Министерство образования и науки
Российской Федерации
Воронежский государственный университет

Программирование в среде Турбо Паскаль

Учебно-методическое пособие

Специальность 011200 – геофизика

Воронеж – 2004

Утверждено Научно-методическим советом геологического факультета (19.02.2004 г. протокол № 4)

Составители: Закутский С.Н., Силкин К.Ю.

Учебно-методическое пособие подготовлено на кафедре геофизики геологического факультета Воронежского государственного университета.

Рекомендуется для студентов 1 курса дневного отделения, обучающихся по специальности 011200 – геофизика.

СОДЕРЖАНИЕ

СРЕДА ТУРБО ПАСКАЛЬ

МИНИМАЛЬНЫЕ СВЕДЕНИЯ ДЛЯ РАБОТЫ С СИСТЕМОЙ ПРОГРАММИРОВАНИЯ

Алгоритмический язык Pascal (Паскаль) был разработан Никлаусом Виртом в 1971 году. Назван в честь Блеза Паскаля. Никлаус Вирт (Wirth) (г.р. 1923) – выдающийся ученый и педагог, с 1968 года профессор информатики Института информатики Швейцарской высшей политехнической школы, Цюрих, Швейцария.

Блез Паскаль (1623 – 1662), французский религиозный философ, писатель, математик и физик. Б. Паскаль изобрел в 1641 первую суммирующую машину

Интегрированная среда программирования Турбо-Паскаль (ИС ТП) представляет собой совокупность программ, обеспечивающих пользователю услуги по набору текстов программ, составленных на одноименном языке, их компилирование, отладку, запуск на выполнение. Ниже приводятся лишь минимальные сведения, необходимые в первую очередь начинающему пользователю для успешной работы с системой.

Чтобы попасть в интегрированную среду (загрузить среду ТП), можно воспользоваться одним из следующих способов:

- набрать в *командной строке маршрут* к файлу turbo.exe. Например, если этот файл расположен на диске D: в каталоге BIN каталога TP, то соответствующая команда будет иметь вид D:\TP\BIN\turbo.exe;
- войти в каталог, содержащий файл turbo.exe и, выделив его, пустить на выполнение;
- с помощью меню пользователя, если в нем предусмотрена команда входа в ТП;
- в рабочем каталоге выбрать файл с расширением .pas, если установлен соответствующий режим среды (например, NC), из которой производится загрузка ТП.

После запуска ТП управление передается программе, называемой *редактором* ТП. Он предназначен для набора и корректировки вводимых текстов (как правило, программ). Вид «распахивающегося» при этом *окна редактора* зависит от произведенного способа загрузки ИС. Если при загрузке указывалось имя какого-либо файла, то в рабочем поле окна редактора будет располагаться текст этого файла. Если файл не указывался, то поле будет пустым. В верхней части экрана располагается меню ТП, войти в него можно либо с помощью мыши, либо нажав клавишу ***F10***. В нижней части экрана приведены назначения некоторых функциональных клавиш.

Работа с редактором

Основные приемы работы с редактором при наборе текста программ (или иных документов) во многом аналогичны приемам, используемым во многих текстовых редакторах. Позиция на *экране*, обрабатываемая редактором по текущей команде, отмечается *курсором*. Перемещение по экрану осуществляется с помощью клавиш ←, →, ↑, ↓ (на одну позицию вправо-влево, на одну строку вверх-вниз); ***Home***[1], ***End***[2] (в *начало* и *конец строки*); ***PageUp***[3], ***PageDown***[4] – на одну страницу вверх или вниз по экрану ***Ctrl***[5]***+PageUp***, ***Ctrl+PageDown*** – к самому первому или самому последнему символу экрана.

Текст программы набирается построчно. Хотя длина стоки не ограничивается, на практике вряд ли целесообразно выходить за пределы экрана (78 символов). Для перехода на *новую строку* после набора очередной строки нажимается ***Enter***[6]. Для удаления ненужных или ошибочно набранных символов следует воспользоваться клавишами ***Delete***[7] (убирается *символ справа от курсора*) или ←***Backspace***[8] (убирается *символ слева от курсора*). *Режим вставки или замены* выбирается клавишей ***Insert***[9] (в режиме вставки набираемые символы раздвигают ранее набранные, а в режиме замены – затирают собой). Поместив курсор в средину какой-либо строки и нажав ***Ctrl+y***, можно убрать всю строку. Если это сделано ошибочно, то сразу же после уничтожения можно восстановить удаленный фрагмент командой ***Alt***[10]***+←Backspase***. Клавишами ***Enter*** и ***Delete*** выполняется разрезание и склеивание строк. Для работы с блоками текста используются следующие *комбинации клавиш*.

- ***Ctrl+k***, ***b*** – пометить начало блока;
- ***Ctrl+k***, ***k*** – пометить конец блока;
- ***Ctrl+k***, ***y*** – стереть блок;
- ***Ctrl+k***, ***c*** – копировать блок
- ***Ctrl+k***, ***w*** – записать блок в дисковый файл;
- ***Ctrl+k***, ***r*** – прочитать блок из файла;
- ***Ctrl+k***, ***p*** – напечатать блок.

После набора текста программы можно поступить двояко: либо приступить к его *компиляции* и *отладке*, либо (этот вариант более предпочтителен) сохранить текст в дисковом файле с расширением .pas. *Сохранение*

---

[1] Home [хоум] – начало.
[2] End [энд] – конец.
[3] Page up [пэйдж ап] – страница вверх.
[4] Page down [пэйдж даун] – страница вниз.
[5] Ctrl (control) [контрол] – управлять.
[6] Enter [энтэр] – ввод.
[7] Delete [дэлит] – удалить.
[8] Backspace [бэкспэйс] – возврат.
[9] Insert [инсэт] – втавить.
[10] Alt (alternate) [олт (олтёнит)] – замена.

текста на диске может быть выполнено либо с помощью функциональной клавиши ***F2***, либо через меню: ***File***[11]***/Save as***[12]. В обоих случаях откроется *диалоговое окно*, в котором следует указать имя, под которым сохраняется набранный текст программы.

П р и м е ч а н и е . Если текст набирается впервые, а имя файла не указывалось, то редактор предлагает *стандартное имя* NONAME00.PAS[13]. Что бы его изменить, следует воспользоваться рассмотренным приемом. При наборе имени не обязательно указывать расширение, т.к. среда подключит его к основному имени автоматически.

Если файл с текстом программы существовал на диске ранее, то после входа в среду ТР загрузить его для работы можно либо через меню (***File/Open***[14]), либо с помощью функциональной клавиши ***F3***.

Выход из среды ТП осуществляется либо через меню (***File/Exit***[15]), либо с помощью клавиш ***Alt+x***.

Компиляция

Исходный текст программы не может быть выполнен компьютером, т.к. для выполнения предусматриваемых программой действий следует текст с *символьного представления* перевести во *внутреннее представление* на язык машины. Для этого и используются специальные программы, называемые *компиляторами*. В среде ТП обратиться к ней можно по команде, связанной с функциональной клавишей ***F9***, либо через меню (***Compile***[16]***/Make***[17]).

П р и м е ч а н и е . В последующем изложении обозначение действия, связанного с функциональной клавишей, производится указанием самой клавиши или соответствующей комбинации клавиш; в круглых скобках будет указан вариант выполнения команды через пункт меню.

В ходе компиляции производится также *проверка синтаксиса* программы, под которым понимается соответствие его правилам языка программирования. При наличии *синтаксических ошибок* компьютер помещает курсор около того места, где обнаружена ошибка, в верхней части экрана выводится сообщение о характере ошибки и компиляция прерывается. Характер ошибки можно попытаться определить по её *коду*, обратившись к какому-либо справочному материалу, например, через *справочную службу* ТП (вызывается с помощью клавиши ***F1***).

Допустим, что при наборе программы какое-либо предложение не завершено точкой с запятой. Анализируя это место, компилятор выдаст

---

[11] File [файл] – файл.
[12] Save as [сэйв эз] – сохранить как.
[13] No name [но нэйм] – без имени.
[14] Open [оупэн] – открыть.
[15] Exit [эксит] – выход.
[16] Compile [компайл] – собирать.
[17] Make [мэйк] – создавать.

сообщение:

```
Error 85 : ";" expected
```[18]

После нажатия любой клавиши информация об ошибке исчезает и восстанавливается режим редактирования. Компиляцию следует продолжать до тех пор, пока все ошибки не будут исправлены. В этом случае компилятор выдаст сообщение о том, что компиляция закончилась успешно:

```
Compile successful : Press any key
```[19]

При этом будет создан новый вариант программы – *исполняемая программа*. Чтобы сохранить его на *диске*, следует воспользоваться командой меню ***Compile/Destination***[20] ***Disk***[21]. При этом на диске появится файл, основное имя которого совпадает с именем файла *исходного текста* программы, а расширение у него будет .exe, т.о. такой файл может быть пущен на выполнение непосредственно из того каталога, в который он помещен. Если же запись на диск не производится (команда меню ***Compile/Destination Memory***[22]), то для запуска программы на выполнение следует воспользоваться командой ***Ctrl+F9*** (***Run***[23]***/Run***). Заметим, что эта команда может использоваться и для компиляции, при этом, если компиляция завершается успешно, происходит автоматический переход к выполнению программы.

Во время работы программы открывается окно выполнения (*окно программы*), в которое помещаются результаты, вывод которых на экран предусмотрен в тексте программы. Чтобы посмотреть эти результаты, следует воспользоваться командой ***Alt+F5*** (***Debug***[24]***/User screen***[25]).Повторное нажатие указанной комбинации клавиш восстанавливает окно редактора. Чтобы окно программы отображалось на экране одновременно с окном редактора (а также с другими окнами), следует воспользоваться командой меню ***Debug/Output***[26]. Любое *активное окно* закрывается командой ***Alt+F3***. *Переход между окнами* обеспечивает команда ***F6***.

Отладка

Программа, успешно прошедшая компиляцию, может содержать *смысловые ошибки*, связанные с неверной организацией алгоритма решения задачи, ошибками в задании операций и др. причинами несинтаксического

---

[18] «Error 85: ":" expected» – «Ошибка 85: ":" ожидается».

[19] «Compile successful: Press any key» – Компиляция прошла успешно: нажмите любую клавишу»

[20] Destnation [дэстинэйшн] – назначение.

[21] Disk [диск] – диск.

[22] Memory [мэмори] – память.

[23] Run [ран] – запустить.

[24] Debug [дибаг] – отлаживать.

[25] User screen [юзэ скрин] – пользовательский экран.

[26] Output [аутпут] – вывод.

характера. Для выявления такого рода ошибок необходимо иметь вариант решения, полученный каким-либо иным и наверняка надежным способом. Проверка результата работы программы с помощью таких вариантов называется *тестированием*. Каждая программа должна быть протестирована.

Другой разновидностью ошибок, возникающих в успешно откомпилированной программе, являются так называемые *ошибки периода выполнения*. Чаще всего они бывают связаны с попыткой выполнения какой-либо операции с недопустимым для неё типом данных. Например, вычисление логарифма отрицательного числа, деления на нуль и др. При обнаружении такой ошибки в компьютере возникнет прерывание, а в окно редактора будет выдано сообщение следующего вида:

```
Error 207 : Invalid floating point operation
```[27]

Код ошибки и сообщение о ее характере, разумеется, зависит от самой ошибки.

Процесс поиска и исправления ошибок смыслового характера называется *отладкой программы*. Для облегчения процесса отладки в ТП имеются специальные средства, некоторые возможности которых рассматриваются ниже.

Установим курсор в той строке программы, с которой начинается участок, работу которого необходимо проверить, и выполним команду ***F4*** (***Run/Go to cursor***[28]). Программа начнет исполняться обычным образом, но остановится перед выполнением операций из строки, на которую указывает курсор. При этом сама строка будет выделена цветной полосой. Далее можно поступить двояко. Одна возможность продолжения отладки состоит в том, что курсор переводится на новую строку и аналогично предыдущему действию проверяется работа нового фрагмента программы. Другая возможность связана с использованием команд ***F7*** (***Run/ Trace into***[29]) или ***F8*** (***Run/Step over***[30]). По этим командам выполняются действия, предусмотренные в выделенной строке, после чего выполнение прекращается, а цветной указатель смещается на следующую строку, т.е. выполняется *пошаговая проверка* работы программы. Отличие команд состоит в том, что при наличии подпрограмм в первом случае (***F7***) пошаговая проверка осуществляется с заходом в подпрограммы, если таковые имеются, а во втором (***F8***) подпрограммы выполняются как один оператор.

В момент прерывания выполнения (в процессе отладки) можно просмотреть значения отдельных переменных. По команде ***Ctrl+F4*** (***Debug/Evaluate***[31]***/modIfy***[32]) на экране распахивается диалоговое окно. Имя переменной, значение которой необходимо узнать, набирается в поле

---

[27] «Error 207: Invalid floating point operation» – «Ошибка 207: недействительная операция над числом с плавающей точкой.

[28] Go to cursor [гоу ту кёсо] – перейти к курсору.

[29] Trace into [трэйс инту] – выполнить с заходом внутрь.

[30] Step over [степ оувэ] – перешагнуть через блок.

[31] Evaluate [ивэлйуэйт] – вычислить.

[32] Modify [модифай] – изменить.

Expression[33] и нажать ***Enter***. Значение указанной переменной появится в поле ***Result***[34]. Окно удаляется с экрана клавишей ***Esc***[35]. Активизировать рассматриваемое окно можно и иным способом: курсор устанавливается на имя переменной и выполняется команда ***Ctrl+F4***. В поле ***Expression*** можно указывать не только имена отдельных переменных, но и имена проверяемых выражений.

За изменением значений отдельных переменных или выражений можно следить в *окне отладчика*, открываемого по команде ***Ctrl+F7*** (***Debug/Add watch***[36]). Чтобы при этом можно было одновременно работать с окнами отладчика и редактора, следует изменить размеры последнего, воспользовавшись командой ***Ctrl+F5*** (***Size***[37]***/Move***[38]). После перехода в *режим изменения окна* его положение на экране выбирается с помощью клавиш ←, →, ↑, ↓, а размер – комбинацией указанных клавиш с ***Shift***[39]. Завершить режим изменения окна следует нажатием ***Enter***. Чтобы продолжить выполнение программы с одновременным просмотром изменения значений переменных или выражений, следует перейти в окно редактора (сделать его активным), нажав ***F6***. Для удаления ненужной переменной или выражения из окна отладчика, следует сделать это окно активным, выделить цветом соответствующую строку и нажать ***Ctrl+y***.

Закрытие активного окна производится командой ***Alt+F3*** (***Window***[40]***/Close***[41]). Выход из режима отладки осуществляется по команде ***Ctrl+F2*** (***Run/Program reset***[42]).

Перечень основных команд, используемых при отладке программ:

- ***F1*** – вызов справочной службы;
- ***F2*** – сохранить текст в дисковом файле;
- ***F3*** – загрузить текст из дискового файла;
- ***F4*** (***Run/Go to cursor***) –выполнить до строки, указанной курсором;
- ***F5*** – распахнуть окно на весь экран (свернуть окно);
- ***F6*** – сделать активным следующее окно;
- ***F7*** (***Run/Trace into***) – пошаговая проверка с заходом в подпрограммы;
- ***F8*** (***Run/Step over***) – пошаговая проверка без захода в подпрограммы;
- ***F9*** (***Compile/Make***) – компиляция программы без загрузки на выполнение;
- ***F10*** – вход в меню ТП;
- ***Ctrl+F2*** (***Run/Program reset***) – выход из режима отладки;

---

[33] Expression [икспрэшн] – выражение.
[34] Result [ризалт] – результат.
[35] Esc (escape) [иск (искэйп)] – отменить.
[36] Add watch [эд уоч] – добавить наблюдение.
[37] Size [сайз] – размер.
[38] Move [мув] – переместить.
[39] Shift [шифт] – сдвиг.
[40] Window [уиндоу] – окно.
[41] Close [клоуз] – закрыть.
[42] Program reset [проуграм рисет] – сбросить программу.

- ***Ctrl+F4*** (***Debug/Evaluate/modify***)
- ***Ctrl+F5*** (***Size/Move***) – вызов режима изменения размеров окна и его положения на экране;
- ***Ctrl+F7*** (***Debug/Add watch***) – добавление переменной в окно отладчика
- ***Ctrl+F9*** (***Run/Run***) – компиляция программы с последующим запуском её на выполнение;
- ***Alt+F3*** (***Window/Close***) – закрытие активного окна;
- ***Alt+F5*** (***Debug/User screen***) – сделать активным окно программы.

Упражнение

1. Загрузить среду TP.
2. Набрать в окне редактора следующий текст программы:

```
PROGRAM Firsr[43];
Const A = 'Результат выполнения первой программы';
Var b,c,d,e : real[44];
BEGIN[45]
WriteLn[46]('Ввести значения b и c');
ReadLn(b, c)
d:=b*b-4*c; e:=SgRt[47](d);
WriteLn(A);
WriteLn(d, e:6:3)
END.
```

3. Выполнить компиляцию и исправить все синтаксические ошибки.
4. Загрузить программу для выполнения и просмотреть содержимое окна программы.
5. Раздвинуть 6 и 7 строки и вставить оператор `Read`[48]`(b, c)`.
6. Вновь загрузить программу на выполнение, приняв для переменных `b` и `c` следующие значения `5` и `2`.
7. Просмотреть содержимое окна программы, обратив внимание на форму вывода значений переменных `d` и `e`.
8. Еще раз загрузить программу на выполнение, приняв для переменных `b` и `c` следующие значения `2` и `5`. Определить причину возникновения прерывания программы в период исполнения.
9. Выйти из среды TP, предварительно сохранив созданный компилятором машинный вариант программы на диске в Вашем каталоге.

---

[43] First [фёст] – первый.
[44] Real [риэл] – вещественный.
[45] Begin [бигин] – начинать.
[46] WriteLn (Write line) [райт лайн] – писать строку.
[47] SqRt (square root) [скуэа рут] – квадратный корень.
[48] Read [рид] – читать.

Контрольные вопросы

1. Что входит в интегрированную среду ТР и как вызвать ее для работы?
2. Какой вид имеет экран дисплея после загрузки ТР?
3. Какое назначение редактора ТР? Перечислите основные приёмы работы с редактором.
4. Перечислите основные этапы работы компилятора и их назначение.
5. Какие разновидности ошибок могут возникать в процессе отладки программы и какие основные приемы их локализации и устранения могут использоваться средствами ТР?
6. Как работать с окнами просмотра значений переменных и отладчика?
7. Перечислите назначение функциональных клавиш при работе редактора ТР.
8. Чем отличаются действия ТР по командам ***F9*** и ***Ctrl+F9***?
9. Как увидеть на экране результат работы программы?

ТЕМА 1. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЛИНЕЙНОЙ СТРУКТУРЫ

Цель работы – составление и отладка программ, содержащих операторы ввода-вывода и присваивания.

Любая программа на языке Турбо Паскаль состоит из *заголовка*, *раздела описаний* и *раздела операторов*. Заголовок программы может отсутствовать. Если же он присутствует, то оформляется по следующему правилу:

Program <имя программы> [параметры программы];

Здесь **program** – *служебное слово*, его присутствие обязательно; <имя программы> – правильный *идентификатор* (здесь и далее угловые скобки означают, что русский текст в них должен быть изменен на соответствующий английский); [параметры программы] – эти параметры определяют форму связи между программой и внешней средой (здесь квадратные скобки означают, что данный элемент может быть опущен). Если *имя программы* или список параметров опущены, то по умолчанию предполагается, что ввод будет осуществляться с клавиатуры, а вывод – на экран дисплея.

Раздел описаний (его присутствие обязательно) содержит характеристики объектов, используемых в разделе исполняемых операторов: *типов*, *констант*, *переменных*, *меток*, *подпрограмм* и т.д. Если какой-либо объект не указан в описательной части, на этапе компиляции будет сделано соответствующее сообщение, а процесс компиляции будет остановлен.

Раздел исполняемых операторов задает действия, выполнение которых приводит к выполнению работы, предусмотренной *алгоритмом* данной программы. Он начинается со служебного слова **begin**, за которым следует последовательность операторов, отделяемых друг от друга точкой с запятой. Раздел заканчивается служебным словом **end**, после которого обязательно ставится точка.

В простейших *программах линейной структуры*, рассмотренных в настоящей работе, для организации вычислительного процесса достаточно ограничиться операторами присваивания и обмена данными. *Оператор присваивания* в общем виде можно представить следующим образом:

<идентификатор>:=<выражение>;

Здесь <идентификатор> – имя объекта, получающего значение; <выражение> – правило, задающее порядок выполнения операций над операндами. Под *операндом* понимается объект, над которым совершается какое-либо действие. В качестве операндов в простейших случаях выступают имена простых переменных, константные значения или имена стандартных функций. Перечень наиболее употребляемых *стандартных функций* приведен в таблице 1. В качестве знаков *операций* при обработке объектов, получающих числовые значения, используются: + (плюс) – сложение; –

(минус) – вычитание; * (умножить) – умножение; / (делить) – деление; **div**[49] – делить нацело; **mod**[50] – вычислить остаток от деления.

Обмен данными наиболее часто реализуется с помощью *стандартных процедур* Read или ReadLn[51] – при вводе данных, Write или WriteLn – при выводе данных. В общем виде оператор, организующий *обмен данными*, можно представить следующим образом:

<Имя процедуры>(<список параметров>);

<Список параметров> при вводе это список имен переменных, разделенных запятыми; при выводе это список переменных или выражений, разделенных запятыми. Значения переменных при вводе с клавиатуры могут располагаться либо в строку, отделяясь, друг от друга пробелами, либо построчно. Подтверждение ввода достигается нажатием ***Enter*** в первом случае после набора всего списка, во втором – после набора каждого значения. Различие между Read и ReadLn поясним примером. Пусть вводятся значения трех простых переменных a, b, c. Пусть для этого используются два следующих друг за другом оператора ввода: ReadLn(a, b); Read(c); пусть все три значения чисел, например -3.4 5.1 -2.5, расположены в одной строке. В этом случае по первому оператору переменные a, b получат

Таблица 1. Некоторые математические (встроенные) функции, входящие в библиотеку TP

Обращение к функции	Тип аргумента	Тип результата	Пояснение
Abs(x)	*r, i*	*r, i*	Вычисление абсолютного значения
ArcTan(x)	*r*	*R*	Вычисление угла (в радианах) по его тангенсу
Cos(x)	*r*	*R*	Вычисление косинуса угла выраженного в радианах
Exp(x)	*r*	*R*	e^x
Frac(x)	*r*	*R*	Вычисление дробной части числа
Int(x)	*r*	*R*	Вычисление целой части числа
Ln(x)	*r*	*R*	Вычисление натурального логарифма
Pi	*r*	*R*	Число π
Sin(x)	*r*	*R*	Вычисление синуса угла выраженного в радианах
Sqr[52](x)	*r*	*R*	Возведение в квадрат
SqRt(x)	*r*	*R*	Вычисление корня квадратного

Здесь r *– вещественный тип;* i *– целый тип.*

[49] Div (divide) [дивайд] – делить.

[50] Mod (module) [модйул] – модуль.

[51] ReadLn (Read line) [рид лайн] – читать строку.

[52] Sqr (square) [скуэа] – квадрат.

значения -3.4 и 5.1. Число -2.5 игнорируется, а программа будет ожидать появления значения c, но в следующей строке экрана. Таким образом, оператор ReadLn в этом случае можно понимать как ввод значений переменных, перечисленных в списке ввода, и переход на новую строку экрана. Если бы оператор был записан в виде Read(a,b,c), то правильным был бы любой способ размещения значений на экране: в одной строке или построчно.

Различие между WriteLn и Write состоит в том, что по первому оператору после окончания процедуры обмена курсор будет переведен в следующую строку, а по второму переход на новую строку по окончании вывода не производится.

Для удобства размещения и чтения выводимых данных предусмотрена возможность представления значений в различной форме. Параметр в списке вывода может иметь один из трех видов: x, x:m, x:m:n, где x – переменная, m и n – целые положительные числа.

Приведем примеры вычисления некоторых математических функций, не входящих в библиотеку:

$$\arcsin x = \operatorname{arctg}\frac{x}{\sqrt{1-x^2}};\ \arccos x = \operatorname{arctg}\frac{\sqrt{1-x^2}}{x};\ \operatorname{tg} x = \frac{\sin x}{\cos x};\ \lg x = \frac{\ln(x)}{\ln(10)};$$

$$a^x = \exp\left[x\ln(a)\right].$$

В качестве примера ниже рассматривается пример программы, вычисляющей значение функции:

$$F(s,t) = e^{-pt}\,\frac{\cos(p\,s)-\sin(p\,s)}{s^2+t^2}.$$

Будем считать, что s и t – переменные, значения которых вводятся с клавиатуры, а a, b – константы, имеющие значения $a = 1$, $b = 15$.

Поскольку в выражении для $F(s,t)$ используется логарифмическая функция, определенная только при положительных значениях аргумента, в программе следует предусмотреть расчет значения аргумента этой функции и вывод на экран сообщения о возникновении аварийной остановки. В тексте программы содержатся пояснения отдельных участков, оформленные в виде комментариев – произвольное сообщение, заключенное в фигурные скобки. Наличие *комментариев* в программе не является обязательным, однако их наличие считается хорошим стилем оформления текста. Компилятор комментарии игнорирует.

```
Const a = 1; b = 15; {Описание констант}
Var F,s,t,Ar_Ln : real; {Описание переменных}
{Завершение описательной части и начало раздела исполняемых
операторов}
BEGIN
  WriteLn('Введите значения переменных s, t');
```

```
  ReadLn(s, t);
  Ar_Ln:=s-a/t;
{Следующие два оператора WriteLn выводят на экран значения
аргумента логарифмической функции и предупреждение о возможности
аврийного прерывания работы программы из-за несоответствия значения
аргумента характеру функции}
  WriteLn('аргумент функции логарифма ArLn = ', Ar_Ln);
  WriteLn('если Ar_Ln< = 0, произойдет аварийная остановка;');
  WriteLn('в этом случае следует ввод повторить заново');
{Далее вычисляется значение функции; в операторе присваивания
использованы библиотечные функции для вычисления логарифма,
экспоненты, косинуса и синуса}
  F:=Ln(Ar_Ln)+Exp(-pi*t)*(Cos(pi*s)-Sin(pi*s))/(s*s+t*t);
  WriteLn('s = ', s:5:2, ' t = ', t:5:2, ' F(s, t) = ', F:6:2)
END.
```

Задания

Вариант 1

Вычислить значения переменных *a* и *b*, пользуясь формулами, приведенными ниже. Значения *x*, *y*, *z* задавать при вводе. На экран вывести значения *a*, *b*, *x*, *y*, *z*.

$$a=\frac{\sqrt{|x-1|}-\sqrt[3]{|y|}}{1+\frac{x^2}{2}+\frac{y^2}{2}};\quad b=x\left[\operatorname{arctg}\left(\frac{z+1}{1+\frac{x^2}{2}+\frac{y^2}{2}}\right)+e^{-(x+2)}\right].$$

Вариант 2

Вычислить значения переменных *a* и *b*, пользуясь формулами, приведенными ниже. Значения *x*, *y*, *z* задавать при вводе. На экран вывести значения *a*, *b*, *x*, *y*, *z*.

$$a=\frac{3+e^{y}-1}{1+\frac{x^2|y-\operatorname{tg}z|}{1+|y-x|}};\quad b=1+|y-x|+\frac{(y-x)^2}{2}+\frac{(y-x)^3}{3}.$$

Вариант 3

Вычислить значения переменных *a* и *b*, пользуясь формулами, приведенными ниже. Значения *x*, *y*, *z* задавать при вводе. На экран вывести значения *a*, *b*, *x*, *y*, *z*.

$$a=(1+y)\frac{x+\dfrac{y}{\left(x^2+4\right)}}{e^{-x-2}+\dfrac{1}{\left(x^2+4\right)}};\quad b=\frac{\dfrac{1}{\left(x^2+4\right)}+\cos(y-2)}{\dfrac{x^4}{2}+\sin^2 z}.$$

Вариант 4

Вычислить значения переменных *a* и *b*, пользуясь формулами, приведенными ниже. Значения *x*, *y*, *z* задавать при вводе. На экран вывести значения *a*, *b*, *x*, *y*, *z*.

$$a=y+\frac{x}{y^2+\left|\dfrac{x^2}{y+\dfrac{x^3}{3}}\right|};\quad b=\left(1+\left|\frac{x^2}{y+\frac{x^3}{3}}\right|+\operatorname{tg}^2\frac{z}{2}\right)\frac{1}{y+x}.$$

Вариант 5

Вычислить значения переменных *a* и *b*, пользуясь формулами, приведенными ниже. Значения *x*, *y*, *z* задавать при вводе. На экран вывести значения *a*, *b*, *x*, *y*, *z*.

$$a=\frac{2\cos(x-p/6)}{\dfrac{1}{2}+\dfrac{\sin^2 y}{3+z^2/5}};\quad b=1+\frac{z^2}{3+z^2/5}-2\cos(x-p/6).$$

Вариант 6

Вычислить значения переменных *a* и *b*, пользуясь формулами, приведенными ниже. Значения *x*, *y*, *z* задавать при вводе. На экран вывести значения *a*, *b*, *x*, *y*, *z*.

$$a=\frac{1+\sin^2(x+y)}{2+\left|x-2x/\left(1+x^2+y^2\right)\right|};\quad b=\cos^2\left(\frac{1}{z}+\frac{2x}{1+x^2+y^2}\right).$$

Вариант 7

Вычислить значения переменных *a* и *b*, пользуясь формулами, приведенными ниже. Значения *x*, *y*, *z* задавать при вводе. На экран вывести значения *a*, *b*, *x*, *y*, *z*.

$$a=\frac{\sqrt{|x-1|}-\sqrt[3]{|y|}}{1+\dfrac{x^2}{2}+\dfrac{y^2}{2}};\quad b=1+|y-x|+\frac{(y-x)^2}{2}+\frac{(y-x)^3}{3}.$$

Вариант 8

Вычислить значения переменных a и b, пользуясь формулами, приведенными ниже. Значения x, y, z задавать при вводе. На экран вывести значения a, b, x, y, z.

$$a=(1+y)\frac{x+\sqrt{y/(x^2+4)}}{e^{-x-2}+\sqrt{1/(x^2+4)}};\quad b=\left(1+\left|\frac{x^2}{y+\frac{x^3}{3}}\right|+\operatorname{tg}^2\frac{z}{2}\right)\frac{1}{y+x}.$$

Контрольные вопросы

1. Из каких разделов состоит текст программы на языке Турбо Паскаль?
2. Что размещается в разделе описаний?
3. Как характеризуются переменные, типы, метки, константы?
4. Чем открывается и чем завершается раздел исполняемых операторов?
5. Как организовать обмен данными между внешними и внутренними устройствами?
6. Какие действия выполняются в операторе присваивания?
7. Что называют встроенными функциями?
8. Что называют комментариями и как они оформляются?

ТЕМА 2. ПРОГРАММИРОВАНИЕ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ

Цель работы – программирование вычислительных процессов, в которых на определенном этапе вычислений производится выбор очередного выполняемого оператора в результате анализа некоторых условий; освоение логического типа и условного оператора.

Алгоритмы и соответствующие им программы, в которых на определенном этапе решения задачи требуется проанализировать некоторое условие и в зависимости от получаемого результата направить вычислительный процесс по одному из альтернативных пути, называются *разветвляющимися*.

Оператор перехода по условию If

При создании разветвляющихся программ используется *условный оператор*. В общем виде этот оператор можно представить следующим образом:

```
If B
  Then C
  [Else D];
```

Здесь **if**[53] , **then**[54], **else**[55] – служебные слова. Квадратные скобки означают, что конструкция **else** D может быть опущена. B – выражение *логического типа*. В качестве значений таких выражений используется константа, обозначаемая одним из зарезервированных слов: **true**[56] или **false**[57]. C и D – операторы. Если в качестве операторов C и D предполагается использовать сложную конструкцию, состоящую из некоторой последовательности высказываний (в т.ч. и другого условного оператора), то следует воспользоваться *операторными скобками* **begin, end**, т.е. оформить такое сложное высказывание, как *составной оператор*. Действие составного оператора состоит в следующем. Если логическое выражение B получает значение **true**, то управление передается сначала оператору C, а далее оператору, следующему сразу же за данным условным оператором. Если логическое выражение B получает значение **false**, то управление будет передано либо оператору D, либо (при отсутствии части **else**) оператору, следующему сразу же за данным условным оператором.

Поясним понятие «логического выражения». Простейшим примером логического выражения является *логическая переменная*. В разделе описаний тип такой переменной определяется с помощью зарезервированного

---

[53] If [иф] – если.
[54] Then [зэн] – тогда.
[55] Else [элс] – иначе.
[56] True [тру] – истинный.
[57] False [фолс] – ложный.

слова **boolean**[58]. Более сложное логическое выражение может быть построено с помощью *операций отношения*: > – больше, < – меньше, >= – больше или равно (не меньше), <= – меньше или равно (не больше), = – равно, <> – не равно.

В самом общем виде логическое выражение строится из логических переменных и отношений с помощью специальных *логических операций*: **not**[59] – логическое отрицание; **and**[60] – логическое умножение (логическое и иначе конъюнкция); **or**[61] – логическое сложение (логическое или иначе дизъюнкция); **xor**[62] – исключающее или. Эти операции применимы только к *логическим операндам*, результат их выполнения определяется по так называемой таблице истинности. При составлении логических выражений и вычислении их значений следует учитывать приоритет логических операций. В первую очередь выполняется отрицание (**not**), затем логическое умножение (**and**), далее логическое сложение и исключающее или (**or, xor**). Операции отношения по сравнению с логическими и арифметическими операциями имеют наиболее низкий приоритет. Для изменения порядка выполнения операций используются круглые скобки.

Таблица 2. Таблица истинности

	Операнд A			
	true	**true**	**false**	**false**
	Операнд B			
	true	**false**	**true**	**false**
not A	**false**	**false**	**true**	**true**
A **and** B	**true**	**false**	**false**	**false**
A **or** B	**true**	**true**	**true**	**false**
A **xor** B	**false**	**false**	**false**	**true**

Оператор безусловного перехода GoTo

При организации ветвлений может оказаться целесообразным использование совместно с условным оператором *оператора безусловного перехода* **GoTo**[63] <метка>. <Метка> в ТП – это произвольный идентификатор, позволяющий именовать некоторый оператор программы, обеспечивая, таким образом, ссылку на него. В качестве меток могут быть использованы также целые числа без знака.

В тексте программы *метка* располагается непосредственно перед помечаемым оператором и отделяется от него двоеточием. Один и тот же оператор может быть помечен несколькими метками, которые в этом случае разделяются друг от друга двоеточиями. В описательной части программы каждая метка должна быть описана. Описание меток начинается служебным словом **Label**[64], за которым следует список меток:

---

[58] Boolean [булин] – булевский (логический).
[59] Not [нот] – не.
[60] And [энд] – и.
[61] Or [о] – или.
[62] Xor (eXclusive OR) [иксо] – исключающее или.
[63] Go to [гоу ту] – перейти к.
[64] Label [лэйбл] – метка.

```
Label Loop[65], Lb1, Lb2;
BEGIN
 {Текст программы}
GoTo Lb1;
 {Текст программы}
Loop:
 {Текст программы}
Lb1: Lb2:
 {Текст программы}
GoTo Lb2;
 {Текст программы}
END.
```

В качестве примера организации ветвлений ниже приводится программа для вычисления функции:

$$U = \frac{\max^2\{x, y, z\} - 2^x \min\{x, y, z\}}{\sin(2) + \max\{x, y, z\} / \min\{x, y, z\}};$$

Значения переменных *x*, *y*, *z* задаются при вводе. При вычислении рассматриваемой функции может возникнуть аномальная ситуация, если минимальное значение из *x*, *y*, *z* окажется равным нулю. В этом случае следует вывести на экран сообщение о невозможности вычисления и передать управление в какую-либо точку программы, например, на ввод исходных данных. Условимся также, что завершить работу программы следует после ввода соответствующего сообщения. Для этого после вычисления функции и вывода требуемых значений следует вывести на экран запрос, ответ на который либо подтверждает необходимость окончания вычислений (например, ***Yes***[66]), либо отрицает (***No***)[67]. Организовать в программе такой ответ можно с помощью какой-либо переменной символьного типа, получающей в качестве значения Y или N.

```
{Пример программы, использующей условный оператор и оператор
безусловного перехода. В программе реализован алгоритм для
вычисления функции:
U = [max(x, y, z)*max(x, y, z)-2^x*min(x, y, z)]/[sin(2)+max(x, y,
z)/min(x, y, z)], где значения x, y, z задаются при вводе. На экран
выводятся значения x, y, z, u}
{----- Описательная часть программы -----}
Label L1;
Var u,x,y,z : real; {функции и текущие значения аргументов}
    max,min : real; {промежуточные переменные}
    a       : char;
```

[65] Loop [луп] – петля.

[66] Yes [йес] – да.

[67] No [ноу] – нет.

```
{----- Исполнительная часть программы -----}
BEGIN
L1: WriteLn; {подготовка строки экрана для ввода сообщения}
Write('x, y, z'); {вывод сообщения о вводе исходной информации}
Read(x, y, z); {ввод исходной информации; при вводе значения
переменных набирать в строку через пробел}
If (x > y) and (x > z)
  Then max:=x {нахождение max(x, y, z)}
  Else If y > z
    Then max:=y
    Else max:=z;
If (x < y) and (x < z)
  Then min:=x {нахождение min(x, y, z)}
  Else If y < z
    Then min:=y
    Else min:=z;
If min  =  0
  Then begin {блокировка аномальной ситуации}
    WriteLn('деление на ноль, повторите ввод');
    GoTo L1;
  End;
u:=exp(x*ln(2)); {вычисление 2^x}
u:=(max*max-u*min)/(sin(2.0)+max/min); {вычисление u}
{----- Организация вывода значений аргументов и функции -----}
WriteLn('x = ', x:6:2, ' y = ', y:6:2, ' z = ', z:6:2, ' u = ',
u:6:2);
WriteLn('закончить вычисления – Y, продолжить - N');
Read(a);
If a  =  'N'
  Then GoTo L1;
ReadLn; {выход из программы в среду, из которой она запускалась на
выполнение}
END. {конец исполнительной части и программы в целом}
```

Задания

Вариант 1

Составить блок-схему и программу для нахождения значений функции, заданной условием:

$$y=\begin{cases} 1\Big/ x+\dfrac{\sin x}{x^2+1} & \text{при } x<0; \\ \left(x^2-1\right)\dfrac{\sin x}{x^{2+1}} & \text{при } x\geq 0. \end{cases}$$

Здесь x – больший корень квадратного уравнения: $ax^2+bx+c = 0$. Значения a, b, c задаются при вводе. На экран вывести значения a, b, c, x, y.

Вариант 2

Составить блок-схему и программу для нахождения значений функции, зависящей от нескольких аргументов. Значения этих аргументов требуется задавать при вводе.

$$u = \begin{cases} \sqrt{|a+b|}, & \text{если } a^2 - b^2 > 1; \\ a+b, & \text{если } a^2 - b^2 \le 1. \end{cases}$$

Здесь

$$a = \frac{4-x^2}{\sqrt{|x|}-\sqrt{|y|}} \sin\frac{x}{y}, \quad \text{если } \sqrt{|x|}-\sqrt{|y|} \neq 0;$$

$$b = \ln(x^2 - y^2) + \sqrt{\left|\frac{x}{y}\right|}, \quad \text{если } x^2 - y^2 > 0.$$

На экран вывести значения a, b, x, y, u.

Вариант 3

Составить блок-схему и программу для нахождения значений функции:

$$v = \sin\left(d\sqrt[5]{x^2+4}\right)e^{2x}$$

,

где x – меньший корень уравнения $ax^2+bx+c = 0$.

Значения a, b, c задаются при вводе. На экран дисплея вывести a, b, c, x, v.

Вариант 4

Составить блок-схему и программу для нахождения значений функции:

$$Q = \frac{\max\left(z, \frac{x_1}{x_2}\right)}{\min\left(z^2+4, x_1 \cdot x_2\right)} \operatorname{arctg}\left(\frac{x_1}{x_2}\right)$$

.

Здесь x_1 – больший, x_2 – меньший из корней уравнения $0.5x^2 - 10x+r = 0$. Значение k определяется при вводе.

$$z = \begin{cases} 0.52k & \text{при } k < 4; \\ 0.85k & \text{при } k \ge 4. \end{cases}$$

На экран вывести значения k, z, x_1, x_2, Q.

Вариант 5

Составить блок- схему и программу для нахождения значений функции:

$$u = \min(\max(a,b),c).$$

Здесь

$$a = \ln\left(\sqrt{e^{xy}} + 2^{|x|}\right); \quad b = \sin(x-2) + \cos^2(y-1); \quad c = \frac{2\sin\left(x^2 - by\right)}{1 + x^2 - 2x} \quad \text{при } x \neq 1.$$

Значения *x, y* задаются при вводе. На экран вывести значения *x, a, b, c, u.*

Вариант 6

Составить блок-схему и программу для вычисления значений функции:

$$p = \begin{cases} 1, & \text{если } x = 0 \quad \text{или } x = 1; \\ 2x, & \text{если } x = 2; \\ x^2, & \text{если } x = 3; \\ 2x^3, & \text{если } x = 4. \end{cases}$$

Здесь x – больший корень уравнения $ax^2 – 9x + c = 0$. Значения a и c задавать при вводе. На экран дисплея вывести значения x и p.

Вариант 7

Составить блок- схему и программу для нахождения значений функции:

$$f = \begin{cases} \dfrac{1}{\sqrt{d}} \operatorname{arctg} \dfrac{b+c}{\sqrt{d}}, & \text{если } d > 0; \\ -\dfrac{1}{b+c}, & \text{если } d = 0; \\ \dfrac{1}{2\sqrt{-d}} \ln \sqrt{-d}, & \text{если } d < 0. \end{cases}$$

Здесь $d = ac – d$, где

$$a = \max\left(\frac{x^2 + y^2}{x}, \min(x, y)\right), \quad b = \min\left(\sqrt{x^2 + y^2}, \frac{x}{y}\right)$$

при условии, что ни один из параметров задачи $a.b$ не равны нулю.

Значения *x, y, c* задаются при вводе. На экран дисплея вывести значения *x, y, c* и *f.*

Вариант 8

Составить блок-схему и программу для вычисления функции:

$$y=\begin{cases} x+\dfrac{\cos(x)}{x^2-1} & \text{при } x<0; \\ \dfrac{(x^2+1)\cos(x)}{x^2} & \text{при } x\geq 0. \end{cases},$$

где x – меньший корень уравнения $ax^2+bx+c = 0$. Значения a, b, c задавать при вводе. На экран дисплея вывести значения a, b, c, x, y.

Контрольные вопросы

1. К какому типу относится значение логического выражения?
2. Из каких операндов состоит логическое выражение?
3. Какие операции используются при вычислении отношений?
4. Какие операции относятся к разряду логических?
5. Как определяется приоритет операций в логическом выражении?
6. Как оформляется условный оператор и как он действует?
7. Для чего используется оператор безусловного перехода?
8. Что можно использовать в качестве меток и как они определяются в тексте программы?

ТЕМА 3. ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ЦИКЛИЧЕСКОЙ СТРУКТУРЫ

Цель работы – составление и отладка программ, содержащих операторы повторения и оператор выбора.

Циклическими называют такие алгоритмы, в которых предусматривается многократное повторение одних и тех же действий при различных значениях некоторой переменной, называемой *счетчиком цикла*. Примером вычислительных процессов циклической структуры являются задачи по накоплению сумм:

$$s = \sum_{i=n}^{N} a_i = a_n + a_{n+1} + \cdots + a_N.$$

Здесь a_i – общий член ряда, являющийся некоторой функцией параметра i, определяющего номер очередного слагаемого ряда; n –начальное значение номера члена ряда (нижний предел суммирования), N – его конечное значение (верхний предел суммирования). Ряд, для которого верхний предел заранее не известен, называют *итерационным*. Вычисление суммы такого ряда возможно лишь в том случае, если сам ряд относится к сходящимся. Для сходящихся рядов сумма может быть рассчитана с определенной заранее задаваемой точностью, при этом количество слагаемых (итераций) заранее неизвестно.

Алгоритм накопления суммы может быть охарактеризован следующим образом. В начале вычислений переменной, обозначающей искомую сумму, присваивается некоторое начальное значение, например, ноль или значение первого слагаемого ряда. Ячейка памяти, в которой будет накапливаться сумма, иногда называют *аккумулятором*. Далее циклически производится переопределение значения суммы путем прибавления к ранее накопленному значению очередного слагаемого. Таким образом, в аккумулятор при каждой итерации помещается новое число, определяющее очередное значение суммы. При этом, разумеется, необходимо каждый раз переопределять и номер слагаемого (номер итерации). Накопление завершается либо при достижении номером очередного слагаемого (счетчика цикла) предельного значения N, либо при достижении заранее задаваемой точности в случае итерационного ряда.

Как правило, общий член ряда можно отнести к одной из следующих разновидностей. Примеры первой разновидности:

$$\frac{x^i}{i!}; \quad \frac{(-1)^i x^{2i-1}}{(2i-1)!}; \quad \frac{x^{2i}}{(2i)!}.$$

Для вычисления очередного слагаемого в подобных случаях целесообразно использовать *рекуррентное соотношение,* при котором новое слагаемое вычисляется на основе значения предыдущего члена ряда. Напри-

мер, если общий член ряда задан в виде $x^i/i!$, то, очевидно, справедливо соотношение:

$$a_{i+1} = \frac{x^{i+1}}{(i+1)!} = \frac{x^i}{i!}\frac{x}{i+1} = a_i \frac{x}{i+1}.$$

Таким образом, значение очередного слагаемого вычисляется через значение предыдущего члена ряда.

Примеры второй разновидности:

$$\frac{\cos(nx)}{n}; \quad \frac{\sin(2n-1)x}{2n-1}; \quad \frac{\cos(2x)}{4n^2-1}.$$

В таких случаях очередное слагаемое при накоплении суммы целесообразно вычислять по общей формуле.

Примеры третьей разновидности:

$$\frac{x^{4n+1}}{4n+1}; \quad \frac{(-1)^n \cos(nx)}{n!}; \quad \frac{(n^2+1)(x/2)^n}{n!}.$$

В подобных случаях рекомендуется выражение общего члена разбить на две части: $a_n = b_n \cdot c_n$. При этом один из сомножителей (например, b_n) рекомендуется вычислять по рекуррентной формуле, а второй (c_n) по общей. Пример: общий член ряда представлен соотношением:

$$a_n = \frac{(-1)^n \cos(nx)}{n!}.$$

Представим его в виде $a_n = b_n \cdot c_n$, где $b_n = (-1)^n/n!$, $c_n = \cos(nx)$. Для первого сомножителя целесообразно использовать рекуррентное соотношение, а для второго – общее.

При организации вычислений в алгоритмах циклической структуры может использоваться один из операторов повторения. В языке Турбо Паскаль имеются три различных оператора, используя которые можно запрограммировать повторяющие фрагменты программы.

Счетный оператор цикла **For**

Счетный оператор цикла **For** имеет следующую структуру:

```
For <I>:=<m> to <n> do
  <оператор>;
```

Здесь **for**[68], **to**[69], **do**[70] – служебные слова; `I` – параметр цикла – простая переменная любого порядкового типа (наиболее часто используется один из целых типов: `integer`[71], `byte`[72] или `word`[73]); `m`, `n` – начальное и

[68] For [фо] – для.

[69] To [ту] – к.

[70] Do [ду] – делать.

[71] Integer [интиджэ] – целочисленная переменная размером 2 байта, со знаком.

конечное значения счетчика цикла: выражения того же типа, что и переменная цикла; <оператор> – произвольный оператор ТП, например, составной. При выполнении этого оператора сначала вычисляется начальное значение счетчика цикла: I:=m. После этого циклически повторяются следующие действия. Проверяется условие $I \le n$, если оно не выполнено, то цикл завершается и управление передается оператору, стоящему непосредственно за данным циклом, иначе выполняется оператор, стоящий после **do**. Переменная цикла увеличивается на единицу, и цикл повторяется заново.

Данный оператор может применяться и в ином виде:

```
For <I>:=<m> downto <n> do
  <оператор>;
```

Замена служебного слова **to** на **downto**[74] означает, что при каждом шаге счетчик цикла должен изменяться на –1. Очевидно, эта модификация будет иметь смысл при $m > n$, а условие выхода из цикла $I = n$.

Оператор цикла с предусловием While

Оператор цикла **While** – *оператор цикла с предпроверкой условия*:

```
While <условие> do
  <оператор>;
```

Здесь **while**[75], **do** – служебные слова; <условие> – выражение логического типа; <оператор> – произвольный оператор ТП, например, составной. Действие оператора: если <условие> получает значение **true**, то выполняется <оператор>, после чего вычисляется значение <условия> и его проверка. Если же <условие> получает значение **false**, то управление передается оператору, стоящему непосредственно за данным циклом.

Оператор цикла с постусловием Repeat–Until

Оператор цикла **Repeat-Until** – *оператор цикла с постпроверкой условия*:

```
Repeat
  <тело цикла>
Until <условие>;
```

Здесь **repeat**[76], **until**[77] – служебные слова; <условие> – выражение логического типа; <тело цикла> – произвольная последовательность операторов ТП. Действие оператора: Последовательность операторов, образующих тело цикла, выполняется хотя бы один раз. После этого вычисляется значение <условия>: если оно равно **false**, то операторы, образующие

---

[72] Byte [байт] – целочисленный тип размером 1 байт, без знака.

[73] Word [уод] – «слово», целочисленный тип размером 2 байта, без знака.

[74] Downto [даунту] – вниз.

[75] While [уайл] – пока.

[76] Repeat [рипит] – повторять.

[77] Until [антил] – до тех пор пока не.

тело цикла, повторяются, в противном случае управление передается оператору, стоящему непосредственно за данным циклом.

Оператор выбора Case

В дополнении к рассмотренным операторам при организации вычислений, в которых предусматривается несколько альтернативных вариантов действий, может оказаться полезным *оператор выбор а* . Этот оператор позволяет выбрать одно из нескольких возможных продолжений программы. Параметр, по которому осуществляется выбор направления алгоритма, называется *ключом выбора.* В качестве последнего допускается использование объектов любого порядкового типа. Структура оператора выбора такова:

```
Case <ключ выбора> of
  <список выбора>
[Else <операторы>]
End;
```

Здесь **case**[78], **of**[79], **else**, **end** – служебные слова; <ключ выбора> – ключ выбора; <операторы> – произвольные операторa Турбо Паскаля; <список выбора> – одна или более конструкций следующего вида:

<константа выбора> : <оператор>;

Здесь <константа выбора> – константа того же типа, что и выражение, использованное для <ключа выбора>. Квадратные скобки означают, что секция **else** может отсутствовать.

Оператор выбора действует следующим образом. Вначале вычисляется выражение, определяющее <ключа выбора>. Затем в последовательности операторов <список выбора> отыскивается такой, которому предшествует константа, значение которой равно вычисленному значению ключа выбора. Найденный оператор выполняется, после чего оператор выбора завершается, и управление передается оператору, следующему непосредственно за данным оператором выбора. Если в списке выбора не окажется константы, значение которой равно значению ключа выбора, управление передается секции **else** (если она присутствует) либо оператору, следующему непосредственно за данным оператором выбора.

В качестве примера, иллюстрирующего использование рассмотренных операторов, ниже приведена программа, в которой решается следующая задача. Требуется вычислить функцию, параметром которой является сумма одного из итерационных рядов:

$$y = \begin{cases} a^s + \cos(a) & \textit{при } s > 0; \\ a^s + \sin(a) & \textit{при } s \le 0. \end{cases}$$

Здесь

---

[78] Case [кэйс] – вариант.

[79] Of [оф] – из.

$$s=\begin{cases}1+\ln(a)x/1!+\ln^2(a)x^2/2!+\ldots+\ln^n(a)x^n/n!+\ldots & \text{при } x\le 0.5;\\ x/1!-x^3/3!+x^5/5!+\ldots+(-1)^n x^n/(2n+1)!+\ldots & \text{при } 0.5<x\le 1.\end{cases}$$

Начальное и конечное значения *x*, шаг изменения Δx, значение *a* и погрешность вычисления требуется задавать при вводе. На экран вывести значения *y*, *x* и количество итераций при вычислении сумм.

```
Var n,c : byte; {n-количество повторений по x; c-ключ выбора для
вычисления y: c = 1, если s<0; c = 2, если s> = 0}
    i,j : word; {параметры циклов}
    x,dx,xmax,a,b,eps,s,y : real;
BEGIN
WriteLn; Write('x, xmax, dx, a, eps = ');
Read(x, xmax, dx, a, eps);
n:=trunc((xmax-x)/dx+1); {определение количества шагов по x; для
преобразования типов использована функция trunc}
For i:=1 to n do {цикл по x}
Begin
  If x < =  0.5
  Then begin {вычисление s при x< = o.5}
    s:=1; b:=ln(a)*x; j:=1;
    While Abs(b/s) > eps do
    Begin
      s:=s+b;
      b:=b*ln(a)*x/j;
      j:=j+1;
    End
  End
  Else begin {вычисление s при x>0.5}
    s:=0; b:=x; j:=1;
    Repeat
      s:=s+b;
      b:=(-1)*x*x/2.0/j/(2.0*j+1);
      j:=j+1;
    Until Abs(b) < eps
    End;
    If s < 0 {вычисление y и вывод результатов}
      Then c:=1
      Else c:=2;
    Case c of
      1 : y:=1/exp(s*ln(a))+sin(s);
      2 : y:=exp(s*ln(a))+cos(a);
    End;
    WriteLn; WriteLn('y = ', y:10:3, ' x = ', x, ' j = ', j);
    x:=x+dx;
  End;
```

```
WriteLn('конец задания'); ReadLn;
END.
```

Задания

Вариант 1

Составить блок-схему и программу для вычисления функции:

$$y=\begin{cases}\dfrac{s}{2}-\ln\left|2\sin\dfrac{s}{2}\right| & \text{при } s\le 0;\\ s^3+\lg\dfrac{s}{3} & \text{при } s>0,\end{cases}$$

где

$$s=\begin{cases}\cos x+\dfrac{\cos 2x}{2}+\ldots\dfrac{\cos nx}{n}+\ldots & \text{при } \dfrac{4p}{5}\le x\le\dfrac{9p}{5};\\ \sin x-\dfrac{\sin 2x}{2}+\ldots+(-1)^{n+1}\dfrac{\sin nx}{n}+\ldots & \text{при } \dfrac{p}{5}\le x\le\dfrac{4p}{5}\end{cases}$$

Шаг изменения x (Δx) задавать при вводе. На экран вывести значения y, s, x и число членов суммирования.

Вариант 2

Составить блок-схему и программу для вычисления функции

$$y=\begin{cases}e^{s\cdot a}\cos(s\sqrt{\left|1-a^2\right|} & \text{при } s<0;\\ \ln\left|s\cdot a\right|-\arcsin\left(\dfrac{s}{\sqrt{\left|1-a^2\right|}}\right) & \text{при } s\ge 0.\end{cases}$$

Здесь

$$s=\begin{cases}1+\dfrac{\cos\dfrac{p}{4}}{1!}x+\ldots\dfrac{\cos\left(n\dfrac{p}{4}\right)}{n!}+\ldots & \text{при } 0.1\le x\le 0.5;\\ 1-\dfrac{x^2}{2!}+\ldots+(-1)^n\dfrac{x^{2n}}{2n!}+\ldots & \text{при } 0.5<x\le 1.\end{cases}$$

Значение a заключено в интервале 0…5 и задается при вводе; шаг изменения x (Δx) задавать при вводе. На экран вывести значения y, x, s, а также число итераций.

Вариант 3

Составить блок-схему и программу для вычисления функции

$$y=\begin{cases}\dfrac{1}{4}\ln\left|\dfrac{1+s}{1-s}\right|+\dfrac{1}{2}\cos s & \text{при } 0.1\le s\le 0.9;\\ e^{\cos(s)}\cos\left(\operatorname{arctg}(s)\right) & \text{при } 0.9<s.\end{cases}$$

Здесь

$$s=\begin{cases}x+\dfrac{x^2}{5}+\ldots+\dfrac{x^{4n+1}}{4n+1}+\ldots & \text{при } 0.5\le x\le 0.8;\\ 1+\dfrac{\cos x}{1!}+\ldots+\dfrac{\cos nx}{n!}+\ldots & \text{при } 0\le x<0.5.\end{cases}$$

Значение Δx задавать при вводе. На экран вывести *y*, *x*, *s* и число итераций.

Вариант 4

Составить блок-схему и программу для вычисления функции

$$y=\begin{cases}(1+a\cdot s^2)e^s & \textit{при } s\le 0;\\ \dfrac{1}{2}\ln\left|1-a\cdot s\cos b+s^2\right| & \textit{при } s>0.\end{cases}$$

Здесь

$$s=\begin{cases}1+\dfrac{3x^2}{1!}+\ldots+\dfrac{2n+1}{n!}x^{2n}+\ldots & \text{при } 0.1\le x\le 0.5;\\ \dfrac{x\cos\dfrac{p}{3}}{1}+\dfrac{x^2\cos\dfrac{2p}{3}}{2}+\ldots+\dfrac{x^n\cos\dfrac{np}{3}}{n}+\ldots & \text{при } 0.5<x\le 1.\end{cases}$$

Значение *a* заключены в интервале от –3 до 5, *b* может принимать значения: {-3, -2, 0, 1, 3}. Конкретные значения *a* и *b*, а также шаг изменения *x* следует задавать при вводе. На экран вывести: в первой строке *a* и *b*, заданные при вводе. Во второй и последующих строках: *y*, *x*, *s*, число итераций.

Вариант 5

Составить блок-схему и программу для вычисления функции:

$$y=\begin{cases}1/2\ln S & \text{при } S>0;\\ 1/4(S^2-p^2/3) & \text{при } S\le 0,\end{cases}$$

где

$$S=\begin{cases}\dfrac{x-1}{x+1}+\dfrac{1}{3}\left(\dfrac{x-1}{x+1}\right)^3+\ldots+\dfrac{1}{2n-1}\left(\dfrac{x-1}{x+1}\right)^{2n-1}+\ldots & \text{при } 0.5\le x\le 1;\\ \dfrac{x^3}{3}-\dfrac{x^5}{15}+\ldots+(-1)^{n+1}\dfrac{x^{2n+1}}{4n^2-1}+\ldots & \text{при } 0\le x<0.5.\end{cases}$$

Шаг изменения x задавать при вводе. На экран вывести значения: y, S, x, число итераций.

Вариант 6

Составить блок-схему и программу для вычисления функции:

$$y=\begin{cases}\dfrac{S\cos a-S^2}{1-2S\cos a+S^2} & \text{при } S<0;\\ a-b & \text{при } S=0;\\ \dfrac{S(b-S)}{(1-S)^3} & \text{при } S>0,\end{cases}$$

где

$$S=\begin{cases}x\cos p/4+x^2\cos(2p/4)+\ldots+x^n\cos(np/4)+\ldots & \text{при } 0.1\le x\le 0.4;\\ 3x+8x^2+\ldots+n(n+2)x^n+\ldots & \text{при } 0.4<x\le 0.8.\end{cases}$$

Значения a и b заключены в интервале –5…5. Конкретные значения этих параметров и значение шага изменения x задавать при вводе. На экран вывести: в первую строку значения a и b, заданные при вводе, во вторую и последующие строки y, S, x, число итераций.

Вариант 7

Значения x изменяются в интервале от –1 до 1. Конкретные значения x_0, x_{max}, Δx из указанного интервала задаются при вводе. Составит блок-схему и программу для вычисления значений функции:

$$y=\begin{cases}e^S-a, & \text{если } S<1.5;\\ e^S+1, & \text{если } S\ge 1.5,\end{cases}$$

где

$$S=\begin{cases}\displaystyle\sum_{n=1}^{\infty}\frac{(x-1)^n}{nx^n} & \text{при } x>0.5;\\ \displaystyle 2\sum_{n=0}^{\infty}\frac{x^{2n+1}}{2n+1} & \text{при } x\le 0.5.\end{cases}$$

Значения a и точность вычисления задавать при вводе. На экран вывести: y, S, x.

Вариант 8

Составить блок-схему и отладить программу для вычисления функции:

$$y=\begin{cases}\dfrac{1+S}{1-S}+\dfrac{1}{2}\arccos S & \text{при } S\le 0.9;\\ \dfrac{\cos S}{e^S} & \text{при } S>0.9,\end{cases}$$

где

$$S=\begin{cases}1+3x^2+\ldots+\dfrac{2n+1}{n!}x^{2n}+\ldots & \text{при } 0.1\le x<0.5;\\ \dfrac{x\cos p/3}{1}+\dfrac{x^2\cos(2p/3)}{2}+\ldots+\dfrac{x^n\cos(np/3)}{n}+\ldots & \text{при } 0.5\le x<1.\end{cases}$$

Начальное значение x и шаг изменения Δx задавать при вводе. На экран вывести значения y, S, x. Суммы вычислять с точностью до 0.01.

Контрольные вопросы

1. Какие алгоритмы называются циклическими?
2. Чем отличается итерационный ряд от ряда с конечным верхним пределом суммирования?
3. Опишите алгоритм, использующийся при накоплении сумм. Как отличаются алгоритмы, используемые при вычислении сумм итерационного ряда и ряда с конечным значением верхнего предела?
4. Какие можно выделить разновидности общих членов ряда?
5. Какие операторы повторения используются в языке Турбо Паскаль?
6. Как оформляется счетный оператор цикла и как он действует?
7. Как оформляются операторы цикла с пред- и постпроверкой условия и как они действуют?
8. Для чего используется оператор выбора, как он оформляется и как действует?

ТЕМА 4. ОБРАБОТКА МАССИВОВ. ОРГАНИЗАЦИЯ ВВОДА И ВЫВОДА С ИСПОЛЬЗОВАНИЕМ ФАЙЛОВ

Цель работы – приобретение навыков в обработке массивов, использовании файловых структур при вводе и выводе данных.

Массивом называют формальное объединение нескольких однотипных значений, рассматриваемых как единое целое. Массивы необходимы в тех случаях, когда требуется связать и одновременно использовать ряд родственных величин (например, результаты измерения напряженности магнитного поля Земли по одиночному профилю или системе взаимоувязанных профилей). Характеристиками массивов являются размер, размерность и тип данных, составляющих содержание массива. *Размер массива* определяется общим количеством элементов. *Размерность* характеризует порядок расположения отдельных элементов в массиве. По размерности различают одномерные массивы (их называют так же *вектор-строками* или *вектор-столбцами*), двумерные массивы (их называют *матрицами*), многомерные массивы, имеющие размерность больше двух. *Тип массива* определяется типом составляющих его элементов.

Положение отдельного элемента в массиве определяется с помощью *индексов*, которые указываются сразу же после имени и заключаются в квадратные скобки: `a[2,3], b[i], c[i+j]`. В качестве индексов могут использоваться целые константы, простые переменные целого типа, выражения целого типа.

Характеристики массивов в программе указываются при их описании. Описание массива может быть выполнено различными способами. В первом способе это делается в разделе описания переменных. При этом необходимо указать имя массива, границы изменения индексов, тип элементов. Таким образом, в этом способе вся информация о массиве приводится сразу. Пример описания по первому способу:

```
Var a : array[80] [1..20,1..3] of real;
```

В этом примере объявляется об использовании двумерного массива с именем `a`, состоящего из 60 элементов вещественного типа. В квадратных скобках указаны границы изменения каждого индекса. Граничные индексы определяют размер памяти, который должен быть зарезервирован для хранения его элементов. Иными словами с помощью граничных индексов определяется предполагаемый максимальный размер массива. Если размерность массива равна двум или более, то граничные индексы разделяются друг от друга запятыми, при этом значения граничных индексов указываются путем использования диапазонного типа. *Диапазонному типу* может быть дано собственное имя, которое может использоваться в последующих описаниях. Например:

---

[80] Array [эрэй] – набор, комплект.

```
Type[81] index  =  -50..50;
Var x : array [index] of real;
```

В тех случаях, когда возможно изменение значений граничных индексов, при описании массива может оказаться целесообразным использование в качестве указателей граничных индексов имен констант. Например:

```
Const nmax = 50;
Var x : array [nmax] of real;
```

Очевидно, если потребуется изменить максимальный размер массива, то в тексте программы достаточно исправить значение константы, используемой для определения граничных индексов.

При втором способе описания сначала определяется новый тип данных, представляющих собой характеристики массивов, которые предполагается использовать в данной программе, а далее имена указанных типов используются для описания соответствующих переменных. Пример:

```
Type Matrix  =  array [1..5,1..5] of word;
Var a,b : Matrix;
```

Для иллюстрации приемов работы с массивами рассмотрим несколько примеров.

Примеры 1 и 2 иллюстрируют организацию ввода и вывода элементов одномерного массива и матрицы.

```
{Пример 1. Организация ввода-вывода элементов одномерного массива}
{ввод с клавиатуры; вывод на экран дисплея}
Var Vector[82] : array [1..10] of integer;
    i,n : byte;
    a : real;
BEGIN
  Write('n = ');
  ReadLn(n); {Определение реальных размеров цикла}
  Write('Vektor = ');
  For i:=1 to n do
    Read(Vector[i]);
  WriteLn; {Переход на новую строку после окончания ввода элементов
массива}
  WriteLn('элементы массива = ');
  For i:=1 to n do
    WriteLn(Vector[i], ' ');
END.

{Пример 2. Организации ввода-вывода элементов матрицы}
{ввод с клавиатуры; вывод на экран дисплея}
Const n = 10;
      m = 10;
```

[81] Type [тайп] – тип.

[82] Vector [векто] – вектор.

```
Var Matrix[83]   : array [1..n,1..m] of byte;
    n1,m1,i,j : integer;
BEGIN
  Write('n1, m1 = ');
  ReadLn(n1, m1); {Определение реальных размеров цикла}
  Writeln('ввод по строкам элементов матрицы');
  For i:=1 to m1 do {Внешний цикл}
    For j:=1 to n1 do {Внутренний (вложенный) цикл}
      Read(Matrix[i,j]);
  WriteLn; {Переход на новую строку после окончания ввода элементов
массива}
  WriteLn('элементы матрицы = ');
  For i:=1 to m1 do {Внешний цикл}
    For j:=1 to n1 do {Внутренний (вложенный) цикл}
      Write(Matrix[i,j], ' ');
END.
```

Как уже было сказано ранее, доступ к отдельному элементу массива осуществляется путем указанием после имени индексного выражения. Для работы с отдельными элементами массивов требуется перебор индексов, что обеспечивается организацией циклов. Если массив многомерный (размерность два и более), то возникает ситуация, когда при изменении одного индекса другие тоже должны пробегать ряд значений. Это приводит к тому, что в теле одного цикла (*внешнего*) помещается один или несколько других циклов (*внутренних*). В итоге возникают так называемые *вложенные циклы* (см. пример 2). Основное правило, которое при этом следует соблюдать, заключается в том, что внутренний цикл должен целиком содержаться во внешнем. Вход в цикл разрешен только через его заголовок.

При организации операций ввода-вывода следует иметь в виду, что реальные размеры используемых массивов могут быть меньше, чем зарезервированный при их описании размер, определяемый, как это было показано выше, значениями граничных индексов. В связи с этим перед процедурой обмена данными следует определить реальные размеры массива (см. примеры 1 и 2).

Для иллюстрации рассмотренных выше понятий ниже приводится программа сортировки матрицы. Исходные данные в рассматриваемом примере представлены матрицей вещественного типа, содержащей положительные и отрицательные элементы. Требуется сформировать два одномерных массива, каждый из которых состоит только из положительных или только из отрицательных элементов. Необходимо также подсчитать количество элементов в обоих массивах, найти в обоих массивах элементы, имеющие максимальные и минимальные значения.

---

[83] Matrix [мэйтрикс] – матрица.

```
{Пример 3: программа по сортировке матрицы}
Type {объявление типа двумерного и одномерного массивов }
     a = array [1..10,1..10] of real;
     b = array [1..100] of real;
Var Matrix : a; {исходная матрица}
    MatPositive[84],MasNegative[85] : b; {одномерные массивы, формируемые
программой}
    MaxPos,MaxNeg,MinPos,MinNeg : real; {максимальные и минимальные
значения среди положительных и отрицательных элементов}
    i,j,n,m   : integer; {переменные, используемые в циклах и при
вводе-выводе элементов матрицы}
    NPos,NNeg : integer; {счетчики количества положительных и
отрицательных элементов}
BEGIN {Начало исполняемой части программы}
{В начале вводится матрица. Для этого полезно вывести подсказку о
необходимости ввода двух переменных, значения которых определяют
фактические размеры матрицы}
Write('n, m = '); ReadLn(n, m);
{Аналогично оформляется ввод значений элементов матрицы}
Write(Matrix = '); {Здесь нужен двойной цикл}
For i:=1 to n do
  For j:=1 to m do
    Read(Matrix[i,j]);
{Сортировка исходной матрицы и формирование двух одномерных
массивов, состоящих из положительных элементов с одновременным
подсчетом количества соответствующих элементов}
NPos:=0; NNeg:=0; {Начальные значения счетчиков}
For i:=1 to n do {Начало внешнего цикла}
  For j:=1 to m do {Начало внутреннего цикла; т.к. быстрее из-
меняется второй индекс, матрица просматривается построчно}
  Begin
    If Matrix[i,j] > =  0
    Then begin
      NPos:=NPos+1;
      MatPos[NPos]:=Matrix[i,j]
    End
    Else begin
      NNeg:=NNeg+1;
      MatNeg[NNeg]:=Matrix[i,j]
    End;
End;
```

[84] Positive [позитив] – положительный.
[85] Negative [нэгэтив] – отрицательный.

```
{Нахождение максимальных и минимальных элементов среди
положительных и отрицательных знчений; сначала в качестве
экстремальных берутся первые элементы}
MaxNeg:=MatNeg[1]; MinNeg:=MatNeg[1];
MaxPos:=MatPos[1]; MinPos:=MatPos[1];
For i:=1 to NNeg do
Begin {обработка отрицательных элементов}
  If MatNeg[i] > MaxNeg
    Then MaxNeg:=MatNeg[i];
  If MatNeg[i] < MinNeg
    Then MinNeg:=MatNeg[i]
End;
For i:=1 to npl do {аналогично обрабатываются положительные
элементы}
Begin
  If MatPos[i] > MaxPos
    Then MaxPos:=MatPos[i];
  If MatPos[i] < MinPos
    Then MinPos:=MatPos[i]
End;
{-----Вывод результатов-----}
WriteLn;
WriteLn('результаты обработки массива положительных элементов');
If NPos  =  0
  Then WriteLn('положительных элемнтов в матрице нет')
  Else begin
    Write('NPos = ',Npos);
    WriteLn(' MaxPos = ',MaxPos:8:2,' MinPos = ',MinPos:8:2);
    WriteLn('MatPos = ');
    For i:=1 to NPos do
      Write(MatPos[i]:8:2);
    WriteLn
end;
WriteLn('результаты обработки массива отрицательных элементов');
If Nneg  =  0
  Then WriteLn('отрицательных элементов в матрице нет')
  Else begin
    Write('NNeg = ',Nneg);
    WriteLn(' MaxNeg = ',MaxNeg:8:2,' MinNeg = ',MinNeg:8:2);
    For i:=1 to notr do Write(MatNeg[i]);
      Write(MatNeg[i]:8:2);
    WriteLn
  End
END.
```

При обработке больших объемов данных использование рассмотренных приемов обмена между внешними и внутренними устройствами ком-

пьютера может потребовать больших затрат времени. Например, при отладке программы ввод с клавиатуры одних и тех же данных потребуется повторять многократно. Более эффективен в таких ситуациях способ обмена, основанный на использовании *файлов*, заранее подготовленных с помощью каких-либо текстовых редакторов. Результаты расчетов также можно поместить в файл, который в дальнейшем может обрабатываться с помощью каких-либо программных средств.

Для того чтобы программа, написанная на Паскале, могла работать с файлом данных, в ней должна быть предусмотрена специальная *файловая переменная*. При использовании наиболее часто встречающихся типов данных (целых, вещественных, символьных и т.д.) такая переменная в описательной части программы характеризуется типом `text`[86]. Например:

```
Var <имя переменной> : text;
```

Переменная типа `text` предназначена для связи программы с одним текстовым файлом. Он представляет собой последовательность символов, разбитую на строки переменной длины. Для работы с файловыми переменными необходимо установить соответствие между рассматриваемой в данный момент переменной и *физическим файлом* – именованной областью памяти на каком-либо внешнем носителе (как правило, на диске). В ТП файловая переменная связывается с именем файла с помощью стандартной процедуры `Assign`[87]:

```
Assign(<файловая переменная>, <имя файла>);
```

Здесь `<файловая переменная>` – правильный идентификатор, объявленный в описательной части программы как переменная файлового типа, `<имя файла>` – текстовое выражение, содержащее имя файла или путь к нему. Пример:

```
Var Data[88],Result[89] : text;
Assign(Data,  'D:\1CURS\Data.dat');
Assign(Result,'D:\1CURS\Result.dat');
```

Перед обращением к файлу необходимо его открыть. Для этого в ТП предусмотрены две следующие процедуры:

`Reset(<имя файловой переменной>);`	открывает существующий файл для чтения из него информации;
`ReWrite(<имя файловой переменной>);`	открывает существующий файл для записи в него информации.

К моменту обращения к подпрограмме `Reset` соответствующий файл уже должен существовать на диске. Подпрограмма `Rewrite`[90] создает но-

---

[86] Text [текст] – текст.

[87] Assign [эсайн] – назначить.

[88] Data [дэйтэ] – данные.

[89] Result [ризалт] – результат.

[90] ReWrite [рирайт] – переписать.

вый файл на диске, если к моменту его выполнения файл с указанным именем не был создан ранее.

В н и м а н и е. При использовании процедуры записи следует иметь в виду, что в тех случаях, когда открываемый для записи файл уже существует, все данные, хранившиеся в нем до обращения, будут стерты.

Чтение и запись данных в файл становятся возможными только после его открытия. Чтение информации из файла производится с помощью операторов:

```
Read(<имя файловой переменной>, <список ввода>);
ReadLn(<имя файловой переменной>, <список ввода>);
```

Для записи в файл может использоваться один из следующих операторов:

```
Write(<имя файловой переменной>, <список ввода>);
WriteLn(<имя файловой переменной>, <список ввода>);
```

После завершения работы с файлами они должны быть закрыты с помощью оператора:

```
Close[91](<имя файловой переменной>);
```

ЗАДАНИЯ

Вариант 1

Составить программу для вычисления элементов матрицы $\mathbf{C} = \{c_{ij}\}$, являющейся разностью заданных матриц $\mathbf{A} = \{a_{ij}\}$ и $\mathbf{B} = \{b_{ij}\}$. Каждый элемент матрицы **C** вычисляется по формуле: $c_{ij} = a_{ij} - b_{ij}$, где $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, m$.

Далее следует вычислить значения элементов вектора $\mathbf{Z} = \{z_1, z_2, z_3, \ldots, z_n\}$, равного произведению матрицы $\mathbf{C} = \{c_{ij}\}$ на заданный вектор $\mathbf{X} = \{x_1, x_2, x_3, \ldots, x_n\}$. Компоненты вектора **Z** вычисляются по формуле:

$$z_i = \sum_{k=1}^{n} c_{ij} x_k, \quad \text{где} \quad i = 1, 2, \ldots, n.$$

При отладке программы рекомендуется использовать следующие исходные данные:

$$\mathbf{A} = \begin{pmatrix} 6.1 & 1.2 & 5.4 \\ 3.1 & 8.4 & 2.7 \\ 4.4 & 2.1 & 3.0 \end{pmatrix}; \quad \mathbf{B} = \begin{pmatrix} 2.0 & -2.5 & 6.0 \\ 1.1 & 3.1 & 8.4 \\ 5.6 & 1.2 & 3.2 \end{pmatrix}; \quad \mathbf{X} = \begin{pmatrix} -1 \\ 2.4 \\ 1 \end{pmatrix}.$$

Максимальные размеры n, m, k принять равными 9. На экран вывести матрицу **C** и вектор **Z.**

---

[91] Close [клоус] – закрывать.

Вариант 2

Составить программу для вычисления функции $d = t^2\sin(t)$ при t, изменяющемся от 1 до 1.5 с шагом 0.1. Вычисленные значения d и t оформить в виде векторов и вывести на экран в виде таблицы (см. табл. 3).

Таблица 3. Значений векторов *d t*

t	*d*	*t*	*d*	*t*	*d*

После вычисления d найти и вывести на экран среднее значение модуля разности между его компонентами и компонентами вектора y, значение которых следует задавать при вводе. Для вычисления средней разности воспользоваться формулой:

$$L = \frac{\sum_{i=1}^{n} |d_i - y_i|}{n}.$$

При отладке программы рекомендуется воспользоваться следующими исходными данными: **y** = {–1.6, 0.9, 1.3, 7.4, –0.2, –0.3}.

Вариант 3

Составить программу для вычисления элементов матрицы $\mathbf{C} = \{c_{ij}\}$, являющейся произведением матрицы $\mathbf{A} = \{a_{ij}\}$ размером $m \times n$ и матрицы $\mathbf{B} = \{b_{ij}\}$ размером $n \times q$. Каждая компонента матрицы **C** вычисляется по формуле:

$$c_{kj} = \sum_{i=1}^{n} a_{ki} \cdot b_{ik}, \quad \text{где} \quad i = 1, 2, ..., n; \quad j = 1, 2, ..., q; \quad k = 1, 2, ..., m.$$

При отладке программы рекомендуется использовать следующие исходные данные: $m = 3$; $n = 4$; $q = 3$.

$$\mathbf{A} = \begin{pmatrix} 3.8 & 0 & 1 & 2 \\ 6 & 0 & 2 & 4 \\ 9 & 0 & 3 & 16 \end{pmatrix}; \quad \mathbf{B} = \begin{pmatrix} 1 & -1 & -2 \\ 3 & 2.2 & 4 \\ 2 & 4 & 1 \\ -2 & 0 & 3 \end{pmatrix}.$$

Результат вычислений вывести на экран в виде:

```
МАТРИЦА C:
C11 C12 C13 C14
C21 C22 C23 C24
C31 C32 C33 C34
C41 C42 C43 C44
```

Вариант 4

Составить программу для нахождения наибольшего элемента прямоугольной матрицы $\mathbf{Z} = \{z_{ij}\}$, $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, m$. Каждый элемент матрицы вычисляется по формуле: $z_{ij} = x_i y_j$, где x_i, y_j – элементы векторов $\mathbf{X} = \{x_1, x_2, \ldots, x_n\}$, $\mathbf{Y} = \{y_1, y_2, \ldots, y_m\}$.

При отладке программы рекомендуется использовать следующие данные. Максимальный размер векторов принять равным: $n = 9$; $m = 10$. $\mathbf{X} = \{-1.1, 2.6, 1.0\}$; $\mathbf{Y} = \{3.2, 2.1, -2.0, 1.1\}$.

Вариант 5

Составить программу для вычисления элементов матрицы $\mathbf{C} = \{c_{ij}\}$, являющейся суммой матриц $\mathbf{A} = \{a_{ij}\}$ и $\mathbf{B} = \{b_{ij}\}$. Каждая компонента матрицы **C** вычисляется по формуле: $c_{ij} = a_{ij} + b_{ij}$, $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, m$. Матрица $\mathbf{A} = \{a_{ij}\}$ задается при вводе, а элементы матрицы $\mathbf{B} = \{a_{ij}\}$ вычисляются по правилу:

$$b_{ij} = \begin{cases} a_{ij} & \text{при } a_{ij} \geq 0; \\ 1 & \text{при } a_{ij} < 0. \end{cases}$$

При отладке программы рекомендуется использовать следующие данные. Граничные индексы принять равными: $n = 6$; $m = 8$.

$$\mathbf{A} = \begin{pmatrix} 3.0 & 0.0 & 1.0 & 2.0 \\ 3.0 & -1.1 & 2.0 & 1.0 \\ -1.0 & 0.0 & 3.0 & 1.0 \\ 1.0 & 3.0 & 2.0 & -2.0 \end{pmatrix}.$$

Вариант 6

Составить программу для вычисления вектора **X**, составленного из положительных элементов матрицы $\mathbf{A} = \{a_{ij}\}$, и вектора **Y**, составленного из отрицательных элементов этой же матрицы. Если значение элемента матрицы равно нулю, оно не записывается ни в один из формируемых векторов. На экран вывести количество элементов в полученных векторах, значения элементов их элементов.

При отладке программы рекомендуется воспользоваться следующими данными: максимальные значения индексов $i = 6$, $j = 5$; матрица $\mathbf{A} = \{a_{ij}\}$:

$$A = \begin{pmatrix} 3.0 & -1.4 & 0.7 & 6.1 \\ -1.1 & 2.5 & 3.8 & -4.0 \\ -5.2 & -3.6 & 0.0 & 8.1 \end{pmatrix}.$$

Вариант 7

Составить программу для вычисления следующей величины:

$$N = \max_i \sum_{j=1}^{m} \left| a_{ij} \right|, \quad \text{где} \quad i = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, m.$$

Матрица **A** считается заданной при вводе. Для решения задачи рекомендуется предварительно составить вспомогательный вектор $\mathbf{Z} = \{z_1, z_2,$

…, $z_n\}$, где $z_i = \sum_{j=1}^{m} \left| a_{ij} \right|$. Далее следует найти максимальный элемент этого вектора, значение которого и определит величину N.

При отладке программы рекомендуется использовать следующие данные. Максимальные значения для индексов принять равными: $n = m = 10$. Матрица $\mathbf{A} = \{a_{ij}\}$:

$$\mathbf{A} = \begin{pmatrix} 3.0 & 0.9 & 1.1 \\ 3.1 & -1.1 & 2.8 \\ -1.4 & 0.5 & 3.7 \\ 1.8 & 3.4 & 2.4 \end{pmatrix}.$$

Вариант 8

Составить программу для вычисления значений функции: $z = \sum_{k=1}^{n} a_k u_k(t)$, где значения вектора u вычисляются по правилу: $u_{k+1} = 2t_i u_k(t_i) - u_{k-1}(t_i)$; $u_1(t_i) = 1$; $u_2(t_i) = t_i$; $k = 1, 2, \ldots, n-1$; $i = 1, 2, \ldots, m$. Значения векторов **A** и **T** задаются при вводе. При отладке программы рекомендуется принять максимальные значения индексов равными: $m = n = 10$. В качестве векторов рекомендуются следующие данные: **T** = {0.5, 0.75, 0.85, 1.0}; **A** = {3, –4.2, 5.3, –6.1, 2.8}.

Контрольные вопросы

1. Какую форму организации данных называют массивами? Назовите их характеристики.
2. Как определяется положение отдельного элемента в массиве?
3. Как в программе могут быть определены характеристики массивов?
4. Что называют вложенными циклами?
5. Какими данными определяются максимальный и рабочий размеры массивов?
6. Для чего используются файловые переменные и как они определяются в программе?
7. Как связать файловую переменную с дисковым файлом?
8. Как открыть файл для чтения и записи?
9. Как закрыть файл?

ТЕМА 5. ПОДПРОГРАММЫ: ФУНКЦИИ И ПРОЦЕДУРЫ

Цель работы – умение составлять и пользоваться подпрограммами (функциями и процедурами).

Подпрограммы представляют собой относительно самостоятельные фрагменты программы, оформленные особым образом и снабженные собственным именем. Упоминание этого имени в тексте основной программы или другой подпрограммы называется *вызовом подпрограммы*. Подпрограммы представляют собой инструмент программирования, с помощью которого любая программа (или подпрограмма) может быть разбита на ряд относительно независимых друг от друга частей. Такое разбиение может оказаться целесообразным, по крайней мере, по двум причинам. Во-первых, это средство экономии памяти. Каждая подпрограмма существует в единственном экземпляре и может быть предназначена для выполнения вычислительного процесса, повторяющегося в различных точках основной программы или другой подпрограммы. Обращаться к подпрограмме можно многократно из соответствующих мест вызывающей программы. При вызове подпрограммы управление передается последовательности операторов, образующих *тело подпрограммы*, а с помощью передаваемых подпрограмме параметров нужным образом модифицируется реализуемый в ней алгоритм.

Во-вторых, использование подпрограмм позволяет применить *методику* так называемого *нисходящего проектирования*. При этой методике алгоритм решения задачи представляется в виде последовательности относительно крупных подпрограмм, реализующих самостоятельные части алгоритма. Подпрограммы в свою очередь могут разбиваться на менее крупные подпрограммы нижнего уровня и т.д. *Последовательное структурирование* может продолжаться до тех пор, пока реализуемые алгоритмы не станут простыми для программирования.

Текст подпрограммы помещается в разделе описания основной программы[92] и состоит из заголовка, раздела описаний и раздела исполняемых операторов. Разделы описаний и исполняемых операторов представляют собой блок. Т.е. можно считать, что подпрограмма состоит из заголовка и блока. Если в программе присутствуют подпрограммы, то возникают *вложенные* блоки. Все описания объектов (констант, типов, переменных, подпрограмм и т.д.) считаются локальными по отношению к тому блоку, в котором они охарактеризованы. Это означает, что соответствующие охарактеризованным объектам имена могут употребляться в принятом смысле (иначе говоря, считаются видимыми) только в той части текста, которая относится к данному блоку. Такой фрагмент текста называется *областью действия* этих *имен*.

---

[92] Если подпрограмма является составной частью другой подпрограммы, то текст внутренней подпрограммы помещается в описательной части внешней подпрограммы.

Таким образом, при введении в программу (или подпрограмму) функций и процедур возникает разделение объектов на глобальные и локальные. *Глобальными* считаются такие *объекты*, которые описаны вне данной подпрограммы. Они оказываются видимыми из любого места программы (в т.ч. и из внутренних подпрограмм). Наоборот, *локальные объекты* видимы лишь внутри того блока, где они охарактеризованы. Кроме объектов, охарактеризованных в описательной части подпрограммы, в них могут быть использованы и объекты, характеристики которых определены в заголовке подпрограммы. Они называются *формальными параметрами* (см. далее) и используются при обмене данными между подпрограммой и основной программой (или внешней и внутренней подпрограммами).

Оформление внутренних блоков для подпрограмм-функций и подпрограмм-процедур идентично, в то время как оформление заголовков имеет свою специфику. Заголовок функции имеет следующий вид:

Function <имя функции>(<список формальных параметров>):<имя типа>;

Здесь <**function**[93]> – служебное слово; <имя функции> – правильный идентификатор, имя, выбираемое пользователем по общим правилам языка программирования; <список формальных параметров> – аргументы функции с указанием их типа, используемые для обмена информацией между функцией и точкой вызова; <имя типа> – тип значения, вырабатываемого функцией.

Список формальных параметров состоит из отдельных частей, отделяемых друг от друга точкой с запятой. Каждая часть включает в себя список параметров одного и того же типа и имеет вид:

<список имен переменных> : <имя типа>;

Здесь <список имен переменных> – имена формальных параметров, разделенных запятыми; <имя типа> – общий тип этих параметров.

В качестве примера ниже приведен текст подпрограммы-функции, предназначенной для вычисления целой степени какого-либо вещественного числа или выражения вещественного типа.

```
Function Power[94](Num : real; Expon : integer) : real;
{Эта подпрограмма-функция вещественное число Num в целочисленную
степень Expon}
Var Count[95] : integer;
    Res   : real;
BEGIN
If Expon = 0
  Then Power:=1
  Else begin
    Res:=Num;
```

[93] Function [фанкшн] – функция.

[94] Power [пауэ] – степень.

[95] Count [каунт] – итог.

```
    For Count:=2 to Abs(Expon) do
      Res:=Res*Num;
    If Expon < 0
      Then Power:=1/Res
      Else Power:=Res
  End
END.
```

Следует обратить внимание на то обстоятельство, что в разделе исполняемых операторов подпрограммы-функции всегда должен присутствовать хотя бы один оператор присваивания, в левой части которого должна быть указана переменная, имя которой совпадает с именем функции (в рассматриваемом примере – Power). Через этот оператор вычисленное значение передается в *точку вызова*.

Для того чтобы воспользоваться подпрограммой-функцией, к ней следует обратиться, задав ее аргументам конкретные значения. Обращение к функции имеет вид:

<Имя функции>(<список фактических параметров>);

Здесь <фактические параметры> – это объекты, указанные в обращении к подпрограмме. В списке они отделяются друг от друга запятыми. В качестве *фактических параметров* могут выступать константы, переменные, арифметические выражения. Соответствие между формальными и фактическими параметрами устанавливается по порядку их следования. Следовательно, списки параметров в имени подпрограммы и обращении к ней должны быть согласованными по числу параметров, порядку их следования и типу принимаемых значений. Например, обращение к указанной подпрограмме Power может быть оформлено в виде:

Hypotenuse[96]:=SgRt(Power(a,2)+Power(b,2));

Описание подпрограммы-процедуры включает заголовок, раздел описаний и раздел исполняемых операторов:

```
Procedure[97] <имя процедуры>(<список формальных параметров>);
<раздел описаний>
BEGIN
<раздел операторов>
END.
```

Таким образом, в отличие от заголовка функции в заголовке процедуры не указывается тип получаемого результата. Эта разница возникает из-за того, что результатом работы процедуры, в отличие от функции, может быть не только единственное значение, но и множество значений. Резуль-

[96] Hypotenuse [хайпотинйуз] – гипотенуза.

[97] Procedure [просиджэ] – процедура.

тат работы процедуры может иметь и не вычислительный характер (например, организация вывода или ввода данных).

Формальные параметры, указываемые в заголовке процедуры, могут быть разделены на две разновидности: *параметры-переменные* и *параметры-значения*. Если какая-либо группа переменных должна рассматриваться как параметры-переменные, то перед этой группой ставится служебное слово **var**. Например:

```
Procedure MyProcedure[98](var a,b : real; c : word);
```

Определение формального параметра тем или иным способом существенно для вызывающей программы. Если формальный параметр объявлен как параметр-переменная, то при вызове подпрограммы ему должен соответствовать фактический параметр в виде переменной нужного типа. Если формальный параметр объявлен как параметр-значение, то при вызове ему может соответствовать либо переменная, либо произвольное выражение соответствующего типа. Замена формальных параметров на фактические при обращении к процедуре осуществляется следующим образом.

Если параметр определен как параметр-значение, то перед вызовом подпрограммы вычисляется значение фактического параметра и полученный результат передается подпрограмме. Любые возможные изменения в подпрограмме параметра-значения никак не воспринимаются вызывающей программой, так как в этом случае изменяется только копия фактического параметра.

Если параметр определен как параметр-переменная, то при вызове подпрограммы передается сама переменная, а не ее копия (фактически в подпрограмму передается адрес переменной). Изменение параметра-переменной приводит к изменению самого фактического параметра в вызывающей программе.

Таким образом, параметры-переменные можно рассматривать как выходные параметры, с помощью которых результат, полученный в подпрограмме, передается в вызывающую программу. Параметры-значения могут использоваться как входные параметры, с их помощью в подпрограмму может передаваться информация, необходимая для работы подпрограммы.

Следует отдельно отметить особенность обмена данными между подпрограммой и вызывающей программой в случае, когда в качестве таких данных должны использоваться массивы. Эта особенность связана с тем обстоятельством, что типом любого параметра в списке формальных параметров может быть только стандартный тип языка или ранее объявленный. Ошибочным, например, будет следующее объявление процедуры:

```
Procedure S(a : array [1..10] of real);
```

Здесь в списке формальных параметров используется ранее не объявленный нестандартный тип-диапазон (1..10). В связи с отмеченной особенностью в тех случаях, когда в качестве данных, используемых при обмене между процедурой и вызывающей программой, предполагается использо-

[98] My [май] – мой.

вать массивы, их тип следует объявить во внешней по отношению к процедуре программе. Например:

```
Type Atype  =  array [1..10] of real;
Procedure S(a : Atype);
```

Следует также иметь в виду, реальный размер массива может быть меньше, чем объявленный при его описании. Поэтому при обмене исходными данными между процедурой и вызывающей программой вместе с самим массивом следует передать и его реальные размеры.

В качестве примера использования процедуры ниже приводятся два варианта программы, в которых по трем векторам (одномерным массивам): $\mathbf{x} = [x_1, x_2, ..., x_n]$; $\mathbf{y} = [y_1, y_2, ..., y_l]$; $\boldsymbol{z} = [z_1, z_2, ..., z_k]$, где n, l, k – длина соответствующего вектора строится матрица **A**, строками которой являются элементы векторов x, y, z, отсортированные предварительно в порядке возрастания значений их элементов. Длина строк в **A** равна минимальной длине одного из трех исходных векторов.

```
{Вариант 1 решения задачи (без включения подпрограммы):}
Var i,j,n,l,k,m    : byte;
    MinX,MinY,MinZ : real;
    x,y,z,xx,yy,zz : array [1..100] of real;
    Matr           : array [1..100,1..3] of real;
BEGIN
Write('Ввести n, l, k '); ReadLn(n, l, k);
WriteLn('Ввести последовательно массивы x, y, z');
For i:=1 to n-1 do
  Read(x[i]);
ReadLn(x[n]);
For i:=1 to l-1 do
  Read(y[i]);
ReadLn(y[l]);
For i:=1 to k-1 do
  Read(z[i]);
ReadLn(z[k]);
{Организация сортировки исходных массивов}
i:=1;
Repeat
  MinX:=x[i];
  For j:=i+1 to n do
    If x[j] < MinX
      Then begin
        MinX:=x[j];
        x[j]:=x[i];
        x[i]:=MinX;
     End;
  i:=i+1;
Until i > n-1;
```

```
i:=1;
Repeat
  MinY:=y[i];
  For j:=i+1 to l do
    If y[j] < MinY
    Then begin
      MinY:=y[j];
      y[j]:=y[i];
      y[i]:=MinY;
    End;
  i:=i+1;
Until i > l-1;
i:=1;
Repeat
  MinZ:=z[i];
  For j:=i+1 to k do
    If z[j] < MinZ
    Then begin
      MinZ:=z[j];
      z[j]:=z[i];
      z[i]:=MinZ;
    End;
  i:=i+1;
Until i > k-1;
{Выбор длины строки в матрице A}
If (n < l) and (n < k)
  Then m:=n
  Else if (l < n) and (l < k)
    Then m:=l
    Else m:=k;
{Формирование матрицы A}
For i:=1 to 3 do
  For j:=1 to m do
    If i=1
      Then Matr[i,j]:=x[j]
      Else if i=2
        Then Matr[i,j]:=y[j]
        Else Matr[i,j]:=z[j];
{Вывод матрицы A}
WriteLn('Матрица результата работы программы');
For i:=1 to 3 do
Begin
  For j:=1 to m-1 do
    Write(Matr[i,j]:5:2, ' ');
  WriteLn(Matr[i,m]:5:2);
End;
```

```
{Конец программы}
Write('Нажми Enter'); ReadLn;
END.

{Вариант 2 решения задачи (с подпрограммой-процедурой)}
Type a = array [1..100] of real;
     b = array [1..100,1..3] of real;
Var i,j,n,l,k,m    : byte;
    x,y,z,xx,yy,zz : a;
    Matr           : b;
{-----------------------------------------}
  Procedure Ord(w : a; nn : byte; var ww : a);
  Var MinW : real;
  Begin {Тело процедуры}
{Организация сортировки исходных массивов}
    i:=1; MinW:=w[1];
    Repeat
      For j:=i+1 to nn do
        If w[j] < MinW
          Then MinW:=w[j];
      ww[i]:=MinW; i:=i+1;
    Until i > nn;
  End;
{-----------------------------------------}
BEGIN
{Начало основной программы}
Write('Ввести n, l, k '); ReadLn(n, l, k);
WriteLn('Ввести последовательно массивы x, y, z');
For i:=1 to n-1 do
  Read(x[i]);
ReadLn(x[n]);
For i:=1 to l-1 do
  Read(y[i]);
ReadLn(y[l]);
For i:=1 to k-1 do
  Read(z[i]);
ReadLn(z[k]);
{Обращение к процедуре Ord для сортировке исходных массивов}
Ord(x,n,xx);
Ord(y,l,yy);
Ord(z,k,zz);
{Выбор длины строки в матрице A}
If (n < l) and (n < k)
  Then m:=n
  Else if (l < n) and (l < k)
    Then m:=l
```

```
    Else m:=k;
{Формирование матрицы A}
For i:=1 to 3 do
  For j:=1 to m do
    If i = 1
      Then Matr[i,j]:=x[j]
      Else if i = 2
        Then Matr[i,j]:=y[j]
        Else Matr[i,j]:=z[j];
{Вывод матрицы A}
WriteLn('Матрица результата работы программы');
For i:=1 to 3 do
  For j:=1 to m-1 do
    Write(Matr[i,j]:5:2, ' ');
WriteLn(Matr[i, m]);
{Конец программы}
WriteLn('Нажми Enter'); ReadLn;
END.
```

Для иллюстрации рассмотренных выше понятий ниже приводится программа *линейной интерполяции*. Содержание задачи, реализованной в программе, заключается в следующем. Исходными данными являются значения функции *y* при некоторых значениях аргумента *x*. При этом известно, что *y* изменяется в зависимости от *x* по *линейному закону*. Требуется вычислить значения функции при заданных значениях аргумента *xa*, называемых *узлами интерполяции*.

```
Uses CRT[99]; {стандартный модуль для очистки экрана}
Const l = 20; {определение максимального размера массива}
Type mas = array [1..l] of real; {определение структуры массивов
путем объявления типа}
{-----------------------------------------}
  Procedure Intln(x,y : mas; n : byte; xa : real; var ya : real);
{заголовок процедуры: x, y-исходные массивы; n-их рабочая длина;
xa-значение аргумента, при котором определяется интерполируемое
значение функции; значения x, y, n, xa будут определены при
обращении к процедуре; ya-искомое значение функции, оно будет
передано в основную программу}
  Var i,i1 : byte;
  Begin {Начало тела процедуры}
    For i:=2 to n do
    Begin
      If (x[i]-xa) >= 0
      Then begin
```

[99] CRT (cathode-ray tube) [сиарти] – катодно-лучевая трубка.

```
        i1:=i-1;
        ya:=y[i1]+(y[i1+1]-y[i1])*(xa-x[i1])/(x[i1+1]-x[i1])
      End
   End
End;
{-----------------------------------------}
{Основная (вызывающая) программа}
Var xx,yy,xint,yint : mas;
    lxy,lxint,i     : byte;
BEGIN {Начало основной программы}
ClrScr;
Write('lxy, lxint = '); ReadLn(lxy, lxint);
{Ввод рабочего размера аргумента (x) и функции(y)}
WriteLn('x, y = '); {Ввод значений аргумента и функции}
For i:=1 to lxy do
  Read(xx[i], yy[i]);
WriteLn;
Write('xint = '); {Ввод значений аргумента, при которых находятся}
For i:=1 to lxint do
  Read(xint[i]); {Интерполированные значения функции}
WriteLn;
For i:=1 to lxint do
  IntLn(xx,yy,lxy,xint[i],yint[i]); {Обращение к процедуре IntLn}
{Переменные, указанные в скобках – это фактические параметры;
значения xx, yy, lxy, xint передаются в процедуру; значение yint
получается из процедуры}
WriteLn('xint, yint = '); {Вывод заголовка результатов}
For i:=1 to lxint do
  Write(i,' ',xint[i]:5:2,' ',yint[i]:5:2,'**'); {Вывод xint, yint}
ReadLn;
END.
```

Задания

Вариант 1

Заданы четыре вектора: $\mathbf{X} = \{x_1, x_2, x_3\}$, $\mathbf{Y} = \{y_1, y_2, y_3\}$, $\mathbf{Z} = \{z_1, z_2, z_3, z_4\}$, $\mathbf{P} = \{p_1, p_2, p_3, p_4\}$. Логической переменной *a* присвоить значение **true**, если скалярное произведение векторов **X** и **Y** больше скалярного произведения **Z** и **P**, значение **false** в противном случае. Скалярное произведение двух векторов (например, **f** и **e**) вычисляется по формуле $(\mathbf{f}\cdot\mathbf{e}) = f_1\cdot e_1 + f_2\cdot e_2 + f_3\cdot e_3 + \ldots$ Вычисление скалярного произведения оформить в виде подпрограммы.

При отладке программы рекомендуется использовать следующие исходные данные: $\mathbf{X} = \{1, 2, 3\}$, $\mathbf{Y} = \{2.5, 6, 3.2\}$, $\mathbf{Z} = \{3.7, 1.2, 6.4, -5.3\}$, $\mathbf{P} = \{-1, 4, 1, -2\}$.

Вариант 2

Найти корни уравнения $ax^2+bx+c = 0$, где a – длина вектора $\boldsymbol{f} = \{f_1, f_2, f_3, f_4, f_5\}$; b – длина вектора $\boldsymbol{h} = \{h_1, h_2, h_3, h_4\}$; c – длина вектора $\boldsymbol{g} = \{g_1, g_2, g_3\}$. Длина вектора вычисляется по формуле: $|d| = \sqrt{\sum_{i=1}^{n} d_i^2}$, где d_i – составляющие вектора. Вычисление длины вектора и корней квадратного уравнения оформить в виде подпрограммы-функции.

При отладке программы рекомендуется использовать следующие исходные данные: $\boldsymbol{f} = \{1, 2, 3, 4, 5\}$; $\boldsymbol{h} = \{42, 75, 16, -34\}$; $\boldsymbol{g} = \{1.2, 0.75, -1.9\}$.

Вариант 3

Заданы две матрицы

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}; \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}.$$

Произведением двух матриц называется новая матрица, определенная по правилу:

$$C_{ij} = \sum_{k=1}^{N} A_{ik} B_{kj}, \quad \text{где } i = 1, 2, ..., M; \quad j = 1, 2, ..., L; \quad k = 1, 2, ..., N.$$

Произведение матриц называется перестановочным, если выполняется условие: $\mathbf{A}\cdot\mathbf{B} = \mathbf{B}\cdot\mathbf{A}$.

Составить программу, проверяющую перестановочность задаваемых матриц. В случае положительного ответа на печать вывести сообщение "AB = BA"; в противном случае на экран вывести текст "Произведение матриц не перестановочно".

Вычисление произведения матриц оформить в виде подпрограммы. При отладке программы в качестве исходных рекомендуется использовать следующие данные:

$$\mathbf{A} = \begin{pmatrix} 1.2 & 5 & -3 \\ 2.7 & 1 & 2 \\ -4 & 6.3 & 2 \end{pmatrix}; \quad \mathbf{B} = \begin{pmatrix} 1 & 2 & 3 \\ 5.1 & 4 & -7 \\ 6 & 8 & 9 \end{pmatrix}.$$

Вариант 4

Заданы два вектора $\mathbf{X} = \{x_1, x_2, x_3, \ldots\}$; $\mathbf{Y} = \{y_1, y_2, y_3, \ldots\}$. Логической переменной a присвоить значение **true**, если длина **X** больше длины **Y**, и **false** в противном случае.

Для вычисления длины вектора составить подпрограмму, пользуясь правилом:

$$|\mathbf{f}| = \sqrt{\sum_{i=1}^{N} f_i^2},$$

где f_i – компоненты вектора **f**.

При отладке программы рекомендуется воспользоваться следующими данными: **X** = {–1.2, 3,5}; **Y** = {1, 2.1, 6, –2, 3}.

Вариант 5

Заданы две матрицы:

$$\mathbf{A}=\begin{pmatrix} a_{11} & a_{12} & a_{13} & \mathbf{L} & a_{1n} \\ \mathbf{M} & & & & \mathbf{M} \\ \mathbf{M} & & & & \mathbf{M} \\ a_{l1} & a_{l2} & a_{l3} & \mathbf{L} & a_{\ln} \end{pmatrix}; \quad \mathbf{B}=\begin{pmatrix} b_{11} & b_{12} & b_{13} & \mathbf{L} & b_{1k} \\ \mathbf{M} & & & & \mathbf{M} \\ \mathbf{M} & & & & \mathbf{M} \\ b_{m1} & b_{m2} & b_{m3} & \mathbf{L} & b_{mk} \end{pmatrix}.$$

Построить таблицу функции $y = cx^2+d$ при x, изменяющемся от 0 до 1 с шагом 0.1. В приведенной формуле c – след матрицы **A**; d – след матрицы **B**. (Следом матрицы называется сумма элементов главной диагонали). След матрицы вычислять с помощью подпрограммы. При отладке программы рекомендуется воспользоваться следующими данными

$$\mathbf{A}=\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}; \quad \mathbf{B}=\begin{pmatrix} 1.2 & 8 & 4 & 1 \\ 7.5 & 3 & 1.2 & 6 \\ -2 & 1 & -1 & 7 \\ 5 & 4 & 3 & 2 \end{pmatrix}.$$

Вариант 6

Заданы две матрицы

$$\mathbf{A}=\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}; \quad \mathbf{B}=\begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{pmatrix}.$$

Определить, какие из заданных матриц являются симметричными.

П р и м е ч а н и е. *Матрица* называется *симметричной*, если *транспонированная матрица* равна исходной. Матрица $\mathbf{A}^T$-размерности $N\times M$ называется транспонированной по отношению к исходной матрице **A** размерности $M\times N$, если между элементами обеих матриц выполняется соотношение $A_{ij}^T = A_{ji}$, где i = 1, 2, …, N; j = 1, 2, …, M.

Транспонирование анализируемых матриц оформить в виде подпрограммы.

При отладке программы рекомендуется воспользоваться следующими исходными данными.

$$\mathbf{A}=\begin{pmatrix} 1 & 2 \\ 2 & 4 \end{pmatrix}; \quad \mathbf{B}=\begin{pmatrix} 1.2 & 3 & 7 \\ 6 & -4 & 2 \\ -1 & 2.3 & 1 \end{pmatrix}.$$

Вариант 7

Заданы два массива $\boldsymbol{a} = \{a_1, a_2, \ldots, a_n\}$; $\boldsymbol{b} = \{b_1, b_2, \ldots, b_m\}$. Переменной S присвоить значение –1, если $\max(a) > \max(b)$; 0, если $\max(a) = \max(b)$; 1, если $\max(a) < \max(b)$.

Поиск максимальных элементов оформить в виде подпрограммы. При отладке программы рекомендуется воспользоваться приведенными данными: $\boldsymbol{a}$ = {-5.2, 8, 1.3, –6 }; $\boldsymbol{b}$ = {–4.7, –3, 2.6, 7, 13, –1}.

Вариант 8

Четыре точки заданы своими координатами: $X\{X_1,X_2\}$; $Y\{Y_1,Y_2\}$; $Z\{Z_1,Z_2\}$; $P\{P_1,P_2\}$. Определить, какие из них находятся на максимальном удалении друг от друга; вывести на экран значение этого расстояния и обозначение точек, соответствующих ему.

Для вычисления расстояния используется формула: $S_{i,j} = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}$, где x_i, y_i – координаты одной точки; x_j, y_j – координаты другой точки. Вычисление расстояния между двумя точками оформить в виде подпрограммы. При отладке программы рекомендуется воспользоваться следующими исходными данными: $X\{4.8, –3\}$; $Y\{–6.5, 1.2\}$; $Z\{–3.7, –4.5\}$; $P\{7.8, 2\}$.

Контрольные вопросы

1. Дайте определение подпрограмме.
2. Какие разновидности подпрограмм используются в ТП и в чем их основные различия?
3. По каким правилам оформляется текст подпрограммы?
4. Почему в заголовке функции указывается тип, а в заголовке процедуры он отсутствует?
5. Какие переменные относятся к глобальным, а какие к локальным?
6. Какие параметры называются формальными, а какие фактическими?
7. В чем различие между параметрами-значениями и параметрами-переменными?
8. Для чего могут использоваться параметры-переменные и параметры-значения?
9. В чем состоит особенность обмена данными между подпрограммой и основной программой при использовании массивов?

ТЕМА 6. МОДУЛИ

Ц е л ь р а б о т ы – знакомство с имеющимися в языке Турбо Паскаль принципами составления программ сложной структуры на основе использования модулей.

При составлении сложных программ мало эффективным оказывается оформление их в виде единого текста, т.к. в этом случае осложняется как процесс его написания, так и отладки. В таких случаях рекомендуется разбивать алгоритм решения задачи на отдельные части. Очевидно, наиболее удобным приемом составления сложной программы из отдельных частей является такой способ, при котором отдельные части компилируются раздельно, а затем «собираются» для последующего совместного использования. В практике программирования может встретиться и такая ситуация, когда при создании новых программ оказывается целесообразным использование написанных ранее подпрограмм. Например, подпрограмма решения системы линейных алгебраических уравнений может быть использована при численном решении дифференциальных уравнений или решения каких-либо аналогичных задач. В языке Турбо Паскаль для независимой разработки отдельных частей программы и последующего связывания их в одну программу используется механизм так называемых модулей.

Модуль – это отдельно создаваемая и отдельно компилируемая программная единица, имеющая собственное имя, которая предназначена, прежде всего, для описания различных объектов (типов данных, переменных, констант, процедур и функций). Все описанные в модуле средства становятся доступными для внешних программ путем специальной ссылки на его имя. Структуру модуля в общем виде можно представить следующим образом:

```
UNIT[100] <имя модуля>;
INTERFASE[101]
  <интерфейсная часть (раздел описаний)>
IMPLEMENTATION[102]
  <исполняемая часть (раздел реализации)>
BEGIN
  <инициирующая часть>
END.
```

Первая строка текста модуля представляется всегда его заголовком. Он состоит из служебного слова **unit** (модуль) и следующего за ним правильного идентификатора (имени модуля). Это имя должно быть уникальным и совпадать с именем того дискового файла, в котором будет разме-

[100] Unit [юнит] – модуль.
[101] Interface [интэфэйс] – взаимодействие.
[102] Implementation [имплэмэнтэйшн] – реализация.

щаться текст исходного модуля. Имя модуля служит для его связи с другими модулями и вызывающей программой.

Раздел описаний начинается служебным словом **interfase** и заканчивается перед зарезервированным словом **implementation**. В этом разделе размещается описание объектов (типов данных, переменных, констант, процедур и функций). Ко всем этим описаниям внешняя программа может обращаться так, как если бы оно было сделано непосредственно в самой программе. Интерфейсную часть можно считать «видимой» частью модуля, так как она определяет объекты, доступные программе, использующей данный модуль. Типы данных, переменные и константы в интерфейсной части описываются по обычным правилам, а для процедур и функций здесь помещается только заголовок, являющийся указанием на то, как к ним следует правильно обращаться.

Раздел реализации открывается всегда служебным словом **implementation**. Все, что описано в интерфейсной части модуля (типы данных, переменные, константы, процедуры и функции), можно использовать и в разделе реализации. Кроме того, здесь помещается описание объектов, которые являются локальными для модуля и недоступны вызывающим программам. Они используются процедурами и функциями, имена которых указаны в интерфейсной части. В этом же разделе помещаются и описания теля процедур и функций. При этом их заголовки могут быть оформлены в краткой форме, без списков формальных параметров, например:

Procedure <имя>;

Functon <имя> : <тип результата>;

Непосредственно за заголовком должен следовать блок подпрограммы.

Для локальных подпрограмм заголовки в разделе реализации оформляются как обычно в полной форме. Следует иметь в виду, что локальные подпрограммы могут использоваться только внутри самого модуля и не могут быть вызваны из другой программы. Последним разделом модуля является *раздел инициализации*, который может отсутствовать. В последнем случае окончание модуля **END** помещается за последней строчкой раздела реализации и обязательно завершается точкой. В разделе инициализации могут размещаться исполняемые операторы, содержащие некоторый фрагмент программы. Эти операторы исполняются до передачи управления вызывающей программе и обычно используются для подготовки ее работы. Например, в них могут инициироваться переменные, открываться нужные файлы, устанавливаться связь с другими компьютерами по коммуникационным каналам и т. д.

Связь основной программы с модулями указывается с помощью объявления

Uses[103] <список модулей>;

[103] Uses [юзэс] – использования.

Здесь **uses** (использует) – служебное слово; <список модулей> – имена модулей, разделенных запятыми, к которым обращается данная программа. Указанное объявление должно открывать раздел основной программы. В качестве примера ниже приводится текст модуля с именем DemoMod, в котором запрограммировано вычисление целой степени *n* вещественного числа *f*. Вычисление степени производится путем последовательного перемножения: $f^n = f^{n-1} \cdot f$. В тексте модуля охарактеризована константа Author[104], которой присваивается в качестве значения фамилия автора модуля (в рассматриваемом примере 'Иванов'). В инициирующую часть помещен вывод сообщения 'Работает модуль DemoMod, автор: Иванов'.

```
UNIT DemoMod;
INTERFASE
  Const Author = 'Иванов';
  Function St(f : real; n : integer) : real;
IMPLEMENTATION
  Function St;
  Var i:integer; b:real;
  Begin
    If n = 0
      Then St:=1
      Else begin
        b:=f;
        For i:=2 to Abs(n) do
          b:=f*b;
        If n < 0
          Then St:=1/b
          Else St:=b
      End
  End;
BEGIN
  Writeln('Работает модуль DemoMod, автор:', Author)
END.
```

После набора текста модуля его следует сохранить в файле DemoMod.pas. Созданный модуль используется в программе, вычисляющей выражение $S = a^5+(a+1)^{-7}$. Значение *a* вводится с клавиатуры.

```
PROGRAM ModTest;
Uses DemoMod;
Var a,rsum : real;
BEGIN
WriteLn('Введите значение a');
Readln(a);
```

[104] Author [ɔco] – автор.

```
Rsum:=St(a,5)+St(a+1,-7);
Writeln('Результат:', rsum)
END.
```

В Турбо Паскале могут использоваться *стандартные модули*, входящие в его библиотеку и содержащие описание различных типов, констант, процедур и функций. Подробное описание этих модулей можно найти в соответствующих инструкциях по языку программирования или учебниках (например, в [1]). Ниже приводится краткая характеристика трех библиотечных модулей, использование которых потребуется при выполнении заданий по рассматриваемой теме.

Модуль System[105] в отличие от других библиотечных (стандартных) модулей подключается к обрабатываемой программе автоматически. Этот модуль является основной библиотекой Турбо Паскаля. Он реализует подпрограммы для всех так называемых встроенных возможностей, таких как ввод/вывод, вычисление часто встречаемых математических функций (синус, косинус, логарифм и т.д.), управление *оверлеями* и *динамическое распределение памяти*).

Модуль CRT содержит процедуры и функции, обеспечивающие *текстовым режимом работы экрана*. С помощью подпрограмм, входящих в этот модуль, можно перемещать курсор в произвольную позицию экрана, менять цвет выводимых символов и фона, создавать отдельные окна.

Модуль Graf содержит набор типов, констант, переменных, процедур и функций для управления *графическим режимом работы экрана*. С помощью программ, входящих в этот модуль, можно создавать разнообразные графические изображения и выводить на экран текстовые надписи стандартными шрифтами или разработанными программистом.

Примеры некоторых процедур, входящих в модули CRT и Graf, приведены в таблице 4.

Таблица 4. Стандартные модули библиотеки Турбо Паскаля

Имя модуля	Имя процедуры или функции	Назначение подпрограммы
Crt	**Procedure** ClrEOL[106]	Удаляет все символы от текущей позиции курсора до конца строки без перемещения курсора
	Procedure ClrScr[107]	Очищает экран (окно) и помещает курсор в верхний угол
	Procedure DelLine[108]	Удаляет строку, на которой находится курсор, и перемещает все строки ниже этой строки на строку вверх. Нижняя строка очищается

[105] System [систим] – система.

[106] ClrEOL (clear end of line) [клиэ иоуэл] – очистить до конца строки.

[107] ClrScr (clear screen) [клиэ скрин] – очистить экран.

[108] DelLine (delete line) [дэлит лайн] – удалить строку.

Имя модуля	Имя процедуры или функции	Назначение подпрограммы
	Procedure GoTo(x, y : byte)	Перемещает курсор в нужное место экрана
Graf	**Procedure** Arc[109] (x, y : integer; StartAngle, EndAngle, Radius : word)	Рисует дугу радиусом Radius[110] от начального угла StartAngle[111] к конечному углу EndAngle, используя X, Y как координаты центра
	Procedure Circle[112](x, y : integer; R : word)	Рисует окружность радиуса R, используя X, Y как координаты центра
	Procedure ClearDevice[113]	Очищает экран
	Procedure DrawPoly[114](NumPoints : word; **var** PolyPoints)	Рисует многоугольник из NumPoints[115] вершин с координатами в PolyPoints текущим цветом
	Procedure FillPoly[116](NumPoints : word; **var** PolyPoints)	Рисует и штрихует многоуголиник, содержащий NumPoints вершин с координатами в PolyPoints
	Procedure InitGraph[117](**var** Driver, Mode : integer; Path : string[118])	Инициализирует графический режим. Переменные Driver[119], Mode[120] должны содержать тип графического драйвера и его режим работы. Допускается указывать Driver = 0 для автоматического определения этих параметров по результатам тестирования аппаратуры. Параметр Path[121] определяет маршрут поиска файла графического драйвера.
	Procedure Line(x1, y1, x2, y2 : integer)	Рисует линию от точки X1, Y1 до точки X2, Y2
	Procedure Rectangle[122] (x1, y1, x2, y2 : integer)	Рисует прямоугольник, используя текущий цвет и тип линии.

[109] Arc [ак] – дуга.
[110] Radius [рэйдйес] – радиус.
[111] Angle [энгл] – угол.
[112] Circle [сёкл] – окружность.
[113] Device [дивайс] – устройство.
[114] Draw poly [дро поли] – рисовать многоугольник.
[115] Num Points [нам поинтс] – количество точек.
[116] Fill poly [фил поли] – заполнять многоугольник.
[117] Init graph [инит грэф] – инициализировать графический режим.
[118] String [стрин] – строка.
[119] Driver [драйвэ] – драйвер.
[120] Mode [моуд] – режим.
[121] Path [пас] – путь.
[122] Rectangle [рэктэнгл] – прямоугольник.

Имя модуля	Имя процедуры или функции	Назначение подпрограммы
	Procedure SetBkColor[123] (Color[124] : word)	Устанавливает цвет фона
	Procedure SetColor (Color: word)	Устанавливает основной цвет, которым будет осуществляться рисование
	Procedure SetFillStyle[125] (Pattern[126], Color : word)	Устанавливает образец штриховки и цвет
	Procedure SetLineStyle[127] (LineStyle, Pattern, Thickness[128] : word)	Устанавливает толщину и стиль линий
	Procedure SetPalette[129] (ColorNum, Color : word)	Заменяет цвет палитры с номером ColorNum на Color

В качестве иллюстративного примера по использованию стандартных модулей ниже приведена программа, в которой реализована задача построения *n* квадратов, вершины которых располагаются на сторонах предыдущего квадрата. Положение вершин очередного квадрата находится путем деления стороны предыдущего квадрата по правилу $x = x_1+(x_2-x_1)*mu/(1+mu)$; $y = y_1+(y_2-y1)*mu/(1+mu)$, где *mu*-параметр деления отрезка. Координаты вершин начального квадрата и значения параметра *mu* задаются при вводе.

```
{Пример программы, использующей два стандартных модуля: Crt
и Graph. }
Uses Graph,Crt; {установление связи со стандартными модулями}
  Var x,y,a,b : array [1..4] of integer; {массивы координат вершин
предыдущего (x, y) и последующего (a, b) квадратов}
      xy      : array [1..5] of PointType; {массив записей типа
PointType, описанного в модуле Graph. Этот массив используется для
построения очередного квадрата с помощью процедуры DrawPoly модуля
Graph}
      c       : char;
      mu      : real;
      i,j     : byte;
      hz,GraphDriver,GraphMode,err : integer;
{-------------------------------}
```

[123] Set back color [сэт бэк кало] – установить цвет фона.
[124] Color [кало] – цвет.
[125] Set fill style [сэт фил стайл] – установить стиль заполнения.
[126] Pattern [патен] – образец.
[127] Set line style [сэт лайн стайл] – установить стиль линии.
[128] Thickness [сикнис] – толщина.
[129] Set palette [сэт палит] – установить палитру.

```
  Procedure Coord(x1,y1,x2,y2 : integer;
                  var xa,ya       : integer);
{Процедура для нахождения координат вершин очередного квадрата
путем пропорционального деления сторон предыдущего квадрата }
  Begin
    xa:=x1+round((x2-x1)*mu/(1+mu));
    ya:=y1+round((y2-y1)*mu/(1+mu));
  End;
{--------------------------------}
{Окончание описательной части; начало исполняемой части}
BEGIN
ClrScr; {Очистка экрана путем вызова процедуры модуля Crt}
GraphDriver:=Detect[130]; {Detect-константа модуля Graph, равна 0,
задает автоматический выбор драйвера графического адаптера}
Write('Ввести координаты квадрата и параметр деления квадрата');
Read(x[1], y[1], x[3], y[3], mu);
x[2]:=x[1]; y[2]:=y[3]; x[4]:=x[3]; y[4]:=y[1];
{Переход к графическому режиму}
InitGraph(GraphDriver,GraphMode,'D:\TP\BGI');
err:=GraphResult;
If GraphResult <> grOk
  Then begin
    WriteLn('Ошибка при включении графического режима', err);
    Halt;
  End;
SetBkColor(Blue[131]); {Задание цвета фона экрана}
SetColor(White[132]); {Задание текущего цвета}
SetLineStyle(0,0,3); {Задание текущего параметра линии}
ClearDevice; {Очистка графического экрана }
Rectangle(x[1],y[1],x[3],y[3]); {Построение исходного квадрата}
{Вычисление координат вершин очередного квадрата и изображение его.
Текущий цвет линий изменяется с изменением номера квадрата}
Sound[133](292);
Delay[134](200);
NoSound;
For j:=1 to 25 do
Begin
  For i:=1 to 3 do
    Coord(x[i],y[i],x[i+1],y[i+1],a[i],b[i]);
```

[130] Detect [дитект] – обнаруживать.
[131] Blue [блу] – синий.
[132] White [уайт] – белый.
[133] Sound [саунд] – звук.
[134] Delay [дилэй] – задержка.

```
  Coord(x[4],y[4],x[1],y[1],a[4],b[4]);
  If j <= 5
  Then begin
    SetColor(Green[135]);
    hz:=330;
  End;
  If (j > 5) and (j <=10)
  Then begin
    SetColor(Red[136]);
    hz:=440;
  End;
  If (j > 10) and (j <= 15)
  Then begin
    SetColor(Magenta[137]);
    hz:=349;
  End;
  If j > 15
  Then begin
    SetColor(Cyan[138]);
    hz:=294;
  End;
  xy[1].x:=a[1]; xy[1].y:=b[1];
  xy[2].x:=a[2]; xy[2].y:=b[2];
  xy[3].x:=a[3]; xy[3].y:=b[3];
  xy[4].x:=a[4]; xy[4].y:=b[4];
  xy[5].x:=a[1]; xy[5].y:=b[1];
  DrawPoly(5,xy);
  Sound(hz);
  Delay(200);
  NoSound;
  x[1]:=a[1]; y[1]:=b[1];
  x[2]:=a[2]; y[2]:=b[2];
  x[3]:=a[3];y[3]:=b[3];
  x[4]:=a[4];y[4]:=b[4];
End;
WriteLn('нажмите Enter');
Readln(c);
CloseGraph; {выход из графического режима}
END.
```

[135] Green [грин] – зеленый.

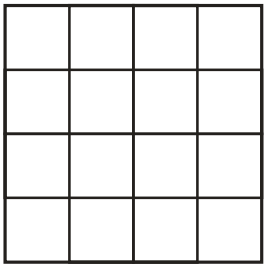
[136] Red [ред] – красный.

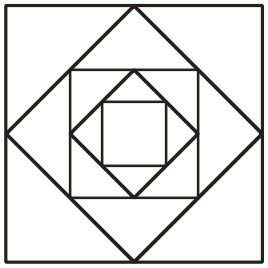
[137] Magenta [маджента] – лиловый.

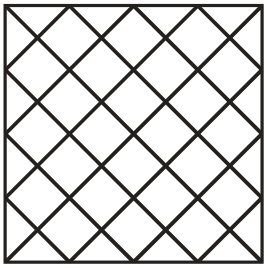
[138] Cyan [сиан] – бирюзовый.

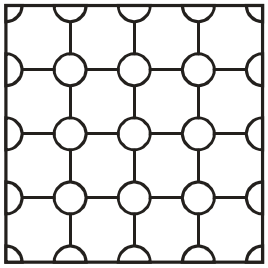
Задания

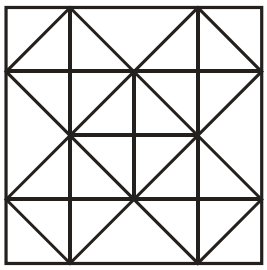
1. В окно редактора ТП загрузите модуль `Dmod` и программу `ModTest`, выполните их компиляцию, после чего пустите программу `ModTest` на выполнение.
2. В окно редактора ТП загрузите программу построения квадрата, пустите несколько раз ее на выполнение, изменяя координаты углов, параметр деления стороны, цвета линий.
3. Составьте программу для построения и изображения рисунков, приведенных в вариантах 1-8.

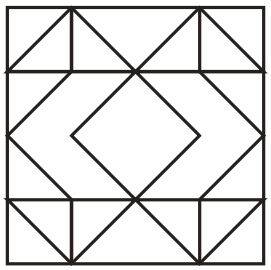

Вариант 1

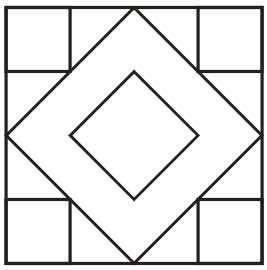

Вариант 2

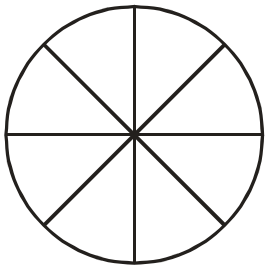

Вариант 3

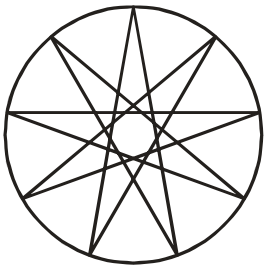

Вариант 4

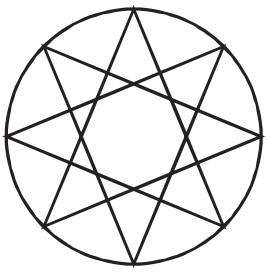

Вариант 5

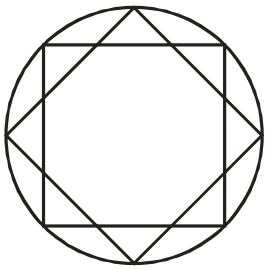

Вариант 6

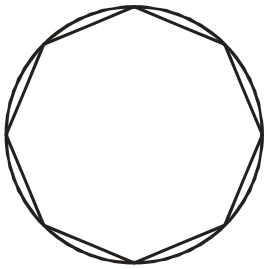

Вариант 7


Вариант 8


Вариант 9


Вариант 10


Вариант 11


Вариант 12

4. Составьте программу для выполнения следующего задания. Задается натуральное число R. Требуется построить фигуры, изображенные в вариантах 9-12. Каждая фигура образована окружностью радиуса R и восемью точками, являющимися вершинами правильного многоугольника, вписанного в окружность и соединенных между собой: вариант 9 через три точки, вариант 10 через две точки, вариант 11 через одну точку, вариант 12 – последовательное соединение точек.

Контрольные вопросы

1. Дайте определение модуля и охарактеризуйте его назначение.
2. Какова структура модуля?

3. Как оформляется заголовок модуля, и для каких целей он используется?
4. Что размещается в интерфейсной части модуля?
5. Что размещается в разделе реализации модуля?
6. Как оформляется описание подпрограмм в разделе реализации модулей?
7. Для каких целей может использоваться раздел инициализации модулей?

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

ЛИТЕРАТУРА

1. Глинский, Ярослав Николаевич. TurboPascal 7.0 и Delphi : Учебное пособие / Я.Н. Глинский, В.Е. Анохин, В.А. Ряжская.—2-е изд. испр. и доп.—СПб: DiaSoft, 2003.—197 с.: ил.—Библиогр.: с.196-197.—ISBN 5-93772-093-8.
2. Епанешников, Алексей Михайлович. Программирование в среде Turbo Pascal 7.0.—4-е изд., испр. и доп. Новая редакция.—М.: Диалог-МИФИ, 1998.—366,[1] с.—ISBN 5-86404-116-5: 19.55.
3. Епанешников, Алексей Михайлович. Программирование в среде Turbo Pascal 7.0.—М.: ДИАЛОГ-МИФИ, 1995.—282,[7]с.: ил.—ISBN 5-86404-029-0: 14.00.
4. Епанешников, Алексей Михайлович. Программирование в среде TURBO PASCAL 7.0 / А.М.Епанешников, В.А.Епанешников.—4-е изд., испр. и доп.—М.: ДИАЛОГ-МИФИ, 2000.—367 с.: ил.—На обл. загл.: Turbo Pascal 7.0.—ISBN 5-86404-116-5: 53.00.
5. Епанешников, Алексей Михайлович. Программирование в среде Turbo Pascal 7.0.— / 3-е изд.,стериотип.—М.: ДИАЛОГ-МИФИ, 1995.—282,[7]с.: ил.—ISBN 5864040290: 14000.
6. Епанешников, Алексей Михайлович. Программирование в среде Turbo Pascal 7.0.—3.изд., стереотип.—М.: Диалог-МИФИ, 1996.—282с.: ил.—ISBN 5-86404-029-0: 14.00.
7. Зеленяк О.П. Практикум программирования на Turbo Pascal: Задачи, алгоритмы и решения / О. П. Зеленяк.—2-е изд., испр. и доп.—М.; СПб.; Киев: DiaSoft, 2002.—310 с.: ил.—Библиогр.: с. 308. - Предм. указ.: с. 309.—ISBN 5-93772-059-8: 92.40.
8. Зубов, Валерий Сергеевич. Программирование на языке TURBO PASCAL (версии 6.0 и 7.0.—3-е изд., испр.—М.: Филинъ, 1997.—320 с.: табл.—ISBN 5-89568-049-6: 28.00.
9. Зуев Е.А. Turbo Pascal: Практ. программирование.—М.: Стрикс, 1997.—334с.: ил., табл.—ISBN 5-85572-189-2: 15.00.
10. Зуев, Евгений Александрович. Язык программирования Turbo Pascal 6.0.—М.: Унитех, 1992.—298,[6]с.: ил.—(Мир Turbo Pascal; Вып.1).—ISBN 5822300014: 1100.00.
11. Кассера, Винфрид. Turbo Pascal 7.0: Пер. с англ. / В. Кассера, Ф. Кассера.—СПб: DiaSoft, 2003.—425 с.: ил.—Пред. указ: с.419-425.—ISBN 5-93772-097-0.
12. Культин, Никита. Turbo Pascal в задачах и примерах / Никита Культин.—СПб. и др.: БХВ-Петербург, 2002.—256 с.: ил.—ISBN 5-8206-0061-4: 54.90.
13. Марченко, Александр Иванович. Программирование в среде Turbo Pascal 7.0 / А.И. Марченко, Л.А. Марченко.—7-е изд.—Киев: Век+, 2003.—458 с.: ил.—ISBN 966-7140-32-6.
14. Немнюгин, Сергей Александрович. Turbo Pascal: Учебник / С. А. Немнюгин.—СПб. и др.: Питер, 2002.—491 с.: ил.—ISBN 5-8046-0137-7: 84.00.

15. Немнюгин, Сергей Андреевич. Turbo Pascal: Практикум / С. А. Немнюгин.—СПб. и др.: Питер, 2001.—253 с.: ил.—ISBN 5-272-00068-4: 47.25.
16. Окулов, Станислав Михайлович. Основы программирования / С. Окулов.—М.: БИНОМ. Лаборатория знаний, 2004.—424 с.: ил., табл.—Библиогр. в тексте: с. 421-424.—ISBN 5-94774-003-6 (в пер.).
17. Пестриков, Виктор Михайловаич. Turbo Pascal 7.0: Изучаем на примерах / В.М. Пестриков, А.Н. Маслобоев.—СПб.: Наука и Техника, 2003.—368 с.: ил.—ISBN 5-94387-074-1.
18. Попов, Владимир Борисович. Turbo Pascal для школьников. Версия 7.0: Учеб.пособие для высших и сред. пед. учеб. заведений физ.-мат. профиля.—М.: Финансы и статистика, 1996.—463с.: ил., табл.—ISBN 5-279-01633-0: 30.00.
19. Рапаков, Георгий Германович. Turbo Pascal для студентов и школьников / Георгий Рапаков, Светлана Ржеуцкая.—СПб.: БХВ-Петербург, 2002.—349 с.: ил., табл.—(Основы информатики).—ISBN 5-94157-240-9: 111.36.
20. Сухарев, Михаил. Turbo Pascal 7.0: Теория и практика программирования / М. Сухарев; Под ред. М.В. Финкова.—СПб.: Наука и Техника, 2003.—574 с.: ил.—ISBN 5-94387-062-8: 140.00.
21. Фаронов, Валерий Васильевич. Turbo Pascal / В.В. Фаронов.—СПб.: БХВ-Петербург, 2003.—1037 с.: ил.+ 1 дискета.—(В подлиннике).—Библиогр.: с. 1035-1037.—ISBN 5-94157-295-6.
22. Фаронов, Валерий Васильевич. Турбо Паскаль: Нач. курс : Учеб. пособие / В. В. Фаронов.—7-е изд., перераб.—М.: Нолидж, 2001.—571 с.: ил.—Загл. обл. и корешка: TurboPascal 7.0.—ISBN 5-89251-054-9: 104.50.
23. Фаронов, Валерий Васильевич. Турбо Паскаль: Начальный курс: Учебное пособие / В. В. Фаронов.—7-е изд., перераб.—М.: Нолидж, 2002.—575 с.: ил.—На обл. загл. : Turbo Pascal 7.0.—ISBN 5-89251-054-9: 135.00.
24. Федоренко Ю. Алгоритмы и программы на Turbo Pascal: Учеб. курс / Ю. Федоренко.—СПб.: Питер, 2001.—240 с.—(Учебный курс).—ISBN 5-318-00102-5: 45.00.
25. Шелест, Вячеслав Дмитриевич. Программирование: [Учебное пособие] / Вячеслав Шелест.—СПб. и др.: БХВ-Санкт-Петербург, 2001.—584 с.: ил.—ISBN 5-94157-058-9: 123.20.
26. Шпак, Юрий Алексеевич. Turbo Pascal 7.0 на примерах / Ю.А. Шпак; Под ред. Ю.С. Ковнатюка.—Киев: Юниор, 2003.—490 с.: ил.+ 1 дискета.—ISBN 966-7323-30-7.

Составители: Закутский Сергей Николаевич,
Силкин Константин Юрьевич

Редактор: Тихомирова О.А.