Государственное образовательное учреждение высшего профессионального образования

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(государственный технический университет)

Кафедра «Проектирование вертолетов»

Маслов А.Д.

Алгоритмические языки и программирование на ЭВМ

Лабораторная работа №1

Основы работы в *Delphi*

Создание и отладка консольного приложения

Утверждено на заседании кафедры

5 июля 2007 г.

г. Москва

2007 г.

Введение

Настоящее пособие предназначено для студентов специальности «Самолето- и вертолетостроение» (специализация «Вертолетостроение»), изучающих дисциплину «Алгоритмические языки и программирование». В нем рассматриваются вопросы, характерные для выполнения всех лабораторных работ (порядок выполнения лабораторных работ, общие сведения о работе в среде программирования *Delphi*, отладка и тестирование программ и др.).

Цель работы – познакомить студентов со средой программирования *Delphi* и привить им практические навыки по созданию в этой среде простейших программ на алгоритмическом языке *Pascal* в форме консольного приложения.

1. Порядок выполнения и защиты лабораторных работ

Каждый студент учебной группы во время выполнения лабораторной работы обеспечивается рабочим местом, на котором установлен работающий персональный компьютер. После ознакомления с основами работы в среде программирования *Delphi* каждый студент перед началом выполнения лабораторной работы №1 должен сформировать в папке с именем, соответствующем номеру учебной группы, индивидуальную папку, в которой будут храниться все разработанные им программы.

В течение каждой лабораторной работы студент обязан написать текст программы в соответствии с индивидуальным заданием, создать файл в среде программирования *Delphi*, содержащий текст программы, выполнить отладку и тестирование программы, получить необходимый результат и защитить его. Индивидуальное задание выдается преподавателем заблаговременно для предварительной подготовки к лабораторной работе (кроме работы №1).

При разработке индивидуальных заданий использованы материалы, предоставленные К.М. Тихоновым (кафедра 701), и разработки кафедры 806.

Документом, удостоверяющим факт выполнения лабораторной работы в полном объеме, служит отчет. Отчет составляется в рабочей тетради по лабораторным работам и должен содержать следующие пункты:

1. Условие задачи.
2. Подготовленный в соответствии с заданием текст программы.
3. Тестовый пример, содержащий исходные данные и просчитанный для них вручную результат.

Отсутствие в отчете на момент начала лабораторной работы любого из пунктов влечет за собой недопуск к выполнению лабораторной работы. Наличие правильно, аккуратно и полностью оформленного отчета, отлаженной и протестированной программы является основанием для защиты лабораторной работы.

При защите лабораторной работы студент предоставляет:

1. Отчет по данной лабораторной работе.

2. Отлаженную программу, содержащуюся на рабочем месте в компьютере в индивидуальной папке студента.

3. Исходные данные и результат расчета на экране монитора, полученные при запуске программы.

В ходе защиты лабораторной работы студент должен свободно ориентироваться в алгоритме и тексте составленной им программы, уметь прокомментировать любой ее фрагмент и объяснить полученный результат. Кроме того, необходимо владеть теоретическим материалом, соответствующим тематике выполняемой лабораторной работы.

2. Общие сведения о среде программирования *Delphi*

Программный продукт под названием *Delphi* — это среда разработки программ, ориентированных на работу в операционной системе *MS Windows*. Слово *Delphi* – это название города в Древней Греции, в котором пророчествовали оракулы. Такое название было выбрано разработчиками *Delphi* для того, чтобы подчеркнуть способность программ, создаваемых в *Delphi*, взаимодействовать с базами данных *Oracle*. В качестве языка программирования в *Delphi* используется язык *Object Pascal*.

Язык *Object Pascal* является развитием языка программирования *Pascal*, названного в честь знаменитого французского математика Блеза Паскаля. Язык *Pascal* был создан профессором Цюрихского университета Никлаусом Виртом в 1971 г. как учебный язык компьютерного программирования. Хорошая структурность языка помогает привить начинающим программистам правильные навыки программирования. В результате *Pascal* быстро получил широкую популярность и стал основным учебным языком во многих университетах, как в США, так и во всем мире. Благодаря богатым функциональным возможностям, легкости составления программ и высокой скорости компиляции *Pascal* стал интенсивно использоваться для создания коммерческих программ.

Компания *Borland* была разработчиком популярной версии этого языка – *Turbo Pascal*. По мере развития операционной системы *Windows* и распространения концепции объектно-ориентированного программирования язык *Pascal* был естественным образом расширен до *Turbo Pascal for Windows* и *Object Pascal for Windows*.

В основе идеологии *Delphi* лежит технология визуального проектирования и событийного программирования (программирования процедур обработки событий), применение которых позволяет существенно сократить время разработки и облегчить процесс создания приложений (программ, работающих в *Windows*). Первая версия *Delphi* была выпущена фирмой *Borland* в 1995 г. Версии программного продукта *Delphi* 5 и *Delphi* 7, в среде которых студенты выполняют лабораторные работы, были выпущены соответственно в 1999 г. и 2002 г.

Среда *Delphi* представляет собой интегрированную оболочку, в которую входит набор

специализированных программ, ответственных за разные этапы создания приложения.

Текст программы готовится в среде *Delphi* с помощью встроенного редактора исходных текстов. Этот редактор специализирован. Он отличается гибкими возможностями цветового выделения различных элементов текста программы (ключевых слов, названий, операций и строк) и предоставляет возможность быстрого ввода часто встречающихся конструкций.

1.1. Запуск *Delphi*

Delphi, как и любая программа *Windows*, может быть запущена двумя способами:

- выбором команды главного меню *Windows*;
- щелчком на ярлыке, который идентифицирует *Delphi* и располагается на рабочем столе. Ярлык для *Delphi* 5 имеет вид  , для *Delphi* 7 -  .

Для запуска *Delphi* 5 первым способом необходимо в главном меню *Windows*, которое открывается нажатием кнопки ***Пуск*** на панели задач, выбрать строку ***Программы***, затем в появившемся списке выбрать строку ***Borland Delphi* 5**, а в следующем списке следует выполнить щелчок на строке ***Delphi* 5** (рис. 2.1). Аналогичным образом запускается и *Delphi* 7.

Удобнее запускать программу щелчком на ярлыке, который предварительно был создан и расположен на рабочем столе (рис. 2.1).

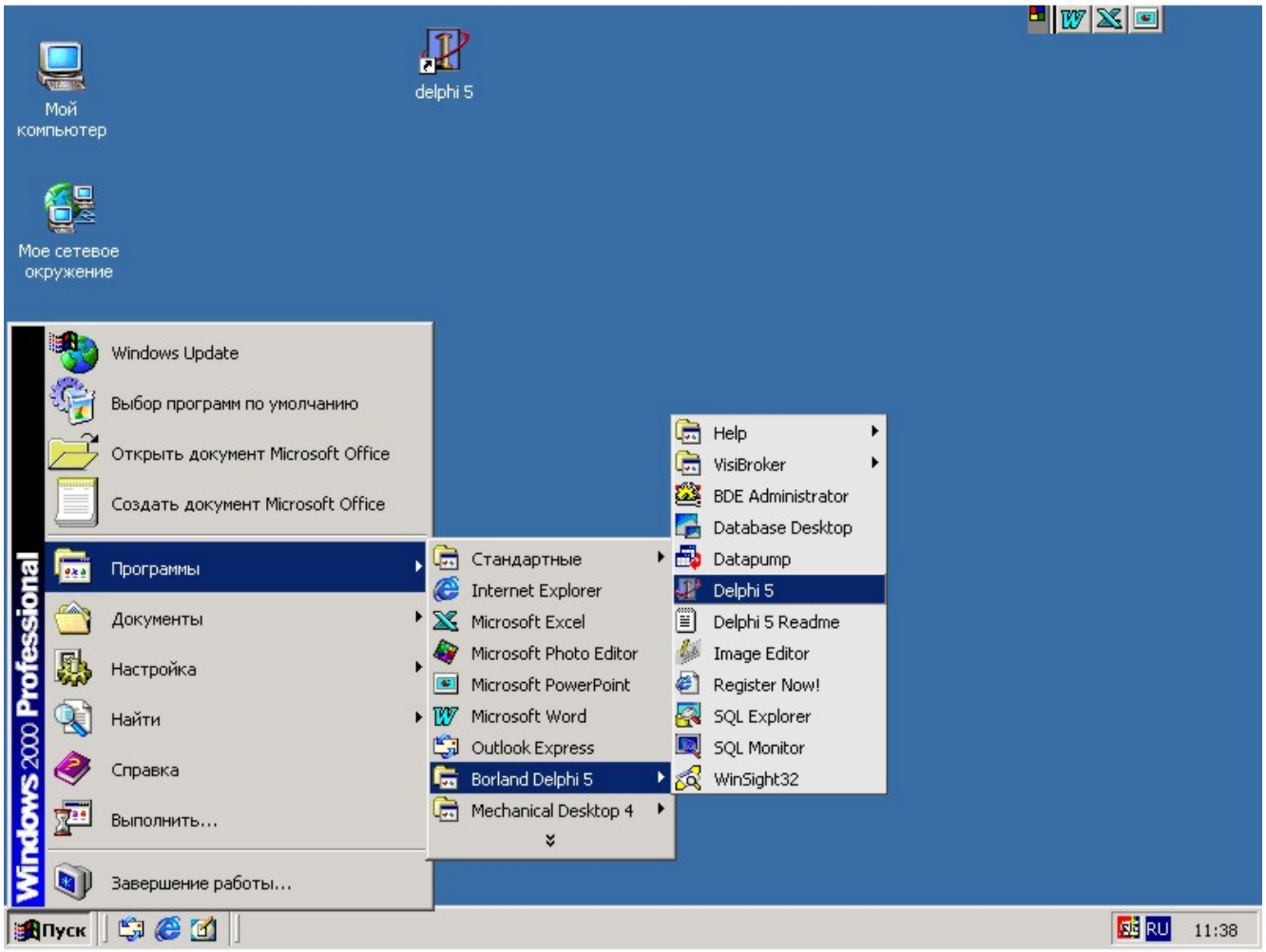


Рис. 2.1. Запуск *Delphi* 5 выбором команды главного меню Windows

2.2. Начало работы в Delphi

Вид экрана после запуска *Delphi* 5 показан на рис. 2.2. На экране появляются четыре окна: главное окно *Delphi*, окно формы *Forml*, окно инспектора объектов *Object Inspector* и окно встроенного редактора кода *Unitl.pas*, которое почти полностью закрыто окном формы.

В главном окне *Delphi* (рис. 2.3) находится меню команд, панель инструментов и палитра компонентов. Окно формы *Forml* представляет собой заготовку (макет) окна разрабатываемого приложения. Окно *Object Inspector* (Инспектор объектов) позволяет изменять свойства (характеристики) объектов: формы, командных кнопок, полей ввода и т. д. После запуска *Delphi* в диалоговом окне инспектора объектов *Object Inspector* находятся свойства формы *Forml*.

Окно редактора кода, которое можно увидеть, отодвинув в сторону окно формы или нажав клавишу ***<F12>***, содержит сформированный *Delphi* шаблон текста (кода) программы. С помощью редактора кода в среде *Delphi* готовится исходный текст программы.

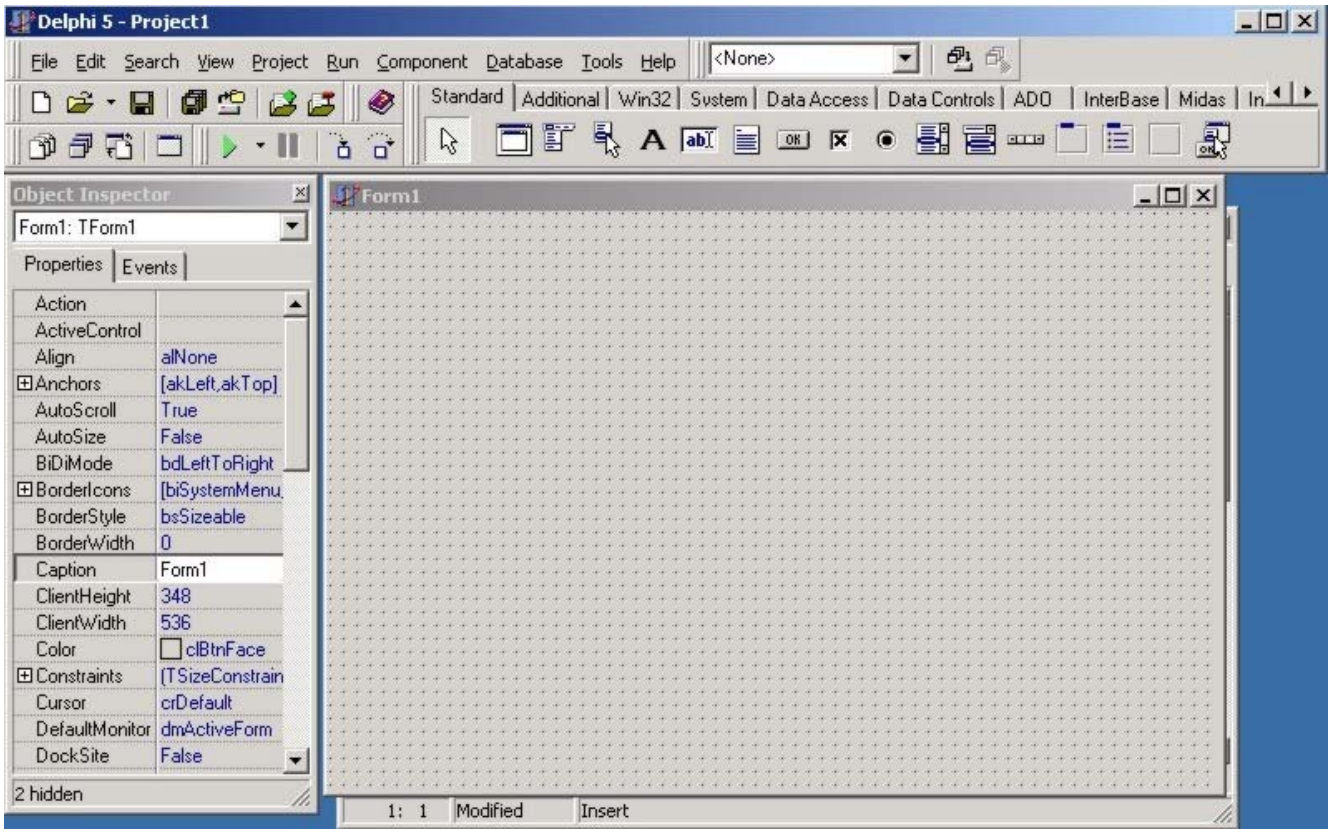


Рис. 2.2. Вид экрана после запуска *Delphi* 5

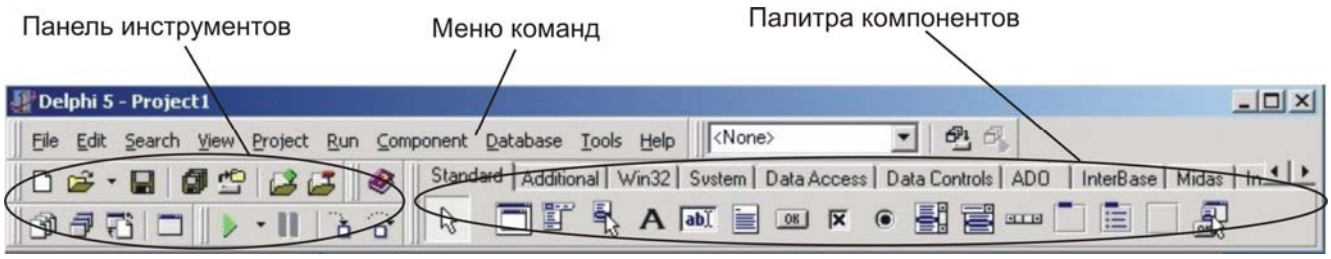


Рис. 2.3. Главное окно *Delphi* 5

Редактор кода *Delphi* автоматически выделяет ключевые слова языка программирования *Object Pascal* (*begin*, *end*, *procedure*, *const*, *var* и другие) полужирным шрифтом, что делает текст программы более выразительным, облегчает восприятие структуры программы. Помимо ключевых слов редактор кода выделяет комментарии. Как только программист напечатает символ начала комментария (открывающую фигурную скобку), так текст, стоящий после этой скобки, изменит свой вид. После того как программист введет закрывающую скобку комментария, текст, находящийся после этой скобки, приобретет обычный вид.

Редактор кода снабжен контекстно-зависимой справочной системой, которая во время набора текста программы автоматически выводит справочную информацию о процедурах и функциях языка программирования.

3. Консольное приложение

В начале 70-х годов, когда Н. Вирт разработал *Pascal*, операционная система *MS Windows* еще не существовала. Большинство программистов создавали свои программы для операционной системы *MS DOS*, в которой основным устройством ввода исходных данных была клавиатура, а устройством вывода — монитор, работающий в режиме отображения символьной информации (буквы, цифры и специальные знаки). Для тех, кто только начинает знакомство с основными операторами языка *Pascal*, в *Delphi* имеется возможность создания простых учебных программ в стиле *MS DOS*. Это так называемые консольные приложения.

3.1. Создание консольного приложения

Чтобы создать консольное приложение в *Delphi* 5, надо из меню ***File*** (Файл) выбрать команду ***New*** (Создать), выбрать значок ***Console Application*** (Консольное приложение) в диалоговом окне *New Items* (Создание программы) (рис. 3.1) и нажать кнопку ***OK***. В результате система *Delphi* 5 автоматически сгенерирует в текстовом редакторе шаблон программного кода будущего консольного приложения (рис. 3.2).

Создать консольное приложение можно также следующим образом. После запуска *Delphi* или выбора из меню ***File*** команды ***New Application*** (Новое приложение) необходимо закрыть ненужные окна: окно стартовой формы (*Form1*) и окно модуля приложения (*Unitl.pas*). При закрытии окна модуля приложения *Delphi* выводит запрос: *save changes to Unit1.pas?* (Сохранить изменения в *Unit1.pas* ?), на который надо ответить «Нет» (нажать кнопку ***No***). В результате на экране остается только главное окно *Delphi* и окно *Object Inspector*, которое тоже можно закрыть. После этого нужно из меню ***Project*** (Проект) выбрать команду ***View Source*** (Просмотр). В результате открывается окно *Projectl.dpr*, в котором находится сформированный *Delphi* шаблон главной процедуры приложения.

Чтобы создать консольное приложение в *Delphi* 7, надо из меню ***File*** (Файл) выбрать команду ***New*** (Создать), команду ***Other…***(Другой), выбрать значок ***Console Application*** (Консольное приложение) в диалоговом окне *New Items* (Создание программы) и нажать кнопку ***OK***.

В окне главной процедуры приложения можно набирать инструкции программы в соответствии с шаблоном, показанным на рис. 3.2.

Как следует из рис. 3.2, программный код консольного приложения начинается с заголовка программы, состоящего из ключевого слова ***program*** и автоматически сгенерированного системой имени программы *Project2*.

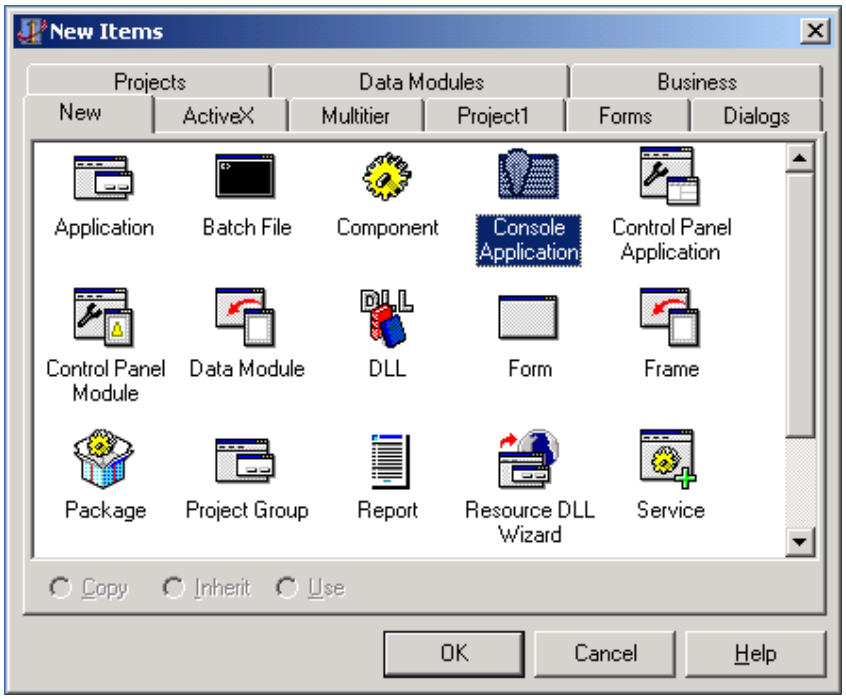


Рис. 3.1. Выбор значка Console Application для создания консольного приложения в среде Delphi 5

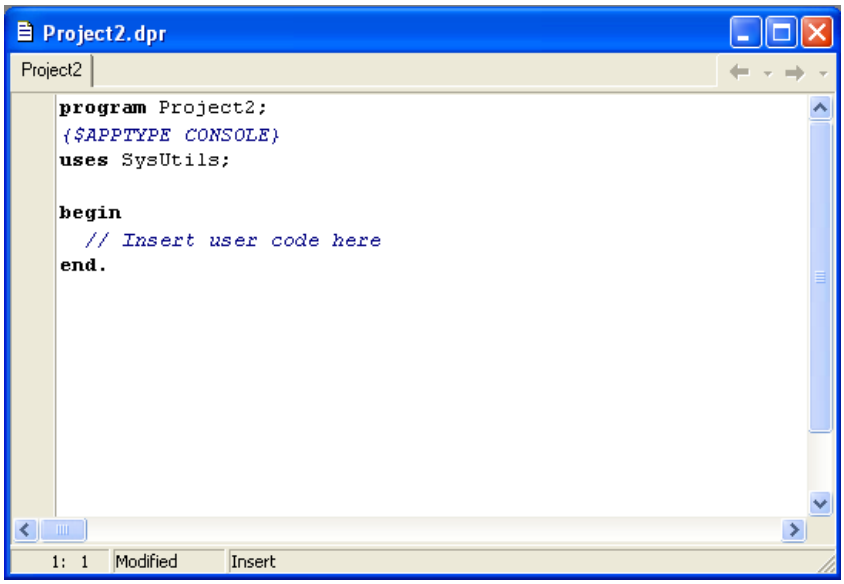


Рис. 3.2. Окно текстового редактора с кодом – заготовкой консольного приложения в среде *Delphi* 5

Далее в виде комментария следует директива компилятора. Она отличается от обычного комментария тем, что сразу за символом *{* (фигурная скобка) следует знак денежной единицы *$*, а следом — слово, задающее определенную настройку компилятора. Директива *{$APPTYPE CONSOLE}* говорит компилятору, что данная программа представляет собой консольное приложение. Хотя настройки компилятора можно задавать и непосредственно в среде *Delphi* 5, нужные настройки не всегда могут быть включены разработчиком вручную, поэтому лучше явно указывать их в тексте там, где они обязательно требуются.

Следующая строка задает подключение с помощью ключевого слова *uses* стандартного модуля *SysUtils*.

Далее идет собственно программа, для написания которой подготовлен логический блок, внутри которого система *Delphi* добавила комментарий *Insert user code here* (Вставьте сюда свой исходный текст). Начало раздела исполняемых инструкций программы (раздела операторов) отмечает инструкция ***begin***. Перед ***begin*** необходимо вставить раздел описания всех данных, используемых в программе. На конец программы указывает инструкция ***end***, за которой следует точка.

Следует отметить, что консольное приложение создается в *Delphi* и, следовательно, в *Windows*, где используется кодировка *ANSI*. Выполняется консольное приложение как программа *DOS*, в которой используется кодировка *ASCII*. Буквы русского алфавита в этих кодировках имеют разные коды. Это приводит к тому, что вместо сообщений на русском языке консольное приложение выводит «абракадабру». Поэтому выводимые на экран сообщения консольных приложений должны записываться латинскими буквами.

3.2. Сохранение программы

Перед первым запуском программы ее исходный текст необходимо сохранить. Файл консольного приложения имеет расширение *.dpr*. Для того чтобы сохранить этот файл, необходимо из меню ***File*** (Файл) выбрать команду ***Save Project As*** (Сохранить проект как). Если проект сохраняется впервые, то в ответ на команду ***Save Project As*** система *Delphi* выводит диалоговое окно *Save Project2 As*. В выпадающем окне Папка следует выбрать локальный диск (как правило C: или D:), в котором находится папка *Студенты* (рис. 3.3). Далее двойным щелчком на значке этой папки ее необходимо открыть, после чего окно *Save Project2 As* примет вид, показанный на рис. 3.4. Затем в окне следует выбрать папку с именем соответствующей учебной группы (см. рис. 3.4) и открыть ее. В папке с именем выбранной учебной группы в начале учебного года требуется создать персональную папку каждого студента группы, в которой на протяжении обучения по дисциплине «Алгоритмические языки и программировании» будут храниться все учебные программы студента. Для создания такой папки следует щелкнуть на значке  и в появившемся редактируемом поле *Новая папка* (рис. 3.5) буквами русского алфавита ввести фамилию и инициалы студента. Для ввода букв русского алфавита необходимо на клавиатуре одновременно нажать клавиши ***<Alt>*** и ***<Shift>***. Процесс создания персональной папки студента завершается после щелчка на значке этой папки. Далее двойным щелчком на значке персональной папки студента ее необходимо открыть, после чего окно *Save Project2 As* примет вид, показанный на рис. 3.6. В редактируемом поле *Имя файла* следует ввести имя программы *ProjectN*, где число *N* соответствует номеру выполняемой лабораторной работы.

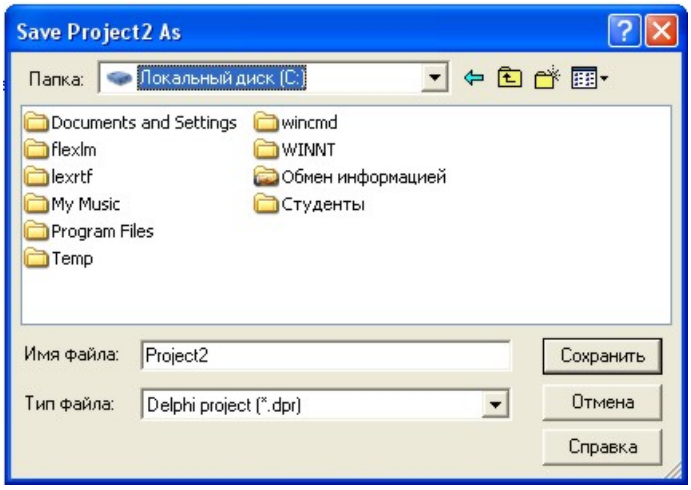


Рис. 3.3. Окно *Save Project2 As* с выбранным в выпадающем окне Папка локальным диском, в котором находится папка Студенты

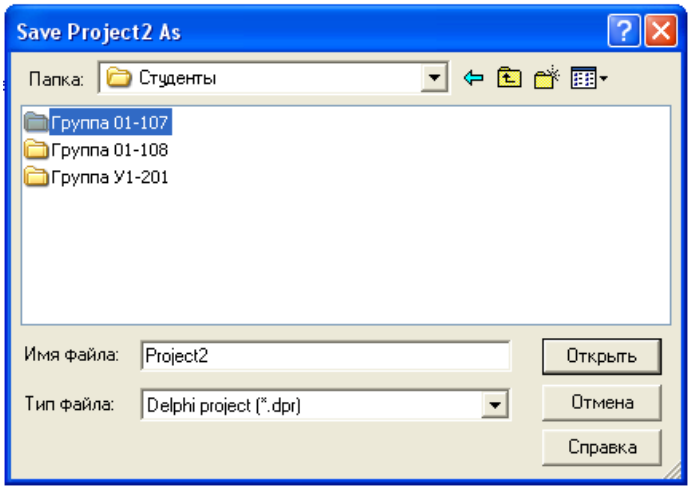


Рис. 3.4. Окно Save Project2 As с открытой папкой Студенты, содержащей папки учебных групп, проходящих обучение в компьютерном классе кафедры 102

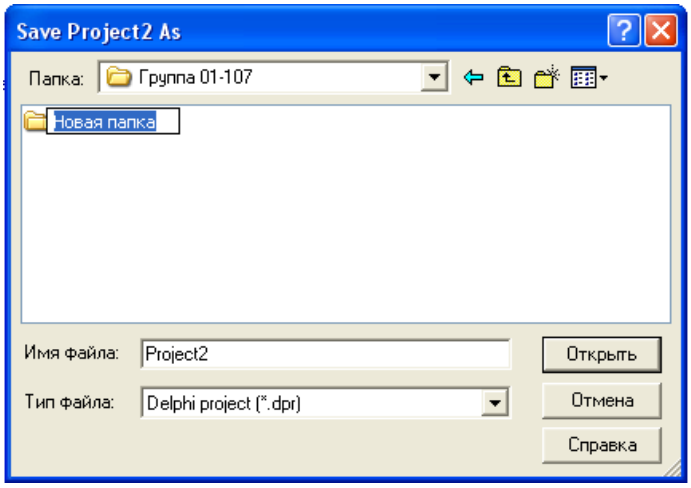


Рис. 3.5. Окно Save Project2 As с открытой папкой *Группа 01-107*, в которой следует создать персональную папку студента группы для хранения учебных программ

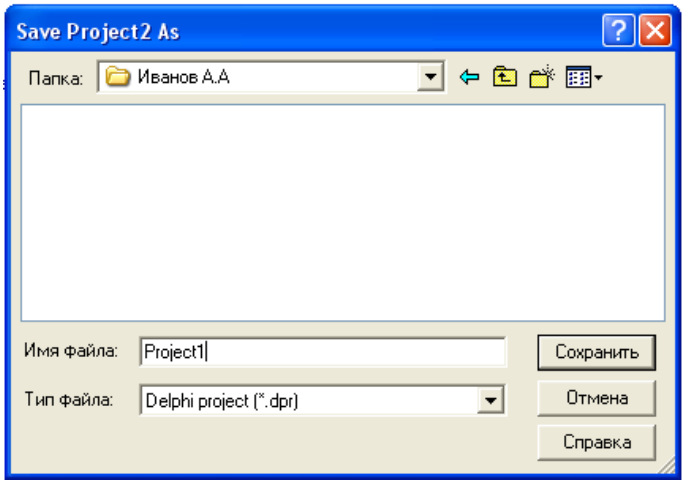


Рис. 3.6. Окно *Save Project2 As* с открытой персональной папкой студента, в которой сохраняется программа под именем *Project1*

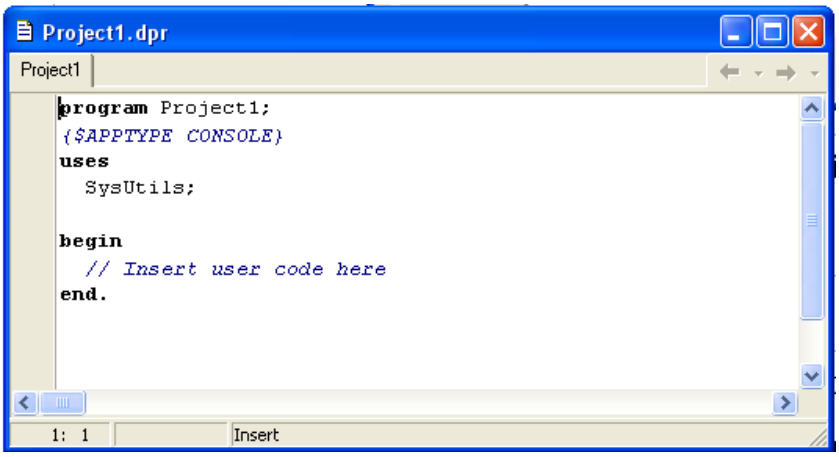


Рис. 3.7. Окно текстового редактора с кодом – заготовкой консольного приложения после сохранения программы под именем *Project1*

Теперь для сохранения программы под именем *Project1* необходимо нажать на выделенную кнопку ***Сохранить*** или нажать на клавишу ***<Enter>***. Окно текстового редактора с кодом – заготовкой консольного приложения после сохранения показано на рис. 3.7.

В ходе отладки программы после внесения в нее соответствующих изменений сохранение программы выполняется из меню ***File*** (Файл) командой ***Save*** (Сохранить) или щелчком на кнопке  , расположенной на панели инструментов.

3.3. Компиляция программы

Процесс компиляции заключается в переводе (трансляции) исходного текста программы, написанной на алгоритмическом языке высокого уровня, каким и является *Pascal*, в набор конкретных инструкций (команд) процессора. Этот процесс выполняется очень быстро с помощью специальной программы, называемой компилятором. **Компилятор – это программа, автоматически преобразующая (транслирующая, компилирующая) исходный код языка высоко-**

го уровня в машинный код и создающая таким образом выполняемый файл.

Компиляция созданного консольное приложение выполняется путем выбора команды ***Compile*** (Компилировать) из меню ***Project*** (Проект). Компилятор просматривает программу от начала. Во время компиляции текст программы автоматически проверяется на отсутствие синтаксических ошибок.

Если обнаруживается ошибка, то процесс компиляции приостанавливается и в нижней части редактора кода на специальной панели отображаются сообщения о найденных ошибках (*Errors*). Вместе с этим в окне редактора кода красным цветом выделяется строка, которая по мнению компилятора содержит синтаксическую ошибку (рис. 3.8). Если размер панели недостаточен, чтобы отобразить сообщения об ошибке в полном объеме, то в конце сообщения указываются три точки. В этом случае можно развернуть окно редактора кода на весь экран или поместить курсор мыши на слово *Error*, в результате чего появится всплывающее окно, содержащее весь текст сообщения об ошибке. Само сообщение (на английском языке) описывает ошибку достаточно подробно, чтобы можно было понять ее причину.

Следует обратить внимание, что строка, выделенная компилятором, не всегда содержит ошибку. Довольно часто ошибочной является инструкция, находящаяся в предыдущей строке. Если в программе есть только одна ошибка, компилятор выводит сообщение о двух. Первая ошибка — это первая от начала текста программы синтаксическая ошибка, обнаруженная компилятором. Наличие этой ошибки приводит к возникновению второй, фатальной ошибки (*Fatal Error*) — невозможности генерации файла исполняемой программы. При наличии нескольких ошибок двойным щелчком на строке с сообщением об ошибке система *Delphi* 5 переключится в редактор, где подсветит строку, в которой эта ошибка обнаружена.

В некоторых случаях кроме обнаруженных компилятором ошибок в нижней части окна редактора кода выводятся предупреждения (*Warnings*) или подсказки (*Hints*). Желательно устранять их источники, руководствуясь принципом «дыма без огня не бывает».

В табл. 3.1 приведены сообщения о наиболее типичных ошибках.

Если синтаксических ошибок в программе нет, компилятор создает исполняемый файл программы, который позже можно будет запустить из *Windows*. Имя исполняемого файла такое же, как и у файла консольного приложения, а расширение — *.exe*. *Delphi* помещает исполняемый файл в ту папку, где уже находится файл с текстом программы, имеющий расширение *.dpr*.

Таблица 3.1

Сообщения об ошибках

Сообщение компилятора	Вероятная причина
Undeclared identifier (Необъявленный идентификатор)	а) Используется переменная, не объявленная после ключевого слова **var** в разделе описания данных программы;
	б) Ошибка при написании имени объявленной переменной. Например, объявлена переменная *Summa*, а в тексте программы написано *Suma*;
	в) Ошибка при написании имени инструкции. Например, вместо ключевого слова ***const*** написано *conts*
Unterminated string (Незавершенная строка)	При записи строковой константы не поставлена завершающая кавычка
Incompatible types ... and ... (Несовместимые типы)	В инструкции присваивания тип выражения не соответствует или не может быть приведен к типу переменной, получающей значение выражения
Missing operator or semicolon (Отсутствует оператор или точка с запятой)	Не поставлена точка с запятой после инструкции программы

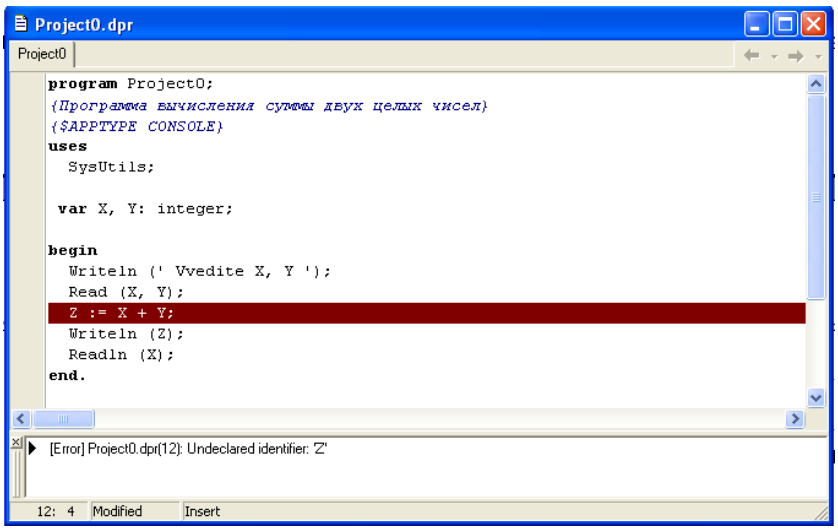


Рис. 3.8. Пример выделения компилятором строки программы, содержащей синтаксическую ошибку – *Необъявленный идентификатор Z*

3.4. Открытие и закрытие файла с программой в форме консольного приложения

Для продолжения разработки консольного приложения или внесения в текст программы некоторых изменений файл приложения необходимо открыть. С этой целью в меню ***File*** (Файл) следует выбрать команду ***Open Project*** (Открыть имеющийся проект) или команду ***Open*** (Открыть имеющийся файл). Последней команде эквивалентна кнопка  , расположенная на панели инструментов. Маленькая стрелка, направленная вниз и помещенная справа от кнопки, позволяет вывести список открывавшихся ранее файлов, из которых можно выбрать файл для повторного открытия.

Перед открытием выбранного консольного приложения уже открытый файл приложения закрывается. Если в нем были произведены изменения, на экране появляется информационное окно с вопросом, следует ли сохранить эти изменения.

В ответ на команду ***Open Project*** система *Delphi* выводит одноименное диалоговое окно *Open Project*. В выпадающем окне *Папка* следует выбрать локальный диск (C: или D:), в котором находится папка *Студенты* (рис. 3.9). Далее двойным щелчком на значке этой папки ее необходимо открыть, после чего окно *Open Project* примет вид, показанный на рис. 3.10. Затем следует выбрать папку с именем соответствующей учебной группы и открыть ее. В этой папке требуется выбрать персональную папку студента группы, в которой хранятся учебные программы студента, и открыть ее. В результате выполненных действий содержимое окна *Open Project* будет иметь вид, показанный на рис. 3.11.

Выбранное имя программы *ProjectN*, где число *N* соответствует номеру выполняемой лабораторной работы, отобразиться в редактируемом поле *Имя файла.* Теперь для открытия программы под выбранным именем необходимо нажать на выделенную кнопку ***Открыть*** или нажать на клавишу ***<Enter>***.

В ответ на команду ***Open*** система *Delphi* выводит одноименное диалоговое окно *Open*. Порядок действий по открытию файла с нужным консольным приложением в этом окне аналогично вышеизложенному.

Закрыть файл, находящийся в окне текстового редактора путем выбора в меню ***File*** (Файл) команды ***Close*** (Закрыть файл) или команды ***Close All*** (Закрыть все файлы). Если в файле были произведены изменения, на экране появляется информационное окно с вопросом, следует ли сохранить эти изменения.

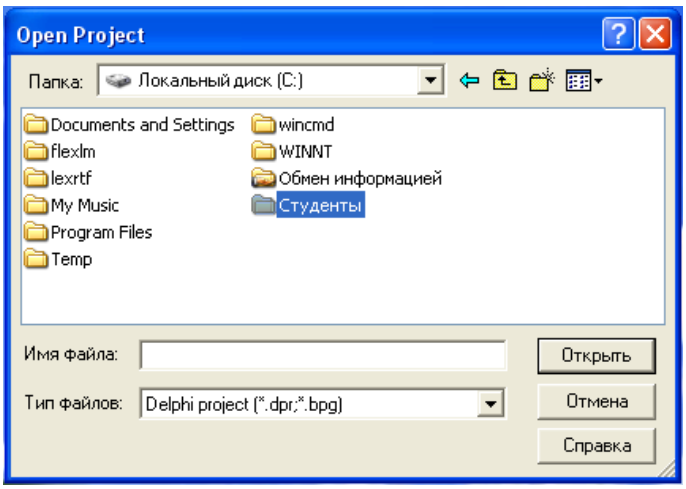


Рис. 3.9. Окно *Open Project* с выбранным в выпадающем окне Папка локальным диском, в котором находится папка Студенты

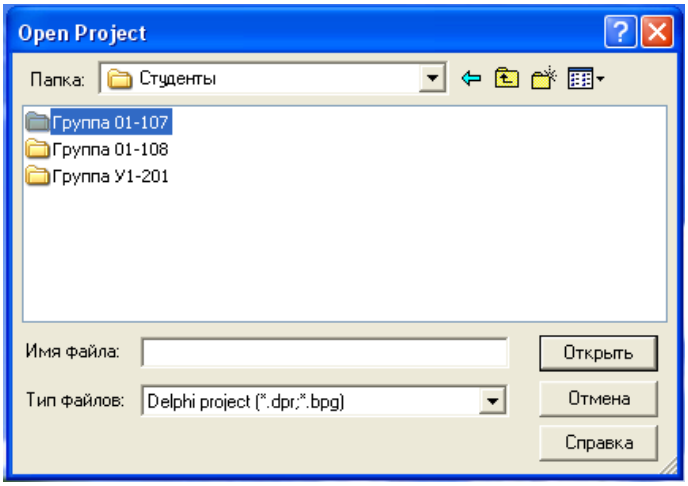


Рис. 3.10. Окно *Open Project* с открытой папкой Студенты, содержащей папки учебных групп, проходящих обучение в компьютерном классе кафедры 102

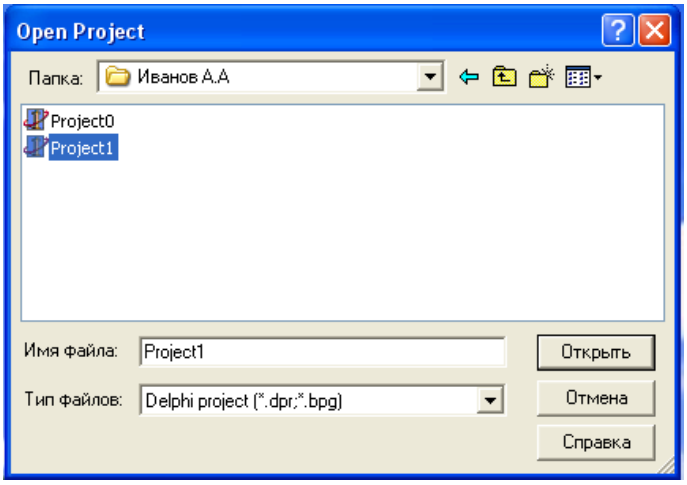


Рис. 3.11. Окно *Open Project* с открытой персональной папкой студента, в которой открывается программа под именем *Project1*

3.5. Запуск программы из среды программирования

Пробный запуск консольного приложения можно выполнить непосредственно из среды разработки, не завершая работу с *Delphi*. Чтобы откомпилировать и автоматически запустить данную программу достаточно выбрать из меню ***Run*** (Выполнение) команду ***Run*** (Выполнить). Можно также щелкнуть на кнопке  , расположенной на панели инструментов или нажать на клавиатуре клавишу ***<F9>***. При запуске консольного приложения (в случае отсутствия ошибок в процессе компиляции) на экране появляется стандартное окно консольного приложения в стиле программы *MS DOS*. Его можно перемещать, по экрану, развернуть на весь экран, свернуть или закрыть (в этом случае программа прекращает работу). На рис. 3.13 показано окно консольного приложения, созданного *Delphi*.

В режиме ввода данных программа переходит в режим ожидания, о чем свидетельствует мигающий курсор. Поэтому рекомендуется в тексте программы предусмотреть вывод на экран информационного сообщения с приглашением к вводу с клавиатуры исходных данных. Пользователь должен ввести запрашиваемую программой информацию и нажать клавишу ***<Enter>***. Результаты работы программы с помощью включенных в ее текст инструкций вывода на экран отображаются в окне консольного приложения.

3.6. Пример создания программы в форме консольного приложения

Для демонстрации возможностей алгоритмического языка *Pascal* и отработки технологии создания консольного приложения в среде *Delphi* 5 или *Delphi* 7, реализуем на практике программу сложения двух целых чисел.

В ходе работы программы с клавиатуры будут вводиться два целых числа, а программа рассчитает и выведет на экран их сумму.

Вначале работы следует создать заготовку консольного приложения (раздел 3.1) пи сохранить ее под именем *Project0* (раздел 3.2).

Далее в редакторе кода перед началом раздела исполняемых инструкций программы надо описать три переменные целого типа (***integer***): две их них предназначены для хранения значений, вводимых пользователем, а третья будет содержать результат сложения. Назовем переменные соответственно *X*, *Y* и *Z*. Начало раздела описания переменных отмечает инструкция ***var***.

В разделе исполняемых инструкций программы сначала с помощью процедуры ***Read*** производится ввод двух чисел с клавиатуры. Их значения записываются в переменные *X* и *Y*, затем с помощью оператора присваивания сумма значений этих переменных помещается в переменную *Z*, а процедура ***Writeln*** выводит значение переменной *Z* на экран. Последний оператор ***Readln*** нужен для того, чтобы после выполнения всех действий программа не сразу закрывала свое окно, а ожидала нажатия клавиши ***<Enter>***. Тогда программист сможем посмотреть на

результат своей работы.

Исходный текст программы с необходимыми комментариями выглядит следующим образом:

```
Program Project0;                  //Заголовок программы
{Программа сложения двух целых чисел}      //Комментарий о назначении программы
{$APPTYPE CONSOLE}                 //Инструкция компилятору, что приложение консольное
uses Sysutils;                     //Подключение стандартного модуля Sysutils

var X, Y, Z; integer;              //Описание переменных
begin                              //Начало раздела исполняемых инструкций
Writeln (' Vvedite X & Y ');       //Вывод на экран приглашения к вводу X и Y
Read(X, Y);                        //Ввод с клавиатуры значений переменных X и Y
Z := X + Y;          //Сложение X и Y и присваивание значения суммы переменной Z
Writeln(Z);          //Вывод на экран значения переменной Z
Readln (X)     //Фиктивный ввод для задержки на экране окна консольного приложения
end.           //Конец программы
```

После сохранения программы и ее запуска из среды программирования *Delphi* 5 путем щелчка на кнопке  или нажатия клавиши ***<F9>*** выполняется компиляция программы. При отсутствии синтаксических ошибок содержимое окна редактора кода с текстом программы примет вид, как показано на рис. 3.12, и на экране появится окно созданного консольного приложения (рис. 3.13) с текстом приглашения к вводу значений переменных (*Vvedite X, Y*). После ввода значений двух чисел, отделенных друг от друга пробелом в результате нажатия клавиши ***<Spacebar>***, например:

2 2

и последующего нажатия клавиши ***<Enter>***, программа выведет на экран результат расчета:

4

и будет ожидать еще одного нажатия клавиши ***<Enter>***, чтобы завершить свою работу.

На рис. 3.13 приведено содержимое окна консольного приложения с результатами работы программы сложения двух целых чисел.

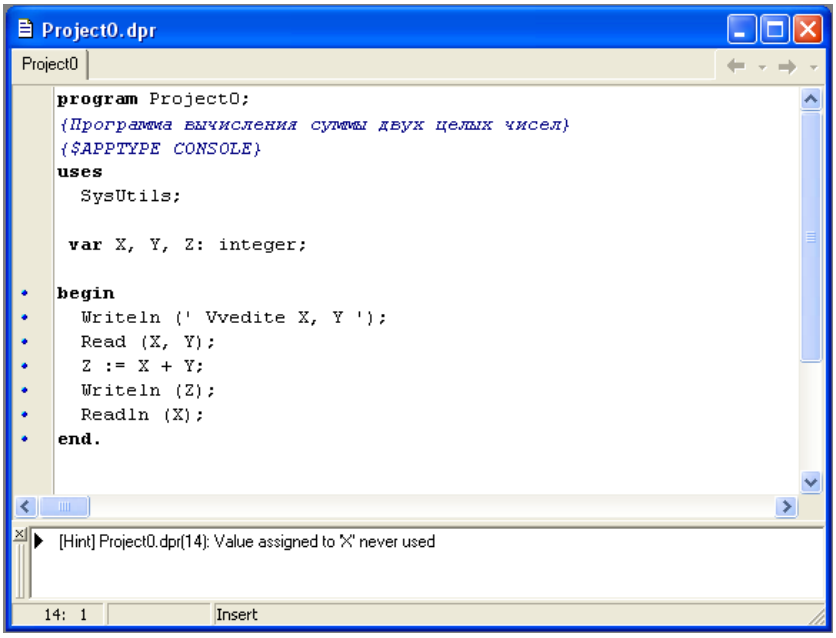


Рис. 3.12. Окно редактора кода с текстом программы после компиляции программы

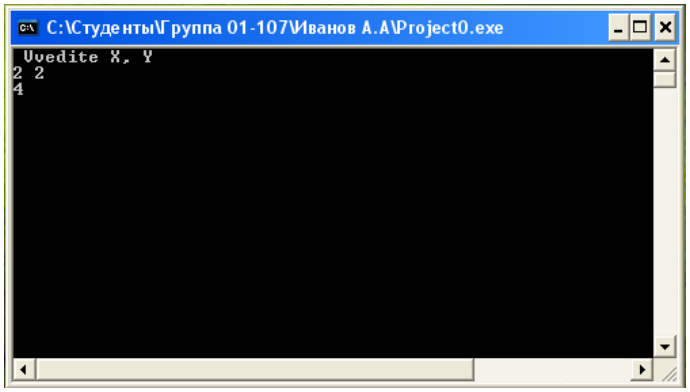


Рис. 3.13. Окно консольного приложения с результатами работы программы

4. Отладка и тестирование программы

Программ без ошибок не существует. Синтаксические ошибки, связанные с неверным вводом команд в редакторе кода, неверной записью идентификаторов и другими некорректными действиями, можно обнаружить простым анализом исходного текста, и они почти всегда фиксируются компилятором *Delphi*.

Ошибки, связанные с неверной реализацией алгоритма (например, когда вместо символа > ошибочно введен символ <), могут привести к возникновению ошибок уже во время работы программы. Кроме того, неверная реализация исходного алгоритма не обязательно приводит к нарушению работоспособности приложения, но может повлечь за собой выдачу неверных результатов или выполнение ошибочных действий.

Во время работы приложения могут возникать ошибки, которые называются ошибками времени выполнения (*run-time errors*). В большинстве случаев причинами этих ошибок являются неверные исходные данные или недопустимые арифметические операции (например, деление на ноль, вычисление квадратного корня из отрицательного числа и т. п.).

Процесс поиска и устранения ошибок в программе называется **отладкой программы**.

Ошибки допускает любой программист: и начинающий, и имеющий тридцатилетний опыт. Различаются только причины этих ошибок. Начинающий разработчик редко берется за большие проекты, и его ошибки связаны, как правило, с плохим знанием операторов языка *Pascal*. Профессиональный программист обычно не допускает ошибок на этапе кодирования. Но он может допускать ошибки на начальных этапах работы, когда будущая программа только проектируется и не все детали алгоритма до конца ясны. При этом очень сложно заранее предусмотреть скрытые взаимосвязи между различными модулями программы, насчитывающей сотни тысяч операторов. Поэтому приходится прибегать к различным программистским приемам, позволяющим повысить надежность приложения и выявлять ошибки как можно раньше. Это служит залогом значительного снижения трудозатрат. Если неверный оператор скрыт в «недрах» программы, и уже написано много нового исходного текста, то изменение работы этого оператора может повлечь за собой лавинообразный эффект влияния на все последующие действия.

Система *Delphi* предоставляет различные средства отладки программ, которые ориентированы на программистов любой квалификации. К их числу относятся выполнение программы по шагам (трассировка программы), возможность контроля значений переменных, использование точек прерывания (точек останова) выполнения программы и др.

Тестирование – это процесс выполнения программы, целью которого является выявление ошибок. Цель тестирования всякой программы состоит в том, чтобы гарантировать ее нормальную работу во всех предполагаемых практических ситуациях. Программисты-новички должны осознавать тот факт, что программу нельзя считать правильно работающей, пока это не

доказано. Если в ходе тестирования выявляется факт наличия ошибки в программе, то она вновь должна быть подвергнута отладке с целью выявления причины обнаруженной ошибки.

Начинающие программисты часто не понимают, каким образом можно определить, что программа прошла тестирование достаточно полно. В этом вопросе следует руководствоваться одному полезному практическому принципу, смысл которого сводится к тому, что тестирование программы необходимо продолжать до тех пор, пока каждая инструкция в ней не будет выполнена хоты бы один раз. Тестовые данные должны обеспечить проверку каждой ветви алгоритма программы. Значительно облегчить процесс тестирования может правильный подбор тестовых данных. При использовании тестовых данных необходимо иметь какой-то способ проверки правильности полученных результатов расчета на ЭВМ. К числу используемых на практике способов такой проверки относятся следующие: вычисление результатов вручную; получение результатов из справочных данных, документации или другого проверенного независимого источника; получение результата на ЭВМ с помощью некоторой программы, составленной на основе другого метода и алгоритма расчета.

Процесс тестирования программы можно разделить на три этапа:

1. Проверка в нормальных условиях; выполняется на основе данных, которые характерны для реальных условий функционирования программы.

2. Проверка в экстремальных условиях; выполняется на основе данных, которые включают граничные значения области изменения входных переменных, которые должны восприниматься программой как правильные данные.

3. Проверка в исключительных ситуациях; выполняется на основе данных, значения которых лежат за пределами допустимой области изменения.

Каждый из вышеперечисленных этапов проверки должен гарантировать получение верных результатов при правильных входных данных и выдачу сообщений об ошибке при неправильных данных.

5. Задание к лабораторной работе №1

1). В соответствии с указаниями в разделах 3.1 и 3.2 настоящего пособия создать в среде программирования *Delphi* 5 или *Delphi* 7 консольное приложение под именем *Project1* и в редакторе кода *Delphi* ввести приведенный ниже текст программы решения неполного квадратного уравнения (с необходимыми комментариями).

2). Выполнить компиляцию программы в соответствии с указаниями в разделе 3.3 и после устранения возможных синтаксических ошибок запустить программу на выполнение (см. раздел 3.5).

3). Провести отладку и тестирование программы в соответствии с рекомендациями в разделе 4.

Текст программы на языке *Pascal*:

```
program Project1;                    { Заголовок программы }
    { программа решения неполного квадратного уравнения ax^2 + b = 0 }
{ Раздел описаний }
    const
     eps=1E-6;                              { задание константы eps=10^-6 }
    var
     a,b,x1,x2,s: real;                     { объявление переменных }
{ Раздел операторов }
    begin
     writeln(' Vvedite a i b ');            {приглашение к вводу}
     read(a,b);                             {ввод исходных данных}
     if abs(a)<eps then                     {если a=0, тогда }
      writeln(' RESHENIA NET: a = 0 ')      { выводим сообщение, что решения нет }
      else begin                           { иначе (a<>0): проверяем знаки a и b }
       s:=b/a;        { вычисление значения вспомогательной переменной S = b/a }
       if s>0 then             { если s>0 (a и b имеют одинаковые знаки), тогда }
    writeln('RESHENIA NET: a i b imeyut odinakovie znaki') { выводим сообщение,
                                                          что решения нет }
       else begin                           { иначе: }
        x1:=sqrt(abs(s)); x2:=-x1;          {вычисляем действительные корни x1, x2}
        writeln('a=',a,'b=',b);             {выводим исходные данные и  }
        writeln('x1=',x1,' x2=',x2);         { результаты расчета - корни x1, x2}
       end;
      end;
     read(a);                          {фиктивный ввод для сохранения окна}
    end.                               {конец программы}
```

Примечание

В связи с приближенным представлением в памяти ЭВМ вещественных чисел при обеспечении недостаточной точности их представления равенство типа $a = 0$ может быть не выполнено. Поэтому программа не будет сообщать об ошибке при введенном в качестве делимого нулевого значения a. Корректной будет программа, в которой вводится константа допустимой точности *eps*, например $eps = 10^{-6}$. Тогда в приведенном выше тексте, если переменная a по модулю меньше *eps*, то в первом условном операторе ***if*** она трактуется как нуль.

Указанный прием можно использовать и в случае возникновения любой другой подобной ситуации.

Список использованных источников

1. Бобровский С. *Delphi* 5: учебный курс – СПб.: Изд-во «Питер», 2000, 640 с.

2. Культин Н.Б. Программирование на *Object Pascal* в *Delphi* 5. – СПб.: БХВ – Санкт-Петербург, 2000, 464 с.

3. Тассел Ван Д. Стиль, разработка, эффективность, отладка и испытание программ: Пер. с англ. – М.: Мир, 1985, 332 с.

Государственное образовательное учреждение высшего профессионального образования
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(государственный технический университет)

Кафедра «Проектирование вертолетов»

Маслов А.Д.

Алгоритмические языки и программирование на ЭВМ

Лабораторная работа № 2

Простейшие конструкции языка *Pascal*

Утверждено на заседании кафедры

5 июля 2007 г.

г. Москва
2007 г.

Введение

Настоящее пособие предназначено для студентов специальности «Самолето- и вертолетостроение» (специализация «Вертолетостроение»), изучающих дисциплину «Алгоритмические языки и программирование».

Целью работы является изучение простейших конструкций алгоритмического языка *Pascal* и приобретение практических навыков по созданию и отладке в среде *Delphi* программ с линейным вычислительным процессом.

1. Основные сведения о языке *Pascal*

Язык программирования относится к группе формальных языков, для которых в отличие от естественных языков однозначно определены синтаксис и семантика. Описание синтаксиса языка включает определение алфавита и правил построения различных конструкций языка из символов алфавита и более простых конструкций.

1.1. Общая характеристика языка

Алфавит языка *Pascal* содержит следующие символы:

- прописные, строчные буквы латинского алфавита (*A..Z*, *a..z*). При этом существенно то, что строчные и прописные буквы не различаются.
- арабские цифры (0..9);
- специальные знаки, состоящие из одного и двух символов:

  , . + - * / = : ; _ < > [ ] { } ( ) $ # <= >= := (* *);
- служебные слова (эти сочетания считаются единым целым и их нельзя использовать в программе в другом качестве):

and, array, begin, const, div, do, downto, else, end, for, function, goto, if, mod, not, of, or, procedure, program, repeat, then, to, type, until, uses, var, while, with и др.

Из символов алфавита в соответствии с правилами синтаксиса строят различные конструкции. Простейшей из них является конструкция *Идентификатор*. Эта конструкция используется для обозначения имен переменных, массивов, процедур, функций и т. п. В языке *Pascal* идентификатор представляет собой последовательность букв латинского алфавита (включая символ подчеркивания) и цифр, которая обязательно начинается с буквы, например: aaaa, b121, Parametr_l, и т. п.

Русские буквы в конструкциях языка использовать нельзя. Они допускаются только при определении строковых и символьных данных.

1.2. Структура программы

Программа на Borland Pascal состоит из трех частей: заголовка, раздела описаний и раздела операторов.

Заголовок программы не является обязательным, он состоит из служебного слова *program* и идентификатора - имени программы.

Раздел описаний содержит описания всех используемых программой данных, подпрограмм и т. д.

Раздел операторов заключается в так называемые операторные скобки *begin ...end* и заканчивается точкой. Между операторными скобками записывают управляющие операторы программы, которые разделяют специальным знаком - точкой с запятой «;».

Для пояснения в тексте программы возможны комментарии, которые обычно помещают в фигурные скобки.

Ниже в качестве примера приведен текст программы решения неполного квадратного уравнения:

```
program Project1;                    { Заголовок программы }
    { программа решения неполного квадратного уравнения ax^2 + b = 0 }
{ Раздел описаний }
    const
     eps=1E-6;                              { задание константы eps=10^-6 }
    var
     a,b,x1,x2,s: real;                     { объявление переменных }
{ Раздел операторов }
    begin
     writeln(' введите a и b ');            {приглашение к вводу}
     read(a,b);                             {ввод исходных данных}
     if abs(a)<eps then                     {если a=0, тогда }
      writeln(' RESHENIA NET: a=0 ')        { выводим сообщение, что решения нет }
      else begin                           { иначе (a<>0): проверяем знаки a и b }
       s:=b/a;         { вычисление значения вспомогательной переменной S = b/a }
       if s>0 then              { если s>0 (a и b имеют одинаковые знаки), тогда }
    writeln('RESHENIA NET: a i b imeyut odinakovie znaki') { выводим сообщение,
                                                             что решения нет }
       else begin                           { иначе: }
        x1:=sqrt(abs(s)); x2:=-x1;          {вычисляем действительные корни x1, x2}
        writeln('a=',a,'b=',b);             {выводим исходные данные и  }
        writeln('x1=',x1,'x2=',x2);         { результаты расчета - корни x1, x2}
       end;
      end;
     read(a);                           {фиктивный ввод для сохранения окна}
    end.                                {конец программы}
```

2. Простейшие конструкции языка *Pascal*

К простейшим конструкциям языка *Pascal* относятся способы представления данных стандартного типа, конструкции выражений, оператор присваивания и операторы ввода-вывода, без которых не обходится ни одна программа.

2.1. Константы и переменные. Типы переменных

Любая программа оперирует с некоторыми данными, используемыми в расчетах или определяющими последовательность выполнения действий. Все данные, с которыми оперирует программа на языке *Pascal*, должны быть описаны.

Данные в программе могут присутствовать в виде констант и переменных.

Константы. Константы - определяются один раз и не изменяются во время выполнения программы.

Используют следующие типы констант:

- целые и вещественные десятичные числа, например, 25, 6.12, 0.125e10 (см. примечание);
- логические константы - *true* (истина) и *false* (ложь);
- символьные константы - записываются либо в апострофах, например 'A', либо в виде соответствующих кодов по таблице кодировки (ASCII – для DOS или ANSI для Windows), что очень важно для букв русского алфавита. Причем в последнем случае перед кодом ставится знак «#».
- строки символов - записываются в апострофах, например 'ABCD';

Примечания. 1. В программировании принято при записи вещественных чисел вместо запятой для разделения целой и дробной части числа использовать точку.

2. Обычно при записи в программе или выполнении операций ввода-вывода вещественные числа записывают в так называемом формате с фиксированной точкой, указывая в начале целую часть числа, а затем, после точки, дробную, например: 0.5, -3.85 . Но иногда бывает удобно задавать числа в формате с плавающей точкой, т.е. в виде мантиссы и порядка. При этом мантиссу записывают перед порядком и отделяют от него строчной или прописной латинской буквой «*e*», например: запись 1.5e-10 соответствует значению $1{,}5 \times 10^{-10}$, а запись 0.5E23 соответствует значению $0{,}5 \times 10^{23}$.

Константы используются в двух формах: как литералы и как поименованные константы.

Литерал представляет собой значение константы, записанное непосредственно в программе (например, в выражении 2 + 5.1 * *x* использованы два литерала «2» и «5.1»).

Поименованные константы объявляются в инструкции раздела описаний *const*. Обращение к ним осуществляется по имени (идентификатору).

Например:

```
const  min = -23; max = 45;          {десятичные константы}
       ch1 = #94;  ch2 = 'a';        {символьные константы}
```

stroka = 'end'; {строковая константа}

Переменные. Переменные - поименованные значения, которые могут изменяться в процессе выполнения программы. Их объявление выполняют в разделе описаний программы в инструкции *var*, причем при этом указывается не только идентификатор переменной, но и ее тип. Обращение к переменным также осуществляют по идентификатору.

Тип переменной определяет возможный набор значений данной переменной, размер ее внутреннего представления и множество операций, которые могут выполняться над переменной.

Различают простые и структурные типы переменных. Простые (скалярные) типы описывают упорядоченные наборы значений. К их числу относятся следующие стандартные типы данных:

- целый тип *Integer* , используется для представления целых чисел;
- вещественный (действительный) тип *Real*, используется для представления чисел, содержащих дробную часть. Во внутреннем представлении ЭВМ мантисса и порядок вещественных чисел хранятся раздельно. Соответственно обработка вещественных чисел в компьютерах выполняется с некоторой конечной точностью, которая зависит от количества двоичных разрядов, отведенных для размещения мантиссы.
- булевский (логический) тип *Boolean*, включает только два значения - *false* (0) и *true* (1), но в памяти значения данного типа занимают целый байт;
- символьный тип *Char*, определяет набор символов (в соответствии с таблицей кодировки ASCII или ANSI).

Примеры объявления переменных:

```
var  m: integer; n: integer;        { переменные целого типа }
     a, b: real;                    { переменные вещественного типа }
     v, u: boolean;                 { переменные логического типа }
     c1, c2: char;                  { переменные символьного типа }
```

2.2 Выражения

Все вычисления и другие преобразования данных в программе записываются в виде выражений. Обычно выражение включает несколько операций, которые выполняются в порядке их приоритетности. Различают:

Арифметические операции: + (сложение), - (вычитание), * (умножение), / (деление вещественное), *div* (деление целочисленное), *mod* (остаток целочисленного деления). Эти операции применяют к вещественным и целым числам, результат – значение целого или вещественного типа.

Операции отношения: > (больше), < (меньше), = (равно), <> (не равно), >= (не меньше),

<= (не больше). Эти операции применяют к числам, символам и некоторым другим типам данных, результат - значение логического типа;

Логические операции: *and* (и), *or* (или), *not* (не). Эти операции выполняют с логическими переменными и константами, результат - значение логического типа;

В табл. 2.1 приведены приоритеты, присвоенные этим операциям. Для изменения порядка выполнения операций в выражении используют круглые скобки. В выражениях также допускается использование стандартных (см. приложение 1) и определенных программистом функций. Им присваивается высший приоритет.

Арифметические операции. Запись выражений, содержащих арифметические операции, выполняется «в строку», порядок выполнения операций определяется скобками.

Таблица 2.1

Операции	**Приоритет**
not	1
*, /, div, mod, and	2
+, -, or	3
>, <, <>, =, <=, >=	4

Особенно внимательно следует программировать выражения, включающие операции различных приоритетов. Например:

- запись $a + b / c$ предполагает, что вначале выполняется операция деления, а затем сложения;
- запись $(a + b) / c * d$ предполагает, что сумма $a + b$ делится на c, а затем умножается на d.

При программировании арифметических выражений также следует учитывать правила выполнения операций, перечисленные ниже.

1. Операции «целочисленное деление» и «определение остатка от деления» применимы только к операндам целых типов, например: 6 *div* 4 = 1, а 6 *mod* 4 = 2. Если в операции участвуют переменные, то они должны быть объявлены как целые.

Для получения при делении целых значений результата с точностью до дробной части необходимо использовать операцию вещественного деления: 6/4= 1.5.

2. При выполнении арифметических операций над числами различных типов выполняется неявное преобразование типов. Если один операнд целого типа, а другой - вещественного, то переменная целого типа преобразуется к вещественному типу; результат операции - значение вещественного типа;

Операции отношения. Операции отношения определены для вещественных и целых чисел, логических значений, кодов символов и др.. Результат этих операций - значение логиче-

ского типа, *true*, если отношение истинно, и *false* - в противном случае.

Следует помнить, что из-за ограниченной разрядной сетки вещественные числа представляются в памяти не точно, и, соответственно, проверка равенства или неравенства вещественных чисел должна выполняться с некоторым допуском, например:

(*x* - *y*) > 1e-10 {вместо x <> y}

(*x* - *y*) < 1e-10 {вместо x = y}

Логические операции. Логические операции *and*, *or* и *not* выполняют над значениями типа *boolean*. Если в логических операциях в качестве операндов используют результаты операций сравнения, которые имеют более низкий приоритет, то необходимы скобки. Например, логическое выражение, которое должно быть истинно, если значение *x* попадает в интервал [a, b], должно быть записано следующим образом:

(*x* >= *a*) *and* (*x* <= *b*).

Результаты выполнения логических операций *and*, *or* и *not* приведены в табл. 2.2.

Таблица 2.2

A	**B**	**A and B**	**A or B**	**not A**	**not B**
False	*False*	*False*	*False*	*True*	*True*
False	*True*	*False*	*True*	*True*	*False*
True	*False*	*False*	*True*	*False*	*True*
True	*True*	*True*	*True*	*False*	*False*

2.3. Встроенные математические функции

Язык *Pascal* содержит многочисленные математические функции. Наиболее часто используемые функции перечислены в табл. 2.3.

Таблица 2.3

Функция	**Возвращаемый результат**
Abs(*x*)	Модуль (абсолютное значение) числа *x*
Exp(*x*)	Вещественное значение числа *e*, возведенное в степень *x*, где *e* — основание натуральных логарифмов
Ln(*x*)	Натуральный логарифм вещественного значения *x*
Sqr(*x*)	Квадрат числа *x*, т. е. x^2
Sqrt(*x*)	Квадратный корень числа *x*, т. е. $x^{1/2}$
Sin(x)	Синус вещественного значения числа *x*, выраженного в радианах
Cos(x)	Косинус вещественного значения числа *x*, выраженного в радианах
Arctan(x)	Арктангенс вещественного значения числа *x*, вычисленный в радианах

Для возведения вещественного значения числа *y* в степень *x,* т. е. y^x, можно использовать арифметическое выражение Exp(*x* * Ln(*y*)). Для вычисления Арксинуса вещественного значения числа *x,* вычисленного в радианах, можно использовать следующее выражение:

$$Arc\sin(x) = Arc\tan\left(\frac{x}{\sqrt{1-x^2}}\right).$$

При этом программист сам должен позаботиться о том, чтобы в выражении для вычисления y^x значение основания y должно быть положительным, а в выражении для вычисления арксинуса значения x не должны превосходить по абсолютной величине единицу.

2.4. Оператор присваивания

С помощью оператора присваивания в программе записываются действия, связанные с изменением значений переменных. При выполнении этого оператора вычисляется выражение, приведенное в правой части, и его результат заносится в переменную, имя которой указано слева. Если оператор присваивания записывается в последовательности операторов, то после него ставится точка с запятой.

Например:

а)
```
Var a,b,c: real;
Begin ...
   c:=(a*a - sin(b))/(a + 25.1); ...
```

б)
```
Var v: boolean; a: integer; b: real;
Begin ...
   a := 8; b := 1.5;
       v := (a > 5) and (b >= 8);  {v получит значение false}...
```

2.5. Операторы ввода - вывода

В язык *Pascal* введены операторы (инструкции), обеспечивающие ввод данных с клавиатуры (*read* и *readln*) и вывод информации на экран монитора (*write* и *writeln*). Эти инструкции консольного ввода-вывода (консоль — это монитор и клавиатура, рассматриваемые как единое устройство ввода-вывода) являются основными инструкциями ввода-вывода при программировании на *Pascal*, например, в *Turbo Pascal* 7.0.

В литературе, посвященной программированию в *Windows*, программы, использующие инструкции консольного ввода-вывода, довольно часто приводятся в качестве примеров.

2.5.1. Инструкции *WRITE* и *WRITELN*

Инструкция *write* предназначена для вывода на экран монитора сообщений и значений переменных. После слова *write* в скобках задается список имен переменных. Кроме имен переменных в список можно включить сообщение — текст, заключенный в одиночные кавычки.

Например

```
write (Suima);
write('Результат вычислений');
write ('Корни уравнения. xl=',xl, ' x2=',x2);
```

После имени переменной через двоеточие можно поместить описание (формат) поля вы-

вода значения переменной.

Для переменной типа *integer* формат — это целое число, определяющее ширину поля вывода (количество позиций на экране).

Например, инструкция

```
write(d:5);
```

показывает, что для вывода значения переменной *d* используется 5 позиций.

Если число такое, что его изображение занимает меньше позиций, чем указано в формате, то перед выводимым числом добавляется такое количество пробелов, чтобы общее количество выведенных символов было равно указанному в формате.

Например, если значение переменной *kol* типа *integer* равно 15, то в результате выполнения инструкции

```
write('Всего изделий:',Kol:5);
```

на экран будет выведено:

```
Всего изделий:   15
```

Для переменных типа *real* формат представляет собой два целых числа, разделенных двоеточием. Первое число определяет ширину поля вывода, второе — количество цифр, стоящих справа от десятичной точки. Если задать только ширину поля, то на экране появится число, представленное в формате с плавающей точкой.

Например, пусть переменные *x*1 и *x*2 типа *real* имеют значения 13.25 и -0.3401, тогда в результате выполнения инструкции

```
write('xl=',xl:5:2,'    x2=',x2:12)
```

на экран будет выведено:

```
xl=13.25   x2=-3.40100E-01
```

Если ширины поля, указанной в формате, недостаточно для вывода значения переменной, то выводится число в формате с плавающей точкой и десятью цифрами после запятой (все поле вывода в этом случае занимает 17 позиций).

После выполнения инструкции *write* курсор остается в той позиции экрана, в которую он переместился после вывода последнего символа, выведенного этой инструкцией. Следующая инструкция *write* начинает вывод именно с этой позиции. Например, в результате выполнения инструкций

```
x:=-2.73;
write('Значение перем');
write('енной:');
write('x=');
write(x:8:5);
```

на экран будет выведено:

```
Значение переменной:x=-2.73000
```

Инструкция *writeln* отличается от инструкции *write* только тем, что после вывода сооб-

щения или значений переменных курсор переводится в начало следующей строки. Например, если значением переменной *x*1 является число -3.561, а значением переменной *x*2 — число 10.345, то результатом выполнения инструкций

```
writein('Значения корней уравнения:'); writeln('xl=',x:7:3);
writeln('x2=',x:7:3);
```

будет следующий текст на экране:

```
Значения корней уравнения:
x1=-3.5610
x2= 10.345
```

2.5.2. Инструкции *READ* и *READLN*

Инструкция *read* предназначена для ввода с клавиатуры значений переменных (исходных данных). В общем виде инструкция выглядит следующим образом:

```
read(Переменная1,Переменная2, ... Переменнаяn)
```

где *Переменная* — имя переменной, значение которой должно быть введено с клавиатуры во время выполнения программы.

Приведем примеры записи инструкции *read*:

```
read(a); read(c,kol);
```

При выполнении инструкции *read* происходит следующее:

1. Программа приостанавливает свою работу и ждет, пока на клавиатуре будут набраны нужные данные и нажата клавиша <*Enter*>.

2. После нажатия клавиши <*Enter*> введенное значение присваивается переменной, имя которой указано в инструкции.

Например, в результате выполнения инструкции

```
read(T);
```

и ввода с клавиатуры строки 21, значением переменной T будет число 21.

Одна инструкция *read* позволяет получить значения нескольких переменных. При этом вводимые числа должны быть набраны в одной строке и разделены пробелами. Например, если тип переменных *a*, *b* и *c* — *real*, то в результате выполнения инструкции *read* (**a**, *b*, *c*); и ввода с клавиатуры строки:

```
4.5 23 0.17
```

переменные будут иметь следующие значения: *a* = 4,5; *b* = 23; *c* = 0,17.

Если в строке набрано больше чисел, чем задано переменных в инструкции *read*, то оставшаяся часть строки будет обработана следующей инструкцией *read*. Например, в результате выполнения инструкций:

```
read(A, B);
read(C);
```

и ввода с клавиатуры строки

```
10 25 18
```

переменные получат следующие значения: $A = 10$; $B = 25$. Инструкция *read* (*C*); присвоит переменной *C* значение 18.

Инструкция *readln* отличается от инструкции *read* тем, что после выделения очередного числа из введенной с клавиатуры строки и присваивания его последней переменной из списка инструкции *readln*, оставшаяся часть строки теряется, и следующая инструкция *read* или *readln* будет требовать нового ввода.

Например, в результате выполнения инструкции:

```
readln(A,B);
read (C);
```

и вводе с клавиатуры строки

```
10 25 18
```

переменные получат следующие значения: $A = 10$; $B = 25$. После чего программа будет ожидать ввода нового числа, чтобы присвоить его переменной *C*.

Перед каждой инструкцией *read* или *readln* следует располагать инструкцию *write*, для того чтобы подсказать пользователю, какие данные ожидает от него программа.

Если тип данных, вводимых с клавиатуры, не соответствует или не может быть приведен к типу переменных, имена которых указаны в инструкции *read* (*readln*), то программа аварийно завершает работу (инструкции, следующие за *read* не выполняются), и на экран выводится сообщение об ошибке.

2.6. Пример программы

В качестве примера использования простейших конструкций языка *Pascal* в программе с линейным вычислительным процессом рассмотрим программу, которая для заданной на плоскости точки (координаты точки x, y вводятся с клавиатуры) выдает значение логической переменной *true*, если точка попадает в ограниченную прямоугольником область (рис. 2.1), или значение *false* - в противном случае.

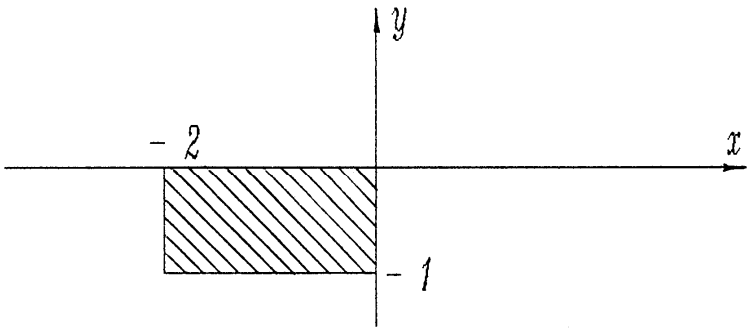


Рис. 2.1. Заданная на плоскости область

Текст программы приведен ниже:

```
program example;                    { Заголовок программы }
    { программа, информирующая о попадании заданной точки с координатами (x, y)
в ограниченную на плоскости область (на рисунке заштрихована) }
{ Раздел описаний }
 var
     x, y: real;                    { объявление переменных }
     z: boolean;
{ Раздел операторов }
    begin
     writeln(' Vvedite koordinati tochki x, y ');          {приглашение к вводу}
     read(x, y);                              {ввод исходных данных}
     z := (x>=-2) and (x<=0) and (y>=-1) and (y<=0); { вычисление значения
                                                       логической переменной z }
     writeln('x = ',x,' y = ',y);             {выводим исходные данные и  }
     write('z = ',z );        { результат расчета – значение переменной z}
     read(x);                              {фиктивный ввод для сохранения окна}
    end.                                   {конец программы}
```

После сохранения программы и ее запуска из среды программирования *Delphi* путем щелчка на кнопке  или нажатия клавиши ***<F9>*** выполняется компиляция программы. При отсутствии синтаксических ошибок на экране появится окно созданного консольного приложения с текстом приглашения к вводу значений переменных (*Vvedite koordinati tochki x, y*). После ввода значений двух чисел, отделенных друг от друга пробелом в результате нажатия клавиши ***<Spacebar>***, например:

-1 -0.8

и последующего нажатия клавиши ***<Enter>***, программа выведет на экран координаты заданной точки *x*, *y* и результат расчета:

z = TRUE

и будет ожидать еще одного нажатия клавиши ***<Enter>***, чтобы завершить свою работу.

На рис. 2.2 приведено содержимое окна консольного приложения с результатами работы программы.

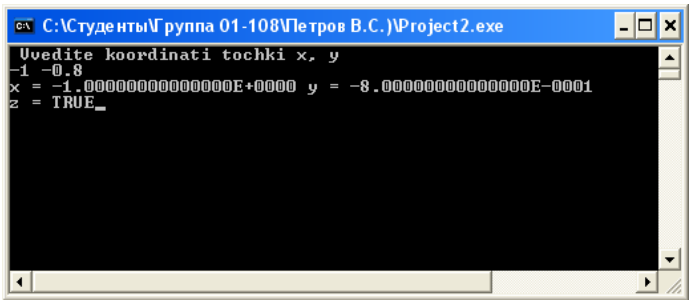


Рис. 2.2. Окно консольного приложения с результатами работы программы

3. Задание к лабораторной работе №2

1). Составить программу, которая для любой точки на плоскости определяет, принадлежит ли эта точка заданной части плоскости (см. рис. в выданном варианте задания). В ходе работы программа должна ввести с клавиатуры координаты точки (x, y), выдать на экран монитора эти координаты в формате с фиксированной десятичной точкой с точностью до двух цифр в дробной части числа и значение логической переменной *true*, если точка попадает в заданную ограниченную область. В противном случае на экран выдается значение логической переменной *false*. Пример выполнения программы приведен в разделе 2.6.

2). Разработку программы выполнить в среде программирования *Delphi* 5 или *Delphi* 7 в форме консольного приложения программу под именем *Project2*.

3). Выполнить компиляцию и провести отладку и тестирование программы.

Методические указания

Для выполнения лабораторной работы №2 следует повторить следующие разделы математики:

- уравнение прямой, проходящей через две заданные на плоскости точки;
- уравнение окружности заданного радиуса, центр которой находится в начале осей координат или смещен относительно начала осей координат.

Список использованных источников

1. Иванова Г.С. Основы программирования: Учебник для вузов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001, 392 с. (Сер. Информатика в техническом университете).

2. Керман Митчелл К. Программирование и отладка в *Delphi*. Учебный курс.: пер. с англ. – М.: Издательский дом «Вильямс», 2002, 672 с.

3. Культин Н.Б. Программирование на *Object Pascal* в *Delphi* 5. – СПб.: БХВ – Санкт-Петербург, 2000, 464 с.

Государственное образовательное учреждение высшего профессионального образования

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(государственный технический университет)

Кафедра «Проектирование вертолетов»

Маслов А.Д.

Алгоритмические языки и программирование на ЭВМ

Лабораторная работа № 3

Управляющие конструкции языка *Pascal*

Утверждено на заседании кафедры

5 июля 2007 г.

г. Москва

2007 г.

Введение

Настоящее пособие предназначено для студентов специальности «Самолето- и вертолетостроение» (специализация «Вертолетостроение»), изучающих дисциплину «Алгоритмические языки и программирование».

Целью работы является изучение управляющих конструкций алгоритмического языка *Pascal* и приобретение практических навыков по созданию и отладке в среде *Delphi* программ с разветвленным вычислительным процессом.

1. Оператор условной передачи управления

Оператор условной передачи управления (условный оператор) используют для программирования ветвлений, т. е. ситуаций, когда возникает необходимость при определенных условиях выполнять различные действия. Условие записывают в виде логического выражения, в зависимости от результата которого осуществляется выбор одной из ветвей: если результат *true*, то выполняется оператор, следующий за служебным словом ***then***, иначе - оператор, следующий за служебным словом ***else***.

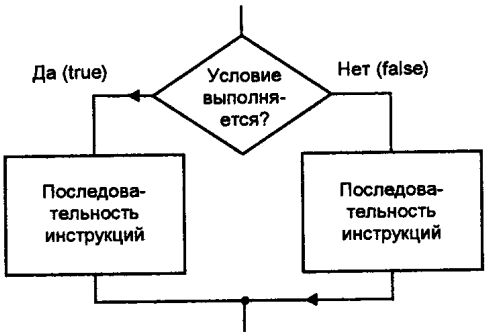


Рис. 1.1. Блок-схема алгоритма, соответствующего действию условного оператора

В каждой ветви (рис. 1.1) допускается запись одной инструкции (оператора), в том числе составного оператора или другого условного оператора. В последнем случае говорят, что один условный оператор вложен в другой условный оператор.

Составным оператором в языке *Pascal* называют последовательность операторов, заключенную в операторные скобки ***begin*** … ***end***.

Операторы последовательности отделяют друг от друга точкой с запятой «;». Перед ***end*** точку с запятой можно не ставить. Перед ***else*** точка с запятой не ставится никогда, так как в этом случае запись условного оператора продолжается.

В общем виде условный оператор записывается так:

```
if условие then
  begin
```

```
{   инструкции, которые надо выполнить,
если условие выполняется, истинно.  }
    end   else
  begin
{ инструкции, которые надо выполнить, если условие не выполняется, ложно. }
  end
```

Выполняется условный оператор следующим образом:

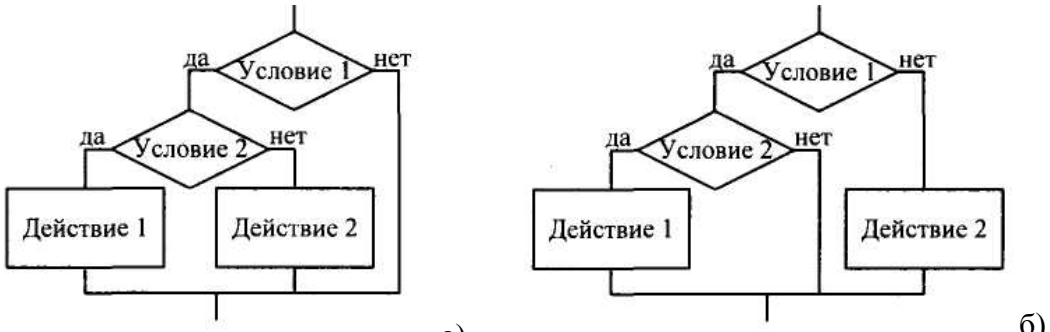
1. Вычисляется значение условия (условие — выражение логического типа, значение которого может быть равно *true* или *false*).

2. Если условие выполняется (значение выражения условие равно *true*), то выполняются инструкции, следующие за словом ***then*** и находящиеся между ***begin*** и ***end***.

3. Если условие не выполняется (значение выражения условие равно *false*), то выполняются инструкции, следующие за словом ***else*** (находящиеся между ***begin*** и ***end***).

Допускается использовать оператор условной передачи управления с неуказанной (пустой) ветвью ***else***.

В случае, если условный оператор вложен в другой условный оператор используется так называемое «правило вложенности»: альтернатива ***else*** всегда относится к ближайшему ***if*** что соответствует варианту алгоритма, показанному на рис. 1.2, а. Если необходимо реализовать вариант алгоритма, изображенный на рис. 1.2, б, то используют операторные скобки.



а) б)

Рис. 1.2. Блок-схемы алгоритмов, соответствующих возможных вариантов действия вложенных условных операторов

Варианту алгоритма а) соответствует следующий фрагмент записи:

```
if условие1 then
   if условие2 then действие1
               else действие2;
```

Варианту алгоритма б) соответствует следующий фрагмент:

```
if условие1 then
 begin
```

```
    if условие2 then
      действие1
  end else действие2;
```

2. Типовые примеры.

Пример 1. Разработать программу, которая вычисляет значение функции, заданной следующим образом:

$$y = \begin{cases} |x|, & |x| \le 1; \\ x^2, & 1 \prec |x| \le 2; \\ 4, & x \succ 2. \end{cases}$$

Программа должна начинаться с ввода значения аргумента. Затем в зависимости от того, в какой интервал попадает введенное значение, вычисляем значение функции по одному из заданных выражений. Блок-схема алгоритма решения данной задачи представлена на рис. 1.3.

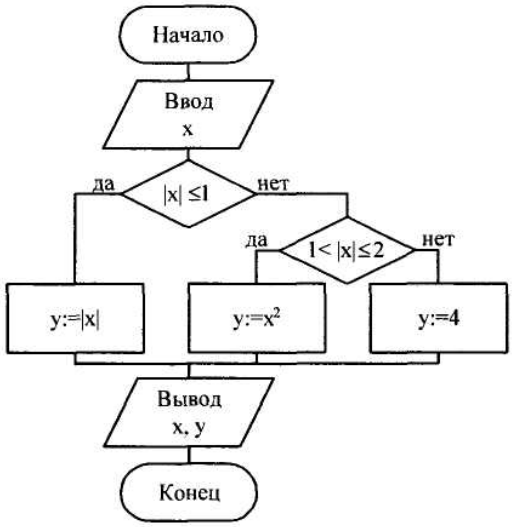


Рис. 1.3. Блок-схема алгоритма программы вычисления функции, заданной на отрезках, в заданной точке

Текст программы имеет следующий вид:

```
program example_1;
{ программа вычисления функции y =(x) }
var x,y:real;
begin
writeln(' Vvedite x ');
readln(x);
if abs(x)<=1 then y:=abs(x)       {первый отрезок - x<=1}
             else
   if (abs(x)>1) and (abs(x)<=2) then
     y:=sqr(x)                    {второй отрезок 1<|x|<=2}
else   y:=4;                      {третий отрезок - x>2}
```

```
        writeln('x = ', x:8:5, ' y =', y:8:5);
end.
```

Пример 2. Разработать программу решения неполного квадратного уравнения вида $a \cdot x^2 + b = 0.$

Очевидно, что решение $x = \pm\sqrt{\frac{-b}{a}}$ существует, если только $a \neq 0$ и коэффициенты a и b имеют одинаковые знаки. Блок-схема алгоритма решения данной задачи представлена на рис. 1.4.

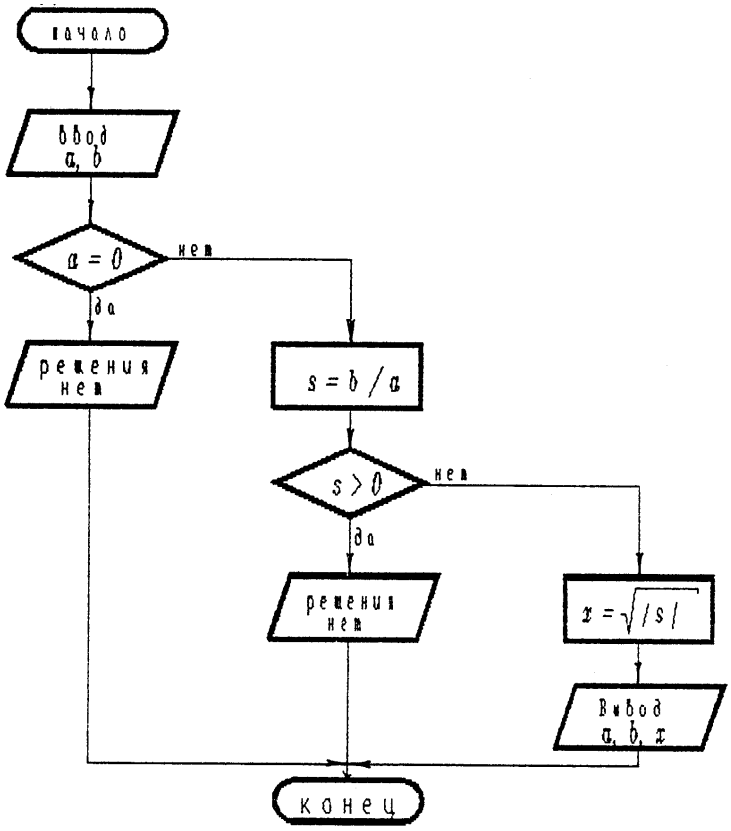


Рис. 1.4. Блок-схема алгоритма программы решения неполного квадратного уравнения.

Текст программы имеет следующий вид:

```
program example_2;
    { программа решения неполного квадратного уравнения ax2 + b = 0 }
    const
     eps=1E-6;
    var
     a,b,x1,x2,s: real;

    begin
     writeln(' Vvedite a i b ');
```

```
 read(a,b);
 if abs(a)<eps then                    {если a=0, тогда }
  writeln(' RESHENIA NET: a = 0 ')      { сообщение, что решения нет }
  else begin                          { иначе (a<>0): проверяем знаки a и b }
   s:=b/a;
   if s>0 then                  { если a и b имеют одинаковые знаки, тогда }
writeln('RESHENIA NET: a i b imeyut odinakovie znaki') { сообщение,
                                                         что решения нет }
   else begin                          { иначе: }
    x1:=sqrt(abs(s)); x2:=-x1;       {вычисление действительных корней x1, x2}
    writeln('a=',a,'b=',b);
    writeln('x1=',x1,' x2=',x2);
   end
  end
end.
```

Примечание. В связи с приближенным представлением в памяти ЭВМ вещественных чисел при обеспечении недостаточной точности их представления равенство типа $a = 0$ может быть не выполнено. Поэтому программа не будет сообщать об ошибке при введенном в качестве делимого нулевого значения a. Корректной будет программа, в которой вводится константа допустимой точности *eps*, например $eps = 10^{-6}$. Тогда в приведенном выше тексте, если переменная a по модулю меньше *eps*, то в первом условном операторе она трактуется как нуль. Указанный прием можно использовать и в случае возникновения любой другой подобной ситуации.

3. Задание к лабораторной работе №3

1). Составить программу, которая вводит с клавиатуры значения переменных x, y, z, выводит на экран значения этих переменных в формате с фиксированной десятичной точкой с точностью до трех цифр в дробной части числа и для каждой из двух заданных математических функций $a, b = f(x, y, z)$ выполняет следующие действия:

- в случае, если функция существует, вычисляет и выводит на экран полученное значение этой функции в формате с фиксированной десятичной точкой с точностью до четырех цифр в дробной части числа;
- в случае, если при введенных исходных данных функция не может существовать, на экран выводится текстовая информация о причине невозможности получения числового результата.

2). Разработку программы выполнить в среде программирования *Delphi* 5 или *Delphi* 7 в форме консольного приложения программу под именем *Project3*.

3). Выполнить компиляцию и провести отладку и тестирование программы.

Методические указания

Для выполнения лабораторной работы №3 следует повторить следующие разделы математики:

- область допустимых значений логарифмических функций;
- область допустимых значений тригонометрических функций (тангенс, арксинус, арккосинус).

Для возведения вещественного значения числа *y* в степень *x,* т. е. y^x, можно использовать арифметическое выражение Exp(*x* * Ln(*y*)). Для вычисления Арксинуса вещественного значения числа *x,* вычисленного в радианах, можно использовать следующее выражение:

$$Arc\sin(x) = Arc\tan\left(\frac{x}{\sqrt{1-x^2}}\right).$$

При этом программист сам должен позаботиться о том, чтобы в выражении для вычисления y^x значение основания *y* должно быть положительным, а в выражении для вычисления арксинуса значения аргумента не должны превосходить по абсолютной величине единицу.

Наиболее часто используемые стандартные математические функции языка *Pascal* перечислены в табл. 3.1.

Таблица 3.1

Функция	**Возвращаемый результат**
Abs(*x*)	Модуль (абсолютное значение) числа *x*
Exp(*x*)	Вещественное значение числа *e*, возведенное в степень *x*, где *e* — основание натуральных логарифмов
Ln(*x*)	Натуральный логарифм вещественного значения *x*
Sqr(*x*)	Квадрат числа *x*, т. е. x^2
Sqrt(*x*)	Квадратный корень числа *x*, т. е. $x^{1/2}$
Sin(x)	Синус вещественного значения числа *x*, выраженного в радианах
Cos(x)	Косинус вещественного значения числа *x*, выраженного в радианах
Arctan(x)	Арктангенс вещественного значения числа *x*, вычисленный в радианах

Список использованных источников

1. Иванова Г.С. Основы программирования: Учебник для вузов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001, 392 с. (Сер. Информатика в техническом университете).

2. Керман Митчелл К. Программирование и отладка в *Delphi*. Учебный курс.: пер. с англ. – М.: Издательский дом «Вильямс», 2002, 672 с.

3. Культин Н.Б. Программирование на Object Pascal в Delphi 5. – СПб.: БХВ – Санкт-Петербург, 2000, 464 с.

Государственное образовательное учреждение высшего профессионального образования

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(государственный технический университет)

Кафедра «Проектирование вертолетов»

Маслов А.Д.

Алгоритмические языки и программирование на ЭВМ

Лабораторная работа № 4

Инструкции языка *Pascal* для организации циклических процессов

Утверждено на заседании кафедры

5 июля 2007 г.

г. Москва

2007 г.

Введение

Настоящее пособие предназначено для студентов специальности «Самолето- и вертолетостроение» (специализация «Вертолетостроение»), изучающих дисциплину «Алгоритмические языки и программирование».

Целью работы является изучение инструкций (операторов) алгоритмического языка *Pascal* для организации циклических (повторяющихся) процессов и приобретение практических навыков по созданию и отладке в среде *Delphi* программ с повторяющимся вычислительным процессом.

1. Операторы повтора (цикла)

Для реализации циклических (повторяющихся) процессов используют операторы циклов. В языке *Pascal* реализованы следующие виды операторов циклов.

- оператор цикла с предварительным условием ***while****;*
- оператор цикла с последующим условием ***repeat****;*
- оператор цикла с параметром ***for***.

Операторы цикла ***while*** и ***repeat*** используют для реализации итерационных циклических процессов, оператор цикла ***for*** – для реализации циклических процессов с заданным количеством повторений.

1.1. Оператор цикла с предварительным условием *while*

В общем виде инструкция цикла с предварительным условием (предусловием) ***while*** записывается следующим образом:

```
while условие do оператор;
```

где *условие* представляет собой выражение логического типа, определяющее условие выполнения цикла.

Оператор цикла с предусловием ***while*** выполняется следующим образом:

1. Вычисляется значение выражения *условие*.

2. Если *условие* не выполняется (значение выражения *условие* равно *false*), то *оператор* тела цикла, стоящий после ключевого слова ***do***, не выполняется, и на этом выполнение инструкции ***while*** завершается.

3. Если *условие* выполняется (значение выражения *условие* равно *true*), то выполняется *оператор* тела цикла и после этого снова проверяется выполнение условия. Если *условие* выполняется, то *оператор* тела цикла выполняется еще раз. И так до тех пор, пока *условие* не станет ложным (*false*).

Для того чтобы цикл завершился, необходимо, чтобы *оператор* тела цикла влиял на значение выражения *условие* (изменял значения переменных, входящих в выражение *условие*).

Блок-схема алгоритма, соответствующего инструкции ***while***, представлена на рис. 1.1.

Если в тело цикла необходимо поместить несколько операторов, то используют составной оператор. Составным оператором в языке *Pascal* называют последовательность операторов, заключенную в операторные скобки ***begin*** … ***end***.

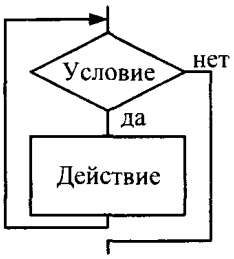


Рис. 1.1. Блок-схема алгоритма, соответствующего действию оператора цикла с предусловием ***while***

1.2. Оператор цикла с последующим условием *repeat*

В общем виде оператор цикла с последующим условием (постусловием) ***repeat*** записывается так:

```
repeat
 оператор1;
 оператор2;
  . . .
 операторN;
until условие;
```

где *условие* представляет собой выражение логического типа, определяющее условие завершения цикла.

Инструкция ***repeat*** выполняется следующим образом:

1. Выполняются операторы тела цикла, находящиеся между ***repeat*** и ***until***.

2. Вычисляется значение выражения *условие*. Если условие ложно, (значение выражения *условие* равно *false*), то повторно выполняются операторы тела цикла.

3. Если *условие* истинно (значение выражения *условие* равно *true*), то выполнение цикла прекращается.

Таким образом, операторы (инструкции) тела цикла выполняются до тех пор, пока *условие* ложно (значение выражения *условие* равно *false*). При этом тело цикла всегда выполняется хотя бы один раз. Для того чтобы цикл завершился, необходимо, чтобы операторы тела цикла влияли на значение выражения *условие* (изменяли значения переменных, входящих в выражение *условие*).

Блок-схема алгоритма, соответствующего инструкции ***repeat***, представлена на рис. 1.2.

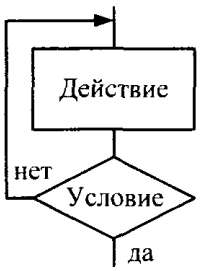


Рис. 1.2. Блок-схема алгоритма, соответствующего действию оператора цикла с постусловием ***repeat***

1.3. Оператор цикла с параметром *for*

Оператор цикла с параметром ***for*** рекомендуется использовать в том случае, если число повторений заранее известно. Цикл ***for*** выполняется, пока переменная (параметр) цикла принимает значения в заданном диапазоне с определенным шагом.

По сравнению с другими языками реализованный в *Pascal* оператор цикла с параметром ***for*** является менее мощным оператором, так как шаг данной конструкции ограничен только +1 или -1.

В общем виде инструкция ***for*** записывается следующим образом:

```
for счетчик := нач_знач to кон_знач_счетчика do оператор;
```

или

```
for счетчик := нач_знач downto кон_знач_счетчика do оператор;
```

где

счетчик – имя переменной - счетчика числа повторений оператора цикла;

нач_знач – выражение, определяющее начальное значение переменной - счетчика циклов;

кон_знач – выражение, определяющее конечное значение переменной - счетчика циклов.

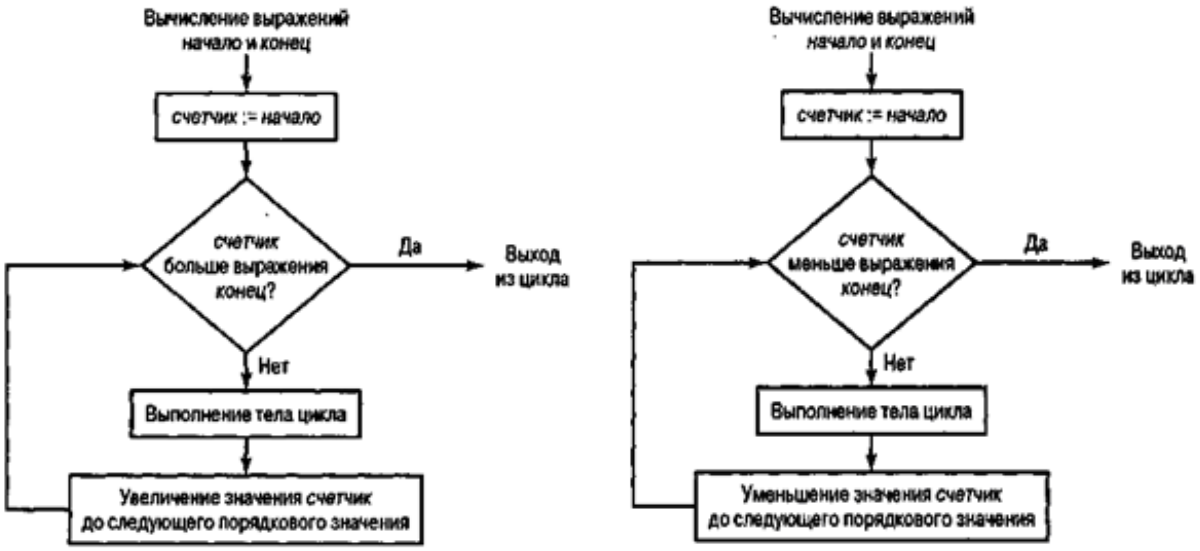
Если используется служебное слово ***to***, то при каждом выполнении цикла переменной цикла присваивается следующее значение порядкового типа переменной. Если используется служебное слово ***downto***, то при каждом выполнении цикла переменной цикла присваивается предыдущее значение порядкового типа переменной. Переменная (параметр) цикла должна иметь порядковый тип. Соответственно начальное и конечное значения должны принадлежать к тому же типу, что и параметр цикла.

Если в тело цикла необходимо поместить несколько операторов, то используют составной оператор (последовательность операторов, заключенная в операторные скобки ***begin*** … ***end***).

Обычно в качестве выражений, определяющих значения начального и конечного состояния счетчика циклов, используют переменные или константы. Если используется служебное

слово ***to*** , то в этом случае оператор тела цикла (или последовательность операторов, находящихся между ***begin*** и ***end***) будет выполнена (*кон_знач* – *нач_знач* + 1) раз. Если начальное значение счетчика превышает конечное значение счетчика, то оператор тела цикла не будет выполнен ни разу.

Блок-схема алгоритма, соответствующего инструкции ***for***, представлена на рис. 1.3.



а) б)

Рис. 1.3. Блок-схема алгоритма, соответствующего действию оператора цикла с параметром ***for*** : а) инкрементного цикла (с увеличивающимся счетчиком циклов); б) декрементного цикла (с уменьшающимся счетчиком циклов).

2. Типовые примеры.

Пример 1. Разработать программу вычисления суммы *n* первых натуральных чисел.

Сумма $\sum_{i=1}^{n} i$ определяется методом накопления. Количество суммируемых чисел известно, поэтому используем цикл с заданным количеством повторений. При каждом проходе к сумме будем добавлять переменную цикла, которая будет изменяться от 1 до *n*. Перед циклом переменную суммы необходимо обнулить. На рис. 2.1 приведена схема алгоритма программы.

Текст программы имеет следующий вид:

```
Program example_1;
 Var
  i, n, S: integer;
Begin
   Writeln('Vvedite n');
   Readln(n);
   S:=0;            // Обнуление суммы перед началом цикла
   for i:=1 to n do
```

```
    S:=S+i;       // Накопление суммы
  Writeln('Summa ', n ,' chisel = ', S);
End.
```

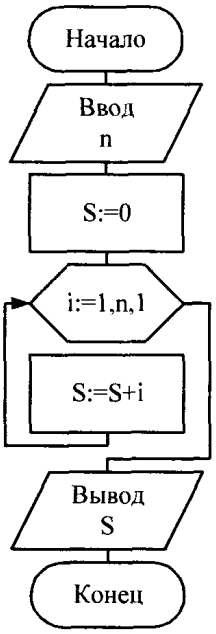


Рис. 2.1. Блок-схема алгоритма программы вычисления суммы n натуральных чисел

Пример 2. Разработать программу, определяющую сумму ряда $1 - 1/x + 1/x^2 - 1/x^3 + \ldots$ с заданной точностью ε.

Из соответствующих разделов математики известно, что суммой ряда называется предел, к которому стремится последовательность частичных сумм данного ряда, если он существует. В этом случае ряд называется сходящимся. Для получения суммы ряда с заданной точностью ε будем накапливать частичную сумму элементов ряда, пока очередной n-й член ряда r_n по модулю не станет меньше ε:

$$| r_n | < \varepsilon.$$

На рис. 2.2 представлен алгоритм определения суммы ряда с заданной точностью. Анализ алгоритма показывает, что в нем присутствует цикл суммирования, который не является ни циклом с предусловием, ни циклом с постусловием, ни циклом с заданным количеством повторений. Для того, чтобы можно было использовать один из имеющихся операторов цикла языка *Pascal*, в данный алгоритм необходимо внести преобразования.

Поскольку заранее количество повторений цикла определить нельзя, преобразуем цикл в алгоритме, например, в цикл с предусловием ***while***. Формирование такого цикла в алгоритме определения суммы ряда с заданной точностью показано на рис. 2.3, а.

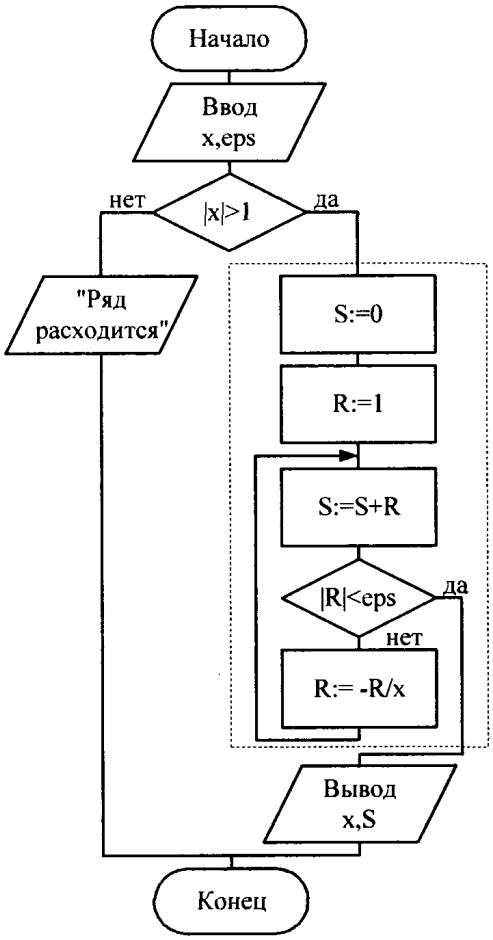


Рис. 2.2. Блок-схема алгоритма определения суммы ряда с заданной точностью

Текст программы имеет следующий вид:

```
Program example_2a;
  Var S, x, R, eps: real;
Begin
  Writeln('Vvedite  x i eps: ');
  Readln(x,eps);
  if x>1 then            {если x > 1, то ищем сумму ряда}
   begin
    S:=1;  R:=1;               // Инициализация цикла
    while abs(R)> eps do
     begin
      R:=-R/x;
      S:=S+R;
     end;
    writeln(' x = ', x:6:2,' S = ', S:8:2,'R =', R:8:6)
   end
  else writeln('Ryad rashoditsya');
End.
```

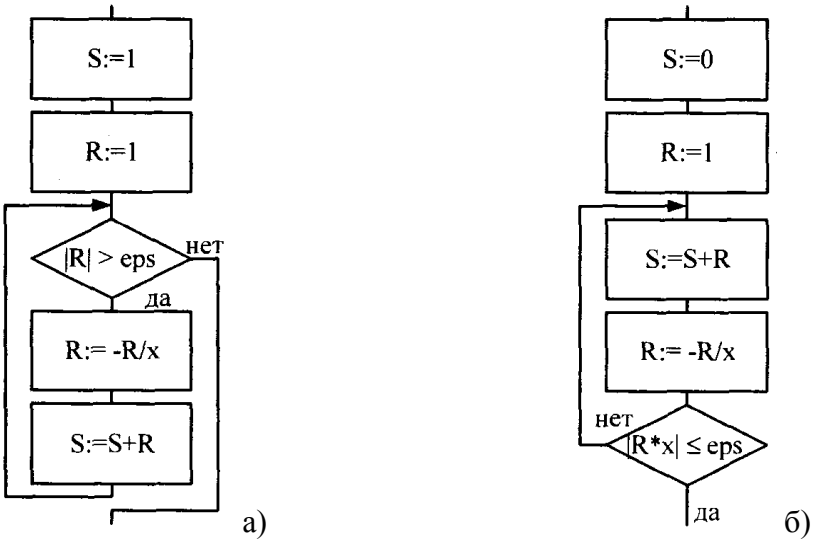


Рис. 2.3. Блок-схемы структурного варианта алгоритма:
а) с использованием цикла ***while*** ; б) с использованием цикла ***repeat***

Тот же алгоритм можно преобразовать так, чтобы цикл можно было реализовать с использованием цикла с постусловием ***repeat*** (рис. 2.3, б). Ниже представлен текст соответствующей программы.

```
Program example_2b;
  var S, R, x, eps: real;
  Begin
   Writeln('Vvedite x i eps:');
   Readln(x,eps);
   if x>1 then
    begin
     S:=0; R:=1;                    // Инициализация цикла
     repeat
      S:=S+R;
      R:=-R/x;
     until abs(R*x)<=eps;
       writeln(' x =', x:6:2,' S = ', S:8:2,' R =', R:8:6)
    else Writeln ('Ryad rashoditsya');
  End.
```

3. Задание к лабораторной работе №4

1). Составить программу вычисления суммы ряда $S=\sum_{i=0}^{\infty} a_i(x)$ с точностью ε = 0,001 и аналитического выражения $f(x)=\lim_{i\to\infty} S$ (см. вариант задания). Значение переменной x вводится с клавиатуры. В результате работы программа должна выводить на экран значения переменной x , суммы S и выражения $f(x)$ для этой суммы в формате с фиксированной десятичной точкой с

точностью до четырех цифр в дробной части числа, а также число повторов (членов ряда), при котором была достигнута требуемая точность вычислений. Сравнение двух результатов позволит оценить расчетную точность вычисления суммы ряда *S*.

2). Составить программу расчета характеристик Международной стандартной атмосферы (давление p, плотность ρ, температура t (°*C*), коэффициент кинематической вязкости v, скорость звука a) для ряда высот H над уровнем моря в диапазоне 0…15 км с шагом $\Delta H = 250$ м.

3). Разработку программ выполнить в среде программирования *Delphi* 5 или *Delphi* 7 в форме консольного приложения программу под именем *Project4.*

4). Выполнить компиляцию и провести отладку и тестирование программ.

Методические указания

Вычисление суммы $S = \sum_{i=0}^{\infty} a_i(x)$ выполнять при помощи оператора цикла ***repeat***.

Для выполнения лабораторной работы №4 следует повторить следующие разделы математики:

- факториалы;
- правила возведения в степень.

Список использованных источников

1. Иванова Г.С. Основы программирования: Учебник для вузов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001, 392 с. (Сер. Информатика в техническом университете).

2. Керман Митчелл К. Программирование и отладка в *Delphi*. Учебный курс.: пер. с англ. – М.: Издательский дом «Вильямс», 2002, 672 с.

3. Культин Н.Б. Программирование на Object Pascal в Delphi 5. – СПб.: БХВ – Санкт-Петербург, 2000, 464 с.

Государственное образовательное учреждение высшего профессионального образования
МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(государственный технический университет)

Кафедра «Проектирование вертолетов»

Маслов А.Д.

Алгоритмические языки и программирование на ЭВМ

Лабораторные работы № 5, 6

Работа со структурными типами данных

Утверждено на заседании кафедры

5 июля 2007 г.

г. Москва

2007 г.

Введение

Настоящее пособие предназначено для студентов специальности «Самолето- и вертолетостроение» (специализация «Вертолетостроение»), изучающих дисциплину «Алгоритмические языки и программирование».

Целью работ является изучение массивов, относящихся к структурным типам данных алгоритмического языка *Pascal*, и приобретение практических навыков по созданию и отладке в среде *Delphi* программ по обработке одномерных и двумерных массивов.

1. Массивы

Массив – это структура данных, которую можно рассматривать как набор переменных одинакового типа, имеющих общее имя. Каждому элементу массива соответствует один или несколько индексов, определяющих положение элемента в массиве. Массивы удобно использовать для хранения однородной по своей природе информации, например, таблиц и списков.

В общем виде инструкция объявления массива выглядит следующим образом:

```
array[нижний_индекс..верхний_индекс] of  тип
```

где

array – зарезервированное слово языка *Pascal*, обозначающее, что переменная является массивом;

нижний_индекс и *верхний_индекс* – целые константы, определяющие диапазон изменения индекса элементов массива и, неявно, количество элементов (размер) массива;

тип – тип элементов массива.

Тип элементов массива – любой допустимый в *Pascal* тип данных, кроме файла.

Тип индекса определяет его допустимые значения. В качестве типа индекса может быть указан любой порядковый тип (*integer*, *char*, *boolean*, перечисляемый тип, а также диапазоны этих типов).

В зависимости от количества типов индексов различают: одномерные, двумерные, трехмерные и *n*-мерные массивы. Двумерные массивы обычно называют матрицами, считая первый индекс - номером строки, а второй – номером столбца.

Объявление переменных типа массив выполняется двумя способами:

– в операторе объявления переменных, например:

```
var a: array[1..5] of integer;       { одномерный массив a из пяти целых чисел }
    b: array[1..3,1..4] of real;     { двумерный массив b или матрица,
                                         состоящая из 3-х строк и 4-х столбцов }
```

– с предварительным объявлением типа, например:

```
type mas = array[1..10] of integer;   {объявляем тип }
var a: mas;                           {объявляем переменную}
```

При объявлении массива удобно использовать именованные константы. Именованная константа объявляется в разделе объявления констант, который обычно располагают перед разделом объявления переменных. Начинается раздел объявления констант словом ***const***. В инструкции объявления именованной константы указывают имя константы и ее значение, которое отделяется от имени символом «равно». Например, чтобы объявить именованную константу *n*, значение которой равно 10, в раздел ***const*** надо записать инструкцию *n* =10. После объявления именованной константы ее можно использовать в программе как обычную числовую или символьную константу. Ниже, в качестве примера, приведено объявление двумерного массива, в котором используются именованные константы.

```
const
  n=3;      { число строк матрицы }
  m=4       { число столбцов матрицы }
type matrix = array[1..n,1..m] of integer;
var a: matrix;
```

Работа с массивом в программе, как правило, сводится к действиям над его элементами. Для обращения к конкретному элементу массива необходимо указать имя массива и значения индексов элемента в квадратных скобках через запятую. В качестве индекса можно использовать константу, а также идентификатор переменной или выражение целого типа, например:

```
a[1,2] := 0;  { присваиваем элементу матрицы a12 нулевое значение }
b[i+1] := 1;    { присваиваем элементу матрицы bi+1 значение 1}
```

Задание индексов в виде идентификатора переменной или выражения позволяет реализовывать последовательную обработку элементов массивов. Причем для этого обычно применяют циклы с заданным количеством повторений ***for***, поскольку интервал изменения индекса определен при объявлении массива. Параметр цикла ***for*** используют в качестве переменной косвенной адресации массива, например:

```
var a: array[1..6] of integer;
    i,j: integer;
...
for i:=1 to 6 do a[i]:= i;      { при i=1 элементу массива a1 присваивается 1,
                                  при i=2 элементу a2 присваивается 2, при i=3
                                  элементу a3 присваивается 3 и т.д.}
```

Количество переменных, необходимых для косвенной адресации массивов, совпадает с размерностью массива. Так, для работы с матрицами используют две переменные, хранящие индексы: одну – для хранения номеров строк, а вторую – номеров столбцов.

Значения элементов массива в программе можно определить также в результате ввода элементов массива с клавиатуры, для чего каждое число помещают на отдельной строке экрана, например:

```
var a: array[1..5] of real;
begin
```

```
for i:=1 to 5 do
Read(a[i]);                    { осуществляем ввод массива }
Readln;                        { очищаем буфер ввода, чтобы далее значения
                                 вводились со следующей строки}
```

Значения элементов массива вводят в порядке обращения к ним из цикла, например для цикла, показанного выше: a_1; a_2, a_3, a_4, a_5. Эти значения могут задаваться в одной строке через пробел или с нажатием клавиши **<*Enter*>** после ввода одного или нескольких чисел.

Например:

а) 2 -6 8 56 34 ↵;

б) 2 ↵

-6 ↵

8 ↵

56 ↵

34 ↵

Здесь символ «↵» означает нажатие клавиши **<*Enter*>.**

Однако, если требуется ввести достаточно большой массив данных, последовательность чисел удобно вводить в виде таблицы построчно так, чтобы числа, соответствующие одной строке матрицы, располагались в отдельной строке экрана, отделенные друг от друга пробелами. Например:

```
var a:array[1..5, L. 7] of real;  { матрица a из 5 строк по 7 элементов }
begin
for i:=1 to 5 do                  { цикл ввода строк массива: a1, a2, a3, a4, a5}
  begin
    for j:=1 to 7 do              { цикл ввода элементов i-й строки: }
    Read(a[i, j]);                { ai,1, ai,2, ai,3, ai,4, ai,5, ai,6, ai,7 }
    Readln;                       {очищаем буфер ввода}
  end;...
```

Аналогичным образом выполняется и вывод значений элементов массива.

2. Типовые примеры.

Пример 1. Разработать программу определения максимального элемента одномерного массива $a(5)$ и его номера. Элементы массива ввести с клавиатуры. На экран вывести исходный массив a, максимальный элемент a_{max} и его номер i_{max}.

Для выполнения операции ввода массива используем цикл с заданным числом повторений. Поиск максимального элемента массива выполним следующим образом.

Запомним в качестве максимального первый элемент, т. е. сохраним величину этого элемента в переменной с именем *amax* и зафиксируем в переменной с именем *imax* его номер. Затем будем последовательно просматривать элементы массива, сравнивая их со значением, хра-

нящимся в *amax*. Если очередной элемент больше значения в *amax*, то сохраняем его в качестве максимального в *amax* и запоминаем его номер в *imax*. Для организации последовательного просмотра используем цикл с заданным числом повторений, изменяя переменную цикла от 2 до 5, так как первый элемент уже учтен. Просмотрев все элементы массива, найдем максимальный элемент и его номер. После этого необходимо вывести на экран исходный массив, максимальный элемент и его номер.

На рис. 2.1 представлена схема алгоритма программы. Поскольку операции ввода-вывода массивов выполняют однотипно, на схеме алгоритма соответствующих циклов, так же как и запросов на ввод данных, обычно не показывают. Вместо этого в схему вставляют блок операции ввода/вывода, в котором указано имя массива и количество элементов, участвующих в операции.

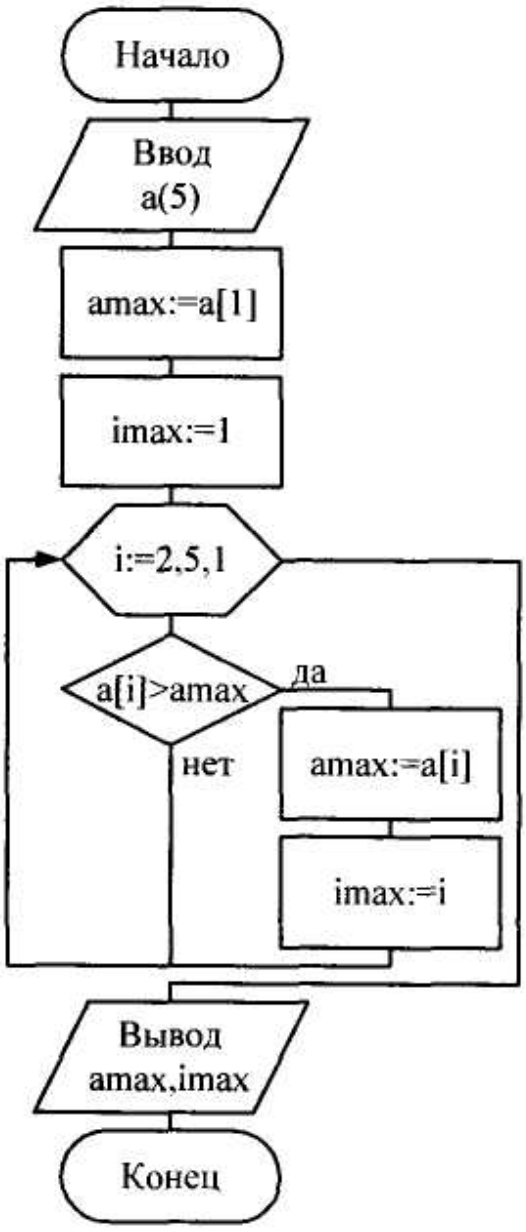


Рис. 2.1. Блок-схема алгоритма поиска максимального элемента массива *a*

Текст программы имеет следующий вид:

```
Program example_1;
Var a: array[1..5] of real;
    amax: real;
    i,imax: integer;
Begin
Writeln('Vvedite 5 elementov massiva:'); {запрос на ввод массива}
{ввод элементов массива}
for i:=1 to 5 do Read(a[i]); Readln;
{поиск максимального элемента}
amax:=a[1]; imax:=1;
for i:=2 to 5 do
  if a[i]>amax then
  begin
   amax:=a[i]; imax:=i;
  end;
{вывод массива}
Writeln('Ishodhiy massiv:');
for i:=1 to 5 do Write(a[i]:5:2); Writeln;
{вывод результата}
Writeln ('Maximum element =' amax:5:2,', Nomer max elementa =', imax:1);
End.
```

Пример 2. Разработать программу вычисления сумм элементов строк матрицы $A(4,5)$. Полученные суммы записать в новый массив B.

Итак, задана матрица, имеющая 4 строки и 5 столбцов. Требуется сформировать одномерный массив B из четырех элементов, который будет содержать суммы элементов строк. Распечатать результат лучше так, чтобы суммы были выведены после соответствующей строки матрицы.

Программа должна начинаться с ввода матрицы. Основной цикл программы – цикл по строкам. Переменная цикла i в нем будет изменяться от 1 до 4. Для каждой i-й строки в этом цикле должно выполняться суммирование элементов. Суммирование будем осуществлять методом накопления, для чего перед суммированием обнулим соответствующий i-й элемент массива B, а затем в цикле выполним добавление элементов строки. После завершения цикла суммирования эту строку и ее сумму можно сразу выводить. На рис. 2.2 представлена схема алгоритма программы (пунктиром выделено суммирование элементов i-й строки).

Текст программы имеет следующий вид:

```
Program example_2;
Var A: array[1..4,1..5] of real;
    B: array[1..4] of real;
```

```
    i,j: integer;
Begin
Writeln('Vvedite elemeti matritsi po strokam:');
for i:=1 to 4 do                          { вводим построчно элементы матрицы }
  begin
    for j:=1 to 5 do read(A[i,j]);
    readln;
  end;
Writeln ('Rezultat:');
for i:=1 to 4 do                          { для каждой строки }
  begin
    B[i]:=0;                              { обнуляем накапливаемую сумму }
    for j:=1 to 5 do B[i]:=B[i]+A[i,j];   { суммируем элементы строки }
    for j:=1 to 5 do Write(A[i,j]:7:2);   { выводим строку }
    Writeln(' Summa = ', B[i]:7:2);     { выводим сумму }
  end;
end.
```

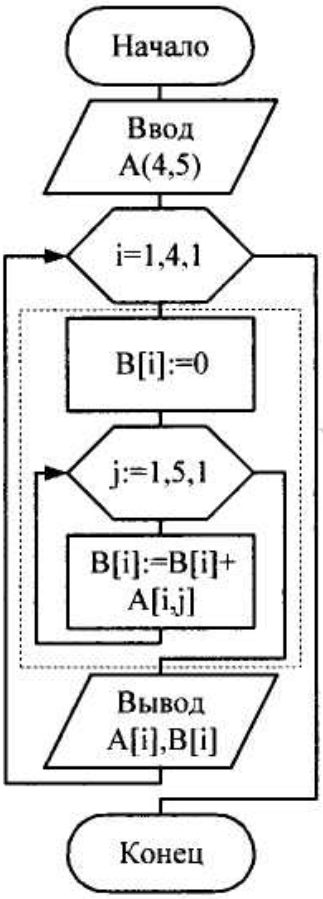


Рис. 2.2. Блок-схема алгоритма нахождения сумм элементов строк матрицы *A*

3. Задание к лабораторным работам №5, 6

1). Составить программу обработки одномерного массива в соответствии с вариантом задания. В результате работы программа должна выводить на экран значения элементов исходного массива, а также результаты расчета в формате с фиксированной десятичной точкой.

2). Составить программу обработки двумерного массива в соответствии с вариантом задания. В результате работы программа должна выводить на экран значения элементов исходной матрицы в виде таблицы по строкам и столбцам, а также результаты расчета в формате с фиксированной десятичной точкой.

3). Разработку программ выполнить в среде программирования *Delphi* 5 или *Delphi* 7 в форме консольного приложения программу соответственно под именем *Project5* и *Project6.*

4). Выполнить компиляцию и провести отладку и тестирование программ.

Методические указания

Для выполнения лабораторных работ №5, 6 следует повторить следующие разделы математики:

- скалярное произведение двух векторов;
- главная и побочная диагонали матрицы;
- след и норма матрицы;
- диагональная и транспонированная матрицы.

Список использованных источников

1. Иванова Г.С. Основы программирования: Учебник для вузов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001, 392 с. (Сер. Информатика в техническом университете).

2. Культин Н.Б. Программирование на Object Pascal в Delphi 5. – СПб.: БХВ – Санкт-Петербург, 2000, 464 с.

Государственное образовательное учреждение высшего профессионального образования

МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(государственный технический университет)

Кафедра «Проектирование вертолетов»

Маслов А.Д.

Алгоритмические языки и программирование на ЭВМ

Лабораторные работы № 7, 8

Процедуры и функции

Утверждено на заседании кафедры

5 июля 2007 г.

г. Москва

2007 г.

Введение

Настоящее пособие предназначено для студентов специальности «Самолето- и вертолетостроение» (специализация «Вертолетостроение»), изучающих дисциплину «Алгоритмические языки и программирование».

Целью работ является изучение принципов создания в программах на алгоритмическом языке *Pascal* подпрограмм и приобретение практических навыков по разработке и отладке в среде *Delphi* процедур и функций.

1. Подпрограммы

Большие программы обычно разрабатывают и отлаживают по частям. Целесообразно при этом, чтобы каждая такая часть, называемая подпрограммой, была оформлена так, чтобы ее можно было использовать при решении аналогичной подзадачи в той же программе или даже при решении других задач. В языке *Pascal* реализованы два типа подпрограмм: процедуры и функции.

Процедуры и функции представляют собой относительно самостоятельные фрагменты программы, соответствующим образом оформленные и снабженные именем (программные блоки). По правилам языка *Pascal* подпрограммы должны быть описаны перед использованием в разделе описаний программного блока, который их использует (основной программы или вызывающей подпрограммы).

Подпрограмма имеет такую же структуру, как и основная программа, т. е. включает заголовок, раздел описаний и раздел операторов, но заканчивается не точкой, а точкой с запятой. Заголовок определяет форму вызова подпрограммы. В разделе описаний объявляют внутренние ресурсы подпрограммы (переменные, типы, внутренние подпрограммы). Описанные в подпрограмме константы, переменные, типы и т. п. действуют только в этой подпрограмме и называются локальными. Раздел операторов содержит инструкции подпрограммы в операторных скобках ***begin***...***end***.

Данные для обработки процедуры и функции получают из вызвавшей их основной программы или подпрограммы через параметры. Результаты же они обычно должны возвращать вызвавшей программе или подпрограмме.

Передача данных через параметры. Список параметров описывается в заголовке подпрограммы. Параметры, перечисленные в этом списке, получили название формальных, так как для их размещения не отводится память. При обращении к подпрограмме для каждого параметра должно быть указано фактическое значение - литерал, константа или переменная, того же типа, что и формальный параметр. Несоответствие типов и количества формальных и фактических параметров выявляется компилятором. Нарушение порядка следования фактических параметров, если это нарушение не связано с несовпадением количества параметров или их типов,

приводит к нарушению логики работы программы и часто может быть обнаружено только при тестировании программы.

В *Pascal* параметры в подпрограмму могут передаваться двумя способами:

- как значения – в подпрограмму передаются копии значений параметров, и никакие изменения этих копий не возвращаются в вызывающую программу;
- как переменные – в подпрограмму передаются адреса фактических параметров, соответственно все изменения этих параметров в подпрограмме на самом деле происходят с переменными, переданными в качестве фактических параметров; такие параметры при описании помечаются служебным словом ***var*** ;

Использование параметров структурных типов, таких как массивы, имеет особенность: тип таких параметров должен быть предварительно объявлен в инструкции описания типа ***type***.

Например:

```
type mas = array[1..10] of real;
. . .
procedure A (M:mas; ...);
```

1.1. Процедуры

Процедура имеет вид

```
procedure имя_процедуры (список_формальных_параметров);
{ тело процедуры }
    раздел_описаний;
 begin
    раздел_операторов
 end;
```

Если в состав программы входит процедура, то такая программа имеет вид

```
program имя_программы;
раздел описания глобальных переменных;
procedure имя_процедуры (список_формальных_параметров);
begin
  раздел_описания_локальных_переменных;
  раздел_операторов_процедуры
end;
begin
  раздел_операторов_основной_программы
end.
```

Работа программы начинается с основной программы. Процедура выполняется при наличии обращения к ней из основной программы. После выполнения процедуры результаты передаются в вызывающую программу. Основная программа продолжает работу, начиная с оператора, стоящего следом за оператором вызова процедуры.

Ниже приведен пример процедуры вычисления суммы $S=\sum_{i=1}^{m}\frac{1}{i}$.

```
procedure Sum(k,m:integer; var S:real);
var i:integer;
 begin
   S:=0;
   for i:=k to m do
     S:=S+1/i
 end;
```

Здесь *k*, *m*, *S* – формальные параметры процедуры. При этом *k*, *m* являются параметрами-значениями, а *S* – параметром-переменной.

Обращение к процедуре (вызов процедуры) осуществляется с помощью отдельного оператора:

имя_процедуры **(***список_фактических_параметров***);**

Между формальными и фактическими параметрами должно быть взаимно однозначное соответствие по количеству, порядку расположения и типу данных. Так, например, обращение к процедуре *Sum* может иметь вид

```
Sum (1,10,z);
```

Оператор вызова процедуры выполняется следующим образом. В теле процедуры формальным параметрам-значениям присваиваются значения соответствующих фактических параметров. Этот процесс называется вызовом по значению. Параметры-переменные заменяются на соответствующие фактические параметры. Этот процесс называется вызовом по имени.

1.2. Функции

Функция имеет вид

```
function имя_функции (список_формальных_параметров): тип;
{ тело функции }
    раздел_описаний;
  begin
    раздел_операторов
  end;
```

Здесь *тип* – тип возвращаемого результата.

В отличие от процедуры функция всегда возвращает в точку вызова скалярное значение, имя типа которого указано в заголовке функции. Поэтому в теле функции обязательно наличие специальной переменной с именем функции, которой должно присваиваться значение результата вычисления функции. Обычно оператор присваивания имени функции значения результата ставится в конце тела функции перед ***end***. Именно это значение и будет возвращено в место вызова функции в качестве ее результата.

Ниже приведен пример функции вычисления суммы квадратов чисел $S=\sum_{i=1}^{n} i^2$.

```
function Sum(k,n:integer):real;
 var i:integer;
     s:real;
 begin
  s:=0;
  for i:=k to n do s:=s+sqr(i);
  Sum:=s
 end;
```

Обращение к функции (вызов функции) осуществляется по имени функции со списком фактических параметров, заключенных в круглые скобки. При этом вызов функции можно осуществлять в составе выражений везде, где возможно использование выражений (в операторе присваивания, в операторе вывода и т. д.), например:

переменная := *имя_функции*(*список_фактических_параметров*);

Так, например, обращение к функции *Sum* можно записать следующим образом:

```
z:=Sum(1,10);
```

2. Типовые примеры.

Пример 1. Разработать программу, которая определяет площадь четырехугольника по заданным длинам сторон и диагонали.

Будем считать площадь четырехугольника как сумму площадей двух треугольников, определенных по формуле Герона. Вычисление площади треугольника оформим как подпрограмму. Исходные данные такой подпрограммы – длины сторон треугольника. Подпрограмма не должна менять значения параметров, поэтому их можно передать как параметры-значения. Результат работы этой подпрограммы – скалярное значение Поэтому она может быть реализована как функция. Однако ее также можно реализовать как процедуру, которая возвращает результат через параметр-переменную. Схемы алгоритма данной программы с использованием подпрограмм обоих типов приведены на рис. 2.1.

Ниже приведены тексты соответствующих программ.

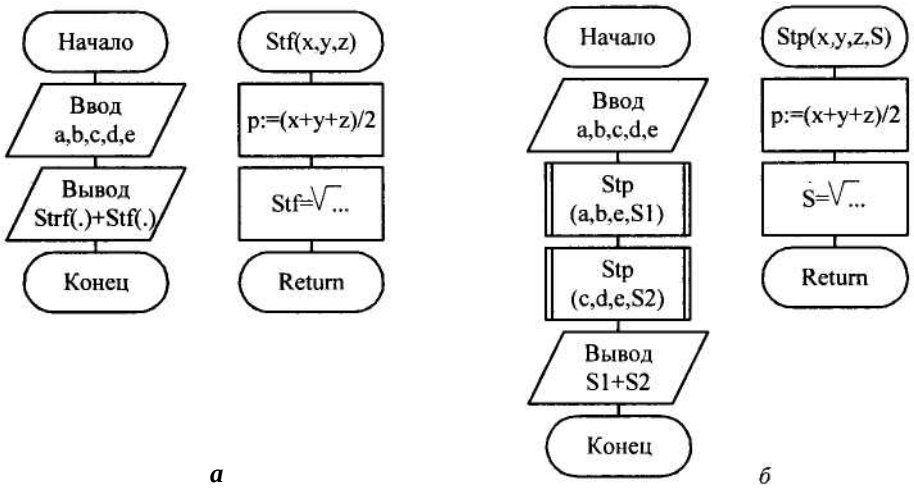


Рис. 2.1. Блок-схемы алгоритмов программы определения площади четырехугольника с использованием функции (*а*) и процедуры (*б*)

Вариант с использованием функции:

```
Program example_1a;
Var A,B,C,D,E: real;                {глобальные переменные}
{описание функции}
Function Stf(X,Y,Z:real):real;
  Varp: real;                       {локальная переменная}
  begin                             {раздел операторов функции}
    p:=(X+Y+Z)/2;
    Stf: =sqrt(p*(p-X) *(p-Y) *(p-Z));
  end;
{раздел операторов основной программы}
begin
Writeln('Vvedite dlini 4 storon i diagonal');
Readln(A,B,C,D,E);
Writeln('Ploschad = ', Stf(A,B,E)+Stf(C,D,E):7:3);
end.
```

Вариант с использованием процедуры:

```
Program example_1b;
Var A,B,C,D,E: real;
    S1,S2: real;                          {глобальные переменные}
{описание процедуры}
Procedure Stp(X,Y,Z:real; var S:real);
 Var p:real;                              {локальная переменная}
 begin                                    {раздел операторов процедуры }
  p:=(X+Y+Z)/2;
```

```
  S: =sqrt(p*(p-X) *(p-Y) *(p-Z));
 end;
{раздел операторов основной программы}
begin
Writeln(''Vvedite dlini 4 storon i diagonal');
Readln(A,B,C,D,E);
Stp(A,B,E,S1);                          {вызов процедуры}
Stp(C,D,E,S2);                          {вызов процедуры}
Writeln('Ploschad = ', S1+S2:7:3);
end.
```

Пример 2. Разработать подпрограмму суммирования элементов массива размерности n, $n < 10$.

Поскольку результат – скалярное значение, то будем использовать подпрограмму-функцию. Описание типа «массив из 10 целых чисел» должно быть выполнено отдельно в разделе описаний. Новый тип *mas* затем используется при объявлении массива *a* и при объявлении типа массива, передаваемого через параметр-значение.

Текст программы имеет следующий вид:

```
Program example_2;
type mas = array[1..10] of integer; {тип «массив из 10 целых чисел»}
Var a: mas;
    i,n:integer;
Function sum(b:mas; n:integer):integer;
 Var s:integer;
     i:integer;
 begin
 s:=0;
 for i:=1 to n do s:=s+b[i];
 sum: =s;
 end;
begin
Readln(n);
for i:=1 to n do Read(a[i]);
Readln;
Writeln('Summa = ',sum(a,n));
end.
```

3. Задание к лабораторным работам №7, 8

1). Пользуясь функцией, составить программу в соответствии с вариантом задания. В результате работы программа должна выводить на экран значения исходных данных и результаты расчета в формате с фиксированной десятичной точкой. Разработанная функция должна быть автономной и при необходимости может быть помещена в библиотеку стандартных модулей пользователя.

2). Пользуясь процедурой, составить программу в соответствии с вариантом задания. В результате работы программа должна выводить на экран значения исходных данных и результаты расчета в формате с фиксированной десятичной точкой. Разработанная процедура должна быть автономной и при необходимости может быть помещена в библиотеку стандартных модулей пользователя.

3). Разработку программ выполнить в среде программирования *Delphi* 5 или *Delphi* 7 в форме консольного приложения программу соответственно под именем *Project7* и *Project8*.

4). Выполнить компиляцию и провести отладку и тестирование программ.

Методические указания

Для выполнения лабораторных работ №7, 8 следует повторить следующие разделы математики:

- скалярное произведение двух векторов;
- главная и побочная диагонали матрицы;
- след и норма матрицы;
- диагональная и транспонированная матрицы.

Список использованных источников

1. Демина И.Д., Ивашенцева Н.А., Мевис А.В. Сборник задач по программированию на языке Паскаль: Учебное пособие / Под ред. А.В. Мевис. – М.: Изд-во МАИ, 1992, 60 с.

2. Иванова Г.С. Основы программирования: Учебник для вузов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2001, 392 с. (Сер. Информатика в техническом университете).

3. Культин Н.Б. Программирование на Object Pascal в Delphi 5. – СПб.: БХВ – Санкт-Петербург, 2000, 464 с.