение присваивается переменной h. Графически структуру этого выражения (всего, что справа от "=") можно представить в виде следующего дерева:

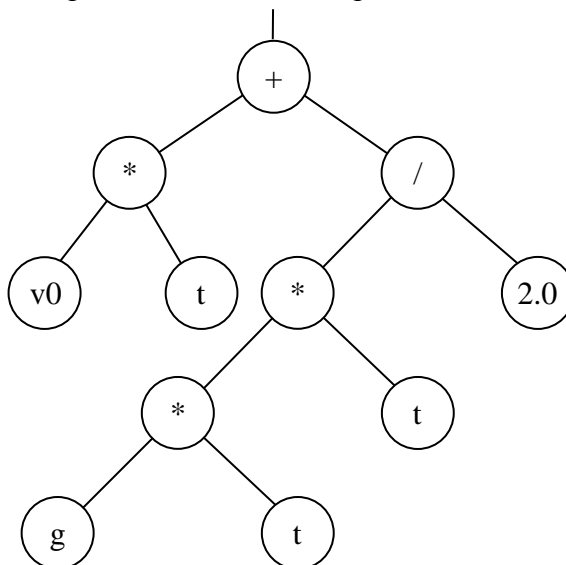


Рис. 2. Дерево разбора выражения

1 Приоритеты операций

При внимательном рассмотрении приведенного выше дерева можно заметить, что операции умножения (*) и деления (/) выполняются *первыми*, а операция сложения (+) выполняется уже *над их результатами*. Это соответствует следующему способу расстановки скобок в первоначальном выражении (сопоставьте с деревом)¹:

```
h = (v0 * t) + (((g * t) * t) / 2.0);
```

Такую интерпретацию данного выражения компилятор выбрал исходя из своего представления о *приоритетах операций*. А именно:

1. Операции * и / более приоритетны, чем + и - (выполняются первыми).
2. Операции * и / равноправны — и поэтому выполняются *в порядке их следования* в тексте программы (слева направо).

Задание 5. Постройте дерево (для того же выражения), аналогичное приведенному на рис. 2, но в предположении, что приоритеты операций инвертированы: + и - более приоритетны, чем * и /.

Ниже приведена таблица приоритетов основных операций C++. Операции с большим численным значением приоритета выполняются раньше.

Таблица 3. Приоритеты операций.

Приоритет	Операторы	Описание
13	++, --	Постфиксный инкремент и декремент
12	++, --	Префиксный инкремент и декремент

¹ "Наивный" компилятор просто выполнял бы все действия по порядку: (((((v0 * t) + g) * t) * t) / 2.0) — что дало бы совсем другой результат.

	~ ! +, - & * Явное и неявное преобразование типа	Побитовая инверсия Логическое «не» Унарные плюс и минус Получение адреса Разыменование
11	*, /, %	Умножение, деление, остаток от деления
10	+, -	Сложение, вычитание
9	>>, <<	Побитовый сдвиг вправо/влево, ввод-вывод
8	<, >, >=, <=	Операторы сравнения
7	==, !=	Операторы проверки на равенство
6	&	Побитовое «и»
5	^	Побитовое «исключающее или»
4		Побитовое «или»
3	&&	Логическое «и»
2		Логическое «или»
1	=, * =, / =, % =, + =, - = > > =, < < =, & =, ^ =, =	Различные варианты присваивания ¹

Задание 6. Расставьте скобки в следующих выражениях согласно приоритетам операций:

1. $a + b * a - b$
2. $x + h > x - h$
3. $i \geq 1 \ \&\& \ j < 4$
4. $! x < 2 \ || \ x > 4$
5. $- x * y$
6. $x == 0 \ || \ x / y > 1 \ \&\& \ y != 0$
7. $a > b != b < c$
8. $n \ \& \ 2 == 0$
9. $a \wedge b \ \& \ c \ | \ d$
10. $k + m < j \ || \ 3 - j \geq k$

2 Контроль типов

Первое правило контроля типов гласит:

Оба аргумента бинарной операции должны иметь один и тот же тип.

Попытаемся, однако, сложить `int` с `double`¹ом:

```
int i = 3;
double d = 2.0;
cout << d + i;           // нарушение?
```

В данном примере операция сложения («+») вызывается с двумя аргументами, имеющие *разные* типы. А это, согласно правилу, *запрещено*².

Однако, такая запись в программе на C++ будет вполне корректной. Дело в том, что при обработке выражения “(d + i)” компилятор автоматически сгенерирует код,

¹ Выше было сказано, что выражение — это “все, что стоит справа от знака “=”. Что же тогда операция “=” делает в таблице приоритетов операций? Выходит, она (операция “=”) тоже может появляться в «справа от знака “=»? Проверьте, допустима ли запись: “a=b=c;”. И как ее следует понимать?

² Хотя бы потому, что в процессоре нет устройства, которое умеет складывать целые числа с числами с плавающей точкой. Числа этих двух форматов обрабатываются отдельными блоками ЦПУ.

осуществляющий *преобразование типа*. В данном случае значение типа `int` будет преобразовано в значение типа `double`.

Направление преобразования `int`->`double` (а не наоборот) компилятор выбрал потому, что тип `double` является более «широким», чем `int`. Общее же правило гласит:

Когда бинарная операция применяется к разным типам, более «узкий» из них преобразуется в более «широкий».

Все эти действия осуществляются в соответствии с иерархией фундаментальных типов C++ (стрелки проведены в направлении «расширения» типов):

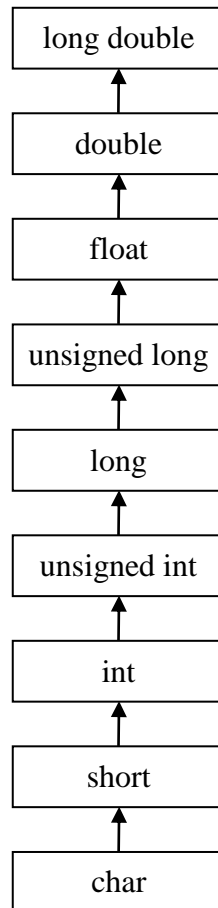


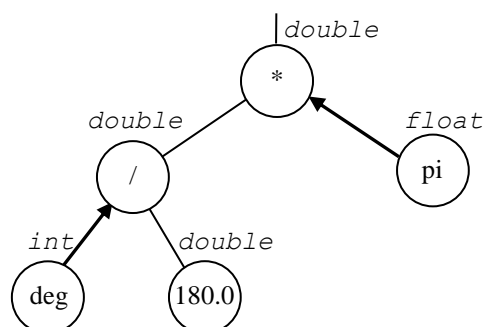
Рис. 3. Иерархия фундаментальных типов

Для примера рассмотрим фрагмент кода, переводящий величину угла в градусах (будем рассматривать всегда целое число градусов) в радианы:

```
int deg = 90;
float pi = 3.14159;
float rad = (deg / 180.0) * pi;
```

Нас интересует выражение, записанное в третьей строке.

Построим дерево его разбора, подписав *типы всех операндов* и отметив жирной стрелкой *преобразование типов*:



Задание 7. Постройте дерево разбора с указанием преобразований типов (аналогично рис. 4) для выражения из следующего фрагмента:

```
short days = 512;
int year = 2000 + days/365;
```

Замечание: более подробно о преобразованиях типов см. раздел «Выражения» главы 2 справочника по C++ под редакцией Шилдта (1_семестр\books\Полный справочник по C++.djvu, с. 70).

Явное приведение типа

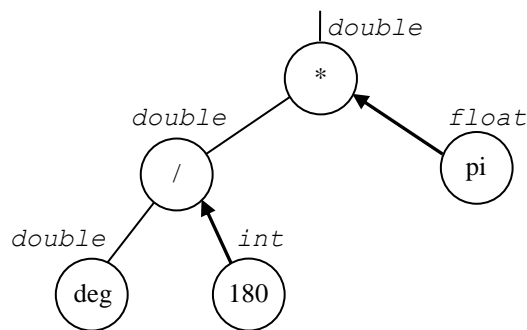
Довольно часто возникает необходимость явно указать, к какому типу следует привести тот или иной операнд в выражении. Это можно сделать при помощи операции явного приведения типа, поместив тип в круглых скобках непосредственно перед операндом:

(тип) операнд

Например, программу перевода из градусов в радианы можно было бы написать так:

```
int deg = 90;
float pi = 3.14159;
float rad = ((double)deg / 180) * pi;
```

В этом случае преобразования типов происходили бы следующим образом (сравните с рис. 4):



Задание 8. По данным социологов, из N опрошенных человек на улицах Харькова за Партию Программистов собираются голосовать M человек. Используя *операцию приведения к типу double*, напишите программу, которая по заданным N и M поможет социологам определить, сколько *процентов* респондентов собираются голосовать за Партию Программистов. Результат представьте в виде целого числа, отбросив дробную часть. Постройте для своей программы дерево разбора выражения с указанием преобразований типов (аналогично приведенному выше).

Контрольные вопросы

1. Какой адрес имеет правая верхняя ячейка на рис. 1?
2. Какое максимальное и минимальное значение имеют целочисленные типы, указанные в таблице из «задания 1» на стр. 2?
3. Почему тип `double` имеет больший размер, чем `float`?
4. Какие Вам известны *другие* применения символа “&” в языке C++, кроме операции взятия адреса?
5. Как размеры переменных влияют на адреса их расположения в памяти?
6. Из каких строк таблицы 3 следует, что умножение должно выполняться раньше сложения?
7. Почему операции логические операции (`||`, `&&`) имеют меньший приоритет, чем операции сравнения (`>`, `==` и др.)?

Оценивание

Содержание отчета

Задание 1. Условие и заполненная таблица.

Задание 2. Условие, исходный код программы, выводящей на экран результаты эксперимента, скриншот ее выполнения. Блок-схема и диаграмма потоков данных не нужны.

Задание 3. Условие, исходный код программы, выводящей на экран результаты эксперимента, скриншот ее выполнения, заполненная таблица. Блок-схема и диаграмма потоков данных не нужны.

Задание 4. Заполненная таблица.

Задание 5. Условие и дерево.

Задание 6. Условие и выражения с правильно расставленными скобками.

Задание 7. Условие и дерево.

Задание 8. Диаграмма потоков данных, блок-схема, исходный код и скриншот выполнения программы. Дерево.

Баллы за задания

Задание	Баллы	
1	обязательное	1
2	обязательное	
3	1	
4	обязательное	1
5	обязательное	
6	1	
7	1	
8	1.5	
Всего	7,5	

Бонусы

Досрочная сдача: +1 балл.

Несвоевременная сдача: макс. балл уменьшается на 1 за каждую просроченную неделю.

Следующая бонус-задача предназначена для тех, кто уверен, что полностью понимает весь изложенный выше материал, без труда ответит на любые вопросы по нему и «с ходу» решит любое из заданий 1-8. Если это — о Вас, то вместо лабы можно сделать приведенную ниже задачу (и получить за это **10 баллов**). Единственная просьба: заимствованный код не использовать. См. также «Критерии_оценивания.doc» по поводу правильной организации исходного кода. Задача принимается в течение двух недель.

Бонус-задача

Написать программу разбора и вычисления значения арифметического выражения.

На входе программы — строка, содержащая числа, скобки «(» и «)» и знаки 4-х арифметических операций: +-* / (по желанию можно добавить и другие операции).

На выходе — результат вычисления.

Литература:

1. <http://algotlist.manual.ru/syntax/revpn.php>
2. http://e-maxx.ru/algo/expressions_parsing