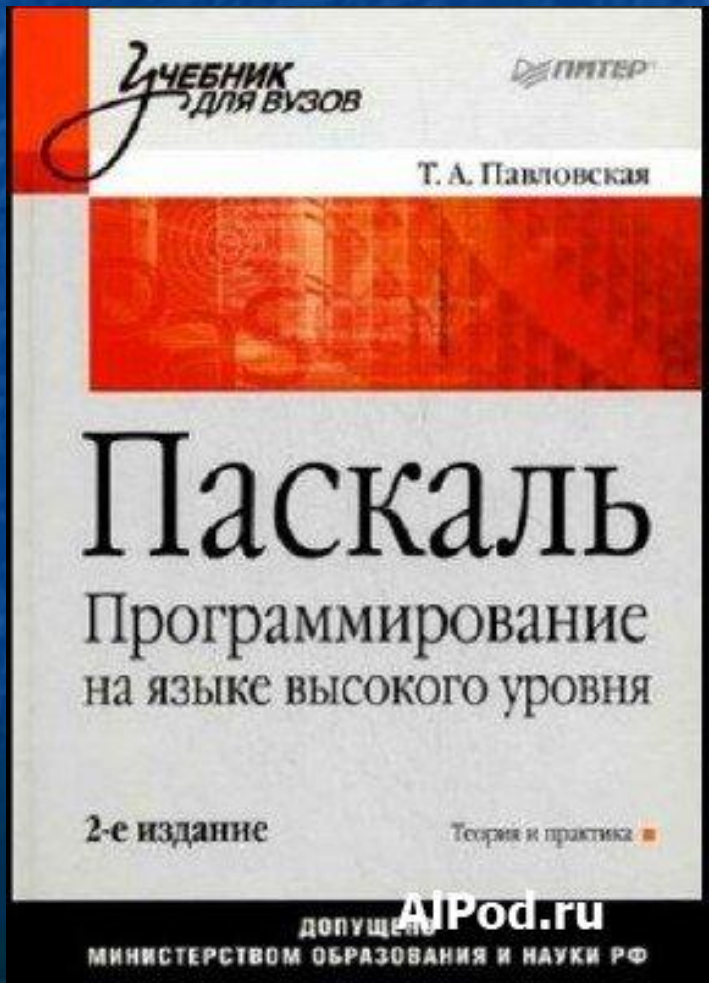
МТИ

МОСКОВСКИЙ ТЕХНОЛОГИЧЕСКИЙ ИНСТИТУТ

Программирование на языке высокого уровня.



Бужинский В.А. ктн доцент
bva2516@mail.ru

Москва 2016

Вебинар № 1/2.

Основные понятия и определения

Основная литература

Ашарина И.В. Объектно-ориентированное программирование в C++ [Электронный ресурс]: учебное пособие/ Ашарина И.В.— Электрон. текстовые данные.— М.: Горячая линия - Телеком, 2012.— 320 с.— Режим доступа: http://www.iprbookshop.ru/12008.

Казанский А.А. Объектно-ориентированное программирование на языке Microsoft Visual C# в среде разработки Microsoft Visual Studio 2008 и .NET Framework. 4.3 [Электронный ресурс]: учебное пособие и практикум/ Казанский А.А.— Электрон. текстовые данные.— М.: Московский государственный строительный университет, ЭБС АСВ, 2011.— 180 с.— Режим доступа: http://www.iprbookshop.ru/19258.

Агапов В.П. Основы программирования на языке C# [Электронный ресурс]: учебное пособие/ Агапов В.П.— Электрон. текстовые данные.— М.: Московский государственный строительный университет, ЭБС АСВ, 2012.— 128 с.— Режим доступа: http://www.iprbookshop.ru/16366.

Высокоуровневый язык программирования — язык программирования, разработанный для быстроты и удобства использования программистом. Основная черта высокоуровневых языков — это абстракция, то есть введение смысловых конструкций, кратко описывающих такие структуры данных и операции над ними, описания которых на машинном коде (или другом низкоуровневом языке программирования) очень длинны и сложны для понимания.

Высокоуровневые языки программирования были разработаны для платформенной независимости сути алгоритмов. Зависимость от платформы перекладывается на инструментальные программы — трансляторы, компилирующие текст, написанный на языке высокого уровня, в элементарные машинные команды (инструкции). Поэтому, для каждой платформы разрабатывается платформенно-уникальный транслятор для каждого высокоуровневого языка, например, переводящий текст, написанный на Delphi в элементарные команды микропроцессоров семейства x86.

Так, высокоуровневые языки стремятся не только облегчить решение сложных программных задач, но и упростить портирование программного обеспечения. Использование разнообразных трансляторов и интерпретаторов обеспечивает связь программ, написанных при помощи языков высокого уровня, с различными операционными системами программируемыми устройствами и оборудованием, и, в идеале, не требует модификации исходного кода (текста, написанного на высокоуровневом языке) для любой платформы.

Программы, написанные на языках высокого уровня, проще для понимания программистом, но менее эффективны, чем их аналоги, создаваемые при помощи низкоуровневых языков. Одним из следствий этого стало добавление поддержки того или иного языка низкого уровня (язык ассемблера) в ряд современных профессиональных высокоуровневых языков программирования.

Примеры: C++, C#, Java, JavaScript, Python, PHP, Ruby, Perl, Паскаль, Delphi, Лисп. Языкам высокого уровня свойственно умение работать с комплексными структурами данных. В большинстве из них интегрирована поддержка строковых типов, объектов, операций файлового ввода-вывода и т. п.

Первым языком программирования высокого уровня считается компьютерный язык Plankalkül, разработанный немецким инженером Конрадом Цузе ещё в период 1942—1946 годах. Однако транслятора для него не существовало до 2000 года. Первым в мире транслятором языка высокого уровня является ПП (Программирующая Программа), он же ПП-1, успешно испытанный в 1954 году. Транслятор ПП-2 (1955 год, 4-й в мире транслятор) уже был оптимизирующим и содержал собственный загрузчик и отладчик, библиотеку стандартных процедур, а транслятор ПП для ЭВМ Стрела-4 уже содержал и компоновщик (linker) из модулей. Однако, широкое применение высокоуровневых языков началось с возникновением Фортрана и созданием компилятора для этого языка (1957).

Основные понятия языка

Состав языка

Для решения задачи на компьютере требуется написать программу. Программа состоит из исполняемых операторов и операторов описания. Исполняемый оператор задает законченное действие, выполняемое над данными. Примеры исполняемых операторов: вывод на экран, занесение числа в память, выход из программы.

Оператор описания, как и следует из его названия, описывает данные, над которыми в программе выполняются действия. Примером описания (конечно, не на Паскале, а на естественном языке) может служить предложение 'В памяти следует отвести место для хранения целого числа, и это место мы будем обозначать А'.

Исполняемые операторы для краткости часто называют просто операторами, а операторы описания — описаниями. Описания должны предшествовать операторам, в которых используются соответствующие данные. Операторы исполняются последовательно, один за другим, если явным образом не задан иной порядок.

Рассмотрим простейшую программу на Паскале. Все, что она делает — вычисляет и выводит на экран сумму двух целых чисел, введенных с клавиатуры.

```
var a, b, sum : integer;                                    { 1 }
begin                                                       { 2 }
    readln(a, b);                                           { 3 }
    sum := a + b;                                           { 4 }
    writeln('Сумма чисел ', a, ' и ', b, ' равна ', sum);   { 5 }
end.                                                        { 6 }
```

В программе шесть строк, каждая из них помечена комментарием с номером (внутри фигурных скобок можно писать все, что угодно).

В строке 1 расположен оператор описания используемых в программе величин. Для каждой из них задается имя, по которому к ней будут обращаться, и ее тип. 'Волшебным словом' var обозначается тот факт, что a, b и sum — переменные, то есть величины, которые во время работы программы могут менять свои значения. Для всех переменных задан целый тип, он обозначается integer. Тип необходим для того, чтобы переменным в памяти было отведено соответствующее место.

```
var a, b, sum : integer;                                    { 1 }
begin                                                       { 2 }
    readln(a, b);                                           { 3 }
    sum := a + b;                                           { 4 }
    writeln('Сумма чисел ', a, ' и ', b, ' равна ', sum);   { 5 }
end.                                                        { 6 }
```

Исполняемые операторы программы располагаются между служебными словами begin и end, которые предназначены для объединения операторов и сами операторами не являются. Операторы отделяются друг от друга точкой с запятой.

Ввод с клавиатуры выполняется в строке 3 с помощью стандартной процедуры с именем readln. В скобках после имени указывается, каким именно переменным будут присвоены значения. Для вывода результатов работы программы в строке 5 используется стандартная процедура writeln. В скобках через запятую перечисляется все, что мы хотим вывести на экран, при этом пояснительный текст заключается в апострофы. Например, если ввести в программу числа 2 и 3, результат будет выглядеть так:

Сумма чисел 2 и 3 равна 5

В строке 4 выполняется вычисление суммы и присваивание ее значения переменной sum. Справа от знака операции присваивания, обозначаемой символами :=, находится выражение — правило вычисления значения.

Чтобы выполнить программу, требуется перевести ее на язык, понятный процессору, — в машинные коды. Этим занимается компилятор. Каждый оператор языка переводится в последовательность машинных команд. Компилятор планирует размещение данных в оперативной памяти в соответствии с операторами описания. Попутно он ищет синтаксические ошибки, то есть ошибки записи операторов. Кроме этого, в Паскале на компилятор возложена еще одна обязанность — подключение к программе стандартных подпрограмм (например, ввода данных или вычисления синуса угла).

МАШИННЫЕ КОМАНДЫ	КОМАНДЫ АССЕМБЛЕРА			
	МЕТКИ	КОМАНДЫ		КОММЕНТАРИЙ
00100100 01011111	СУМНЕЧЕТ	**MOVE.L**	(A7) +,A2	Заслать адрес возврата из стека в A2.
00100010 01011111		**MOVE.L**	(A7) +,A1	Заслать адрес первого числа в A1.
00110010 00011111		**MOVE.W**	(A7) +,D2	Заслать n в D1.
01000010 01000010		**CLR.W**	D2	Обнулить D2.
01001110 11111010 00000000 00001110		**JMP**	СЧЕТЧИК	Перейти в конец цикла n = 0?
00001000 00101001 00000000 00000000 00000000 00000001	ЦИКЛ	**BTST**	0,1(A1)	Если число по адресу A1 четное...
01100111 00000010		**BEQ.S**	СЛЕД.	...то перейти на СЛЕД
11010100 01010001		**ADD.W**	(A1),D2	...если нет, прибавить число к D2.
01010100 01001001	СЛЕД.	**ADDQ.W**	#2,A1	Записать в A1 адрес следующего числа.
01010001 11001001 11111111 11110010	СЧЕТЧИК	**DBF**	D1,ЦИКЛ	Уменьшить D1; пока не -1, переход на ЦИКЛ.
00111110 10000010		**MOVE.W**	D2,–(A7)	Протолкнуть сумму из D2 в стек
01001110 11010010		**JMP**	(A2)	Перейти по адресу возврата.

Алфавит и лексемы

Все тексты на языке пишутся с помощью его алфавита. Алфавит Паскаля включает:
прописные и строчные латинские буквы, знак подчеркивания _ ;
цифры от 0 до 9 ;
специальные символы, например +, *, { и @ ;
пробельные символы: пробел, табуляцию и переход на новую строку.

Из символов составляются лексемы (tokens), то есть минимальные единицы языка, имеющие самостоятельный смысл:
константы ;
имена (идентификаторы);
ключевые слова ;
знаки операций ;
разделители (скобки, точка, запятая, пробельные символы).

Лексемы языка программирования аналогичны словам естественного языка. Например, лексемами являются число 128, имя Vasia, ключевое слово goto и знак операции сложения +. Компилятор при синтаксическом разборе текста программы определяет границы одних лексем по другим, например разделителям или знакам операций. Из лексем строятся выражения и операторы.

Константы

Константа — величина, не изменяющая свое значение в процессе работы программы. Классификация констант Паскаля приведена в шапке таблица 1.1. Нижние строки таблицы представляют собой примеры соответствующих констант.

Таблица 1.1. Классификация констант Паскаля

Целые		Вещественные		Символьные	Строковые
Десятичные	Шестнадцатеричные	С плавающей точкой	С порядком		
2	$0101	-0.26	1.2e4	'k'	'абырвалг'
15	$FFA4	.005	0.1E-5	#186	'I''m fine'
		21.		^M	

Десятичные **целые константы** представляются в естественной форме. Шестнадцатеричная константа состоит из шестнадцатеричных цифр (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F), предваряемых знаком $.

Вещественные константы записываются с точкой перед дробной частью. Либо целая, либо дробная часть могут отсутствовать. Вещественная константа с порядком представляется в виде **мантиссы** и **порядка**. Мантисса записывается слева от знака E или e, порядок — справа от знака. Значение константы равно произведению мантиссы и возведенного в указанную в порядке степень числа 10. Пробелы внутри числа не допускаются.

Символьные константы служат для представления любого символа из набора, используемого в данном компьютере. Так как под каждый символ отводится 1 байт, всего используется 256 символов. Каждому символу соответствует свой код. В операционной системе MS-DOS для кодировки символов используется стандарт ASCII, являющийся международным только в первой половине кодов (от 0 до 127), вторая половина кодов (от 128 до 255) является национальной и различна для разных стран. Первые 32 символа являются **управляющими**: хотя многие из них имеют графическое представление, предназначены они для передачи управляющих сигналов внешним устройствам, например монитору, принтеру или модему. Символьные константы записываются в одной из трех форм:
Символ, заключенный в апострофы.
Десятичный код символа, предваряемый знаком #. Применяется для представления символов, отсутствующих на.
Буква, предваряемая знаком ^. Используется для представления управляющих символов. Код буквы должен быть на 64 больше, чем код представляемого таким образом символа.

Строковая константа — это последовательность любых ASCII-символов, расположенная на одной строке и заключенная в апострофы. Если требуется представить сам апостроф, он дублируется. Максимальная длина строковой константы — 126 символов.

Имена, ключевые слова и знаки операций

Имена в программах служат той же цели, что и имена людей, — чтобы обращаться к программным объектам и различать их, то есть идентифицировать. Поэтому имена также называют **идентификаторами**.

Имена дает программист, при этом следует соблюдать следующие правила:
имя должно начинаться с буквы (или знака подчеркивания);
имя должно содержать только буквы, знак подчеркивания и цифры;
прописные и строчные буквы не различаются;
длина имени практически не ограничена.

Например, правильными именами будут Vasia, A, A13, A_and_B и _____, а неправильными — 2late, Big gig и Sюр. Имена даются элементам программы, к которым требуется обращаться: переменным, константам, процедурам, функциям, меткам и т. д. **Ключевые** (**зарезервированные**) **слова** — это идентификаторы, имеющие специальное значение для компилятора. Их можно использовать только в том смысле, в котором они определены. Например, для оператора перехода определено ключевое слово goto, а для описания переменных — var. Имена, создаваемые программистом, не должны совпадать с ключевыми словами.

Знак операции — это один или более символов, определяющих действие над операндами. Внутри знака операции пробелы не допускаются. Например, операция сравнения 'меньше или равно' обозначается <=, а целочисленное деление записывается как div. Операции делятся на **унарные** (с одним операндом) и **бинарные** (с двумя).

Типы данных

Данные, с которыми работает программа, хранятся в оперативной памяти. Компилятору необходимо точно знать, сколько места они занимают, как именно закодированы и какие действия с ними можно выполнять. Все это задается при описании данных с помощью типа. Тип данных однозначно определяет:

внутреннее представление данных, а следовательно и множество их возможных значений;
допустимые действия над данными (операции и функции).

Например, целые и вещественные числа, даже если они занимают одинаковый объем памяти, имеют совершенно разные диапазоны возможных значений; целые числа можно умножать друг на друга, а, например, символы — нельзя. Каждое выражение в программе имеет определенный тип. Компилятор использует информацию о типе при проверке допустимости описанных в программе действий.

Стандартные	Определяемые программистом		
	Простые	Составные	
Логические			
Целые			
Вещественные	**Перечисляемый**	Массивы	Файлы
Символьный	**Интервальный**	Строки	Процедурные типы
Строковый	Адресные	Записи	Объекты
Адресный		Множества	
Файловые			

Стандартные типы не требуют предварительного определения. Для каждого типа существует ключевое слово, которое используется при описании переменных, констант и т. д. Если же тип данных определяет сам программист, он описывает его характеристики и дает ему имя, которое затем применяется точно так же, как имена стандартных типов. Описание собственного типа данных должно задавать всю информацию, необходимую для его использования: внутреннее представление и допустимые действия.

Логические типы

Внутреннее представление. Основной логический тип данных Паскаля называется boolean. Величины этого типа занимают в памяти 1 байт и могут принимать всего два значения: true (истина) или false (ложь). Внутреннее представление значения false — 0 (нуль), значения true — 1.

Операции. К величинам логического типа применяются логические операции and, or, xor и not (таблица 1.3). Для наглядности вместо значения false в таблице используется 0, а вместо true — 1.

a	b	a and b	a or b	a xor b	not a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Величины логического типа можно сравнивать между собой с помощью операций отношения, перечисленных в таблице. Результат этих операций имеет логический тип.

Операция	Знак операции	Операция	Знак операции
Больше	>	Меньше или равно	<=
Больше или равно	>=	Равно	=
Меньше	<	Не равно	<>

Целые типы

Внутреннее представление. Целые числа представляются в компьютере в двоичной системе счисления. В Паскале определены несколько целых типов данных, отличающиеся длиной и наличием знака: старший двоичный разряд либо воспринимается как знаковый, либо является обычным разрядом числа. Внутреннее представление определяет диапазоны допустимых значений величин (от нулей до единиц во всех двоичных разрядах).

Тип	Название	Размер	Знак	Диапазон значений
integer	Целое	2 байта	Есть	$-32\,768 .. 32\,767 (-2^{15} .. 2^{15}-1)$
shortint	Короткое целое	1 байт	Есть	$-128 .. 127 (-2^{7} .. 2^{7}-1)$
byte	Байт	1 байт	Нет	$0 .. 255 (0 .. 2^{8}-1)$
word	Слово	2 байта	Нет	$0 .. 65\,535 (0 .. 2^{16}-1)$
longint	Длинное целое	4 байта	Есть	$-2\,147\,483\,648 .. 2\,147\,483\,647 (-2^{31} .. 2^{31}-1)$

Операции.

С целыми величинами можно выполнять арифметические операции.

Результат их выполнения всегда целый (при делении дробная часть отбрасывается).

Операция	Знак операции	Операция	Знак операции
Сложение	+	Деление	div
Вычитание	–	Остаток от деления	mod
Умножение	*		