Лекция № 2

Основные типы данных

Основным средством хранения данных, обрабатываемых программой, являются ячейки памяти. Вся память компьютера является однородной и представляет собой единый массив адресуемых ячеек — байтов, представляющих собой группу из восьми бит, носителей минимальной порции информации.

Определение. Бит — двоичная единица; минимальное количество информации, соответствует той информации, которая доставляется пользователю при приеме *одного из двух* равновероятностных сообщений.

Байт — минимальная адресуемая ячейка памяти; объем: 1 байт = 8 бит.

Переменная базового типа представляет собой байт или группу байт, которые обрабатываются как единое целое и имеющее имя — *идентификатор*. Информация, хранящаяся в этой группе, воспринимается как единое *число* целого или вещественного типа. Типы данных (базовые) используемые в языке С (и С++) представлены в таблице 1 (см. Приложение).

Описание переменных

Описание переменных в программе С (и С++) выполняется следующим образом:

```
[класс памяти] <тип> <список переменных>;
```

Примеры описания:

```
int v;  // переменная целого типа
long int w;  // переменная типа целого длинного
unsigned int i, j, k;  // несколько переменных целого типа без знака
float x, y, z;   // несколько переменных вещественного типа
long double p, r, s, t;  // несколько переменных вещественного типа
                         // максимальной точности
char f;   // переменная для хранения символов
```

При написании текста программы в некоторых случаях, например при выделении памяти, требуется учитывать размер переменной, то есть, сколько байт она занимает. Тогда можно не указывать конкретное число (то есть константу), а заставить проделать это компилятор, обратившись к функции `sizeof ()`.

Пример:
```
k = sizeof (long double) * 10; // k сохранить количество байт необходимых
                               // для запоминания 10 элементов типа long double
m = sizeof (n);  // в m сохранится количество байт, которые занимает n
```

В языке С (и С++) нет переменных логического типа. Но надо, что отметить в последующих версиях такой тип введен.

Операторы в языке С++

Оператор представляет собой составную конструкцию и является основным строительным боком программы.

Определение. *Операнд* — то над чем выполняется действие.
Операция — суть действие над операндом. Операции обладают приоритетами.
Выражение — объединение операций и операндов.

Пример: `-56  -a  a+b  c/d`

Операции выполняются слева направо, если имеют один приоритет. Но операции языка С (и С++) имеют приоритеты, то есть операции с бо́льшим приоритетом выполняются раньше остальных. Приоритеты операция языка С (и С++) указаны в таблице 2 (см. Приложение).
Операции делятся на три группы:

- Унарные — выполняемые над одним операндом;

- Бинарные — выполняемые над двумя операндами;
- Тернарные — выполняемые над тремя операндами.

Порядок выполнения операций определяется приоритетом операций и их расположением слева направо. Так же этот порядок можно изменять, используя круглые скобки, которые задают выражению, заключенному в них, более высокий приоритет. То есть, как и в математике, выражение, заключенное в скобки, имеет более высокий приоритет и вычисляется предварительно, а уже потом его значение участвует в операциях указанных между скобок.

Следует особо отметить унарные операции автоматического инкрементирования "+ +" и декрементирования "– –". При воздействии на операнд такие операции приводят к увеличению или уменьшению текущего его значения на единицу. Так при применении к целому числу его значение увеличивается или уменьшается на единицу. А при воздействии на типизированный указатель увеличивает его значение так, чтобы указатель стал ссылаться на следующий или предыдущий элемент. При этом операции инкрементирования и декрементирования разделяют на два типа:

- префиксные — воздействуют на операнд и изменяют его значение до того как операнд будет участвовать в дальнейших вычислениях;
- постфиксные — воздействуют на операнд и изменяют его значение после того как операнд будет участвовать в дальнейших вычислениях.

Операции инкрементирования и декрементирования не рекомендуется применять к одной и той же переменной *многократно в одном выражении*, так как это усложняет чтение такого оператора, да точный порядок выполнения этих пре/постфиксных операций зависит от конкретного компилятора. Помимо этого надо быть осторожным с применением этих операций в логических выражениях. Вычисление логических выражений оптимизируется, и часть логического выражения может не вычисляться и, как следствие, пре/постфиксная операция оказавшаяся в этой части тоже не будет выполнена.

Итак, в языке С (и С++) следующие конструкции являются эквивалентными, а значение переменной будет изменяться одинаково, если она поучаствует в одном из следующих операторах:

Увеличение	i = i + 1;	i + = 1;	+ +i;	i+ +;
Уменьшение	i = i – 1;	i – = 1;	– –i;	i– –;

Но результаты операторов, в которых используются префиксные или постфиксные операции будут различны:

Увеличение	a = b / c * (+ +i);	Значение **i** изменяется и новое значение участвует в вычислениях	a = b / c * (i++);	Вычисляется выражение, а затем изменяется значение **i**
Уменьшение	a = b / c * (– –i);		a = b / c * (i– –);	

Определение. *Оператор* — законченная инструкция для компьютера. Признаком оператора является наличие знака "**;**" на конце (то есть, как правило это выражение с "**;**" на конце). "**;**" — входит в синтаксис оператора, является его неотъемлемой частью (в языке Pascal "**;**" — разделитель).

Итак, операторы это основные строительные блоки программы, а *программа* — суть последовательность операторов с добавлением небольшого количества знаков пунктуации.

В языке С++ *операторы*, помимо основного их действия, *возвращают значение*, которое может использоваться как операнд в других операторах. Более того операция «**,**» в языке С (и С++) позволяет объединить несколько операторов в один, причем, его значение будет равно значению последнего выполняемого оператора из числа объединенных (то есть самого правого).

Операторы языка С (и С++) перечислены в таблице 3 (см. Приложение).

Оператор присваивания

Основным оператором в языке программирования является *оператор присваивания*. Его назначение достаточно просто — указать, какие вычисления необходимо выполнить и куда положить результат. Общий вид оператора присваивания:

<переменная> = *<выражение>*;

Примеры: `y = 5.8 + a/b - f + x*y;` `t = 2*3.141516*R;`

Результат работы оператора: вычисленное значение *выражения* в правой части преобразуется к типу *переменной* в левой части и затем помещается в эту переменную.

Значением оператора присваивания является результат значения выражения в правой части, преобразованный к типу объекта в левой части. То есть, вычисленное значение выражения укладывается в переменную, указанную в левой части, и доступно как значение оператора. Оператор присваивания, как операция, может использоваться в выражениях наравне с любыми операциями. Как отмечалось ранее в языке С (и С++) можно объединить несколько операторов в один перечисли их через запятую, а значение такого оператора будет равно значению последнего, правого. На основе оператора присваивания правильными будут и такие конструкции:

`i = 0, j = 1;` — обеспечивает присваивание переменным `i`, `j` и имеет свое значение равное 1, но оно никак не используется;

`x = (i = 0, j = 1);` — здесь значение всего оператора (а точнее правого оператора) присваивается `x`, то есть: `x = 1`.

Условный оператор

Программа представляет собой последовательный список операторов, которые и выполняются в такой последовательности. Но, как правило, в ходе выполнения вычислений приходится выполнять анализ промежуточных результатов и менять линейный порядок выполнения операторов. Условный оператор и предназначен для организации ветвящихся вычислений. Общий вид условного оператора:

```
if (выражение) оператор;
```

Выражение, указанное в круглых скобках, в общем случае, является целочисленным. Если же это не так, то результат вычисления этого выражения приводится к целочисленному типу.

Записываться это выражение может в форме сходной с другими языками, в виде логического выражения, например `a>b`. Правила языка С (и С++) предполагают, что результатом логического выражения будет целое число общего вида (то есть 1–2 байта, знаковое или беззнаковое), равное **1** при выполнении условия, и **0** — при невыполнении.

Помимо этого, *выражение* может быть простым арифметическим выражением целочисленного типа. Если значение выражения не целочисленное, то оно приводится к целочисленному типу.

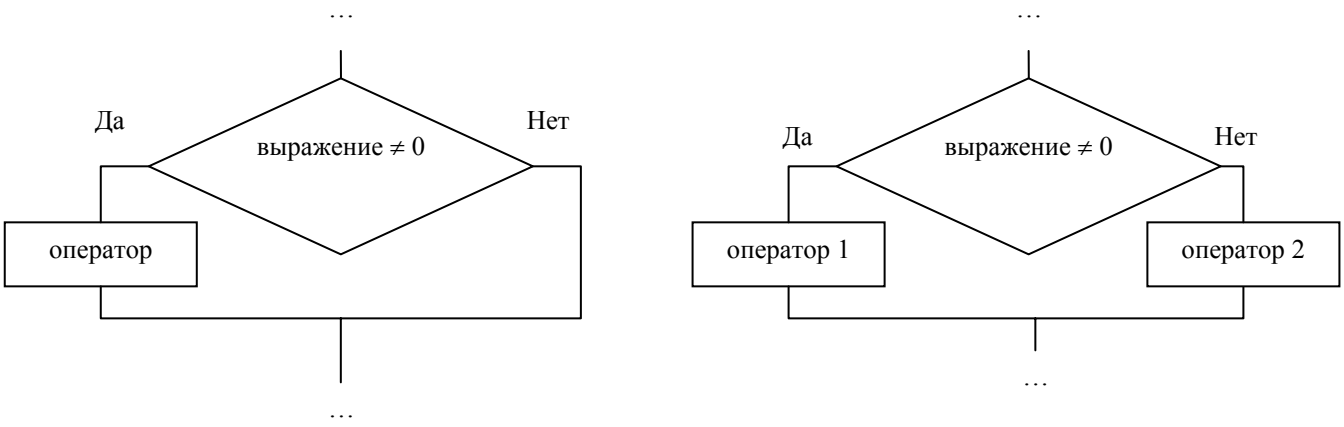
Порядок выполнения условного оператора следующий. Если значение *выражения* равно 1 (в общем случае отлично от нуля; если выражение логическое и оно истинно), то *оператор* выполняется. Если значение *выражения* равно 0 (если выражение логическое и оно ложно), то *оператор* не выполняется, а программа переходит к выполнению оператора, следующего за условным.

Указанная форма условного оператора является сокращенной формой оператора. Полная форма условного оператора:

```
if (выражение) оператор 1;
else оператор 2;
```

Порядок выполнения: если логическое выражение истинно (или целочисленное арифметическое отлично от нуля) то выполняется *оператор 1*; иначе выполняется *оператор 2*. После этого выполняется оператор, следующий за условным оператором.

Логику работы условного оператора полной формы и сокращенной легко увидеть в графическом отображении работы алгоритма — в виде блок–схемы.



а) сокращенная форма б) полная форма

Рис. Блок–схемы работы условного оператора

Если *оператор* составной, то используют блоковые скобки “**{}**”. То есть в самом общем виде условный оператор имеет вид:

```
if (выражение)
 {оператор 1;
  оператор 2;
  …
  оператор N;
 }
else
 {оператор 1;
  оператор 2;
  …
  оператор M;
 };
```

Необходимо учитывать, что выполнение *выражений*, содержащих логические операции, оптимизируется. То есть оно может выполняться не полностью. Так если два логических выражения соединены операцией **или** (знак “||”),а левая часть **истина**, то нет необходимости вычислять правую (ведь результат и так будет **истина**). А если два логических выражения соединены операцией **и** (знак “&&”) а левая часть **ложь**, то нет необходимости вычислять правую (ведь результат и так будет **ложь**). Результаты операций логических **и** и **или** указаны в таблице 4 (см. Приложение).

Поэтому при формировании логических выражений с использованием префиксных/постфиксных операций необходимо быть осторожным. Так как, если часть выражения не будет выполняться, то и не отработает операция префиксного/постфиксного изменения переменной, указанная в этой части.

В языке С (и С++) приведение результата логического выражения к целочисленному значению делает возможным смешанные логико-арифметические операции, например:

```
f = (a > b) * (c + d) + (a <= b) * (c - d);
```

В простейших случаях вместо условного оператора может быть использована тернарная операция. В общем виде эта операция имеет вид:

```
операнд_1 ? операнд_2 : операнд_3
```

и возвращает значение `операнда_2` или `операнда_3`, в зависимости от значения `операнда_1`. Пример использования:

```
f = (a > b) ? (c + d) : (c - d);
```

Здесь значение переменной `f` будет равно `c + d`, если условие выполняется (то есть `a > b`), а в противном случае (то есть `a ≤ b`) переменной `f` будет присвоено результат вычисления `c - d`.

Приложение

Таблица 1

Базовые типы переменных языка С++

	Тип	Длина	Пределы значений	Примеры	
символьные	unsigned char	8 bits	0 до 255	'A'; 'ю'; '\007'	
	char	8 bits	-128 до 127	'g'; 'Д'	
целые	enum	16 bits	-32 768 до 32 767	128; -15	
	unsigned int	16 bits	0 до 65 535	117; 8; 0; 32001	
	short int	16 bits	-32 768 до 32 767	-31017; 115	
	int	16 bits	-32 768 до 32 767	-32135; 2135	
	unsigned long	32 bits	0 до 4 294 967 295	294967295	
	long	32 bits	-2 147 483 648 до 2 147 483 647	-14748364	
С плавающей запятой	float	32 bits	$\pm 3.4 * 10^{-38}$ до $\pm 3.4 * 10^{+38}$	2.71828	
	double	64 bits	$\pm 1.7 * 10^{-308}$ до $\pm 1.7 * 10^{+308}$	-3.1415e7	
	long double	80 bits	$\pm 3.4 * 10^{-4932}$ до $\pm 1.1 * 10^{+4932}$	-101e-1999	

Таблица 2

Операции Си в порядке убывания приоритета

Операция	**Назначение**	**Порядок**
[]	задание элемента массива	слева направо
()	вызов функции	слева направо
.	выбор поля структуры	слева направо
–>	выделения поля структуры с помощью указателя	слева направо
++, --	постфиксное, префиксное увеличение и уменьшение на 1, если одно и тоже встречается в одном выражении, то постфиксное имеет более высокий приоритет	справа налево
sizeof	определение размера в байтах	справа налево
(тип)	приведение к типу	справа налево
~	побитовое отрицание	справа налево
!	логическое не	справа налево
-	унарный минус	справа налево
&	определение адреса	справа налево
*	обращение по адресу	справа налево
*, /, %	умножение, деление, остаток от деления	слева направо
+, –	сложение, вычитание	слева направо
<<, >>	сдвиг	слева направо
<, >, <=, >=	сравнение	слева направо
==, !=	равно, неравно	слева направо
&	побитовое И	слева направо
^	побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ	слева направо
|	побитовое ИЛИ	слева направо
&&	логическое И	слева направо
||	логическое ИЛИ	слева направо
?:	условная операция (тернарная)	справа налево
=, +=, –=, *=, /=, <<=, >>=, &= ^=, |=	присваивание, замещение	справа налево
,	предписывает последовательность вычислительных выражений	слева направо

Таблица 3

Операторы языка С++

Оператор	Назначение	Формат	Пример
присваивания	переменной присваивается значение выражения	переменная = выражение;	a = 3.1415; b = 2.7 + a;
условный	если *выражение* истинно, то выполняется *оператор*	**if** (*выражение*) *оператор*;	if (a > b) c = d + h;
	если *выражение* истинно, то выполняется *оператор1*; если *выражение* ложно, то выполняется *оператор2*.	**if** (*выражение*) *оператор1*; **else** *оператор2*;	if (a > b) c = d + h; else f = t * m;

переключения	если *выражение* равно *конст_k*, то выполнить *операт_k;* если нет ни одного совпадения,то выполнить *операторы*.	**switch** (выражение) {
		case конст_1 : операт_1;
		case конст_2 : операт_2;
		…
		case конст_N : операт_N;
		default : операторы;
		}
			switch (x) {
			case 'A': printf ("Буква А");
			break;
			case 'B': printf ("Буква В");
			break;
			default : printf ("Другая буква")
			}

цикла	пока *выражение* истинно, выполнять *оператор*	**while** (*выражение*)
		оператор;
			while (k < n) {
			y = y * x;
			k++;
			}

	выполнять *оператор*, пока *выражение* истинно	**do**
		оператор
		while (*выражение*);
			i = 0;
			do
			a [i] = i * i;
			while (++i <= 100);

	выполнить *выраж1*; пока *выраж2* истинно, выполнять *оператор* и *выраж3*	**for** (*выраж1*; *выраж2*; *выраж3*) *оператор*;	**for** (i = 0; i <= 100; i++) a[i] = i * i;
завершения	прекратить выполнение switch, while, do, for	**break**;	
продолжения	продолжить выполнение while, do, for	**Continue**;	
возврата	прекратить выполнение функции	**return**;	**return** x + y;
перехода	перейти на оператор с *меткой*	**goto** *метка*;	**goto** BACK;

Таблица 4

Логические операции
логические "И" (&&) и "ИЛИ" (||)

&&		
B / **A**	**истина**	**ложь**
истина	**истина**	**ложь**
ложь	**ложь**	**ложь**

||		
B / **A**	**истина**	**ложь**
истина	**истина**	**истина**
ложь	**истина**	**ложь**