Литература

1. Абрамов С.А., Гнездилова Г.Г. Задачи по программированию. М.: Наука, 1988.

2. Быкадоров Ю.А., Кузнецов А.Т. Информатика и вычислительная техника 10–11. – Мн.: Народная асвета,– 1977.

3. Бородич Ю.С., Вальвачев А.Н., Кузьмич А.И. Паскаль для персональных компьютеров. Мн.: Вышэйшая школа, 1991.

4. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. М.: Мир, 1981.

5. Зима В.С., Абрамов С.А. Начала программирования на языке Паскаль.

6. Офицеров Д.В., Старых В.А. Программирование в интегрированной среде -Паскаль. Справочное пособие. Мн.: Беларусь, 1992.

7. Офицеров Д.В., Долгий А.Б., Старых В.А. Программирование на персональных компьютерах . Практикум. Мн.: Вышэйшая школа, 1993.

8. Пильщиков В.Н. Сборник упражнений по языку Паскаль. М.: Наука, 1989.

Литература к лабораторным занятиям

1. Вабишчэвіч С.В., Кузняцоў А.Ц. Асновы алгарытмізацыі. Частка 1. Алгарытмічная мова. Лабараторны практыкум.– Мн.:БДПУ імя М.Танка, 1999.

2. Вабішчэвіч С.В. Уводзіны ў праграмаванне на мове Паскаль: Лабараторны практыкум.– Мн.:БДПУ імя М.Танка, 2000.

3. Дадатковыя магчымасці мовы Turbo Pascal. Лабараторны практыкум //Аўт.-склад. Л.М. Краўчанка.– Мн.: БДПУ імя М.Танка, 2000.

1. Понятие о структурном программировании

С общей точки зрения процесс решения задачи на ЭВМ включает в себя следующие этапы:

1) постановка задачи;

2) построение математической модели;

3) разработка алгоритма;

4) составление программы;

5) реализация программы на ЭВМ;

6) анализ результатов.

Для организации процесса проектирования кодирования программ со сложной иерархической структурой служит структурное программирование, которое включает три главные составляющие:

1) проектирование сверху вниз;

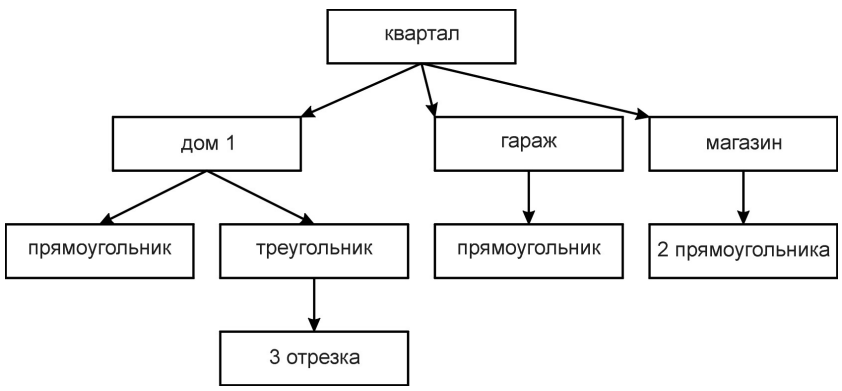
2) модульное программирование;

3) структурное моделирование.

Проектирование сверху вниз по-другому называют «методом последовательных уточнений», т. к. при этом сначала предусматривается определение задачи в общих чертах, а затем идет постепенное уточнение структуры путем выделения более мелких деталей. На очередном шаге каждая подзадача в свою очередь разбивается на ряд задач (модулей), пока они не станут настолько простыми, что программирование их не составит особого труда. Таким образом, при этом действует метод «от общего к частному».

Разбивание задачи на модули (модуляризация) – непременный компонент любого большого программного проекта. Общепризнанно, что с ростом размеров проекта качество выполнения модуляризации играет очень важную роль в конечном успехе проекта.

Пример. Спроектировать квартал.



Однако прежде чем приступать к программированию, следует проанализировать проект с целью выявления логических ошибок, которые могут возникнуть при проектировании. Такие ошибки легче устранить на стадии программирования, чем исправлять программу.

Заключительный этап структурного программирования — написание программы — в основе которого лежит метод структурного программирования. Этот метод позволяет получать программы более удобные для тестирования, модификации и использования. Он основан на принципе, разработанном итальянскими учеными (Бом, Джакони), согласно которому любая программа произвольного размера и сложности может быть написана с использованием ограниченного набора базисных структур.

В теории структурного моделирования доказано, что любая правильная программа эквивалентна программе, которая содержит в качестве логических структур только:

последовательность двух и более операторов;

условие перехода к одному из двух операторов;

повторение операторов, пока условие истинно (цикл «пока»).

Правила структурного программирования:

1. В каждой структуре используется один вход и один выход;
2. Алгоритм может быть построен так, что все его структуры образовывают линейную цепочку;
3. Следует максимально использовать модульный принцип.

0.1 Особенности написания программ

Может создаться впечатление, что поскольку программа будет обрабатываться машиной, то главное, чтобы она была правильной. В этом случае машина разберется с программой, какой бы запутанной она ни была. Это правильно, но главное, что в первую очередь программы читаются людьми (разработчиками, пользователями и т. д.)

На первое место выступает не просто правильность программы, но и ее удобочитаемость. Стиль программирования, — это выражение опыта общения идей занимающихся разработкой и использованием программ. Индивидуальный стиль хорош у художника, но программист должен придерживаться особого стиля, чтобы его программы были доступны другим. Для этого используются комментарии, правильный выбор имен переменных, размещения программы.

Предпочтительнее размечать каждый оператор в отдельной строке. Для выявления структуры программы рекомендуется делать отступы разных уровней от левого края программы.

В зависимости от целей программы, условий эксплуатации, ресурсов вычислительной техники критерии ее эффективности могут быть различны:

размер памяти;

скорость выполнения;

удобочитаемость и простота реализации.

Часто эти критерии бывают противоречивыми и которому отдать предпочтение зависит от конкретной ситуации.

0.2 Алгоритм и его свойства

Понятие алгоритма относится к числу фундаментальных математических понятий. Точное понятие алгоритма в математике определяется по-другому, вместе с тем для ознакомления с методами алгоритмизации в связи с записью программы для компьютеров нет необходимости обращаться к строгому определению этого понятия. ***Под алгоритмом понимают точное и полное предписание о последовательности выполнения конечного числа команд (действий, инструкций) исполнителю (человеку или автомату) для решения любой задачи из данного класса однотипных задач***. Слово «алгоритм» возникло от algorithmi – латинской формы написания имени великого средневекового ученого, деятельность которого протекала в Багдаде, аль-Хорезми (IX в). Он описал десятичную систему счисления и впервые сформулировал правила выполнения в ней четырех основных арифметических действий. В латинских переводах эти правила начинались со слов «Алхоризми сказал». Постепенно люди забыли, что аль-Хорезми – это автор правил, и стали сами эти правила называть алгоритмами. Сами правила арифметических действий и сегодня служат простейшими примерами математических алгоритмов. Алгоритмы составляются как для задач вычислительного характера, так и логических. Алгоритмы могут составляться для обработки слов, букв, разных предметов, таблиц и т.д. Часто для решения одной и той же задачи можно составить несколько алгоритмов. Существуют различные способы оценки качеств алгоритмов. Можно сказать, что лучше тот алгоритм, который быстрее приводит к результату.

Рассмотрим основные свойств алгоритмов:

- дискретность: описываемый алгоритмом процесс должен быть разбит на последовательность отдельных действий. Возникающее пр этом описание представляет собой последовательность четко

разделенных друг от друга указаний, образующих прерывную (или, как говорят, дискретную) структуру алгоритмического процесса – только выполнив требования одного указания, можно перейти к следующему;

- детерминированность или определенность: алгоритм должен настолько точно и понятно определять последовательность действий, чтобы не оставалось никакой неясности для исполнителя, и после выполнения алгоритма при заданных исходных данных всегда должен быть определенный, однозначный результат;

- массовость: алгоритм решает не одну лишь индивидуальную задачу, а некоторую серию задач данного типа, имеющих разные исходные данные;

- конечность: завершение работы алгоритма в целом происходит за конечное число шагов;

- результативность: алгоритм должен содержать четкое указание об окончании выполнения последовательности действий и о том, что следует считать результатом его выполнения. При решении любой индивидуальной задачи через конечное число шагов алгоритм должен останавливаться и выдавать результат.

Алгоритмы записывают в виде формул, схем, словесных правил, наставлений, рецептов, изображают графически и т.д. Программа для компьютера – это особая форма записи алгоритмов. Запись алгоритма включает отдельные шаги – этапы выполнения алгоритма, состоящие из выполнения одной простой команды или проверке условия.

Алгоритмы бывают разных видов: линейные (алгоритмы следования), ветвления, циклические. Рецепт приготовления блюда, инструкция являются примерами линейных алгоритмов; во время движения автомобиля может встретиться развилка, и в зависимости от направления дальнейшего движения водитель выбирает нужную дорогу (разветвляющийся алгоритм), в стакане сахар размешивается до тех пор, пока он полностью не растворится (многократное выполнение определенного набора действий) – циклический алгоритм.

Для каждого конкретного алгоритма, кроме его свойств, можно выделить *7* ***характеризующих алгоритм параметров:***

1) совокупность возможных *исходных данных;*
2) совокупность возможных *результатов;*
3) совокупность возможных *промежуточных результатов;*
4) правило *начала;*
5) правила непосредственной *обработки* данных;
6) правило *окончания;*
7) правило *извлечения (сообщения) результата.*

Алгоритм решения квадратного уравнения с этой точки зрения можно охарактеризовать следующим образом:

1) исходными данными являются коэффициенты *a, Ь, c.* Их возможные значения — множество действительных чисел; кроме того, коэффициент при x^2 не должен равняться нулю;

2) совокупность возможных результатов: один корень, два корня или сообщение, что решения нет;

3) промежуточным результатом является значение дискриминанта;

4) правило начала заключается во вводе значений коэффициентов уравнения и проверке, что $a \neq 0$;

5) правила непосредственной обработки данных — соответствующие формулы математики;

6) правило окончания — получение одного из возможных результатов;

7)правило извлечения (сообщения) результата: чаще всего — вывод полученного результата на экран (возможно вывести его в файл или использовать для решения другой задач.

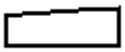
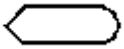
0.3 Блок-схема

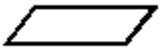
Блок-схемой называют наглядное графическое изображение алгоритма, когда отдельные его действия (этапы) изображаются при помощи различных геометрических фигур (блоков), а связи между этапами указываются при помощи стрелок, соединяющих эти фигуры.

Рассмотрим перечень условных обозначений, наиболее часто используемые для представления алгоритмов в графической форме. Для решения задач их вполне достаточно.

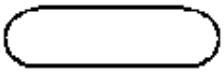
Необходимо также отметить, что представленные в перечне геометрические фигуры, в соответствии с логикой решения задачи, в блок-схеме алгоритма соединяются линиями связи. На

концах последних должны обязательно присутствовать стрелки, если их направление снизу вверх или справа налево, то есть отличное от стандартного направления (сверху вниз и слева направо), когда стрелки могут и отсутствовать.

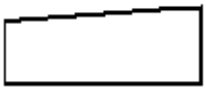
Особенно хотелось бы остановиться на вводе-выводе данных. Как было отмечено выше ввод данных с клавиатуры обозначается  вывод на экран дисплея обозначается 

Символ  отображает данные, носитель данных не определен.

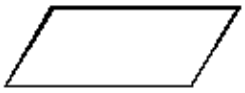
Этот символ можно использовать для ввода-вывода данных в общем виде. Ввод исходных данных и вывод результатов можно изображать в виде параллелограмма. Внутри него пишется слово "ввод" или "вывод" ("печать") и перечисляются переменные, подлежащие вводу или выводу.



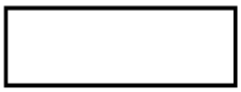
Символ отображает вход из внешней среды в алгоритм или программу и выход во внешнюю среду в соответствии с пояснительной надписью — "начало", "конец", "вход", "выход".



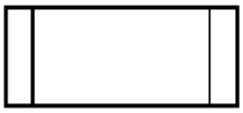
Указание на ручной ввод данных (например, с клавиатуры)



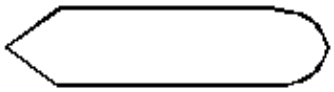
Символ отображает данные, носитель данных не определен



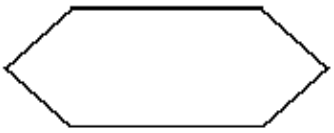
Символ обозначает операцию или группу операций над данными, которая приводит к изменению их значения.



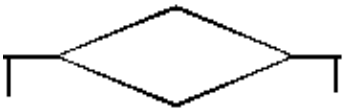
Символ, которым заменяют одну или несколько команд, определенных в другом месте (во вспомогательном алгоритме, в подпрограмме)



Указание на вывод данных на экран дисплея.



При организации цикла с параметром Внутри него указать заголовок цикла или Модификации индекса, но можно Обойтись прямоугольниками и ромбом.



Проверка условия

Разрыв линий потока на странице, на разных страницах

Пример блок-схемы циклического алгоритма высиления значения факториала числа p или p!.

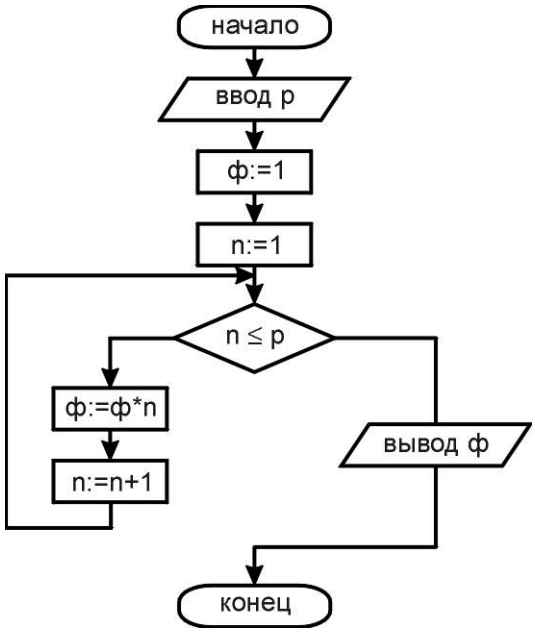


Рис.1

1. Общие сведения о языке программирования ПАСКАЛЬ

1.1 Название языка Паскаль

Язык Паскаль получил свое название в честь великого французского ученого, физика-математика Блеза Паскаля, который в 1642 г. изобрел счетную машину для арифметических операций – Паскалево колесо. История создания языка Паскаль начинается с 1965 года, когда международная федерация по обработке информации IFIP предложила нескольким специалистам в области информатики принять участие в разработке нового языка программирования – приемника АЛГОЛА-60. Среди них был швейцарский ученый, работавший доцентом информатики Стенфордского университета Николаус Вирт. В конце 1968 года Вирт и сотоварищи из швейцарского федерального института технологии в Цюрихе разработали первую версию Паскаля, а спустя 2 года –1-й вариант компилятора. В 1971 году Вирт выпустил описание своего языка. Создавая Паскаль, Вирт преследовал две цели: во-первых, разработать язык, пригодный для обучения программированию как систематической дисциплине; во-вторых, реализация языка должна быть эффективной и надежной на существующих вычислительных машинах. Одним из достоинств языка Паскаль является то, что он воплотил в себе идею

структурного программирования, суть которой заключается в том, что с помощью нескольких конструкций можно выразить в принципе любые алгоритмы: линейные, ветвление, циклические конструкции.

При создании и совершенствовании языка Вирт ввел много новшеств, в частности, изобрел синтаксические диаграммы, с помощью которых удобно представлять конструкции языка, первый ввел в алфавит квадратные скобки, высказал идею решения проблемы переносимости программ в виде Пи-системы, которая заключается в том, что написанная на Паскале программа транслируется в Пи-код, в машинный язык некоторой идеальной машины, а затем интерпретируется на реальных машинных языках. Он выявил и ряд недостатков: отсутствие операции возведения в степень, понятие отдельно транслируемого модуля, затрудняющее тем самым создание больших программ и др.

Кроме авторской версии, стали появляться различные его расширения и диалекты. Например: УКСД Паскаль, Паскаль-80, ЭППЛ-Паскаль, -Паскаль, Квик-Паскаль. Особую популярность на ПЭВМ в настоящее время получило семейство Паскаль-систем, названное -Паскаль и разработанное фирмой BORLAND. Данное семейство, работающее с различными операционными системами CP/M, MSXDOS, имеет высокую производительность, т. к. используется ускоренная однопроходная процедура компиляции. Последняя версия -Паскаль 7.0. Паскаль стала прародителем более поздних языков программирования.

Через 10 лет после Паскаля Вирт создал язык МОДУЛА-2, в котором особое внимание уделяется построению программ как набора независимых модулей. На Паскаль опирались все 4 языка, принятые в 70-х годов Министерством обороны США для разработки своего универсального высокоуровневого языка, который в дальнейшем получил название АДА.

1.2 Алфавит языка Паскаль

Алфавит языка Паскаль состоит из букв русского и латинского алфавита, арабских цифр, знаков операций (+,-,*,/,=,<,>,:=), ограничителей (. , : , ; , ',[],(),). Действительные числа изображаются в естественной и полулогарифмической форме (например, 2*E*+5=2*10^5 это 200000). Допустимый диапазон изменения целых и вещественных чисел зависит от конкретной реализации языка.

Особую роль при записи программ на языке программирования играют величины (данные). Рассмотрим основные характеристики величины: имя, тип, вид и значение.

Имя величины – это ее обозначение в алгоритме. Для ЭВМ имя означает также место в памяти, где хранится значение величины. Именем может быть любая последовательность букв, цифр и знаков подчеркивания, начинающихся с буквы.

Тип величины (данного) задает множество допустимых значений величины и множество применимых к ней операций.

Вид величины характеризует ее использование в алгоритме, роль содержащейся в величине информации. В алгоритмическом языке имеются различные виды величин: аргументы (исходные данные), результаты, промежуточные величины.

Имя, тип, вид величины указываются в ее описании. Эти характеристики не могут меняться в ходе работы алгоритмов

1.3 Типы данных

Паскаль характеризуется разветвленной структурой типов данных (рис.3.3.1)

Рис.3.3.1 Структура типов данных

Простые типы данных. В языке Паскаль типы данных разделяются на простые и структурированные, строки, указатели и др..

Простые типы являются базовыми. На их основе строятся более сложные структурные типы данных.

К простым типам относятся порядковые и вещественные типы. Порядковые типы отличаются тем, что каждый из них имеет конечное число возможных значений. Эти значения можно определенным образом упорядочить (отсюда -название типов) и, следовательно, с каждым из них можно сопоставить некоторое целое число - порядковый номер значения.

Т. к. данные простого типа (константы, переменные, функции, выражения) имеют одно значение, то простые типы также называют скалярными. Скалярный тип определяет упорядоченное множество значений переменных, констант, функций и выражений,принадлежащих к данному типу, форму представления в машине значений этих величин и операций, которые могут выполняться над ними. Скалярный тип может быть предопределенным (стандартным) или задаваться пользователем с помощью перечисления его возможных значений в разделе описания типов. Поэтому в последнем случае тип называют перечисляемым.

Стандартные простые типы данных. К ним относятся:

1) вещественный (*real*); 2) целочисленный (*integer*); 3) логический (*boolean*); 4) символьный (*char*).

Примеры	Обозначения	Границы	Требуется памяти (байт)
целый	*byte*	0..255	1
	word	0..65535	2
	integer	-32768..32767	2
	shortint	-128..127	1
	longint	-2147483648..2147483647	4
вещественный	*real*	2.9*E*-39..1.7*E*38	6
символьный	*char*	кодовая таблица ПЭВМ	1
логический	*Boolean*	true, false	1

Диапазон допустимых ***вещественных*** значений (*real)* от *E*-38 до *E*+38 с мантиссой, занимающей 11 двоичных разрядов. Числа этого типа занимают 6 байт памяти. Чтобы некоторая переменная в программе относилась к вещественному типу, ее имя а разделе описания переменных должно быть описано как *real*:

var c, sk:real;.

Тип выражения опередеяется типом входящих в него операндов и видом операций, проводимых над ними. Результат операций +, -, * будет действительным числом (вещественным), если хотя бы один операнд вещественного типа. Результат операции деления – **всегда действительное число**, даже если оба операнда целого типа. Существует встроенная константа вещественного типа, которая составляет значение числа π, которая обозначается **pi**.

Значениями ***целочисленного*** или целого типа являются элементы подмножества целых чисел. Диапазон допустимых целых чисел в десятичной записи от -32768 до 32767. Определена стандартная константа ***maxint, равная*** *32767***.** В разделе описания переменных указываются имена:

var c,sp,q:integer;..

Арифметическое выражение будет давать целый результат, если все входящие в него операнды относятся к целому типу и к ним применены операции -,+, *, а также *div* или *mod*. Как правило, данные целого типа редко используются в вычислениях. В программировании они применяются для обозначения индексов в массивах, организации счетчиков. Подмножество целых чисел от 0 до 255 обозначают специальным словом *byte*. Кроме этого используется специальное обозначение **отрезочного** типа – две точки. Например, обозначение 5..16 обозначает что переменные, описанные таким образом – это целые числа от 5 до 16.

Для вещественных и целых типов выделяются другие подмножества, которые обозначаются специальными служебными словами. С ними можно ознакомиться в справочниках.

Логические (булевы) значения. Обозначаются стандартными именами *true* и *false*. Установлено, что *false* меньше *true*. Переменная этого типа занимает 1 байт памяти. Логическая переменная – это переменная, принимающая одно из значений, и в разделе описания переменных она должна быть описана так:

var c,sp, q1:boolean;..

Значение *true* и *false* получают в результате выполнения операций сравнения <,>,<=,>=,<>,=. Операнды этих операций могут быть вещественного, целого типа. Следует помнить, что к операндам вещественного типа очень осторожно следует применять операцию =, т. к. условие может не выполняться за счет неточного представления действительных чисел в памяти ЭВМ и неизбежных ошибок округления при вычислении выражений вещественного типа. В некоторых случаях необходимо применять вместо записи *A1=A2* запись *ABS(A1-A2)<E*, где *E* – некоторая величина, характеризующая допустимую погрешность округления.

Помимо операций отношения, существуют ***3 логические операции,*** применяемые только к операндам булевого типа. Это *not*, *and*, *or*.

Операция *not* является одноместной. Ее результат – *true*, если значение операнда – *false*. Операции *and* и *or* двухместные.

not		*and*			*or*		
арг	рез	арг	арг	рез	арг	арг	рез
true	*false*	*true*	*true*	*true*	*true*	*true*	*true*
false	*true*	*true*	*false*	*false*	*true*	*false*	*true*
		false	*true*	*false*	*false*	*true*	*true*
		false	*false*	*false*	*false*	*false*	*false*

Логические операции и операции отношения часто встречаются в одном выражении, причем отношения, стоящие слева и справа от логической операции, имеют более высокий приоритет и выполняются в следующем порядке: в первую очередь *not*, затем *and*, *or* в последнюю очередь. С помощью скобок порядок выполнения может быть изменен.

Например:

1 6 4 2 5 3

(-3>=5) OR NOT (7<9) AND (0<=3). Имеет значение *false*.

Символьный тип задает конечное и упорядоченное множество символов. Значение символьной константы заключается в апострофы. Например: *'к','1','+'*; *var c,k:char*. Все переменные этого типа упорядочены по кодам.

1.4 Перечисляемые типы данных

В специальном разделе – разделе описания типов – программист может сам определять некоторые типы данных, перечислить те значения, которые будут принимать переменные этого типа. Переменные и константы скалярного типа, задаваемые перечислением, не могут быть параметрами операторов ввода и вывода в языке Паскаль. Они используются для управления логикой программы в операторах цикла, условных операторах. Они в какой-то мере играют роль комментариев.

Каждое значение типа задается указанием, обозначающим это значение идентификатора. Например:

type day = (sat, sun, mon, tue, wed, thu, fri);.

Скалярный тип *day* включает в себя перечисленные значения *day*, *const*. Переменные должны быть описаны в разделе описания переменных:

var paday , st: day;.

Переменные *paday* и *st* объявлены как переменные типа *day*, тогда можно записать следующие операции присваивания:

paday := mon; st := tue;.

Переменные типа day при выполнении программы могут принимать только одно из 7 указанных значений. Объект, указанный в списке перечисления может присутствовать не более чем в одном описании. Имена объектов, указанных в описании перечисляемого типа являются константами этого типа. Для перечисляемого типа данных существенен порядок указанных объектов. К данным перечисляемого типа применимы операции отношения. Порядковый номер объекта вычисляется с помощью функции *ord*. Примеры:

sun<paday; ord(sat)=0; ord(tue)=3;.

1.5 Арифметические операции. Функции. Выражения

Основными элементами, из которых конструируется исполняемая часть программы, являются константы, переменные и обращения к функциям. Каждый из этих элементов характеризуется своим значением и принадлежит к какому-либо типу данных. С помощью знаков операций и скобок из них можно составлять выражения, которые фактически представляют собой правила получения новых значений.

Частным случаем выражения может быть просто одиночный элемент, т.е. константа, переменная или обращение к функции. Значение такого выражения имеет, естественно, тот же тип, что и сам элемент. В более общем случае выражение состоит из нескольких элементов (операндов) и знаков операций, а тип его значения определяется типом операндов и видом примененных к ним операций

Над числовыми величинами могут выполняться операции сложения, вычитания, умножения, деления, целочисленного деления (*div*), вычисление остатка от целочисленного деления *mod*. Если операции +,-, * выполняются над целыми числами, результат получается целочисленный. При выполнении операции *div* и *mod* над вещественными числами исходные данные сначала приводятся к целому типу. Результат выполнения этих операций целочисленный. Приоритет операций в порядке убывания:

1– *, /
2– *div*
3– *mod*
4– +, -

Например: *3 div 4+5=5*; *7 mod 2+3=4.*

Арифметические операции

Обозначение	Действие	Тип аргументов	Тип результата

а+в	сложение *а* и *в*	только целый	целый
		только вещественный	вещественный
		один целый, другой вещественный	вещественный
а–в	вычитание *в* из *а*	только целый	целый
		только вещественный	вещественный
		один целый, другой вещественный	вещественный
*а*в*	умножение *а* на *в*	только целый	целый
		только вещественный	вещественный
		один целый, другой вещественный	вещественный
а/в	деление *а* на *в*	только целый	вещественный
		только вещественный	вещественный
		один целый, другой вещественный	вещественный
а mod в	остаток от деления *а* на *в*	только целый	целый
а div в	целочисленное деление *а* на *в*	только целый	целый

1.6 Основные математические функции

Обозначение	Действие	Тип аргумента x	Тип результата y
abs(x)	определяет модуль величины *x*	целый	целый
		вещественный	вещественный
sin(x)	определяет синус *x*	целый	только вещественный
		вещественный	
cos(x)	определяет косинус *x*	целый	только вещественный
		вещественный	
arctan(x)	определяет арктангенс *x*	целый	только вещественный
		вещественный	
exp(x)	*e* возводит в степень *x*	целый	только вещественный
		вещественный	
ln(x)	определяет натуральный логарифм *x (x > 0)*	целый	только вещественный
		вещественный	
sqr(x)	возводит *x* в квадрат	целый	целый
		вещественный	вещественный
sqrt(x)	определяет корень из *x (x 0)*	целый	только вещественный
		вещественный	
frac(x)	определяет дробную часть *x*	целый	только вещественный
		вещественный	
int(x)	определяет целую часть *x*	целый	только вещественный
		вещественный	
trunc(x)	определяет целую часть *x*	целый	только целый
		вещественный	
round(x)	находит целое, ближайшее к *x*	целый	только целый
		вещественный	
odd(x)	проверяет на нечетность *x*	целый	логический
random(x)	определяет случайное целое значение	целый	целый в пределах *0 y < x*
random	определяет случайное	без аргумента	вещественный в преде-

	вещественное значение		лах *0 y < 1*
pi	определяет число π	без аргумента	вещественный

Результат выполнения функции *int(X)*, *trunc(X)*, *frac(X)* для $X < 0$ отличается от привычного значения целой и дробной части. При выполнении *int(x)* и *trunc(x)* цифры после запятой отбрасываются. Например: *int(-3.72)=-3.0 trunc(-3.72)=-3.*

При вычислении функции *frac(x)* отбрасываются все цифры до запятой. Например: *frac(-19.42)=-0.42*.

Результатом функции *round(x)* является ближайшее целое к числу x. Например: *round(1.25)=1*, *round(-12.84)=-13*.

Отметим, что в Паскале ***нет операции возведения в произвольную степень***. Ее можно записать следующим образом: $x^y = x\,x^y = \left(e^{\ln x}\right)^y = e^{y \ln x}$ или x^y =***exp(y*ln(x))***, при этом число $x > 0$.

Если у нас небольшая степень, то лучше выполнить небходимое количество умножений или использавать функцию *sqr(x)*: x^5 =*sqr(sqr(x))*x*.

1.7 Структура программы

Программа на языке Паскаль состоит из заголовка, блока и заканчивается точкой. Блок в свою очередь содержит раздел описаний и раздел операторов.

Раздел операторов представляет собой последовательность операторов, разделенных точкой с запятой, и ограничивается служебными словами *begin*, *end*.

program имя *[(*имя файла*)];* — заголовок

uses *crt, graph;*

label метка*,* метка*;*

const имя*_const* = константа*;* имя *const* = константа*;*

type имя типа = тип*;*

var имя переменной,имя переменной:тип*;* — раздел описаний

procedure <заголовок>*;*

<блок>*;*

function <заголовок>*;*

<блок>*;*

BEGIN

<*операторы* >

END. — тело программы

Раздел подключения встроенных библиотек uses используется при работе с системой Паскаль, Борланд Паскаль.На ЭВМ Корвет в системе Экспресс Паскаль этот раздел не используется.

Раздел *uses* описывает список имен, подключаемых стандартных и пользовательских модулей, например:

crt – содержит средства управления дисплеем и клавиатурой ЭВМ;

printer – открывает доступ к печатающему устройству;

graph – позволяет использовать графику в Паскале;

dos – позволяет использовать команды работы с операционной системой.

Раздел констант, раздел переменных, раздел процедур и функций составляют раздел описаний. Раздел описаний и раздел операторов составляют блок программы.

Порядок следования разделов строго определен, однако в некоторых описаниях языка порядок следований *const*, *label*, *type*, *var* произволен. В заголовке программы после служебного слова program указывается имя программы, не имеющее смысла внутри программы. После имени программы в круглых скобках следует список файлов (наборов данных, размещающихся на внешних устройствах), с которыми взаимодействует программа. Обычно используют имена стандартного входного *input* и выходного *output*: *program gons (input, output);*.

Во многих версиях языка имена стандартных файлов могут отсутствовать. Они принимаются по умолчанию: *program gons;*.

Кроме того в последних версиях языка слово *program* может отсутствовать.

После заголовка программы следует раздел описания меток *label*. Любой оператор в программе можно выделить, поставив перед ним метку (целое число без знака, содержащее не более 4-х цифр). Метка от оператора отделяется двоеточием. Например:

25:read(x,y,z);.

Появление меток в программе дает возможность ссылаться на эти метки в специальных операторах управления (*goto*) и изменять естественный ход выполнения программы. Все метки должны быть описаны в разделе *label*.

```
program cons;
label
  1,25,100;
```

Этот раздел может отсутствовать, если в программе меток нет. За разделом меток следует раздел описания констант *const*.

Константы в программе должны быть представлены именем, тогда в разделе констант этим именам должны быть присвоены некоторые значения.

Например: *const д=3.4; m=3; n=4; s='заголовок таблицы';*.

Это позволяет сгруппировать в начале программы постоянные величины, зависящие от машины или характерные для данной задачи. Здесь их легче изменять, не изменяя саму программу.

Раздел описания типов (*type*) служит для определения программистом собственных простых и структурных типов данных. Каждая величина в программе должна быть сопоставлена с одним и только одним типом.

Тип переменной должен быть описан в специальном разделе **var**. Раздел описаний процедур и функций присутствует в программе, если программист, помимо стандартных процедур и функций, использует свои, являющиеся самостоятельными программным единицами, к которым осуществляется обращение из основной программы. В состав программы могут быть включены комментарии, т. е. тексты, поясняющие программу, но не влияющие на ход выполнения. ***Комментарии*** заключаются в специальные скобки. Например:

```
goto 20; {выход из цикла} или
(*выход из цикла*)  или
 /*выход из цикла*/
```

Операторы в исполнительной части программы определяют, какие действия должны быть выполнены над данными. С каждым элементом данных обычно связывают имя и значение. Имя используется для обозначения элемента данных. ***В качестве имени*** в Паскале используется идентификатор – последовательность букв или цифр, начинающаяся с буквы. Хотя длина идентификатора не оговаривается, реально идентификатор не может переноситься со строки на строку. Обычно используют не более 80 символов. Кроме того, в большинстве реализаций Паскаля идентификаторы различаются по первым 6–8 символам. Все идентификаторы должны быть описаны в разделе описаний.

Для того чтобы постоянно не нажимать Alt-F5 для просмотра результата можно в конце программ перед END записать оператор readln;. Этот оператор без параметров означает ожидание нажания клавиши ввода. После нажатия этой клавиши система возвращает пользователя из окна вывода в текстовый редактор.

2. Текстовый режим работы в системе Паскаль

Для работы с экраном в текстовом режиме предназначен модуль ***crt*****,** в котором используются готовые операторы предназначен целый ряд процедур, функций и переменных, который подключается с помощью команды uses crt, записанной сразу после заголовка программы.

Стандартные процедуры или функции модуля crt:

gotoxy (x,y) – перемещает курсор в строку x позиции y (x,y – целые);

clrscr – очистка экрана;

delay (m_s) – задержка программы на m_s миллисекунд (m_s – *integer*);

keypressed – функция, результат *true* – клавиша нажата, *false* – клавиша не нажата.

textcolor(c) – выбор текущего цвета букв (с – номер цвета до 7);

textbackground (cf) – выбор цвета фона.

insline – вставляет пустую строку на месте расположения курсора;

delline – удаляет строку, в которой находится курсор и перемещает все расположенные;

clreol – очищает текущую строку до конца с позиции курсора

window (x1,y1,x2,y2) – отображает на экране окно в текстовом режиме, определяемое как текущее. Координаты окна *x1*,*y1*,*x2*,*y2* всегда отсчитываются от левого верхнего угла экрана и должны удовлетворять следующим условиям:

1<=*x1* < *x2* <=*X*max;

1<=*y1* < *y2* <=*Y*max.

Если эти условия не выполняются, то окно создано не будет. Для большинства видеоадаптеров Xmax=80, Ymax=25. Для адаптеров EGA и VGA специальными командами он может принимать еще 43 или 50.

После выполнения процедуры window окно становится текущим. Это значит, что все операции с экраном относятся к той его части, которая определена координатами *x1*,*y1*,*x2*,*y2*. При этом перемещение курсора происходит только в пределах текущего окна, и позиция с координатами (1,1) является левым верхним углом окна. Часть экрана вне окна становится недоступной для других процедур и функций. При завершении работы программы, использующей окна, происходит автоматическое восстановление параметров текстового режима.

Пример. Окно с тенью и вывод по диагонали 3 слов «информатика».

```
hjogram zz1;
uses crt;
BEGIN
  textbackground(2);
  clrscr;
  window(10,4,60,18);
  textbackground(0);
  clrscr;
  window(11,3,61,17);
  textbackground(4);
  clrscr;
        gotoxy(1,1);      writeln('информатика');
        gotoxy(2,2);      writeln('информатика');
        gotoxy(3,3);      writeln('информатика');

  readln;
END.
```