Министерство образования и науки Украины
Национальный технический университет
"Харьковский политехнический институт"

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к практическим занятиям
по курсу "Информатика"
(модуль "Алгоритмический язык ТурбоПаскаль")
для студентов специальности
8.100501 "Подвижной состав и специальная техника
железнодорожного транспорта"

Утверждено
редакционно–издательским
советом университета,
протокол № 1 от 30.03.07.

Харьков НТУ "ХПИ" 2007

Методические указания к практическим занятиям по курсу "Информатика" (модуль "Алгоритмический язык ТурбоПаскаль") 8.100501 "Подвижной состав и специальная техника железнодорожного транспорта". / Сост. Ю. В. Веретельник, В. И. Сериков, Е. В. Пелешко. – Харьков: НТУ «ХПИ», 2007. – 56 с. – Рус. яз.

Составители: Ю. В. Веретельник
В. И. Сериков
Е. В. Пелешко

Рецензент Н. А. Ткачук

Кафедра теории и систем автоматизированного проектирования механизмов и машин

СОДЕРЖАНИЕ

ВСТУПЛЕНИЕ

Повсеместное использование персональных компьютеров в системах автоматизированного проектирования гусеничных и колесных машин требует от современного инженера овладения навыками использования вычислительной техники. Одной из дисциплин, непосредственно связанной с применением компьютеров, является "Информатика". Основным разделом курса является изучение методики разработки алгоритмов и языка программирования ТурбоПаскаль для использования в создании вычислительных програм при проектировании транспортных средств, в курсовом и дипломном проектировании.

Паскаль – один из наиболее распространенных языков программирования 80–90–х годов, поддерживающий самые современные методологии проектирования программ (нисходящее, модульное проектирование, структурное программирование). Новую жизнь в язык вдохнула фирма Борланд, разработавшая на его базе семейство Паскаль–систем, называемых ТурбоПаскалем. Компиляторы фирмы поддерживаются многими распространенными операционными системами персональных ЭВМ, такими как CP/M–80, MSDOS, MSX DOS, средами Windows и т.п.

Интегрированная среда, обеспечивающая многооконную разработку программной системы, обширный набор встроенных в нее средств компиляции и отладки, доступный для работы через легко осваиваемое меню – все это обеспечивает высокую производительность труда программиста.

Язык Паскаль разработан с учетом принципов структурного программирования, которое на современном этапе признано действенным методом рационализации труда программиста. Для структурированных программ характерны легкость отладки и корректировки, низкая частота ошибок. Кроме этого, такие программы легко сопровождать и модифицировать без участия разработчиков.

Паскаль обладает полным набором структурных типов данных, таких, как простые переменные, массивы, файлы, множества, записи, записи с вариантами, ссылочные переменные. Введение структурных типов данных способствует созданию эффективных алгоритмов.

Особо следует отметить надежность Паскаль–программ, которая достигается иногда за счет избыточности, например, обязательного описания переменных и соответствующих типов. Надежность достигается также за счет простоты и естественности конструкций языка, соответствующих логическому мышлению разработчика программ.

Целью методических указаний является улучшение подготовки студентов специальности "Подвижной состав и специальная техника железнодорожного транспорта" в ходе изучения ими языка ТурбоПаскаль для разработки вычислительных программ при проектировании, а также исследовании систем автоматического регулирования железнодорожного транспорта.

Методические указания состоят из восьми разделов, в которых изложен необходимый теоретический материал и приведены многочисленные примеры и за-

дания, которые дают возможность наглядно продемонстрировать практическое применение рассматриваемой темы. Последовательность тем подчинена задачам практики.

ТЕХНИКА БЕЗОПАСНОСТИ ПРИ РАБОТЕ С ЭВМ

1. К работе на ЭВМ допускаются студенты, изучившие правила техники безопасности, имеющие навыки работы с системой и текст программы для реализации.

2. Подготовка ЭВМ к работе, техническое обслуживание и ремонт производятся персоналом кафедры, имеющим соответствующую подготовку. Поэтому о любых неполадках в работе ЭВМ необходимо сообщить преподавателю.

3. Студентам при работе на ЭВМ разрешается пользоваться только клавиатурой, манипулятором «мышь» и дисплеем. Использование других устройств без разрешения преподавателя запрещается.

4. ***Категорически запрещается*** пользоваться дискетами, не проверенными преподавателем.

1. ОСНОВНЫЕ КОМАНДЫ СРЕДЫ ТУРБОПАСКАЛЯ

1.1 Секция File

Секция содержит подменю, изображенное на рисунке 1.1.

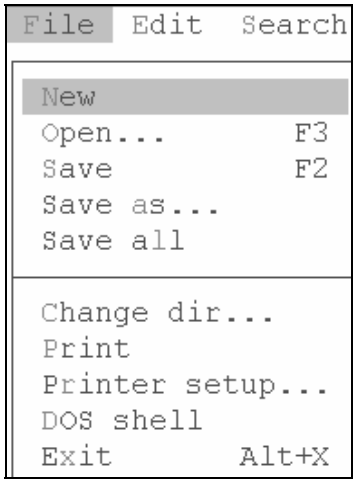

Рисунок 1.1 – Подменю секции File

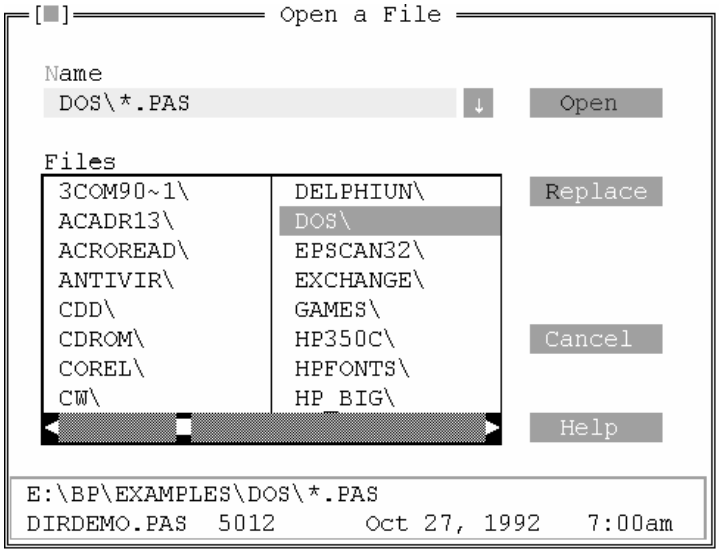

Рисунок 1.2 – Диалоговое окно Open a File

Эта секция меню предоставляет возможности для открытия и загрузки соответствующих файлов, создания новых файлов, выхода в оболочку DOS, сохранения файлов на диске и выхода из ТурбоПаскаля. С командами New и Save as Вы уже знакомы, рассмотрим остальные команды.

Команда Open открывает диалоговое окно Open a File (рисунок 1.2), в котором можно выбрать файл для открытия в окне Edit.

Диалоговое окно содержит поле ввода, список файлов, информационную

панель, стандартные «кнопки» CANCEL и HELP, две другие кнопки (**OPEN**, **REPLACE**), список предыстории, приписанный к входному полю Name.

Входное поле **Name** – это место для размещения (набора с помощью клавиатуры) имени файла, который Вы хотите загрузить в окно редактирования. Здесь же можно поместить маску для использования в качестве фильтра содержимого поля **Files**.

Поле **Files** содержит имена файлов в текущем каталоге, в соответствии с маской, установленной в поле Name. Например, если в поле **Name** записано *.PAS, то в поле Files появятся имена всех файлов каталога, содержащие расширение .PAS.

Список предыстории добавляет к входному полю все имена, которые появлялись в нем во время последних вызовов диалогового окна **Open a File**. В список предыстории можно войти в том случае, если справа от входного поля **Name** Вы видите стрелку «вниз». Для входа в список следует нажать курсор «вниз». Этот список используется для повторного вхождения в текст, в который Вы уже входили.

Выбор нужного элемента осуществляется курсором, при этом подсвечивается выбранная позиция. Затем следует нажать клавишу **ENTER**. Выбранное имя файла попадает во входное поле **Name**. Вывод текста файла в редакционное окно осуществляется нажатием клавиши **ENTER**. Если выбор не сделан, для выхода из списка предыстории нажмите клавишу **ESC**.

Информационная панель отображает путевое имя выбранного файла, его имя, дату, время создания и размер.

Кнопка **OPEN** открывает новое окно редактирования и помещает выбранный текст в это окно.

Кнопка **REPLACE** заменяет файл в активном окне редактирования выбранным файлом, не открывая новое окно.

Кнопка **CANCEL** отменяет все действия и выводит из диалогового окна.

Кнопка **HELP** выводит окно с подсказкой.

Переключения внутри диалогового окна осуществляются клавишей TAB. Внутри списочных полей переключение осуществляется курсором.

Команда New Вам уже знакома.

Команда Save записывает файл из активного окна на диск. Если файл имеет имя по умолчанию (NONAME00. PAS), ТурбоПаскаль открывает диалоговое окно **Save File As** для переименования файла и сохранения его в другом каталоге или на другом диске.

Диалоговое окно содержит входное поле, список файлов, информационную панель, стандартные переключатели **OK**, **CANCEL**, **HELP** и список предыстории.

Во входном поле **Save file as** записывается имя, под которым Вы собираетесь запомнить файл (либо файловая маска для поля Files).

Поле **Files** содержательно не отличается от одноименного поля, описанного ниже в этой главе. То же относится и к остальным элементам диалогового окна:

информационной панели, переключателям **CANCEL** и **HELP.**

Переключатель **OK** служит для подтверждения выполненных действий.

Команда Save as Вам уже знакома.

Команда Save All запоминает все файлы в открытых окнах. Если среди файлов были поименованные по умолчанию как NONAME00.PAS, ТурбоПаскаль для них проведет диалог с использованием диалогового окна **Save File as**.

Команда Change dir открывает диалоговое окно **Change Directory**, в котором можно задать каталог, который Вы хотите сделать текущим. Текущий каталог – тот, который используется ТурбоПаскалем для запоминания файлов и их поиска.

Входное поле **Directory Name** предназначено для впечатывания путевого имени нового каталога.

Поле **Directory Tree** содержит дерево каталогов. Здесь Вы указываете имя каталога, который хотите сделать текущим. Сделав выбор курсором, нажмите клавишу **ENTER**, затем либо выберите **OK**, либо выйдите из меню клавишей **ESC**.

Переключатель **Chdir** инициирует изменение каталога как только Вы выбрали имя нового каталога в дереве или напечатали его во входном поле. Если Вы передумали, выберите CANCEL.

Переключатель **Revert** возвращает к предыдущему каталогу, если Вы еще не вышли из диалогового окна.

Команда Print печатает содержимое активного окна редактирования. ТурбоПаскаль расширяет знаки TAB (заменяет их подходящим числом пробелов) и затем посылает содержимое окна в DOS Print Handler. Команда становится неправомочной, если окно не может быть напечатано. Для печатания выбранного текста наберите Ctrl–K–P.

Команда DOS Shell выводит из среды ТурбоПаскаля в среду DOS, для выполнения команд DOS или входа в другую программу. Для возвращения в среду ТурбоПаскаля наберите в ответ на подсказку DOS слово **EXIT**, затем нажмите клавишу **ENTER**.

1.2 Секция Edit

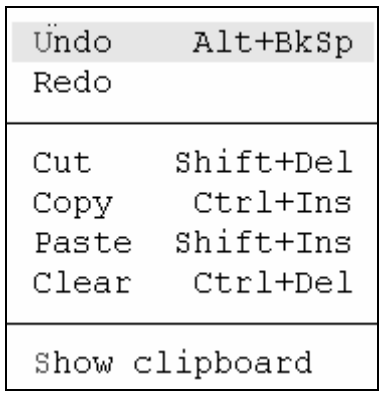

Рисунок 1.3 – Секция Edit

Секция содержит подменю, изображенное на рисунке 1.3.

Секция меню **EDIT** обеспечивает команды для выделения, копирования и вставки фрагментов текста в редакционные окна. Можно также открыть «окно вставок» **Clipboard** (окно, в которое ТурбоПаскаль помещает для хранения все выделенные фрагменты) для обозрения или редактирования его содержимого.

Команда Restore Line возвращает последнюю отредактированную строку. Она также убирает **Ctrl–**

Y или текст, напечатанный на пустой строке. Команда работает только на последней модифицированной или убранной строке.

Команда Cut убирает выделенный текст из документа и помещает его в **Clipboard.** Для выделения текста используйте клавишу **Shift** и клавишу управления курсором (одновременно). Для вклеивания выделенного фрагмента из **Clipboard** в другое место используется команда **Paste** (см. далее). Текст остается в **Clipboard**, и его можно вклеивать неограниченное число раз (пока он остается выделенным в **Clipboard**).

Команда Copy не изменяет выбранный текст, помещая копию выделенного фрагмента в **Clipboard**. Выделенный фрагмент может быть вклеен в любое место с использованием команды **Paste**.

Команда Paste вставляет выбранный текст из **Clipboard** в текущее окно по позиции курсора. Вклеивается текст, выделенный в **Clipboard**.

Команда Clear удаляет выбранный текст, не помещая его в **Clipboard**. Текст, убранный этой командой, теряется. Команда **Paste** не возвращает его.

Команда Show Clipboard открывает окно **Clipboard**, в котором запоминается текст, выбранный редакционных окон с помощью команд меню **EDIT** (**Cut, Copy, Copy Example**). Последний выбранный текст добавляется к **Clipboard**, не убирая предыдущего. Только что добавленный текст выделен (подсвечен) в **Clipboard**, именно этот текст будет использован ТурбоПаскалем при выполнении команды **Paste**. Текст в окне **Clipboard** можно отредактировать. После использования команды **Paste** текст остается в окне **Clipboard** и может использоваться для вклеивания сколько угодно раз (пока он остается подсвеченным).

1.3 Секция Search

Секция содержит подменю, изображенное на рисунке 1.4.

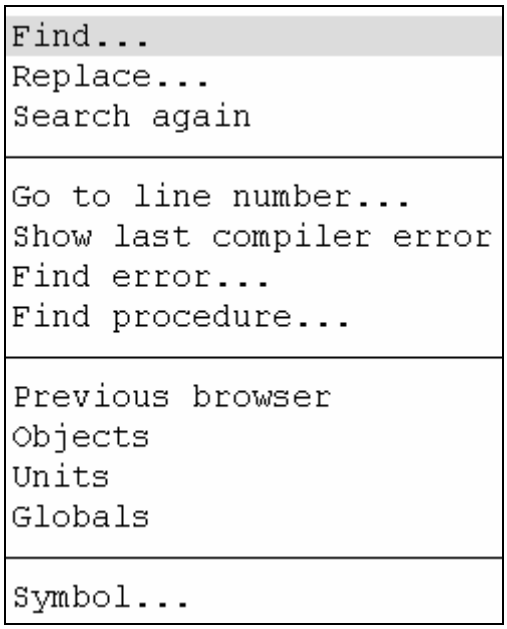

Рисунок 1.4 – Подменю секции Search

Меню **Search** обеспечивает набор команд для поиска и замены текста, поиска процедур и место расположения ошибок в Вашей программе.

Команда Find (Ctrl–Q–F) открывает диалоговое окно **Find**, содержащее входное поле со списком предыстории, три набора переключателей, группу управляющих установок и стандартные кнопки **OK**, **CANCEL** и **HELP**.

Для быстрого входа в диалоговое окно достаточно нажать клавиши **Ctrl–Q–F.**

Для выхода в список предыстории при высвеченном **Text to find** достаточно нажать клавишу управления курсором

«вниз». Откроется окно, содержащее список использованных Вами ранее текстов. Выбор нужного осуществляется курсором, после чего необходимо нажать клавишу **ENTER**. Выход из списка при невыбранной позиции осуществляется нажатием клавиши **ESC**.

Переключение между полями, наборами и клавишами диалогового окна производится клавишей **TAB**, внутри наборов – курсором.

Следует запомнить, что установки, отмеченные прямыми скобками, независимы, т. е. они могут быть выбраны для одновременного воздействия. Переключатели, отмеченные круглыми скобками, альтернативны, т. е. активен только один из них.

Для изменения активности установки внутри поля достаточно «наехать» на нее курсором и нажать клавишу **ПРОБЕЛ**, также работают и с переключателями. Включенная установка содержит знак **X** внутри прямых скобок ([**X**]), выключенная не содержит ([ ]). Для переключателей таким распознающим знаком является точка внутри круглых скобок.

Входное поле Text to Find предназначено для ввода искомой строки, после чего следует выбрать OK для начала поиска.

Если Вы хотите ввести строку, которую Вы уже искали, используйте клавишу управления курсором «вниз». Откроется дополнительное окно, содержащее список предыстории, в котором можно сделать выбор.

Поле **Directions** переключает направление поиска от текущей позиции курсора: вперед (Forward) или назад, к началу текста (Backward). По умолчанию выбрано направление вперед (Forward).

Поле **Scope** определяет объем файла, подвергаемый просмотру при поиске. По умолчанию переключатель устанавливается в позицию Global – просматривается весь файл. Если установить Selected Text, поиск будет происходить только в пределах помеченного блока текста в направлении, указанном в Direction.

Отметка текста осуществляется посредством блоковых команд, сведенных в таблицу 1.1.

Поле **Origin** задает начало поиска. При выборе From cursor поиск начинается с текущей позиции курсора в направлении, заданном в Direction.

При выборе **Entire Scope** просматривается либо целый блок выбранного текста, либо весь файл (независимо от того, где установлен курсор).

Поле **Options** представляет группу установок, задающих тип строк, которые ищет Турбо Паскаль.

При включенном **Case Sensitive** различаются верхний и нижний регистры.

Таблица 1.1

Назначение команды	Используемые клавиши
Маркировка начала блока	Clrl–K–B
Маркировка конца блока	Clrl–K–K
Маркировка единственного	Clrl–K–T
Копирование блока	Ctrl–K–C
Передвижение блока	Clrl–K–V
Исключение блока	Ctrl–K–Y
Чтение блока с диска	Ctrl–K–R
Запись блока на диск	Clrl–K–W
Показать/Скрыть блок	Ctrl–K–H
Печать блока	Clrl–K–P
Индентация блока (отступ)	Ctrl–K–l
Деидеитация блока (убрать	Clrl–K–U

При выборе **Whole words only** Турбо Паскаль ищет только слова, т.е. искомая строка должна иметь пунктуацию или пробелы с обеих сторон. При включенном **Regular Expression** ТурбоПаскаль распознает в искомой строке специальные символы, показанные в таблице 1.2.

Таблица 1.2

Символ	**Толкование символа**
^	Циркумфлекс в начале выражения соответствует началу строки
$	Знак доллара в конце выражения соответствует концу строки
.	Точка используется вместо любого символа
*	Звездочка после символа соответствует любому числу символов (включая ноль). Например, bo* соответствует bot, b, boo, а также be
+	Знак плюс после символа соответствует одному или более (но не нулевому) появлению символа. Например, bo + соответствует bot и boo, но neb или be
[...]	Символы в скобках соответствуют любому символу, который появился в скобках, по не другим
[^]	Циркумфлекс в скобках в начале строки означает NOT. Т.е. [^bot] соответствует любому символу, кроме b, o или 1
[–]	Дефис в скобках означает диапазон символов. Например, [Ь–o] соответствует любому символу от b до o
\	Бэкслэш перед спецсимволом сообщает ТурбоПаскалю, что этот символ нужно воспринимать буквально, а не как спецсимвол. Например, \^ соответствует символу ^, а не символу начала строки

Команда Replace открывает диалоговое окно **Replace**, в котором печатается как текст, подлежащий замене, так и новый текст. Как и в случае других диалого-

вых окон к полям записи могут быть вызваны списки предыстории.

Почти все поля идентичны полям диалогового окна **Find** за исключением установки **Promt on replace** в поле **Options** и кнопки **Change all**. [ ] **Promt on replace**. Установка обеспечивает подсказку ТурбоПаскаля каждый раз перед заменой найденной текстовой строки на новую. Кнопка **Change all** используется для начала поиска. При нахождении заданного Вами текста ТурбоПаскаль спрашивает, хотите ли Вы сделать замену. При нажатой кнопке **Change all** подтверждение требуется для каждой замены.

Команда Search again повторяет последние команды **Find Replace**. Все установки, сделанные в окне **Find**, остаются действующими.

1.4 Секция Run

Секция содержит подменю, изображенное на рисунке 1.5.

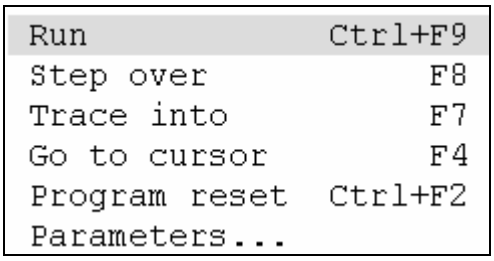

Рисунок 1.5 – Подменю секции Run

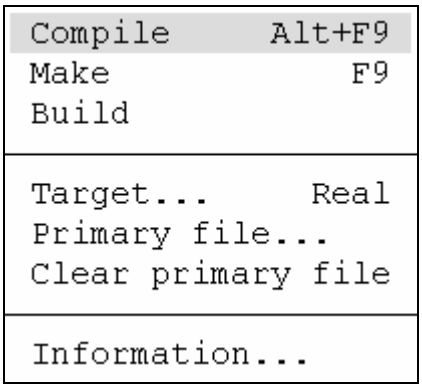

Рисунок 1.6 – Подменю секции Compile

Команды меню **Run** позволяют запускать программу на выполнение, начинать и заканчивать сеанс отладки.

Команда **Step over** позволяет производить пошаговое выполнение программы без захода в подпрограммы (в том числе процедуры и функции).

Команда **Trace into** позволяет производить пошаговое выполнение программы с заходом в подпрограммы.

1.5 Секция Compile

Секция содержит подменю, изображенное на рисунке 1.6.

Меню **Compile** содержит набор функций, обеспечивающих функции компиляции и выполнения Вашей программы.

Команда Compile компилирует файл в активном редакционном окне. При компиляции или выполнении команды Make на экране высвечивается окно состояния с результатами.

После завершения компиляции или команды Make для ликвидации окна состояния компиляции достаточно нажать любую клавишу. При возникновении ошибки в верхней части редакционного окна появляется сообщение.

Команда Make включает встроенный **Project Manager** для создания файла

.**EXE**.

Файлы рекомпилируются в соответствии со следующими правилами:

• Если Compile/Primary File содержит в списке первичный файл, он компилируется, в противном случае компилируется последний файл, загруженный в редактор. ТурбоПаскаль проверяет все файлы, от которых зависит компилируемый файл.

• Если исходный файл для данного модуля модифицировался после того, как объектный код (.TPU) файла был создан, модуль перекомпилируется.

• Если интерфейс для данного модуля изменен, все другие модули, от него зависящие, перекомпилируются.

• Если модуль включает файл типа Include и он новее, чем файл с расширением .TPU данного модуля, модуль перекомпилируется.

Команда Build перестраивает все файлы независимо от их новизны. Команда идентична команде **Make**, но не является условной (**Make** перестраивает только файлы, не являющиеся текущими).

1.6 Секция Debug

Секция содержит двухуровневое подменю, изображенное на рисунке 1.7.

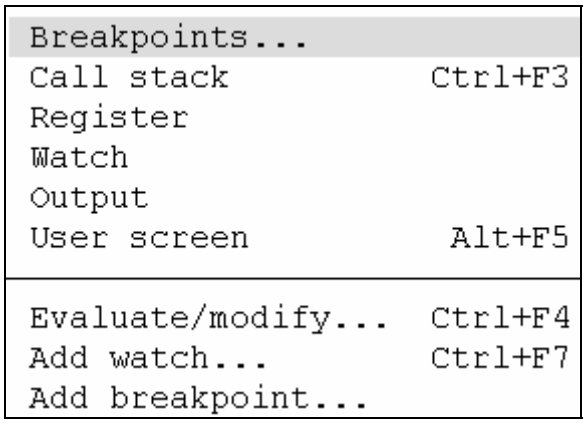

Рисунок 1.7 – Подменю секции Debug

Команды меню **Debug** управляют всеми средствами интегрированного отладчика. Можно изменять установки, сделанные по умолчанию с помощью меню **Options/Debugger.**

Команда Evaluate/modify. С помощью этой команды можно:

• вычислить значение переменной или выражения;

• наблюдать за значениями, принимаемыми переменной или другим элементом данных;

• изменить значение простых элементов данных. Команда открывает диалоговый бокс **Evaluate and Modify**.

Входное поле **Expression** показывает слово, помеченное курсором в редакционном окне. Это слово является наблюдаемым выражением по умолчанию. Для вычисления выражения по умолчанию необходимо нажать клавишу **ENTER**. Можно отредактировать это выражение или заменить его, в т. ч. выбрав нужное из списка предыстории, раскрываемого нажатием клавиши управления курсором «вниз».

Если отладчик может оценить выражение, он показывает значение в поле **Result.**

Если выражение ссылается на переменную или простой элемент данных,

Вы можете передвинуть курсор в позицию **New Value** и ввести выражение как новое значение.

Для удаления окна нажмите клавишу **ESC**. Если Вы изменили содержимое поля **New Value** и не нажали **ENTER**, отладчик проигнорирует изменения.

Поле **Result** высвечивает значение выражения, введенного во входное поле **Expression**.

Поле **New Value** предназначено для ввода нового значения. Кнопка **Evaluate** используется для вычисления выражения, введенного в бокс **Expression**.

Кнопка **Modify** используется для изменения значения переменной или простого элемента данных, введенного в **Expression**, на значение, введенное в **New Value.**

Команда Watches

*Команда **Add watch*** помещает наблюдаемое выражение в окно **Watch**. При выборе **Add Watch** отладчик открывает диалоговое окно **Add Watch**. Во входном поле **Watch expression** высвечивается выражение по умолчанию (то, на которое указывает курсор в редакционном окне). Для поиска и выбора другого выражения (из числа уже использовавшихся) можно открыть список предыстории.

Если Вы вводите допустимое выражение, нажав клавишу **ENTER** или задействовав **ОК,** отладчик добавляет выражение и его текущее значение в окно **Watch**.

Если окно **Watch** является активным, для введения нового выражения для наблюдения нужно нажать клавишу **INS**.

*Команды **Delete Watch** и **Edit watch.*** Эти команды становятся доступными в строке состояния, когда идет работа с окном **Watch.** Команда **Delete Watch** убирает текущее выражение **Watch** из окна **Watch** (текущее выражение помечено в окне Watch). Того же эффекта можно достичь, нажав DEL или Ctrl–Y.

Команда **Edit watch** дает возможность редактировать выражение в окне **Watch**. Команда открывает диалоговое окно **Edit Watch**, содержащее копию текущего наблюдаемого выражения. В поле **Watch expression** редактируется выражение, заданное по умолчанию (выражением по умолчанию выбирается выражение, на которое установлен курсор в редакционном окне). Можно использовать список предыстории. Отредактированное выражение заменяет исходную версию при нажатии клавиши **ENTER** или выборе кнопки **ОК**.

Команда Toggle breakpoint. Этой командой можно установить или снять безусловную точку прерывания (точку останова при отладке) на строке, позиционируемой курсором. Установленная точка отмечается подсветкой. После повторного нажатия Ctrl–F8 точка прерывания снимается.

2 ОСНОВЫ ЯЗЫКА

2.1 Синтаксические диаграммы

В настоящее время одним из наиболее распространенных формальных ме-

тодов, которые используются для описания синтаксических правил языков программирования, используются диаграммы, называемые синтаксическими. Н. Вирт – создатель языка Паскаль, популяризовал синтаксические диаграммы и поэтому они носят название синтаксических диаграмм Вирта. На синтаксических диаграммах использованы два вида четырехугольников – с прямыми и округленными углами. В прямоугольники заключаются элементы языка, значение которых должно быть определено (так называемые «нетерминальные» символы). В четырехугольники с округленными углами (или иногда в кружки) размещаются так называемые терминальные символы, или иероглифы языка. Значение их в определении не нуждается. Стрелки на диаграмме показывают направление движения по диаграмме при раскрытии структуры понятия, записанного при входе в диаграмму.

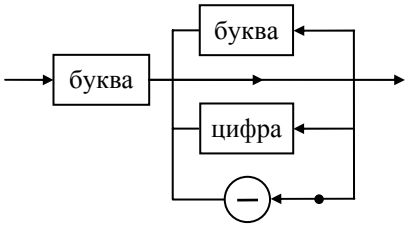

Рисунок 2.1. Пример синтаксической диаграммы

Пример синтаксической диаграммы для определения синтаксиса понятия «идентификатор» показан на рисунке 2.1.

Чтобы получить правильные грамматические конструкции языка, используя синтаксические диаграммы, нужно идти по путям, указанным стрелками, от одного четырехугольника к другому до тех пор, пока не встретится выход.

Там, где предусмотрено более одного направления движения, можно выбрать любое. Если по пути встретилась ссылка к другой синтаксической диаграмме, то надо войти в эту новую диаграмму, пройти по ней, выйти из нее и возвратиться на старое место в первоначальной диаграмме. Если по пути движения встретилась точка, то это означает, что данный путь является характерным только для ТурбоПаскаля и является расширением стандарта.

В ТурбоПаскале в идентификаторах могут использоваться не только заглавные, но и строчные латинские буквы, так как компиляторы не делают между ними различия. Диаграмма для строчных букв аналогична диаграмме для заглавных.

2.2 Структура Паскаль–программы

По стандарту языка каждая Паскаль–программа начинается со слова **Program**, за которым следует имя программы. Это имя идентифицирует программу, но не имеет никакого значения внутри нее. Операторы, составляющие тело программы, должны быть ограничены так называемыми операторными скобками **begin** и **end**. После end в конце программы обязательно присутствие «.» (точки), которая служит для компилятора признаком логического конца программы.

Простейшая Паскаль–программа имеет вид:

```
Program First (Output);
begin
  Writeln ('Наша первая программа')
end.
```

В качестве первой строки программы записывается ее заголовок. Кроме имени программы в заголовке могут записываться имена файловых переменных, используемых в данной программе. В конце заголовка программы обязательно ставится символ « ; ». Из синтаксической диаграммы, изображенной на рисунке 2.2, видно, что все секции являются необязательными, за исключением секции операторов.

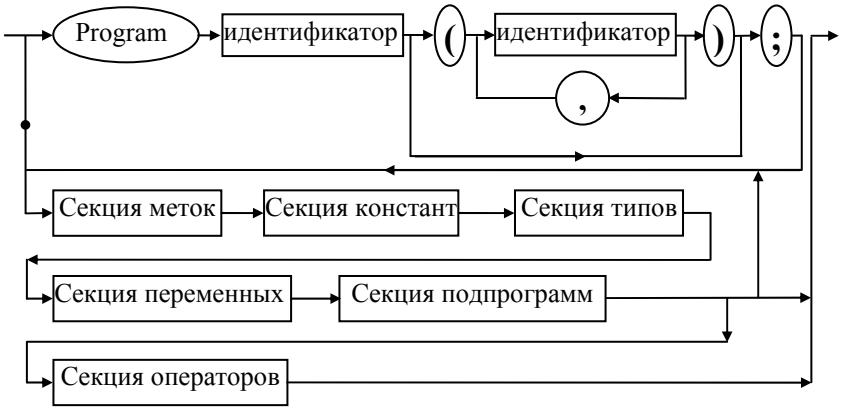

Рисунок 2.2 – Диаграмма Паскаль-программы

Оператор представляет собой составной оператор, содержащий один или последовательность операторов, заключенных в операторные скобки **begin** и **end**. Паскаль–программы могут содержать одну или более операторных скобок **begin**–**end**. Каждой скобке **begin** должна соответствовать **end**. В программе допустима произвольная вложенность операторных скобок. Единственным ограничением при использовании вложенных скобок является следующее: каждая более внешняя пара операторных скобок должна полностью включать более внутреннюю пару операторных скобок, то есть они не должны перекрываться.

Например,

```
begin
  begin
    (операторы 1–х)
  end
  begin
    (операторы 2–х)
  end
end.
```

Рекомендуется первую пару операторных скобок Паскаль–программы размещать на том же уровне, что и слово **Program**. Секции описании можно сдвигать вправо от слова **Program**. С целью наглядности рекомендуется каждую следующую пару операторных скобок сдвигать вправо.

В ТурбоПаскале заголовок программы и параметры оператора **Program** не обязательны. Если заголовок присутствует, то он проверяется на правильность синтаксиса, но никакого влияния на программу не оказывает. Кроме того, на гло-

бальном уровне программы (то есть на ограниченном рамками процедуры) ТурбоПаскаль допускает произвольный порядок следования секций **label**, **const, type**, **var** и произвольное их количество.

Одна из возможных структур Паскаль–программы имеет вид:

```
var    I: integer;
const  Pi=3.1415;
var    Pi2: real;
type   f2=text;
label  1,2;
type   f3=file of real;
begin
   { операторы}
end.
```

Точка с запятой используется в Паскале для разделения двух идущих друг за другом операторов.

За зарезервированным словом **begin** никогда не ставится точка с запятой. Можно опустить точку с запятой и перед зарезервированным словом **end**, так как **begin** и **end** аналогичны скобкам.

В том случае, если перед **end** все же стоит точка с запятой, то подразумевается наличие пустого оператора.

2.3 Константы

Константой в Паскале может быть идентификатор константы, целое или действительное число, строка. Число без знака считается вещественным, если в его состав входит десятичная точка или символ «**E**». Все остальные числа считаются целыми. Символ «**E**» при представлении чисел в форме с плавающей точкой заменяет основание степени 10, а непосредственно за ним следует порядок.

В стандарте Паскаля предусмотрены именованные константы. Именованная константа – это фиксированное значение, которому при объявлении константы в секции констант дается имя. Например, пусть дано объявление:

```
const Speed=120;
      Pi=3.14159;
      Beta=Speed;
```

Присваивание имен константам делает программу более удобной для понимания и внесения исправлений в программу. При изменении констант достаточно изменить их значения в секции констант. Синтаксическая диаграмма именованной константы приведена на рисунке 2.3.

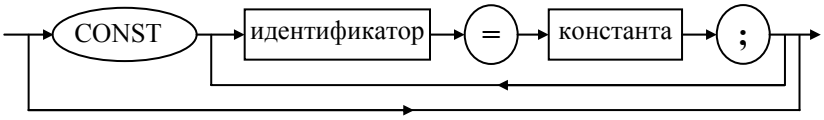

Рисунок 2.3 – Синтаксическая диаграмма именованной константы

2.4 Переменные. Типы переменных

Переменная – это величина, обращение к которой производится по имени. Любая встречающаяся в Паскаль–программе переменная должна быть объявлена в секции переменных. Исключение составляют только предопределенные имена, т.е. имена с заранее определенным компилятором смыслом. Предпочтительнее использование осмысленных имен, так как это делает программу более доступной для понимания. Диаграмма секции переменных приведена на рисунке 2.4.

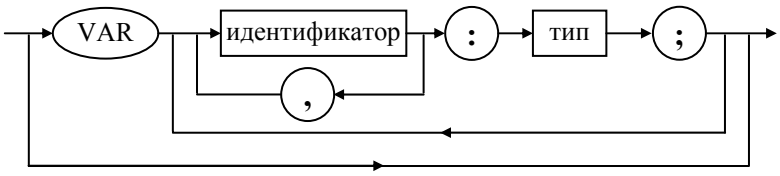

Рисунок 2.4 – Диаграмма секции переменных

С помощью объявления имен в секции переменных устанавливается не только факт существования переменной, но и задастся ее тип. В стандарте языка предопределены четыре стандартных типа переменных:

– **integer** (целый); значения – все целые числа;
– **real** (вещественный); значения – все вещественные числа;
– **boolean** (логический); значения – **true** или **false**;
– **char** (символьный); значения – элементы конечного и упорядоченного множества символов.

Размер переменных в зависимости от их типа приведен в таблице 2.1.

Таблица 2.1

Тип	Диапазон значений	Количество значащих цифр	Размер в байтах
real	2.9E–39..1.7E38	11–12	6
single	1.5E–45..3.4E38	7–8	4
double	5.0E–324..1.7E308	15–16	8
extended	3.4E–4932..1.1E4932	19–20	10
comp	–9.2E18..9.2E18	19–20	8
Shortint	–128..127	Signed	8–bit
Integer	–32768..32767	Signed	16–bit
Longint	–2147483648..2147483647	Signed	32–bit
Byte	0..255	Unsigned	8–bit
Word	0..65535	Unsigned	16–bit
Boolean			8–bit

Элементами множества символов являются символы на устройствах ввода–вывода, поэтому одного, стандартного множества нет. Однако вне зависимости от реализации для множества символов должны выполняться следующие условия:

– упорядоченность в алфавитном порядке множества прописных латинских букв A,B,C,...Z (непрерывность для этого множества не требуется);

– множество десятичных цифр должно быть упорядоченным и непрерывным;

– в множество должен быть включен символ «пробел». Для кода ASCII все эти требования выполняются. Каждый символ этого кода имеет неотрицательный порядковый номер. Т.е. множество символов является порядковым. Множество чисел типа **real** порядковым не является.

Примеры объявления переменных:

```
Var  A: rea1;
     BI,CI: integer;
     L: boolean;
     S: char:
```

3 ВЫРАЖЕНИЯ И ФУНКЦИИ В ПАСКАЛЕ

3.1 Выражения

Н. Вирт пишет: «Выражения представляют собой конструкции, задающие правила получения значения переменных и образования путем применения операций новых значений». Выражения состоят из операндов и операций.

Существуют следующие типы операций:

– арифметические;

– отношения;

– логические (булевы);

– операции над множествами.

В таблице 3.1 приведены арифметические операции, типы операндов и результатов.

Таблица 3.1

Операция	Действие	Тип операн-	Тип результата
+	сложение	integer, real	integer, real
–	вычитание	integer, real	integer, real
*	умножение	integer, real	integer, real
/	деление	integer, real	real
div	деление нацело	integer	integer
mod	вычисление остатка от деления	integer	integer

Операции «+» или «–» (в отличие от всех остальных арифметических операции) могут быть использованы с одним операндом. Ряд арифметических операций может использоваться с операндами типа **integer** или **real**, результат при этом

будет типа **real**, если хотя бы один из операндов имеет тип **real**.

Примеры записи арифметических выражений:

10+4 {равно 14};

9*(–3) {равно –27};

10 **div** 3 {равно 3};

15 **div** 3 {равно 5}.

Операция **div** вычисляет целую часть частного, а его остаток можно найти с помощью операции **mod**. Требование стандарта Паскаля состоит в том, чтобы результат выполнения операции **mod** был положительным, независимо от знаков операндов. Для двух положительных целых операндов A и B для этой операции выполняется равенство вида:

A mod B = A – (A div B) * B

Операции «*», «/», «+», «–» необязательно окружать разделителями, например, пробелами, но пробелы обязательны при употреблении зарезервированных слов **mod**, **div**, поскольку **AmodB** является идентификатором, а **A mod B** – выражением, используемым для вычисления остатка от деления **A** на **B**.

При вычислении арифметических выражений приняты следующие правила, определяющие приоритет операторов. Операции «*», «/», **div**, **mod** имеют более высокий приоритет, чем операции «+» и «–». Операции с более высоким приоритетом выполняются раньше. Если операции имеют одинаковый приоритет, то операция, стоящая левее, выполняется первой.

В Паскале предусмотрены 3 булевы (логические) операции:

– **not** – отрицание (логическая инверсия);

– **and** – конъюнкция (логическое умножение):

– **or** – дизъюнкций (логическое сложение);

Эти операции являются фундаментом булевой логики, разработанной в XIX веке математиком Джорджем Булем. Результат операции **and** является истинным, если оба ее операнда истинны. Результат операции **or** является истинным, если какой–либо из ее операндов истинен. Операция **not** имеет один операнд и образует его логическое отрицание. Результаты логических операций сведены в таблицу 3.2.

Таблица 3.2

	Переменная	Значения			
Операция	X	True	True	False	False
	Y	False	True	False	True
Отрицание	**NOT** X	False	False	True	True
Конъюнкция (логическое умножение)	X **AND** Y	False	True	False	False
Дизъюнкция (логическое сложение)	X **OR** Y	True	True	False	True

Булевы выражения могут быть сложными. Операнды этих выражений

должны быть булева типа. Самой приоритетной булевой операцией является операция **not,** за ней следует операция **and**, а затем операция **or.**

Пример булевых выражений: **not** C **and** K, где C и K– булевы переменные.

Булев тип в языке Паскаль определен таким образом, что false меньше true (при этом порядковый номер false равен нулю, true – единице). Поэтому к булевым операндам применимы операции отношения. Примеры выражений:

(X=0) and (Y=0) {при X=5, Y= –2 выражение имеет значение false}.

В таблице 3.3 приведен порядок выполнения операций в порядке убывания приоритетности

Таблица 3.3

Приоритетность	Тип операции	Имя
0	инверсия	not
1	мультипликативные операции	+, /. div, mod. and
2	аддитивные операции	+, –, or
3	операции отношения	=, <>, >, >=, <, <=

3.2 Стандартные функции

Для выполнения часто встречающихся вычислительных операций и преобразования данных, относящихся к разным типам, существуют заранее определенные стандартные функции. Ниже приведена таблица типов аргументов и результатов стандартных функций языка Паскаль. Во всех тригонометрических функциях (**Sin**, **Cos**, **Arctan**) аргумент задается в радианах. Действия функций следующие: **Abs**(X) – вычисляет абсолютное значение X; **Exp**(X) – ***e*** возводится в степень X; **Ln**(X) – вычисляется натуральный логарифм X; **Sqr**(X) – X возводится в квадрат; **Sqrt**(X) – вычисляется квадратный корень из X.

Для стандартных функций **Arctan**, **Cos**, **Sin**, **Exp**, **Ln**, **Sqr**, **Sqrt** аргумент может быть целым или вещественным, но результат – всегда вещественный.

Функции **Trunc**, **Round**, **Chr**, **Ord** принято называть функциями преобразования типов.

Trunc(X) – X имеет тип real. Результатом является целая часть числа X. Например, **Trunc**(6.7) = 6, **Trunc**(–6.7) = – 6.

Round(X) – X имеет тип real. Результатом является ближайшее к X целое число. Например, **Round**(4.8) = 5, **Round**(4.5) = 5.

Chr(I) – I имеет тип integer. Результатом является символ, порядковый номер которого на множестве символов равен I. Например, **Chr**(66) = B.

Ord(C) – имеет порядковый тип. Результатом является номер (код) C в упорядоченной последовательности элементов. Например, **Ord**('B')=66, **Ord**(False) = 0.

Для функций **Ord** и **Chr** на множестве символов справедливо соотношение **Chr**(**Ord**(C))=C.

Функции **Pred, Succ** принято называть функциями порядковых типов.

Pred(K) – **К** имеет порядковый тип. Результатом является предшествующий **К** элемент на порядковом множестве. **Pred**(K) считается неопределенным, если **К** является первым по порядку элементом множества. Например: **Pred**('B')='A', **Pred**(10)=9.

Succ(K) – **К** имеет порядковый тип. Результатом является следующий по порядку за **К** элемент на порядковом множестве. **Succ**(K) считается неопределенным, если **К** является последним на порядковом множестве элемент. Например, **Succ**('A')='B', **Succ**(false)=true

Функции **Odd**, **Eoln**, **Eof** являются булевыми, так как результат этих функций имеет тип **boolean**.

Odd(X) – X имеет целый тип. Если X – нечетный, то результат принимает значение **true**, если четный – **false**.

Eo1n(X) – X – файловая переменная. Результат принимает значение **true,** если при чтении текстового файла достигнут конец текущей строки. В остальных случаях результат равен **false**.

Eof(X) – X – файловая переменная. Результат принимает значение **true**, если при чтении текстового файла достигнут конец файла, в противных случаях результат равен **false**.

Большинство функций в качестве аргументов может использовать числа, переменные, выражения, типы которых должны соответствовать указанным в таблице 3.4.

Таблица 3.4

	integer	real	boolean	char	file
integer	Pred, Succ, Abs, Sqr	Trunc, Round	Ord	Ord	
real	Sin, Cos, Arctan, Ln, Exp, Sqr	Abs, Sqr, Sin, Cos, Arctan, Ln, Exp, Sqrt			
boolean	Odd		Pred, Succ		Eof, Eoln
char	Chr			Pred, Succ	

В ТурбоПаскале к арифметическим функциям добавлены три функции: **Pi** – возвращает значение числа **Pi**; **Int** – возвращает целую часть аргумента; **Frac** – возвращает дробную часть аргумента.

Для переменных порядкового типа в ТурбоПаскале введены дополнительные функции: **Dec –** уменьшает значение переменной; **Inc** – увеличивает значение переменной. Пример:

```
var D, R, A: char;
    S, B: real;
    K, l: integer;
begin
```

```
B:=1.75;
S:=lnt(B) ; { Значение S равно 1 }
A:='P';
R:==lnc(A) ; { Значение R равно 'R' }
D:=Dec(R) : { Значение D равно 'P' }
I:= 25;
К := lnc(l) ; {Значение К равно 26 }
end.
```

Для того, чтобы возвести в любую степень число необходимо воспользоваться известной формулой $a^x = e^{x \cdot \ln a}$, т.е. **exp(x*ln(a)).**

Задания для контроля. Напишите выражения в том виде, в котором его необходимо записать в Паскаль–программе:

1) $c = d : 6 \cdot V$; 2) $\sqrt{\frac{a \cdot b^{3,1}}{a+b}}$; 3) $y = \ln(x)\frac{e^c}{a+b}$.

4 ОПЕРАТОРЫ ПАСКАЛЯ

4.1 Простые и структурные операторы

Алгоритм решения задачи, записанный на языке Паскаль, представляет собой последовательность операторов. **Оператор** – это основной элемент входного языка. Операторы делятся на простые и сложные (структурные).

Простые операторы не содержат внутри себя других операторов. **Структурные** представляют собой конструкции, в состав которых входят простые операторы. К простым операторам относятся оператор присваивания, пустой оператор, оператор перехода и оператор вызова процедуры.

К структурным операторам отнесены составной оператор, условный оператор, все операторы цикла, оператор варианта, оператор присоединения.

Все операторы языка Паскаль представлены на диаграмме (рисунок 4.1) в следующем порядке:

– оператор присваивания;
– оператор процедуры (или вызова процедуры);
– составной оператор;
– условный оператор (**if**);
– оператор варианта (**case**);
– оператор цикла с предусловием (**while**);
– оператор цикла с постусловием (**repeat**);
– оператор цикла с параметром (**for**);
– оператор присоединения (**with**);
– оператор перехода (**goto**);
– пустой оператор (оператор не выполняет ни какого действия, поэтому на диаграмме он представлен прямой линией).

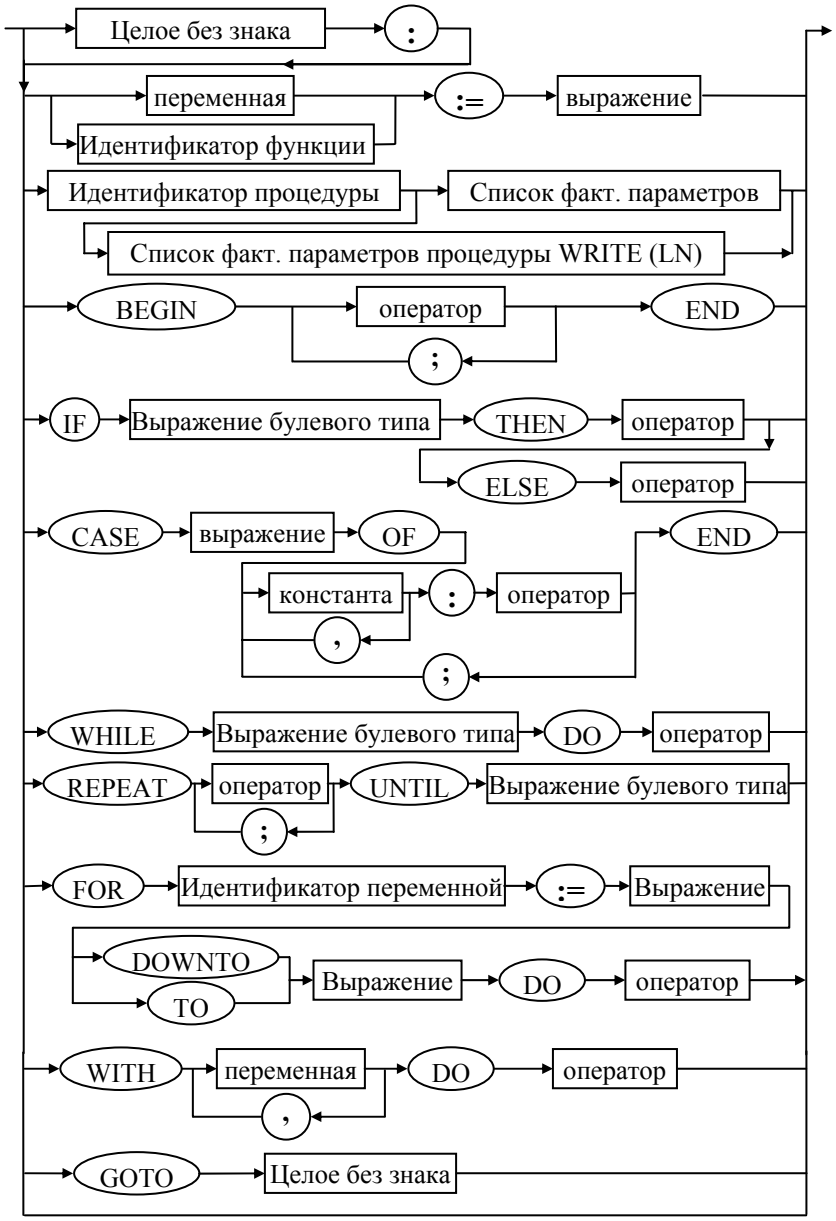

Рисунок 4.1 – Диаграмма оператора

4.2 Оператор присваивания

Форма записи оператора имеет вид: переменная **:=** выражение. Знак «**:=**» называется знаком операции присваивания.

Действие оператора состоит в том, что вначале вычисляется значение выражения и после этого вновь вычисленное значение присваивается переменной. Переменная и выражение должны иметь один и тот же тип. Примеры:

A:=0;

K := 2 * K + M + Sqr (X + 4);

Фрагмент программы:

```
Var    A, B: Integer;
       Rez: Real;
begin
  Rez := A+B; (* При выполнении этого оператора вычисляется значение
```

выражения справа (оно имеет целый тип). Вычисленное значение преобразуется к вещественному типу и после этого присваивается переменной REZ *)

A := Rez; (* Переменная Rez – вещественного типа, а A – целого, поэтому операция присвоения ошибочна, так как в Паскале запрещено присваивать значение выражения вещественного типа переменной целого типа*)

Пример 4.1. Приведем программу вычисления удельной касательной силы тяги электровоза по следующей формуле [7]:

$$f_k = \frac{\gamma\left(2115 - 39{,}34 \cdot V + 0{,}1965 \cdot V^2\right) \cdot 1000}{m_c + m_л}, \qquad (4.1)$$

где приняты следующие обозначения: γ – коэффициент ослабления поля; V – скорость; m_c, $m_л$ – масса состава и локомотива соответственно.

```
Program TEST;
Var fk, gam, V, ms, ml: Real;
Begin
   V:=62.5; {км/ч} gam:=1; ms:=100{т}; ml:=200 {т};
   fk:=(gam*(2115–39.34*V+0.1965*sqr(V))*1000)/(ms+ml);
end.
```

4.3 Оператор перехода

Форма записи оператора: **Goto** <метка>. **Оператор перехода** – простой оператор, указывающий, что дальнейшая работа программы должна продолжаться с оператора, на котором стоит метка. **Метка** – целое число без знака. Не более четырех цифр. Метки программы должны быть обязательно описаны в секции метки (рисунок 4.2). Оператор перехода следует использовать в исключительных ситуациях, когда приходится нарушать естественную структуру алгоритма.

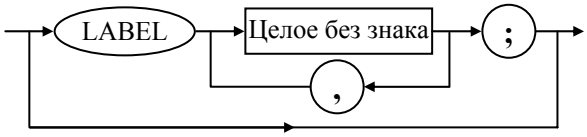

Рисунок 4.2 – Диаграмма секции меток

4.4 Оператор вызова процедур

Форма записи оператора процедур имеет вид: имя_процедуры [(список фактических параметров)]. Квадратные скобки означают, что список фактических параметров может отсутствовать. Понятие процедуры, ее структура, способ вызова будут рассмотрены ниже. В первую очередь рассмотрим стандартные процедуры ввода–вывода, так как трудно реализовать простейшую Паскаль программу без организации ввода–вывода данных.

4.5 Ввод-вывод в Паскаль-стандарте

Задачу обеспечения взаимодействия человека и ЭВМ принято называть вводом–выводом. В Паскале эти функции реализованы при помощи четырех стандартных процедур: **Read, Readln, Write, Writeln.**

Для ввода информации используются две процедуры вида:

Read (C1, C2, ... Cn), **Readln** (C1, C2, ... Cn)

Read – процедура читает данные с именем Ci. Действие процедуры **Read** оканчивается, как только исчерпается список переменных C1,..., Cn.

Действие процедуры **Readln** аналогично. Однако после исчерпания переменных (фактических параметров) действие процедуры заканчивается переходом курсора на начало следующей строки.

Операторы вывода имеют вид:

Write (P1, P2, ... Pn), **Writeln** (P1, P2, ... Pn)

P1, ..., Pn – имена переменных, составляющих список вывода.

Процедура **Write** реализует вывод значений переменных P1,..., Pn.

Процедура **Writeln** реализует вывод значений переменных P1,..., Pn в одну строку и переходит к началу следующей строки. Процедура **Writeln** без параметров реализует пропуск строки (перевод строки).

Таблица 4.1

Примеры	Значение	Устройство вы-
Write (I:4)	I = 5	_5
Write (I:5, J:4)	I = – l, J = 297	_ _ _ – 1_297
Write (I, J)	I = – l, J = –	_ – l _.. _ – 435
Write (I:1, J:l)	I = – 1, J = –	– 1 – 435
Write(I:l, ',', J:l)	I = – l, J = –	– 1, – 435
Write ('Метод Га-		Метод Гаусса
Write(' I = ', l:2)	I = – l	I = – l

При этом следует помнить, что параметры P1...Pn могут быть вида:

E

E : m

E : m : n ,

где **E** – выражение, значение которого необходимо вывести, m, n– выражения типа **integer**, необязательные параметры, указывающие соответственно ширину выводимого поля и количество дробных цифр.

Выражение **E** может быть типа real, integer, char, boolean, а также может быть строкой символов. Конструкция вида **E : m : n** может использоваться только для данных типа **real**. Для данных остальных типов употребляется конструкция вида **E : m**.

Если выводимое данное имеет меньше знаков, чем **m**, то оно дополняется слева пробелами. Если больше, то выводится столько знаков, сколько необходимо

для корректного представления результата.

Если параметры **m** и **n** опущены, то подразумевается их некоторые, зависящие от реализации, значения.

Пример 4.2. Приведем программу вычисления удельной касательной силы тяги электровоза по формуле (4.1), которая будет читать значения исходный данных, а выведет результат с соответствующим комментарием.

```
Program Test;
Var fk, gam, V, ms, ml: Real;
Begin
   Write('Скорость электровоза =');
   Readln(V);
   Write('Коэффициент ослабления поля =');
   Readln(gam);
   Write('Масса состава и локомотива =');
   Readln(ms, ml);
   fk:=(gam*(2115–39.34*V+0.1965*sqr(V))*1000)/(ms+ml);
   Writeln('Удельная касательная силы тяги электровоза =', fk,'{Н/м});
   Readln;
end.
```

Задание для контроля. Написать программу, которая предложит пользователю ввести скорость поезда в начале и конце торможения $V_{нач}$ и $V_{кон}$ (км/ч), тормозной путь $S_{тр}$ (м) и выведет на экран соответствующее значение удельной тормозной силы, вычисляемой по следующей формуле [7]:

$$b_T = \frac{40{,}58 \cdot \left(V_{кон}^2 - V_{нач}^2\right)}{S_{TP}} - \left(w_X + g \cdot i\right), \qquad (4.2)$$

где w_X – основное удельное сопротивление движению локомотива на выбеге, Н/т; i – уклон пути, %; g – ускорение силы тяжести, м/с^2.

4.6 Условный оператор IF

В Паскале существуют средства, с помощью которых можно организовать ветвление в программе. Одним из таких операторов является оператор **If**, представленный на синтаксической диаграмме (рисунок 4.1).

Действие оператора состоит в следующем: вычисляется значение выражения булева типа. Если оно истинно, то выполняется оператор, следующий за словом **then**. Если значение ложно, выполняется оператор, следующий за словом **else**. Если после зарезервированных слов **then** или **else** нужно выполнить несколько операторов, то их нужно объединить в составной оператор с помощью операторных скобок **begin**, **end**.

Условные операторы могут быть вложенными, степень их вложенности Паскалем не ограничена. Перед зарезервированным словом **else** не ставится точка с запятой. Кроме этого **else** – часть в конструкции условного оператора, которая

может опускаться. В этом случае, если булево выражение имеет значение **false**, никакие действия не выполняются, управление в программе передается оператору, следующему за условным оператором.

Необходимо отметить, что оператор, фигурирующий в синтаксической диаграмме условного оператора, может быть составным.

Пример 4.3. Напишем программу, которая определяет и печатает наибольшее из двух чисел

```
Program Max;
Var    A, B: Integer;
begin Read (A,B);
   if A > B
      then Writeln (A)
      else Writeln (B)
end.
```

Для наглядности желательно соблюдать вид записи условных операторов. Альтернативные части этого оператора должны сдвигаться по отношению к условию:

```
if Условие
        then оператор
        else оператор
```

Если операторы составные – то их расположение может быть следующее:

```
if условие
  then begin
       операторы
  end
  else begin
       операторы
  end
```

4.7 Оператор варианта CASE и его расширение в ТурбоПаскале

Более общим случаем условного оператора является оператор варианта **case**. Условный оператор в зависимости от значения булева выражения обеспечивает выполнение одного из двух возможных операторов. Оператор **case** дает возможность выполнять один из нескольких операторов в зависимости от значения выражения, фигурирующего на синтаксической диаграмме (рисунок 4.1) и часто называемого селектором. Другими словами **case** выполняет роль переключателя в зависимости от значения селектора. Селектор может быть целого, символьного или булева типа.

Константы на синтаксической диаграмме (рисунок 4.1) называют метками вариантов. Метки должны иметь тот же тип, что и селектор. После каждой метки (последовательности меток) после знака двоеточия записывается оператор, который может быть составным.

Если для некоторых значений варианта никаких действий производить не надо, то записывается пустой оператор. Каждое, отличное от других, значение селектора может появиться только в одном элементе **case**–списка.

Действие оператора состоит в следующем: вычисляется значение выражения (селектора). После этого выполняется только тот оператор, который имеет метку варианта, значение которой совпадает со значением селектора.

Желательно, чтобы любое возможное значение селектора было указано среди констант **case**–оператора. В ситуации, когда какая–то константа отсутствует, управление передается операторам, следующим за служебным словом **else**.

Пример 4.4. Рассмотрим Паскаль–программу, в которой вычисляется значение характеристики [7], которые определяются значением скорости (значение скоростей приведены в таблице). Пусть значение скорости, в свою очередь, определяется значением параметра K (таблица 4.2).

Таблица 4.2

K	V
1	V<10 км/ч
2	10<V<22 км/ч
3	22 ≤ V<46,7 км/ч
4	46,7 ≤ V<48,5 км/ч
5	48,5 ≤ V<52 км/ч
6	52 ≤ V<56 км/ч
7	56 ≤ V<60 км/ч
8	V ≥ 60 км/ч

Программа на Паскале имеет вид:

```
Program Test;
Var V, i, U, W, gam: real;
    K: integer;
Begin
   Writeln ('Введите значение напряжения на проводе');
   Readln (U);
   Writeln ('Введите значение параметра K, в соответствии с таблицей (1.1)');
   Readln (K);
   Case K of
   1: Begin Writeln('Введите значение скорости'); Readln(V); i:=685–9*v; end;
   2: Begin Writeln('Введите значение скорости'); Readln(V); i:=1223–3*v; end;
   3: Begin Writeln('Введите значение скорости'); Readln(V); i:=2443–6.3*v; end;
```

```
    4: Begin Writeln('Введите значение скорости'); Readln(V); i:=6291–89*v;
end;
    5: Begin Writeln('Введите значение скорости'); Readln(V); i:=7746–108.6*v;
end;
    6: Begin Writeln('Введите значение скорости'); Readln(V); i:=8140–150*v;
end;
    7: Begin Writeln('Введите значение скорости'); Readln(V); i:=8400–100*v;
end;
    8: Begin Writeln('Введите значение скорости'); Readln(V);
            i:=gam*(9490–158*V+0.782*sqr(V)); end;
    else Writeln ('Значение К неверно');
    end;
    W:=U*I;
    Writeln ('Мощность W=', W);
    Readln;
End.
```

В приведенной программе предусмотрена обработка ситуации, когда вместо цифр 1 – 8 считана другая цифра. В этом случае, можно выдавать сообщение «значение параметра К неверно». Для выдачи такого сообщения в программе в case–оператор после строки с меткой 4 добавлен оператор **Writeln** ('значение ....'). Оператор, следующий за словом **else**, называется оператором умолчания и выполняется, если ни одно значение метки не соответствует значению селектора.

4.8 Операторы цикла

Для реализации циклов в Паскале предусмотрены три оператора. Операторы **while** и **repeat** используются в том случае, если число повторений оператора (может быть составного) неизвестно. Оператор **for** используется, если число повторений оператора известно заранее.

Действие оператора цикла с постусловием **repeat** заключается в следующем. Выполняется оператор или последовательность операторов, следующих за словом **repeat**. Вычисляется значение логического выражения, расположенного за ключевым словом **until**. Если значение ложно, то повторяется выполнение оператора (последовательности операторов), следующих за ключевым словом **repeat**. В случае последовательности операторов, операторные скобки **begin**, **end** не нужны. Если значение выражения – истинно, то выполнение указанных операторов прекращается.

Пример 4.5. Пусть необходимо рассчитать удельный эффективный расход топлива ДВС в диапазоне изменения угловой скорости коленчатого вала двигателя от минимального значения ω_{min} = 50 рад/с до максимального значения $\omega_{max} = \omega_N$ = 300 рад/с с шагом $\Delta\omega$ = 5 рад/с по следующей формуле

$$g_e = g_{eN}\left[1{,}55 - 1{,}55\frac{\omega}{\omega_N} + \left(\frac{\omega}{\omega_N}\right)^2\right], \qquad (4.3)$$

где g_{eN} – удельный эффективный расход топлива в режиме максимальной мощности ДВС – примем равным 80 мг/кДж.

Паскаль–программа , реализующая данный пример, будет иметь следующий вид:

```
Program Test;
Const geN=80; Wmin=50; WN=300;dW{ΔW}=5;
Var    W, ge: integer;
begin W:=Wmin;
   Repeat
      ge:=geN*(1.55–1.55*W/WN+sqr(W/WN));
      Writeln('  W=',W,'  ge=',ge);
      W:=W+dW
   Until W>WN;
end.
```

Действие оператора цикла с предусловием **while** состоит в следующем. Вычисляется значение логического выражения. Если оно истинно, то выполняется оператор (или составной оператор), следующий за ключевым словом **do**. Если оно ложно, то на этом выполнении цикла прекращается и выполняется оператор, следующий за **while**–конструкцией.

Реализация примера 4.5 с использованием **while**–конструкции будет иметь следующий вид:

```
Program Test;
Const geN=80; Wmin=50; WN=300;dW{ΔW}=5;
Var    W, ge: integer;
begin W:=Wmin;
   while W<=WN do
      begin
      ge:=geN*(1.55–1.55*W/WN+sqr(W/WN));
      Writeln('  W=',W,'  ge=',ge);
      W:=W+dW
      end;
end.
```

Идентификатор оператора **for** определяет управляющую переменную цикла. Управляющая переменная принимает значения, начиная с первого (им является выражение после символа « := ») и кончая последним (им является выражение после ключевых слов **to** или **downto**). При каждом новом значении управляющей переменной выполняется оператор (или составной оператор), следующий за ключевым словом **do**. Типы управляющей переменной и выражения должны совпа-

дать. Управляющая переменная должна быть порядкового типа, т.е. для которого определены функции **Succ** (в случае ключевого слова **to**) или **Pred** (в случае ключевого слова **downto**).

Если используется ключевое слово **to,** значение управляющей переменной возрастает в ходе выполнения цикла, если используется ключевое слово **downto,** значение управляющей переменной убывает.

Если первое значение управляющей переменной превышает последнее (для **to**) или меньше последнего (для **downto**), то оператор, стоящий после ключевого слова **do**, не выполняется ни разу.

Пример 4.6. Написать Паскаль–программу вычисления факториала числа.

```
Program Fact;
Var    F, I, N: integer;
begin F:=l;
   Read (N);
   for l:=l to N do F:=F*I;
   Writeln(F)
end.
```

Пример 4.7. Распечатать последовательность латинских букв от 'A' до 'Z'

```
Program Letter;
Var    L:char; I: integer;
begin  for I:='A' to 'Z' do Write(I) ;
       Writeln
end.
```

Задание для контроля. Написать программу, реализующую пример 4.5, с использованием оператора **for** .

5 ПРОСТЫЕ И СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ

5.1 Перечислимый и ограниченный тип

Ранее рассматривались простые стандартные типы данных – **integer**, **boolean, char**, **real**. Первые три типа данных являются порядковыми, т.е. к переменным этих типов применимы стандартные функции **Succ** и **Pred.**

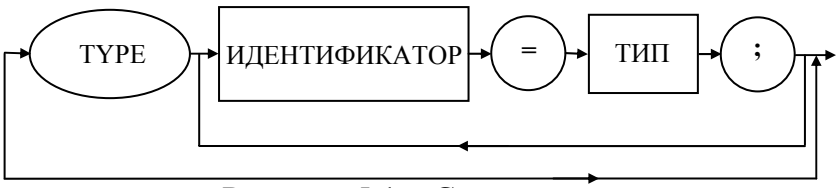

Рисунок 5.1 – Секция типов

Кроме этих типов в Паскале разрешено введение новых типов. Эти новые простые типы должны быть описаны в секции типов, представленной на синтаксической диаграмме (рисунок 5.1). В соответствии со стандартом языка эта секция располагается между секцией констант и секцией переменных. Введение новых типов расширяет возможности языка Паскаль, повышает читабельность про-

грамм.

Допустим необходимо представить последовательность месяцев года. Пользуясь предопределенными типами данных, можно использовать прием, который ставит в соответствие число 1 – январю, 2 – февралю и т.д. Однако такое представление неудобно, т.к. надо трактовать числовые значения. Удобнее было бы написать **Month := MAY.**

Чтобы можно было в программе манипулировать с названиями месяцев, а не числами, в Паскале разрешено ввести новый тип. Переменные, имеющие этот тип, могут принимать значения месяцев года.

```
type
Month = (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV,
DEC) ;
var M: Month;
```

В этом случае создается новый тип данных, состоящий из последовательности значений, заключенных в круглые скобки. Такой тип данных принято называть перечислимым (см. рисунок 4.1, среднюю ветвь). Переменной перечислимого типа может быть назначено любое значение в пределах типа.

В Паскале отсутствуют средства, которые бы позволяли осуществлять непосредственный ввод–вывод переменных перечислимого типа. Если записать

```
M:=APR;
Write (M) ;
```

то слова APR выведено не будет. Поэтому выводить значения переменных перечислимого типа нужно программным путем. Однако можно вывести выражение вида **Write(Ord(M))**. При этом будет выведено число 3, соответствующее порядковому номеру идентификатора **APR** в списке значений. Переменные перечислимого типа могут быть использованы в булевых выражениях. Кроме этого к данным этого типа применимы операции сравнения, например:

```
if M>MAY and M<SEP then Writeln ('Летний месяц')
```

Пример 5.1. Пусть перечень специальностей кафедры колесных и гусеничных машин задан следующими обозначениями: "Гусеничные и колесные машины" – GKM; "Электрические системы и комплексы транспортных средств" – ESKTS; "Информационные технологии проектирования" – ITP. Написать программу, в которой переменной перечислимого типа присваивается одно из значений GKM, ESKTS, ITP в зависимости от введенного одного символа. Если введенные символы ошибочны, то выдать соответствующее сообщение.

```
program TM;
type spes = (KGM,ETTS,AT) ;
var S: spes;
   Well: boolean;
   C: char;
begin
```

```
    Read (C);
    Well := false;
    case C of
    'G': begin Well:= true; S:= KGM; end;
    'E': begin Well:=true; S:= ETTS; end;
    'I': begin Well:=true; S:= AT; end;
    end; { case}
    Writeln(Ord(S));
    Readln;
  if not Well then Writeln ('Ошибочный символ');
end.
```

Упорядоченность элементов перечислимого типа определяется порядком их следования. Самый левый элемент имеет минимальное значение, а наиболее правый – максимальное.

Кроме перечислимого типа в Паскале разрешено введение так называемого ограниченного или интервального типа, форма записи которого представлена на нижней ветви синтаксической диаграммы (рисунок 4.1). Например: **Type year = 1900..2000; letter = 'A'..'Z'; spes= 1..3;.**

Левая и правая константы задают диапазон значений и их называют соответственно нижней и верхней границей ограниченного типа. Значения этих констант должны удовлетворять условию: **левая константа <= правая константа**.

Введение ограниченного типа улучшает читабельность программ, так как представляет диапазон значений, которые может принимать переменная этого типа. Константы в определении ограниченного типа должны относиться к одному и тому же базовому типу (целому, символьному, порядковому). Базовый тип констант определяет допустимость соответствующих операций над данными ограниченного типа. И перечислимый, и ограниченный типы переменных относят к простому типу.

Простой – это такой тип, который может представлять только одно значение. Например, переменная типа **integer** может хранить только одно целое число.

5.2 Структурированные типы данных

Наряду с простыми типами Паскаль содержит ряд структурированных типов данных, которые могут представлять совокупности значений. В Паскале имеются следующие структурированные типы: массивы, файлы, множества, записи.

Переменные этих типов имеют структуры, которые определяются Н.Виртом, как «совокупность связанных данных и множество правил, определяющих их организацию и способ доступа к элементам данных». Выбором той или иной структуры данных мы определяем и алгоритм обработки данных, от которого зависит эффективность их обработки.

5.2.1 Массивы

Массив – это упорядоченный набор переменных одного типа. Массивы содержат фиксированное число компонент, которое задается при определении переменных типа массив. Тип компонент массива – базовый. Тип индекса (индексов) в описании заключается в квадратные скобки и относится к простому.

Ограничений на количество индексов нет. Индекс задает место элемента в массиве. Тип компоненты массива может быть любым.

Примеры описания массивов:

Mas: array [1..15] of real ; (* описан массив из 15 вещественных чисел *)

Spes: array [(GKM, ESKTS, ITP)] of integer; (* описан массив целых чисел, индексы элементов массива имеют перечислимый тип и принимают значения названий специальностей GKM, ESKTS, ITP*)

B : array ['A'..'Z'] of boolean; (* описан массив элементов булевого типа, тип индексов – ограниченный символьный *)

C : array [1..3, 1..5] of real; (* описан двумерный массив с вещественных чисел, содержащий три строки и пять столбцов *)

D : array [(BLACK, WHITE)] of 11..20; (* описан массив D целых чисел с индексами BLACK, WHITE. Каждый элемент массива может принимать значения от 11 до 20*)

К первому элементу массива Mas можно обратиться Mas[l], ко второму – Mas[2] и т.д. Пример цикла, который обнуляет элементы массива Mas имеет вид:

for I:=1 to 15 do Mas[I]:=0;

Переменная Mas **[I]** называется переменной с индексом. В качестве индекса может быть любое выражение, имеющее тот же тип, что и индекс.

Пример 5.2. Написать программу для вычисления суммы элементов массива из 100 вещественных чисел.

```
Program Test;
Const N=100;
Type  Mas = array [1..N] of real;
var   Sum: real ; M: Mas;
      I: integer;
begin
   Sum:=0;
   for I:=l to N do
   Sum:=Sum+M[I];
   Writeln('Sum=',Sum);
end.
```

В Паскале многомерный массив можно описать как одномерный, элементами которого являются массивы. Упомянутую выше матрицу **C** можно описать как одномерный массив из трех элементов типа строка, каждая из которых является массивом из пяти элементов:

Type Mas = array [1..3] of array [1..5] of real;

Var A, B : Mas;

При последовательной записи каждой строки матрицы в память соблюдается правило размещения двумерного массива по строкам. Ссылка на элемент матрицы, лежащей на пересечении i–той строки и j–ого столбца, выглядит следующим образом: **A[I,J]**

Пример 5.3. Написать программу вычисления произведения двумерных веществ матриц A, размерностью (N,M), и B, размерностью (M,L). Результирующая матрица C будет иметь размерность (N,L), причем каждый ее элемент будет вычисляться по формуле

$$c_{ij} = \sum_{k=1}^{M} a_{ik} \cdot b_{kj}, \qquad (i = 1,..., N;\ j = 1,..., L).$$

```
program Test;
const  N=5; M=2; L=3;
var    A: array [1..N,1..M] of real ;
       B: array [1..M,1..L] of real ;
       C: array [1..N,1..L] of real ;
       i,j,k: integer;
begin {Ввод массивов A, B}
   for i:=1 to N do
       for j:=1 to L do
       begin  C[i,j]:=0;
              for k:=1 to M do C[i,j]:=C[i,j]+A[i,k]*B[k,j];
              Writeln('C',i,j,'=',C[i,j]);
       end;
end.
```

5.5.2 Символьные строки. Тип String в ТурбоПаскале

Строка – это массив, компоненты которого имеют тип **char** и тип индекса имеет нижнюю границу, равную 1.

Строки и строковые константы можно использовать в операторах присваивания, а также в процедурах Write, Writeln в качестве фактических параметров.

К строкам применимы все 6 операций отношений, но при этом строки должны иметь одинаковую длину.

В ТурбоПаскале введен тип данных **String**. Тип данных **String** иногда называют стринговым. Примеры описания стринговых переменных:

var Name: string[20]; Title: string[40]; Rez: string [70] ;

Память, отведенная для хранения значений переменной Name, составляет 21 байт. В каждом байте хранится одна литера (литера – это цифра, буква, точка или какой-нибудь другой знак). Каждая литера представляет собой значение типа **char**. Один байт переменной Name содержит ее текущую длину. Это значение не

должно превышать 255. Например:

Name := 'Автор';

Title := ' Программирование на языке Паскаль ';

При присваивании значение стринговой переменной берется в кавычки (апострофы, но не парные кавычки). Переменная **Title** занимает всего 34 байта, один байт содержит ее текущую длину. В стринговых выражениях используется только одна операция – конкатенация (слияние), которая обозначается знаком "+".

Пример. **Rez := Name+Title;**

Значение выражения **Rez: 'Автор Программирование на языке Паскаль'.** К стринговым переменным могут применяться следующие операции сравнения: **=, < >, >, >=, <, <=.** При этом стринговые переменные должны иметь одинаковую длину, в противном случае фиксируется ошибка. Пример:

```
var Age: string [3] ;...
Age := 'тридцать';...
```

Переменной Age будет присвоено значение «три», т.к. максимальная длина переменной Age не должна превышать трех. Лишние правые литеры усекаются.

Для переменных типа String в ТурбоПаскале допускается применение процедур Read, Readln, Write, Writeln.

5.5.3 Записи

Запись – наиболее общий и гибкий структурированный тип в Паскале. Запись состоит из фиксированного числа компонент, называемых полями, которые могут быть различных типов. Этим запись существенно отличается от массива, все компоненты которого должны быть одного и того же типа.

Пример 5.4.

```
type Date = record
        Year: integer;
        Month : 1 ..12 ;
        Day: 1..31
      end;
{Тип Date включает три поля: Year, Month, Day.}
type Book = record
        Title: string [40] ;
        Author: string [50] ;
        Entry: Date
      end;
var Dl: Date;
    b: Book;
```

Объявление типа запись осуществляется с помощью ключевого слова **record**, за которым следует список полей записи. Завершается описание этой структуры ключевым словом **end**. Для каждого поля должны быть указаны имя и тип.

Именем поля может быть любой идентификатор.

Тип **Book** содержит три поля, одно из которых имеет ранее определенный тип **Date.**

В Паскале разрешено использовать массивы записей.

Чтобы обратиться к отдельной компоненте записи, необходимо задать имя записи, за ним точку и сразу за точкой написать название нужного поля, например,

```
Dl.day := 25;
B.title := 'computer games';
B.author := 'Levy';
```

6 ПРОЦЕДУРЫ И ФУНКЦИИ

6.1 Подпрограмма в Паскале

В Паскале существует два вида подпрограмм – процедуры и функции. Для того, чтобы подпрограммы ввести в Паскаль–программу, их надо описать в секции подпрограмм. Эта секция помещает в программе за секцией описания переменных. Каждая процедура или функция описывается только однажды, но может использоваться многократно.

Наличие всех секций (кроме секции оператора) в блоке подпрограмм необязательно. В ТурбоПаскале разрешено любое количество секций типов в блоке подпрограмм.

Каждая подпрограмма (процедура или функция) имеет имя, которое определяется по правилам образования идентификаторов. Список параметров, которые принято называть формальными, содержит перечень исходных данных, с которыми работает подпрограмма, а также идентификаторов, содержащих значения результатов.

В блоке функции или процедуры можно использовать любые переменные, описанные во внешнем блоке или блоке главной программы. Эти переменные принято называть глобальными.

Кроме этого, в блоке можно описать дополнительные переменные, которые будут использоваться только в данной функции или процедуре. Эти переменные временные и называются локальными. Вызывающей программе недоступны значения локальных переменных.

Если, например, переменная с именем **T** является глобальной, т.е. была описана в главной программе, то все подпрограммы, входящие в состав главной программы могут обращаться к переменной **T**. Но, если переменная **T** была описана в конкретной подпрограмме, то в этой подпрограмме значение глобальной переменной с именем **T** становится недоступным , а выбирается значение локальной переменной с именем **T**.

По соображениям надежности не рекомендуется использовать одинаковые идентификаторы в вызывающей программе и подпрограмме. Если программа (функция или процедура) является вложенной, то к ней можно обращаться только

внутри программы (например, с именем **К**), в которой она описана. Причем обращение к ней может происходить, как из самой программы **К**, так и из описанных в **К** других подпрограмм.

6.2 Функции

Функции обычно используются для описания подпрограммы, в результате выполнения которой значение результата получает одна переменная. Эта переменная – имя функции. Поэтому в заголовке функции через двоеточие описывается тип функции. В теле функции должен быть оператор, который присваивает имени функции значение результата. Обращение к функции происходит аналогично обращению к стандартным функциям типа SQR, SIN и др. Обращение к функции происходит в выражении. Для этого в выражении записывается имя функции, вслед за которым в круглых скобках перечисляются фактические параметры, т.е. параметры, которые необходимы для вычисления значения функции. Для каждого формального параметра, входящего в список формальных параметров (в заголовке функции) при обращении к функции должен быть указан фактический параметр. Порядок следования, количество, тип формальных параметров должны строго соответствовать порядку следования, количеству, типу фактических параметров.

Пример 6.1. Написать Паскаль–программу вычисления эффективного крутящего момента двигателя внутреннего сгорания по формуле

$$M_e(\omega) = \frac{N_e(\omega)}{\omega} \qquad (6.1)$$

для значений угловой скорости коленчатого вала двигателя ω, изменяющейся в пределах от 50 рад/с до 300 рад/с с шагом $\Delta\omega$= 10 рад/с. Реализовать с помощью подпрограммы–функции вычисление значений функции $N_e(\omega)$ по формуле:

$$N_e(\omega) = N_{eN}\left[a_\omega \frac{\omega}{\omega_N} + b_\omega\left(\frac{\omega}{\omega_N}\right)^2 - \left(\frac{\omega}{\omega_N}\right)^3\right], \qquad (4.2)$$

где ω_N – максимальная угловая скорость коленчатого вала ДВС – равна 300 рад/с; N_{eN} – максимальная эффективная мощность ДВС; коэффициент a_Δ= 0,15. В формуле (6.2) коэффициенты a_ω, b_ω для двухтактного дизеля соответственно равны: a_ω= 0,8; b_ω= 1,2. Программа должна печатать таблицу значений функции $M_e(\omega)$.

```
Program MOMENT;
Const wN=300.; aw=0.8; bw=1.2; wmin=50.; dw=10.;
Var    w, Me, NeN: real ;
Function Ne (w: real ): real ;
Var    v: real;
begin
```

```
        v:=w/wN;
        Ne:=NeN*(aw*v+sqr(v)*(bw–v));
end;
begin    w:=wmin;
        Write ('NeN=');Readln (NeN);
          Repeat
            Me:=Ne(w)/w;
            Writeln('  w=',w,'  Me=',Me);
            w:=w+dw
          Until w>wN;
end.
```

6.3 Процедуры

Не всегда можно подпрограмму реализовать как функцию (например, работа с массивами, сортировка в памяти и др.). Если по алгоритму требуется выдача более чем одного результата, то, как правило, используются **подпрограммы–процедуры**. Отличия процедур от функций:

– способ вызова процедуры отличается от вызова функции, так как вызов производится с помощью специального оператора вызова процедуры;

– в заголовке процедуры отсутствует описание типа ее имени, так как имя процедуры никак не связано с результатом (одним или несколькими), вызываемым ею;

– для передачи результатов процедуры употребляются формальные параметры в заголовке процедуры, описанные с помощью ключевого слова **var**.

В описаниях программ–процедур и функций различаются два типа параметров – параметры–переменные и параметры–значения. Ключевое слово **var** перед идентификатором в заголовке подпрограммы обозначает параметр–переменную; так как в процедурах имя процедуры не используется для передачи результата, то для этой цели используются формальные параметры–переменные. В списке фактических параметров формальным параметрам–переменным должны соответствовать ссылки, а параметрам–значениям – выражения. Формальные параметры–переменные передают свои значения соответствующим фактическим параметрам при возврате из процедуры в вызывающую программу. Отличаются эти параметры–переменные от остальных формальных параметров, не помеченных ключевым словом **var**, тем, что они могут передать значения и вернуть результат.

Пример 6.2. Написать процедуру вычисления эффективной мощности и эффективного крутящего момента двигателя внутреннего сгорания соответственно по формулам (6.1) и (6.2) для значений угловой скорости коленчатого вала двигателя ω, изменяющейся в пределах от 50 рад/с до 300 рад/с с шагом $\Delta\omega = 10$ рад/с.

```
program DVS;
Const wN=300.; aw=0.8; bw=1.2; wmin=50.; dw=10.;
```

```
Var    w, Me, Ne, NeN: real ;
procedure EFF(w: real; Var N, M: real);
Var    v: real;
begin
       v:=w/wN;
       N:=NeN*(aw*v+sqr(v)*(bw–v));
       M:=N/w;
end;
begin    w:=wmin;
       Write ('NeN=');Readln (NeN);
          Repeat
             EFF(w, Ne, Me);
             Writeln('  w=',w,'  Ne=',Ne,'  Me=',Me);
             w:=w+dw
          Until w>wN;
end.
```

6.4 Процедурный тип данных

Процедурный тип данных необходим в ситуациях, когда при вызове старшего блока ему в подчинение передается один из нескольких однотипных блоков, т. е. может быть выбран любой из них. Например, старший блок (процедура или функция) вычисляет определенный интеграл и находится в модуле М, а подчиненные блоки описывают разные подынтегральные функции и даны в основной программе.

Формат описания данных процедурного типа имеет следующий вид для подчиненных блоков–функций

TYPE имя_типа = Function (список формальных параметров): тип функции;

и для подчиненных блоков–процедур

TYPE имя_типа = Procedure (список формальных параметров);

Переменной процедурного типа можно присваивать имена лишь тех процедур или функций, для которых установлен дальний вызов **{$F+}** [5,6].

Пример 6.3. Используя единый блок суммирования, вычислим с точностью E=0,0001 следующие суммы:

$$S_1 = \frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \frac{1}{3\cdot 4} + \ldots + \frac{1}{j\cdot(j+1)} + \ldots;$$

$$S_2 = \frac{1}{1^2} + \frac{1}{3^2} + \frac{1}{5^2} + \ldots + \frac{1}{(2i-1)^2} + \ldots;$$

$$S_3 = \frac{1}{1\cdot 2\cdot 3} + \frac{1}{2\cdot 3\cdot 4} + \frac{1}{3\cdot 4\cdot 5} + \ldots + \frac{1}{k\cdot(k+1)\cdot(k+2)} + \ldots\ .$$

```
Program Summing;
Const   E=0.001;
Type    Fun = Function (m: word): real; {объявление процедурного типа}
Var     S1, S2, S3: real;
   Function Sum(f: Fun; E: real): real; {начало блока суммирования}
   Var  S: real; m: word;
   Begin        S:=0;  m:=1;
        Repeat S:=S+f(m);
        m:=m+1
        until f(m)<E;
        Sum:=S;
   End; {конец старшего блока}
{$F+}                          (* включение дальнего вызова*)
   Function F1(j: word): real;          {распространяется на три}
   Begin        F1:=1/(j*(j+1));        {нижележащих блока–функции}
   End;
   Function F2(i: word): real;
   Begin        F2:=1/sqr(2*i–1);
   End;
   Function F3(k: word): real;
   Begin        F3:=1/(k*(k+1)*(k+2));
   End;
{$F–}   (* отключение дальнего вызова*)
Begin   {начало основной программы}
  S1:=Sum(F1, E);
  S2:=Sum(F2, E);
  S3:=Sum(F3, E);
  Writeln ( 'S1=', S1, 'S2=', S2, 'S3=', S3);
End.
```

В примере 6.3 **F1, F2, F3** являются блоками–параметрами.

Подчиненные блоки, используемые как параметры, не могут быть вложенными. Заголовки блоков – параметров общего старшего блока (в примере – **Sum**) должны быть подобны; лишь имена блоков и их параметров (в примере **i,j,k**) могут различаться, остальное совпадает. Это подобие позволяет описать общий вид заголовка как процедурный тип, имеющий целью контроль правильности обращений к старшему блоку. Имя блока в описании типа опущено как несуществующий элемент. Например, третью строку программы нужно читать так: "Тип **Fun** – это какая–то вещественная функция одного аргумента типа **word**".

Таким образом, процедурный тип указывает:

– класс подчиненного блока;

– сколько у него параметров;

– тип и порядок записи параметров;
– для блока–функции – тип результата.

Пример 6.4. Пусть заданы допустимая погрешность E и границы a, b ($a < b$) области поиска локального максимума функции F. Составим программу, которая с помощью функции Xmax (описана ниже) находит максимальное значение эффективного крутящего момента двигателя внутреннего сгорания Memax по формуле (6.1) на интервале изменения угловой скорости коленчатого вала двигателя ω от 50 рад/с до 300 рад/с.

```
Program Maximum;
Const   wN=300.; aw=0.8; bw=1.2;
Var     E, NeN, Memax: real;
Type    fx = Function (x: real): real;
Function Xmax (F:fx; E:real; Var a,b: real): real;
  {Начало блока поиска локального максимума}
Var     h: real;
Begin
  Repeat        h:=(b–a)/3;
        If F(a+h) < F(b–h) Then a:=a+h
                           Else b:=b–h
  Until h < E;
  Xmax:= a+h;
End;    {Конец блока поиска локального максимума}
{$F+}
Function Me (w: real ): real ;              {Функция вычисления}
Var     Ne,v: real;                 {эффективного крутящего}
Begin                                       {момента ДВС}
  v:=w/wN;
  Ne:=NeN*(aw*v+sqr(v)*(bw–v));
  Me:=Ne/w;
end;
{$F–}
Begin           {начало основной программы}
  Write ('Введите точность вычисления ='); Readln (E);
  Write ('NeN='); Readln (NeN);
  Memax:= Xmax( Me, E, 50, 300);
  Writeln (‘Максимальное значение эффективного крутящего момента
  ДВС = ‘, Memax);
End.
```

В данном примере в роли абстрактной функции *F* выступает конкретная функция Me. Функцию Xmax можно использовать для отыскания локального максимума любой действительной функции одного действительного аргумента.

7. БИБЛИОТЕКА GRAPH

7.1 Инициализация графического режима

Отметим прежде всего, что система координат в графическом режиме не соответствует системе координат текстового режима. Текстовый режим обеспечивает, как правило, выдачу символов в 25 строк и 80 столбцов. Графические же режимы обеспечивают выдачу точек, различную для различных средств:

640 x 200 для EGA;
640 x 350 для EGA;
640 x 480 для VGA.

Для вывода графических изображений на экран ТурбоПаскаль предоставляет пользователю библиотеку Graph. Это означает, что программа должна содержать объявление этой библиотеки после заголовка.

```
Program имя_программы;
uses Graph; { обязательно для работы в графическом режиме}
```

Работа программы начинается с инициализации графического режима процедурой **InitGraph** и завершается процедурой **CloseGraph**.

Любая программа имеет вид:

```
Program Имя_программы;
uses Graph;
var
grDriver, grMode, errCode: Integer;
{эти переменные используются процедурой InitGraph}
begin { тело программы }
grDriver := Detect; { определение номера драйвера }
InitGraph(grDriver,grMode, 'Путь к драйверу ');
 {путь к драйверу, например, 'C:\TP\BCI'}
errCode := GraphResult;
if errCode = grOK then
begin
   { режим открыт и Вы можете работать}
   .............
   CloseGraph; { закрывает режим графики }
end
else
begin { режим не удалось открыть. Почему ?}
   writelnC...'); { в этом месте Вы сообщаете причину
   неудачи, которую узнаете в результате анализа
   переменной errorCode}
end;
end.
```

Переменная ErrorCode сохраняет значение, полученное функцией GraphRe-

sult, которое может быть числом в диапазоне от 0 до 14.

О безошибочной работе свидетельствует значение, равное 0 (мнемоника этой константы – grOk); остальные значения указывают на причину невозможности инициализации графического режима:

– 1 (grNolnitGraph) – графика не инициализирована (используйте InitGraph),
– 2 (grNotDetected) – не обнаружено графическое устройство,
– 3 (grFileNotFound) – не найден драйвер устройства,
– 4 (grInvalidDriver) – неверный драйвер.

7.2 Построение графиков функций

При выводе графиков на экран возникают проблемы выбора приращения аргумента и масштабирования; необходимо оценить границы изменения аргумента и функции, соотнести вещественным их значениям целочисленные значения номеров позиций на экране.

Приведенная ниже программа выводит на отрезке $[x_n, x_k]$ график любой заданной функции $F(x)$, для которой заданы значения абсолютного минимума $F_{\min}$ и абсолютного максимума $F_{\max}$ (на отрезке). Если фактические значения функции окажутся больше заданного $F_{\max}$ (меньше $F_{\min}$), график обрезается сверху (снизу).

Если значения $F_{\min}$ и $F_{\max}$ завышены по абсолютной величине, то экран недоиспользуется по вертикали. Алгоритм построен таким образом, что ось ординат (*y*) не вычерчивается, если $x_n > 0$ или $x_k < 0$, чтобы использовать весь экран по горизонтали.

Пример 7.1. Вычертить график функции $y = \dfrac{\sin(x)}{x}$ на отрезке $[x_n, x_k]$.

```
Program Grafic;
Uses Graph;
Const n=200; {количество точек}
VargrDriver, grMode: Integer;
{эти переменные используются процедурой InitGraph}
    x, y, x0, y0, xm, ym, j: Integer;
    cvet, fon: Word;          {цвет и фон графики соответственно}
    xn, xk, dx, xx, Fmax, Fmin, Razmax: Real;
Function F(x: Real): Real;      {выводимая на график функция}
Begin
  if x=0 then F := 1 else F := sin(x)/x;
End;
Begin { тело программы }
  Writeln ('Fmax, Fmin, xn, xk = ');
  Readln (Fmax, Fmin, xn, xk);
```

```
    Writeln ('Цвет графика, фон = ');
    Readln (cvet, fon);
    dx := (xk–xn)/n;
    Razmax := Fmax–Fmin;
    if  Fmax<0  then  Razmax := –Fmin;
    if  Fmin>0  then  Razmax := Fmax;
    grDriver := Detect;
    InitGraph(grDriver,grMode, ' ');
    SetColor (cvet);
    SetBkColor (fon);
    xm := GetmaxX;
    ym := GetmaxY;
    x0 := Round (–xn / (xk–xn)*xm);   {x0 задает положение оси y, y0 задает
    положение оси x }
    y0 := Round (Fmax / Razmax*ym);
    if  Fmax<0  then  y0 := 0;
    Line (0, y0, xm, y0);   {вычерчивание оси x}
    Line (x0, 0, x0, ym);   {вычерчивание оси y }
    For  j := 0  to  n  do
       Begin
         xx := xn + j*dx;
         x := Round (j / n * xm);
         y := Round (y0 – F(xx) / Razmax * ym);
      { PutPixel (x, y, cvet); } {вывод графика в виде точек}
         if  j=0  then MoveTo (x, y)  else  LineTo (x, y);
         {вывод графика в виде ломаной линии}
       End;
    Readln;
    CloseGraph;
End.
```

8. КРАТКАЯ СПРАВКА

8.1 Процедуры ТурбоПаскаля

Append (var имя файла: text) – открывает существующий текстовый файл для присоединения.

Arc (координата X центра, координата Y центра : Integer; начальный угол, конечный угол: Word; радиус: Word) – рисует геометрический образ дуги.

Assign (var имя файла; внешнее имя: string) – назначает имя внешнего файла файловой переменной.

Bar (координата X левой верхней вершины, координата Y левой верхней вершины, координата X правой нижней вершины, координата Y правой нижней вершины: Integer) – рисует геометрический образ сплошного бруса.

Bar3D (координата X левой верхней вершины, координата Y левой верхней вершины, координата X правой нижней вершины, координата Y правой нижней вершины : Integer, глубина : Word; вершина : Boolean) – рисует геометрический образ трехмерного бруса. Параметр вершина может принимать два значения: TRUE (мнемоника константы – TopOn) – есть верхняя площадка, FALSE (мнемоника константы – TopOff) – нет верхней площадки.
Circle (координата X центра, координата Y центра : Integer; радиус : Word) – рисует геометрический образ окружности.
Close (var имя_файла) – закрывает открытый файл.
CloseGraph – завершает работу в графике.
ClrScr – очищает активное окно, заполняя его цветом, заданным в TextBackGround.
Dec (var переменная 1;переменная 2: Longint]) – уменьшает переменную 1 на значение переменной 2, если вторая переменная не задана, то уменьшает на 1.
Delay (длительность в миллисекундах: Word) – задерживает выполнение следующей операции.
Delete (var строка : string; номер символа, с которого начинается удаление: Integer; число удаляемых символов: Integer) – удаляет подстроку из строки.
DetectGraph (var драйвер, режим : Integer) – проверяет технику и определяет драйвер.
DrawPoly (число вершин: Word; var массив точек) – рисует геометрический образ контура из линий.
Ellipse (координата X центра, координата Y центра : Integer; начальный угол, конечный угол: Word; Xрадиус, Yрадиус: Word) – рисует геометрический образ эллиптической дуги.
Exec (полное имя, командная строка: string) – выполняет заданную программу.
Exit – осуществляет немедленный выход из текущего блока, если текущим блоком является программа, она завершается.
Ellipse (координата X центра, координата Y центра : Integer; Xрадиус, Yрадиус: Word) – рисует геометрический образ заполненного эллипса.
FillPoly (число вершин : Word; var массив точек) – рисует геометрический образ заполненного контура.
F1oodFill (координата X, координата Y, цвет границы : Word) – заполняет область.
GetDate (var год, месяц, день, день недели: word) – возвращает набор текущей даты в операционной системе.
GetTime (var часы, минуты, секунды, сотые: word) –возвращает время, установленное в операционной системе.
GoToXY (X,Y:Byte) – позиционирует курсор в текстовом режиме (внутри окна) в точку с координатами X,Y.
GraphDefaults – восстанавливает графическую систему по параметрам по умол-

чанию, позиционирует курсор.
Halt[(код выхода : word)] – останавливает программу.
Inc (var переменная 1 [;переменная 2: Longint]) –увеличивает переменную 1 на значение переменной 2, если вторая переменная не задана, то увеличивает на 1.
InitGraph (var номер драйвера : Integer; var режим графики Integer; путь к драйверу: string) – инициализирует графическую систему.
Insert (подстрока: string; var строка: string; позиция: Integer) вставляет подстроку в строку, начиная с заданной позиции.
Line(координата X начала линии, координата Y начала линии, координата X конца линии, координата Y конца линии : Integer) – рисует геометрический образ линии (из точки к точке).
LineRel (приращение по координате X, приращение по координате Y : Integer) – рисует геометрический образ линии (от указателя к точке в приращениях).
LineTo (координата X конечной точки, координата Y конечной точки: Integer) – рисует геометрический образ линии (от указателя к точке в координатах).
MoveRel (приращение по X, приращение по Y: Integer) – перемещает курсор на заданное приращение.
MoveTo (координата X, координата Y : Integer) – перемещает курсор в заданную точку.
OutTextXY (координата X точки выдачи текста, координата Y точки выдачи текста: Integer; текстовая строка: siring) – выдает строку, начиная с заданной точки.
PieSlice (координата X центра, координата Y центра : Integer; начальный угол, конечный угол: Word; радиус: Word) – рисует геометрический образ заполненного сектора.
PutPixel(координата X, координата Y: Integer; код цвета : word) – устанавливает пиксель в заданную точку.
Randomize – инициализирует встроенный генератор случайных чисел.
Read([var файл: text;] переменная1 [,переменная2,...,переменная n]) – читает одно или более значений в одну или более переменных.
Readln – выполняет процедуру Read, затем переходит к следующей строке файла.
Rectangle(координната X левой верхней вершины, координата Y левой верхней вершины, координата X правой нижней вершины, координата Y правой нижней вершины : Integer) – рисует геометрический образ прямоугольника.
Reset(var имя файла [:file; размер записи :word]) – открывает существующий файл для чтения.
RestoreCrtMode – восстанавливает текстовый режим.
Rewrite(var имя файла:file[;размер записи:word]) – создаст и открывает новый файл для записи.
Sector(координата X центра, координата Y центра : Integer; начальный угол, конечный угол, Xрадиус, Yрадиус : Word) – рисует геометрический образ заполненного эллиптического сектора.

SetBkColor(Цвет : word) – устанавливает текущий фоновый цвет.
SetCo1or(цвет : Word) – устанавливает текущий цвет.
SetGraphMode(режим : Integer) – устанавливает систему в графический режим и очищает экран, допустимый режим зависит от используемого оборудования.
SetLineStyle(тип линии: word; образ: word; толщина: word) – устанавливает текущую ширину и тип линии.
Str(число [:widlh[:decimals]]; var строка: string) – преобразовывает числовое значение в строку.
TextBackGround (цвет : byte) – выбирает фоновый цвет.
TextColor(цвет: byte) – выбирает цвет выдаваемых символов текста в текстовом режиме.
TextMode(режим : Integer) – задает режим выдачи текста.
Val(Строка:string; var переменная; var код: Integer) – превращает значение строки в его цифровое значение.
Window (координата x левого верхнего угла, координата y левого верхнего угла, координата x правого нижнего угла, координата y правого нижнего угла: byte) – определяет текстовое окно на экране.
Write([var файл: text;] переменная1, переменная2, ..., переменная n]) – записывает одно или более значений в файл, без необязательных параметров выдает на экран указанные значения .
Writeln – выполняет процедуру Write, затем выдаст маркер end–of–file (конец_файла) в файл, без необязательных параметров выдает на экран указанные значения с новой строки.

8.2 Функции ТурбоПаскаля

Abs (аргумент) : тот же тип, что и у параметра – возвращает абсолютное значение аргумента.
ArcTan (аргумент : real): real – возвращает арктангенс аргумента.
Chr(номер : Byte) : Char – возвращает символ с заданным порядковым номером.
Concat(строка1 [строка2, строка3, ..., строкаn]:string): string – осуществляет конкатенацию строк.
Copy (строка : string; номер_символа_начала_копирования: Integer; число_копиру–емых_символов: Integer): string–возвращает подстроку заданной строки.
Cos (аргумент: real): real – вычисляет косинус аргумента.
Eof (var имя_файла): Boolean – возвращает статус конца файла для объявленного в type файла.
Eoln [(var имя_файла: text)]: Boolean – возвращает статус конца строки текстового файла.
Exp(аргумент :real): real – возвращает экспоненту аргумента.
Frac(число : real): real – возвращает дробную часть числа.
GetGraphMode: Integer – выдает текущий режим.

GetMaxColor : word – выдает максимальный цвет в SetColor.
GetMaxMode: Integer – выдает максимальный номер режима.
GetMaxX : Integer – выдает разрешающую способность по X.
GetMaxY : Integer – выдает разрешающую способность по Y.
GetPixel(координата_X, координата_Y :Integer): word – выдает значение цвета пикселя в заданной точке.
GetX : Integer – выдает координату X текущего указателя.
GetY : Integer – выдает координату Y текущего указателя.
GraphResult : Integer – выдаст ошибочный код для последней графической операции.
Int (число: real): real – возвращает целую часть числа.
KeyPressed : Boolean – возвращает значение TRUE, если на ключевой панели нажата клавиша.
Ln (число: real): real – возвращает натуральный и логарифм аргумента.
Odd (аргумент : Longint) : Boolean – проверяет, является ли аргумент нечетным числом.
Ord(элемент): Longint – возвращает порядковый номер элемента порядкового типа.
Pi: real – возвращает значение числа пи.
Pos (подстрока :string; строка : string): Byte – ищет подстроку в строке и выдает номер позиции.
Random (верхний_предел : word) : тот же тип, что и у параметра – выдает случайное число в диапазоне от нуля до заданного параметра.
ReadKey : Char – считывает символ с ключевой панели.
Round (число: real):Longint – округляет вещественное число до целого.
Sin (аргумент: real): real – возвращает синус аргумента.
Sqr (аргумент): тип тот же, что и у аргумента – возвращает квадрат аргумента.
Sqrt (аргумент: rea1) :rea1 – возвращает квадратный корень аргумента.
Trunc(число: real):Longint – обрезает значение типа real до значения типа Integer.
UpCase(Символ: Char) :char – преобразовывает символ в символ верхнего регистра.

9. ВОПРОСЫ И ЗАДАНИЯ ДЛЯ КОНТРОЛЯ.

Глава 1

1. Из каких компонент состоит интегрированная инструментальная среда Турбо Паскаль?
2. Как запустить среду Турбо Паскаль?
3. Назовите пункты строки меню среды Турбо Паскаль.
4. Как выбрать команду из пункта меню с помощью клавиш на клавиатуре?
5. Как создать новый документ?
6. Как указать путь к нужной папке? Как сохранить файл в свою папку?
7. Как отправить программу на исполнение?
8. Как отключить окно среды Турбо Паскаль и посмотреть результат выполнения программы? Как вернуться в окно редактора?

9. Как можно задержать исполнение программы до нажатия клавиши ввода?

10. Как очистить экран? Изменить цвет фона? Цвет символов?

11. Как сохранить изменения в программе? Как сделать резервную копию программы?

12. Как выйти из среды Турбо Паскаль?

13. Как открыть текст программы из Вашей папки?

14. Как в программе переместиться на одно слово влево? Вправо?

15. Как перейти в начало строки? В конец строки?

16. Как попасть на первую строку? На последнюю?

17. Как переместиться на один экран вниз? Вверх?

18. Найдите и замените один идентификатор другим.

19. Как осуществить поиск нужного фрагмента но всему документу?

20. Что такое блок?

21. Покажите два способа выделения блока. Как снять–вернуть выделение?

22. Приведите два способа перемещения, копирования и удаления блока.

23. Как переключиться между открытыми окнами?

24. Как закрыть окно документа?

Глава 2, 3, 4

1. Вычислить высоту треугольника, опущенную на сторону a, по известным значениям длин его сторон a, b, c.

2. Определить координату середины отрезка (a, b), если $a = 0.5$, $b = 2$.

3. Вычислить объем цилиндра с радиусом основания r и высотой h.

4. Определить расстояние, пройденное физическим телом за время t, если тело движется с постоянным ускорением a и имеет в начальный момент времени скорость V_0

5. Определить время свободного падения физического тела с высоты H.

6. Вычислить площадь прямоугольного треугольника, а также по выбору пользователя:

а) длину гипотенузы по двум его катетам;

б) длину одного из его катетов по гипотенузе и второму катету.

7. Дано натуральное n. По выбору пользователя определить:

а) Сколько цифр в числе n?

б) Чему равна сумма его цифр?

в) Найти первую цифру числа n.

8. Определить, какая из двух точек – $M_1(x_1, y_1)$ или $M_2(x_2, y_2)$ – расположена ближе к началу координат. Вывести на экран дисплея координаты этой точки.

9. Определить, какая из двух фигур (круг или квадрат) имеет большую площадь. Известно, что сторона квадрата равна a, радиус круга r. Вывести на экран название и значение площади большей фигуры.

10. Определить, попадает ли точка M (X, Y) в круг радиусом r с центром в

точке (X_0, Y_0).

11. Сможет ли шар радиуса R пройти в ромбообразное отверстие со стороной P и острым углом Q?

12. Проверить, можно ли из четырех данных отрезков составить параллелограмм.

13. Вычислить множество значений функции $y = x^2 + b$ для x, изменяющихся от – 10 до 10 с шагом 2, при $b = 5$.

14. Вычислить k первых членов арифметической прогрессии, заданных рекуррентной формулой $a_{n+1} = a_n + 2$, где a_n – n–й член арифметической прогрессии.

15. Вычислить произведение m членов арифметической прогрессии, если известны значения первого члена a_1 и разность арифметической прогрессии h.

16. Сформировать последовательность, элементы которой вычисляются по формуле $a_n = \frac{n}{n+1}$, n=1,2,...,20.

17. Вычислить значение n! для n=7.

18. Не используя стандартные функции (за исключением Abs), вычислить с точностью Eps > 0: $y = e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + ... + \frac{x^n}{n!} + ...$ Считать, что требуемая точность достигнута, если очередное слагаемое по модулю меньше Eps, – все последующие слагаемые можно уже не учитывать.

19. Найти первую степень числа 3, превышающую данное целое число a.

20. Найти наибольшую степень числа 2, делящую данное целое число a.

21. Проверить, содержит ли квадрат данного натурального числа n цифру 3 в своей записи.

22. Найти наименьшее положительное число x, удовлетворяющее условию $1+x > 1$.

Глава 5

Обработать на ЭВМ массив в соответствии с вариантом задания.

Вариант	Массив	Действия	Условия и ограничения
1	2	3	4
1	X(100)	Вычислить сумму и количество элементов массива X.	$0 \le x[i] \le 1$
2	A(80)	Вычислить среднее арифметическое значение элемента массива A	$a\,[i] > 0$
3	X(70)	Переписать элементы массива X в массив Y и подсчитать их количество.	$-1\ x[i] \le 1$
4	B(50)	Определить максимальный элемент массива B и его порядковый номер.	$x\,[i] > 0$

Продолжение

1	2	3	4
5	C(40)	Вычислить минимальный элемент массива C и его номер.	$x\,[i] < 0$
6	D(80)	Найти максимальный и минимальный элементы массива D и поменять их местами.	
7	Y(20)	Вычислить среднее геометрическое элемента массива Y.	$y\,[i] > 0$
8	Z(30)	Расположить в массиве R сначала положительные, а затем отрицательные элементы массива Z.	
9	N(50)	Определить сумму элементов массива N, кратных трем.	$n[i]/3*3 = n[i]$
10	X(N)	Вычислить сумму и количество элементов массива X.	$N \le 40$

Задание Б.

Вариант	Матрица	Действия	Условия и ограничения
1	2	3	4
1	A(10,15)	Вычислить и запомнить сумму и число положительных элементов каждого столбца матрицы. Результаты отобразить в виде двух строк.	$a\,[i, j]>0$
2	A(N,M)	Вычислить и запомнить суммы и числа элементов каждой строки матрицы. Результаты отобразить в виде двух столбцов.	N <= 20 M <= 15
3	B(N,N)	Вычислить сумму и число элементов матрицы, находящихся под главной диагональю и над ней.	$\underline{N \le 12}$
4	C(N,N)	Вычислить сумму и число положительных элементов матрицы, находящихся над главной диагональю.	$\underline{c[i,j]>0\ N \le 12}$
5	D(K,K)	Записать на место отрицательных элементов матрицы нули и отобразить ее в общепринятом виде.	$K \le 10$
6	D(10,10)	Записать на место отрицательных элементов матрицы нули, а на место положительных – единицы. Отобразить нижнюю треугольную матрицу в общепринятом виде.	

Продолжение

1	2	3	4
7	F(N,M)	Найти в каждой строке матрицы максимальный и минимальный элементы и поместить их на место первого и последнего элемента строки соответственно. Матрицу вывести в общепринятом виде.	N ≤20 M ≤10
8	F(10,8)	Транспонировать матрицу и вывести на печать элементы главной диагонали и диагонали, расположенной под главной.	
9	N(10,10)	Для целочисленной матрицы найти для каждой строки число элементов, кратных пяти, и наибольший из полученных результатов.	n_{ij} /5*5 = n_{ij}
10	P(N,N)	Найти в каждой строке матрицы наибольший элемент и поменять его местами с элементом главной диагонали. Отпечатать полученную матрицу в общепринятом виде.	N ≤15

Глава 6

1. Описать функцию Stepen (*x*, *n*) от вещественного *x* и натурального *n*, вычисляющую (посредством умножения) величину x^n, и использовать ее для вычисления $b = 2.7^k + (a+1)^{-5}$.

2. Даны отрезки *a*, *b*, *c* и *d*. Для каждой тройки этих отрезков, из которых можно построить треугольник, напечатать площадь данного треугольника. Определить процедуру Plo (*x*, *y*, *z*), печатающую площадь треугольника со сторонами *x*, *y* и *z*, если такой треугольник существует.

3. Описать процедуру Socr (*a*, *b*, *p*, *q*) от целых параметров ($b \neq 0$), которая приводит дробь $\frac{a}{b}$ к несократимому виду $\frac{P}{q}$.

4. Пусть процедура Socr(a,b,p,q) от целых параметров (b≠0) приводит дробь $\frac{a}{b}$ к несократимому виду $\frac{P}{q}$. Описать данную процедуру и использовать ее для приведения дроби $1+\frac{1}{2}+\frac{1}{3}+...+\frac{1}{20}$ к несократимому виду $\frac{c}{d}$.

5. Пусть процедура maxmin (*x*, *y*) присваивает параметру *x* большее из вещественных чисел *x* и *y*, а параметру *y* – меньшее. Описать данную процедуру и использовать ее для перераспределения значений вещественных переменных *a*, *b* и с так, чтобы стало $a \geq b \geq c$.

6. Описать функцию $F(m,n)=\frac{n!m!}{(n+m)!}$, где n и m – неотрицательные целые числа.

Глава 7

1. Изобразить окружность диаметром d, перемещающуюся по вертикали через центр экрана.

2. Построить график функции $y=x*cos(x)+sin(x)$, для $x \in [-4,4]$ с шагом $h = 0,1$.

3. Изобразить квадрат со стороной a, перемещающийся по горизонтали на расстоянии 100 точек от начала координат.

4. Построить график функции $y = -6x^2 + 3x$.

5. Изобразить прямоугольник с длиной основания L и высотой H, перемещающийся по диагонали экрана.

СПИСОК ЛИТЕРАТУРЫ

1. Епанешников А.М., Епанешников В.А Программирование в среде Turbo Pascal 7.0. – 3–е изд.. стереотип. – М.: ДИАЛОГ МИФИ, 1998. – 282с.

2. Зуев Е.А. Программирование на языке Turbo Pascal 6.0,7.0. – М.: Веста: Радио и связь, 1993. – 304с.

3. Зуев Е.А. Turbo Pascal. Практическое программирование. – М.: Стрикс, 1997. – 334с.

4. Турбо Паскаль 7.0 – К.: Издательская группа BHV, 1996. – 448с.

5. Зубов В.С. Программирование на языке TURBO PASCAL (версии 6.0 и 7.0). Издание 2–е, переработанное и дополненное. – М.: Информационно – издательский дом «Филинъ», 1997. – 320с.

6. Довгаль С.И., Литвинов Б.Ю., Сбитнев А.И. Персональные ЭВМ: ТурбоПаскаль V 6.0, Объектное проектирование, Локальные сети. – Киев: «Информсистема сервис», 1993. – 440 с.

7. Основы электрической тяги, системы и режимы тяговых сетей постоянного тока: Учеб.пособие / Под ред. Омельяненко В.И. – Харьков: НТУ «ХПИ», 2002. – 164 с.

Навчальне видання

Методичні вказівки к практичним заняттям з курсу „Інформатика” (модуль „Алгоритмічна мова ТурбоПаскаль”) для студентів спеціальності 8.100501 „Рухомий склад і спеціальна техніка залізничного транспорту”

Російською мовою

Укладачі: **Веретельник** Юрій Вікторович
Сєриков Володимир Іванович
Пелешко Євген Віталійович

Відповідальний за випуск М. А. Ткачук
Роботу рекомендував до видання В. К. Бєлов

В авторській редакції

План 2007 р., поз. 50/

Підписано до друку __.__.__. Формат 60×84 1/16. Папір офсетний.
Друк – ризографія. Гарнітура Times New Roman. Ум. друк. арк. 3,5.
Обл. вид. арк. 4,0. Наклад 100 прим. Зам № . Ціна договірна.

Видавничий центр НТУ „ХПІ”, 61002 Харків, вул. Фрунзе, 21
Свідоцтво про державну реєстрацію ДК №116 від 10.07.2000 р.

Друкарня НТУ „ХПІ”, 61002 Харків, вул. Фрунзе, 21