Глава 2

Среда программирования

В этой главе…

- ✓ Интегрированная среда программирования
- ✓ Организация работы ИСП в Windows
- ✓ Текстовый редактор ИСП
- ✓ Отладка программы
- ✓ Технология работы с модулями
- ✓ Директивы компиляции
- ✓ Обработка кода завершения программы

2.1. Интегрированная среда программирования

Изложение материала ориентировано на использование интегрированной среды программирования (ИСП) Borland (Turbo) Pascal 7.0 под управлением операционной системы Windows. Рассматриваемая ИСП объединяет (интегрирует) в своем составе все необходимые функции от написания до выполнения готовой программы с целью получения результатов. Ниже рассматриваются наиболее популярные и практически значимые из них.

2.1.1. Главное окно ИСП

После запуска ИСП на экране появляется ее главное окно. В верхней строке окна находится главное меню ИСП. Оно содержит перечень пунктов, наименования которых являются условными обозначениями групп сходных команд:

- File – операции с файлами текстов программ;
- Edit – редактирование текстов программ;
- Search – поиск заданных объектов в тексте программы;
- Run – выполнение программы;
- Compile – компиляция программы;
- Debug – отладка программы;
- Tools – вспомогательные программы;
- Options – параметры ИСП;
- Window – операции с окнами редактора;
- Help – справочная служба.

Для навигации в пределах главного меню используется *селектор* – цветной прямоугольник, управляя которым, можно указывать на те или иные объекты. Чтобы попасть в главное меню (перевести в него селектор), следует нажать клавишу <F10>, чтобы выйти из него – клавишу <Esc>.

Все пункты главного меню имеют подчиненные выпадающие меню с наименованиями нескольких сходных по своему содержанию команд. Если селектор находится в главном меню, то можно выбрать любой его пункт и соответствующее подчиненное меню с помощью клавиш управления курсором и последующим нажатием клавиши <Enter>.

Удобно также попадать в главное меню и перемещаться между его подчиненными выпадающими меню, нажимая <Alt+"горячая" клавиша>. "Горячие" клавиши совпадают с теми буквами в наименованиях пунктов меню, которые выделены красным цветом.

В подчиненных меню можно пользоваться клавишами управления курсором для перемещения селектора вверх и вниз по перечню команд и клавишей <Enter>, когда будет выбрана нужная. И тут тоже проще пользоваться "горячими" клавишами, совпадающими с выделенными буквами.

Нижняя строка главного окна ИСП — *информационная*. Она используется для размещения подсказок по функциям основных клавиш.

В главном окне ИСП можно пользоваться и мышью. В частности, для выбора пункта главного меню или команды из подчиненного меню достаточно щелкнуть на соответствующем пункте левой кнопкой мыши.

Команды из подчиненных меню могут иметь в конце своего наименования многоточие "..." либо символ "▶". В первом случае выбор команды связан с появлением диалогового окна, в котором должны быть установлены параметры выполнения этой команды. Во втором случае выбор команды приводит к появлению дополнительного меню.

В дальнейшем для обозначения той или иной команды используется цепочка наименований пунктов последовательных меню, начиная с главного и продолжая подчиненными. При этом между отдельными пунктами используется знак "⇨".

Любая команда может быть указана также с помощью соответствующих "горячих" клавиш. При этом используются условные знаки "+" и "&". Знак "+" обозначает одновременное нажатие, знак "&" — последовательное. Например, команде Window⇨Close соответствуют "горячие" клавиши <Alt+W&C>. Эта комбинация должна набираться следующим образом: удерживая в нажатом состоянии клавишу <Alt>, кратковременно нажимают клавишу с буквой "W", затем все клавиши отпускают, после чего нажимают клавишу с буквой "C".

Во многих случаях для команд из подчиненных меню рядом с наименованием указываются комбинации клавиш, как например, <Alt+F3> рядом с Window⇨Close. Это значит, что выбор такой команды для ускорения работы может быть заменен нажатием указанных клавиш.

2.1.2. Управляющие элементы диалоговых окон

Диалоговые окна появляются на экране при выборе тех команд из подчиненных меню, наименования которых оканчиваются многоточием. Они предназначены для установки параметров выполнения этих команд. Наименование выполняемой команды находится на верхней рамке окна. Управляющими элементами диалоговых окон являются поля и кнопки.

Поля диалоговых окон предназначены для установки значений параметров выполнения команды. Основные разновидности полей таковы.

- **Поля, значения в которые вводятся с помощью клавиатуры**; при этом предоставляется дополнительная возможность выбора из списка предыдущих значений, который раскрывается нажатием клавиши <↓>; к таким полям относятся, например, поле Name диалогового окна Open a File (команда File⇨Open...), поле Parameter диалогового окна Program Parameters (команда Run⇨Parameters...) и др.
- **Поля с прокручиваемым набором значений**; при этом указание нужного из них осуществляется с помощью селектора, перемещаемого клавишами управления курсором; к таким полям относятся, например, поле Directory tree с разворачиваемым деревом каталогов в диалогового окне Change Directory (команда File⇨Change dir...), поле Files с прокручиваемым списком файлов в диалоговом окне Primary File (команда Compile⇨Primary File...) и др.
- **Поля переключателей**; например, поле Target Platform диалогового окна Target (команда Compile⇨Target...), поле Screen sizes диалогового окна Preferences (команда Options⇨Environment⇨Preferences...), поле Direction диалогового окна Find (команда Search⇨Find...) и др.; изменение положения переключателя осуществляется с помощью клавиш управления курсором.
- **Поля флажков**; например, поле Runtime errors диалогового окна Compiler Options (команда Options⇨Compiler...), поле Options диалогового окна Replace (команда Search⇨Replace...) и др.; установка либо снятие флажка осуществляется нажатием клавиши пробела.

Чтобы с тем или иным элементом диалогового окна возможно было выполнение тех или иных действий (ввод значения, прокручивание списка, выбор (нажатие) кнопки и установка флажка), этот элемент должен быть предварительно активизирован. Такой активный элемент выделяется в окне альтернативным цветом. Для последовательной активизации элементов окна в прямом направлении нажмите клавишу <Tab>, в обратном — клавиши <Shift+Tab>.

“Нажатие” той или иной кнопки диалогового окна инициирует определенный вариант выполнения соответствующей команды и реализуется применением к ней клавиши <Enter>, когда она (данная кнопка) находится в активном состоянии. Например, “нажатие” кнопки Cancel отменяет процесс выполнения команды, “нажатие” кнопки Ok — наоборот, приводит к выполнению команды. При этом в обоих случаях диалоговое окно закрывается. “Нажатие” кнопки Help вызывает окно контекстной справки и др. Отметим, что для отмены выполнения команды и закрытия диалогового окна чаще удобнее пользоваться клавишей <Esc>.

При выполнении операций с элементами диалоговых окон иногда бывает удобнее использовать мышь. Особенности здесь таковы:

- чтобы активизировать элемент и выполнить соответствующую ему операцию, как правило, достаточно щелкнуть на нем левой кнопкой мыши;
- для установки нужного значения переключателя или флажка достаточно щелкнуть на нем левой кнопкой мыши;
- дополнительные элементы диалоговых окон, специально ориентированные на мышь:
  - кнопка "■" — закрытие окна без выполнения соответствующей окну команды;
  - кнопки со стрелками "◀", "▶", "▼", "▲" — прокручивание содержимого поля с набором значений в соответствующую сторону;
  - кнопка "↓" — раскрытие списка;
  - ползунок "■" на полосе прокрутки, перетягивание которого при удерживаемой левой кнопке мыши в нужную сторону обеспечивает быструю плавную прокрутку содержимого поля с набором значений;
  - части полосы прокрутки по обе стороны ползунка "■", щелкая на которых, можно выполнять дискретную прокрутку содержимого поля с набором значений.

2.1.3. Меню File

В этом меню сгруппированы команды для выполнения операций с файлами текстов программ.

- Команда File⇨New (<Alt+F&N>). Открытие нового пустого окна текстового редактора для вновь создаваемой программы.

  Имя файла по умолчанию для новой программы имеет вид `NONAMExx.PAS`, где `xx` — условное обозначение номера от `00` до `99`. Может быть открыто одновременно несколько окон для параллельного создания разных программ.

- Команда File⇨Open... (<F3>). Выбор нужного файла на диске и открытие окна редактора с текстом соответствующей программы. Может быть открыто одновременно несколько окон для параллельной работы над разными программами.

  Все действия выполняются в диалоговом окне Open a File. Имя нужного файла с текстом программы вводится в поле Name. Это имя можно также выбрать из списка, нажав клавишу <↓>. Кроме того, это имя можно найти в поле Files, перемещаясь по дереву каталогов. При этом следует учитывать, что подчиненные каталоги отображаются в этом поле в виде

"<*имя*>\", родительский — в виде "..\". Чтобы попасть в любой из них, достаточно установить на него селектор и нажать клавишу <Enter>. Для смены диска необходимо ввести его имя в поле Name. Полный путь к выбираемому каталогу или файлу, а также их характеристики отображаются в нижней строке окна Open a File.

Следует помнить, что по умолчанию в поле Files отображаются файлы со стандартным расширением PAS. Для того чтобы видеть в этом поле файлы с любыми расширениями, в поле Name нужно ввести шаблон "*.*" и нажать клавишу <Enter>, для файлов с расширением TXT следует использовать шаблон "*.TXT" и т.п.

При использовании кнопки Open открывается новое окно с текстом программы, при использовании кнопки Replace — текст программы помещается в то же самое окно, замещая предыдущую. Если сделать попытку замещения, не сохранив предыдущую программу, на экране появляется окно сообщений Information с предложением сначала сохранить предыдущую программу "Save?". На это предложение следует дать ответ, щелкнув на одной из кнопок Yes — "Да, сохранить", No — "Нет, не сохранять" или Cancel — "Отменить открытие".

- Команда File⇨Save (<F2>). Запись на диск (сохранение) текста редактируемой программы.

  Текст программы, который находится в текущем окне редактора, будет сохранен на диске в том же файле и в том же каталоге, где он хранился первоначально до начала сеанса редактирования.

  Для файлов NONAMExx.PAS появляется диалоговое окно Save File As, в котором сначала следует указать для нового файла нужное имя. Новый файл при этом будет сохранен в текущем каталоге. При желании для сохранения нового файла в поле Files можно выбрать какой-либо другой каталог. Расширение в имени файла указывать не обязательно, поскольку расширение PAS при этом присваивается по умолчанию. Если файл с указанным именем в указанном каталоге уже существует, то на экране появляется окно предупреждения Warning с предложением "Overwrite?", что значит "Заменить существующий файл?". На это предложение следует дать ответ, щелкнув на одной из кнопок: Yes — "Да, заменить", No — "Нет, не заменять" или Cancel — "Отменить переименование".

- Команда File⇨Save as... (<Alt+F&A>). Сохранение текста программы в другом файле.

  Появляется диалоговое окно Save File As, в котором можно указать имя другого файла для сохраняемой программы. Под этим именем новый файл

появится в текущем каталоге. При желании для нового файла можно указать и любой другой каталог. Существующий же файл с этой программой останется без изменений.

- Команда File⇨Save all (<Alt+F&L>). Сохранение текстов всех редактируемых программ в соответствующих файлах.
- Команда File⇨Change dir... (<Alt+F&C>). Выбор и установка нового текущего каталога.

  Имя каталога, выбираемого в качестве текущего, вводится в поле Directory name диалогового окна Change Directory. Наиболее удобным образом это осуществить, развернув дерево каталогов в поле Directory tree. При необходимости там же можно выбрать и другой диск, если развернуть пункт Drives.

  Установив селектор на выбранный по дереву каталог, его наименование перемещают в поле Directory name посредством щелчка на кнопке Chdir, после чего щелчок на кнопке OK переводит этот каталог в статус текущего.

  Результаты поиска и выбора нужного нового текущего каталога можно аннулировать и тем самым оставить текущий каталог прежним, если щелкнуть на кнопке Revert.
- Команда File⇨DOS shell (<Alt+F&D>). Временный выход из ИСП в DOS.

  Это может понадобиться для выполнения команд из командной строки операционной системы. Чтобы вернуться в ИСП, выполните команду `EXIT`.
- Команда File⇨Exit (<Alt+X>). Завершение работы в ИСП.

2.1.4. Меню Edit

В этом меню сгруппированы команды для выполнения действий с выделенным фрагментом текста программы, находящейся в текущем окне редактора. При этом используется текстовый буфер обмена (clipboard).

- Команда Edit⇨Undo (<Alt+Backspace>). Отмена последней из операций редактирования текста (откат).

  При соответствующей настройке однократный откат позволяет отменить и более чем одну из предыдущих однотипных операций. Для этого в диалоговом окне Editor Options, появляющемся после выбора команды Options⇨Environment⇨Editor..., предварительно должен быть отмечен пункт Group Undo.
- Команда Edit⇨Redo (<Alt+E&R>). Отмена последней из команд Edit⇨Undo (откат отката).

- Команда Edit⇨Cut (<Shift+Delete>). Перемещение выделенного текста из окна редактора в буфер обмена.
- Команда Edit⇨Copy (<Ctrl+Insert>). Копирование выделенного текста из окна редактора в буфер обмена.
- Команда Edit⇨Paste (<Shift+Insert>). Копирование выделенного текста из буфера обмена в окно редактора.
- Команда Edit⇨Clear (<Ctrl+Delete>). Удаление выделенного текста из окна редактора.
- Команда Edit⇨Show clipboard. Переход в окно с содержимым буфера обмена, что может понадобиться для выбора и выделения в нем нужного текста.

2.1.5. Меню Search

Данное меню содержит различные команды, связанные с поиском заданных объектов в тексте программы, находящейся в текущем окне редактора.

- Команда Search⇨Find... (<Alt+S&F>). Поиск в тексте программы заданного фрагмента.

  Подготовительные действия выполняются в диалоговом окне Find. Основные из них: ввести искомый фрагмент в поле Text to find, в поле Direction установить переключатель направления поиска вперед (Forward) или назад (Backward) по тексту, а в поле Origin установить переключатель начала поиска от текущего положения курсора (From cursor) или от начала текста (Entire scope).
- Команда Search⇨Replace... (<Alt+S&R>). Поиск в тексте программы и замена одного заданного фрагмента другим.

  Подготовительные действия выполняются в диалоговом окне Replace. Основные из них: ввести заменяемый фрагмент в поле Text to find, заменяющий фрагмент – в поле New text, в поле Direction установить переключатель направления замен вперед (Forward) или назад (Backward), а в поле Origin установить переключатель начала замен от текущего положения курсора (From cursor) или от начала текста (Entire scope).

  Процесс замен можно настроить таким образом, чтобы по каждой конкретной замене в появляющемся окне сообщений Information выдавался запрос о необходимости ее выполнения “Replace this occurrence?”. На каждый такой запрос придется давать ответ щелчком на одной из кнопок Yes – “Да”, No – “Нет” или Cancel – “Отменить процесс замен”. Соответствующая настройка состоит в том, что в поле Options устанавливается флажок на пункте Prompt on replace.

Щелчок на кнопке OK диалогового окна приводит к выполнению одной очередной замены, а щелчок на кнопке Change all – к выполнению всех возможных замен в заданном направлении.

- Команда Search⇨Search again (<Alt+S&S>). Повтор последней выполненной операции поиска или замены с теми же самыми параметрами.
- Команда Search⇨Go to line number... (<Alt+S&G>). Перемещает текстовый курсор в строку программы с заданным порядковым номером (делает эту строку текущей).

  Требуемый номер строки программы указывается в поле Enter new line number диалогового окна Go to Line Number. Команда удобна тем, что позволяет быстро находить нужные фиксированные точки программы.

2.1.6. Меню Run

В составе данного меню имеются разнообразные команды, связанные с запуском на выполнение программы, текст которой находится в текущем окне редактора.

- Команда Run⇨Run (<Ctrl+F9>). Компиляция и выполнение программы.

  Если программа уже была откомпилирована ранее, то ее выполнение начинается сразу же.

  Прекратить выполнение "зациклившейся" программы можно с помощью *Диспетчера задач Windows*, окно которого вызывается нажатием клавиш <Alt+Ctrl+Delete>. В нем следует выбрать вкладку Приложения, выделить задачу "Borland (Turbo) Pascal 7.0" и щелкнуть на кнопке Снять задачу.
- Команда Run⇨Step over (<F8>). Выполнение операторов очередной строки текста программы с последующей приостановкой. При этом строки подпрограмм не отслеживаются.

  Очередная выполняемая строка отмечается альтернативным цветом, т.е. *треком выполнения*. В момент приостановки есть возможность проверить и проанализировать текущие значения переменных программы.
- Команда Run⇨Trace into (<F7>). Выполнение операторов очередной строки текста программы с последующей приостановкой. При этом отслеживаются также строки подпрограмм.

  В остальном данная команда аналогична команде Run⇨Step over.
- Команда Run⇨Go to cursor (<F4>). Выполнение программы до текущей строки.

  Текущей строкой считается та, в которой находится курсор текстового редактора ИСП. Программа при этом или начинает выполняться сначала, или продолжает выполняться от места положения трека выполнения.

Таким образом, можно получить точку временной остановки программы для контроля текущих значений переменных.

- Команда Run⇨Program reset (<Ctrl+F2>). Отказ от текущего сеанса отладки программы.

  При этом трек выполнения программы снимается, память, занятая отлаживаемой программой, освобождается.

- Команда Run⇨Parameters... (<Alt+R+A>). Установка параметров командной строки запуска программы на выполнение.

  Соответствующие действия выполняются в диалоговом окне Program Parameters. В поле Parameter этого окна следует ввести строку параметров выполняемой программы, после чего щелкнуть на кнопке OK. С помощью этого средства поддерживаются стандартные возможности запуска программ, реализованные в рамках любой операционной системы, о чем идет речь ниже.

Параметры программы указываются в командной строке запуска программы на выполнение. В Windows для этого можно поступить одним из следующих способов.

- Откройте папку с EXE-файлом программы. Щелкните на ней правой кнопкой мыши и в открывшемся контекстном меню выберите пункт Свойства. В соответствующем диалоговом окне откройте вкладку Программа и дополните строку Команда: значениями параметров, указав их через пробел. Закройте окно Свойства, щелкнув на кнопке OK. Теперь выполните эту программу одним из известных способов (например, дважды щелкнув на имени программы или выбрав пункт Открыть в соответствующем контекстном меню).
- Откройте Главное меню Windows, выберите команду Все программы⇨Стандартные⇨Командная строка. В окне Командная строка с помощью команд MS DOS перейдите в каталог с EXE-файлом нужной программы. Наберите командную строку ее выполнения, указав через пробелы значения параметров, после чего нажмите клавишу <Enter>.
- Откройте Главное меню Windows и выберите команду Выполнить.... В диалоговом окне Запуск программы щелкните на кнопке Обзор.... В диалоговом окне Обзор, используя разворачивающееся дерево в поле Папка:, установите нужный каталог, укажите имя EXE-файла в поле Имя файла: и щелкните на кнопке Открыть. В диалоговом окне Запуск программы командную строку в поле Открыть: дополните значениями параметров и щелкните на кнопке OK.

В самой Паскаль-программе для обслуживания строки передаваемых ей параметров используются следующие стандартные подпрограммы:

- функция `ParamCount: Word`, значением которой является количество параметров, переданных программе в командной строке;
- функция `ParamStr(Index): String`, значением которой является заданный параметр командной строки:
  - в общем случае параметр `Index` представляет собой выражение типа `Word`, указывающее порядковый номер нужного параметра;
  - если `Index` превышает число параметров, то значением функции является пустая строка;
  - если `Index=0`, то значением функции является расширенное имя файла выполняемой программы.

2.1.7. Меню Compile

Меню содержит разнообразные команды компиляции программы.

- Команда Compile⇨Compile (<Alt+F9>). Компиляция программы, находящейся в текущем окне текстового редактора ИСП, и создание `EXE`-файла с выполняемой программой.

  В результате выполнения команды на экране появляется окно со служебной информацией и сообщением об успешном (successful) завершении компиляции. Чтобы удалить окно, достаточно нажать произвольную клавишу. Если будет найдена синтаксическая ошибка, то компилятор в специальной строке сообщит о ее типе и установит курсор в окне редактора на место найденной ошибки.

  Имя созданного `EXE`-файла совпадает с именем исходного файла с текстом программы, и по умолчанию этот файл будет помещен в тот же каталог, где находится исходный файл с текстом программы. Если же необходимо размещать создаваемые `EXE`-файлы в другом каталоге, то его следует указать в поле EXE&TPU directory диалогового окна Directories, получаемого в результате выбора команды Options⇨Directories....

  При работе в ИСП Borland Pascal `EXE`-файл всегда размещается только на диске. А вот при использовании ИСП Turbo Pascal для окончательного определения места расположения создаваемого `EXE`-файла дополнительно должна выбираться команда Compile⇨Destination (<Alt+C&D>). В этом случае возможны два варианта сохранения выполняемой программы: или на диске (Disk), или в памяти (Memory). При выборе последнего варианта после завершения работы с ИСП выполняемая программа будет утрачена.

- Команды Compile⇨Make (<F9>), Compile⇨Build (<Alt+C&B>), Compile⇨Primary file... (<Alt+C&P>) и Compile⇨Clear primary file (<Alt+C&L>) используются для компиляции программ, имеющих модульную структуру.
- Команда Compile⇨Target... (<Alt+C&T>). Установка режима компиляции программы в ИСП Borland Pascal.

  Выполняется в диалоговом окне Target, в котором переключатель Target Platform следует установить в положение Real mode Application, после чего щелкнуть на кнопке OK.

2.1.8. Меню Debug

Данное меню содержит ряд команд, предназначенных для выполнения функций, связанных с отладкой программы.

- Команда Debug⇨Breakpoints (<Alt+D&B>). Открывает диалоговое окно Breakpoints со списком контрольных точек с целью просмотра и редактирования их параметров.
- Команда Debug⇨Call stack (<Ctrl+F3>). Открывает окно Call stack с информацией о текущем состоянии программного стека.

  В процессе выполнения программы в этом окне отображаются вызовы подпрограмм с конкретными значениями фактических параметров обращения.
- Команда Debug⇨Wath (<Alt+D&W>). Открывает окно наблюдения Watches.

  В процессе отладки программы в этом окне отображаются текущие значения заданных переменных или выражений.
- Команда Debug⇨Output (<Alt+D&O>). Открывает окно выполнения программы Output.
- Команда Debug⇨User screen (<Alt+F5>). Открывает окно выполнения программы Output в полноэкранном режиме.
- Команда Debug⇨Evaluate/modify... (<Ctrl+F4>). Открывает диалоговое окно Evaluate and Modify для оперативного контроля и модификации значений переменных отлаживаемой программы.
- Команда Debug⇨Add watch... (<Ctrl+F7>). Открывает диалоговое окно Add Watch для ввода переменной или выражения, помещаемого в окно наблюдения Watches.
- Команда Debug⇨Add breakpoint... (<Alt+D&P>). Открывает диалоговое окно установки контрольных точек программы Add Breakpoint.

2.1.9. Меню Options

Данное меню содержит команды настройки ИСП.

- Команда Options⇨Compiler... (Alt+O&C). Открывает диалоговое окно настройки компилятора Compiler Options.
- Команда Options⇨Memory sizes... (Alt+O&M). Открывает диалоговое окно установки размеров программного стека и динамической памяти Memory sizes.
- Команда Options⇨Linker... (Alt+O&L). Открывает диалоговое окно настройки компоновщика программы Linker.
- Команда Options⇨Debugger... (Alt+O&B). Открывает диалоговое окно настройки процесса отладки программы Debugging/Browsing.
- Команда Options⇨Directories... (Alt+O&D). Открывает диалоговое окно назначения каталогов ИСП Directories.
- Команда Options⇨Browser... (Alt+O&W). Открывает диалоговое окно настройки параметров браузера Browser Options.
- Команда Options⇨Environment (Alt+O&E). Переход в дополнительное меню настройки параметров среды программирования.
- Команда Options⇨Open... (Alt+O&O). Открывает диалоговое окно Open Options для чтения файла конфигурации, сохраняющего настройки ИСП.
- Команда Options⇨Save (Alt+O&S). Сохранение настроек ИСП в том же файле конфигурации.
- Команда Options⇨Save as... (Alt+O&A). Открывает диалоговое окно Save Options As для сохранения настроек ИСП в новом файле конфигурации.

2.1.10. Меню Window

Данное меню содержит функции для управления окнами.

- Команда Window⇨Tile (Alt+W&T). Автоматически уменьшает размеры всех открытых окон с программами и располагает их равномерно в главном окне ИСП таким образом, что каждое из них видно полностью.
- Команда Window⇨Cascade (Alt+W&A). Все открытые окна с программами располагает друг за другом в главном окне ИСП, причем таким образом, что видно рамку каждого из них.
- Команда Window⇨Size/Move (<Ctrl+F5>). Изменяет размеры (<Shift+клавиши управления курсором>) и перемещает окно редактора (клавиши управления курсором).

  Клавиши <Home>, <End>, <PageUp> и <PageDown> перемещают окно в крайние положения (влево, вправо, вверх и вниз соответственно).

- Команда Window⇨Zoom (<F5>). Разворачивает окно редактора до максимально возможных размеров в пределах главного окна ИСП. Повторный выбор команды восстанавливает его предыдущие размеры.
- Команда Window⇨Next (<F6>). Переход к очередному окну редактора из числа открытых в прямом направлении.
- Команда Window⇨Previous (<Shift+F6>). Переход к очередному окну редактора из числа открытых в обратном направлении.
- Команда Window⇨Close (<Alt+F3>). Закрывает текущее окно редактора.
- Команда Window⇨Close all (<Alt+W&O>). Закрывает все окна редактора ИСП.
- Команда Window⇨List... (<Alt+0>). Открывает диалоговое окно Window List с перечнем всех открытых окон текстового редактора ИСП.

  При этом оказывается возможным быстро найти и выбрать нужное окно либо для перехода в него, либо для его закрытия.

2.1.11. Меню Help

Данное меню содержит команды получения справочных данных по языковым средствам Паскаля и возможностям ИСП. Эта информация предоставляется программисту в специальном окне, которое называется окном помощи Help.

Текст, находящийся в окне помощи, может содержать выделенные термины. Они являются ссылками на соответствующие разделы справки, раскрывающие содержание этих терминов. Двойной щелчок мышью на ссылке вызывает появление окна с нужным разделом.

Двойной щелчок мышью на обычном невыделенном слове приводит к его поиску в алфавитном перечне терминов. В результате курсор окна помощи устанавливается в том месте перечня, которое оказывается наиболее близким к данному слову. Если же данное слово является заголовком раздела справки, то последний и будет получен.

Любое окно помощи может быть закрыто нажатием клавиши <Esc>.

- Команда Help⇨Contents (<Alt+H&C>). Открывает окно помощи с общим оглавлением справочных данных.
- Команда Help⇨Index (<Shift+F1>). Открывает окно помощи с алфавитным перечнем терминов.
- Команда Help⇨Topic Search (<Ctrl+F1>). Осуществляет поиск в программе ближайшего к текстовому курсору термина и открывает окно помощи с соответствующим разделом справочных данных.
- Команда Help⇨Previous topic (<Alt+F1>). Позволяет получить окно помощи с предыдущей справкой. Возможен неоднократный выбор данной команды.

При этом окна помощи выводятся в порядке, обратном процессу их получения.

- Команда Help⇨Using help (<Alt+H&H>). Открывает окно помощи с информацией о порядке пользования справочной системой.

Выбор следующих команд приводит к открытию окон помощи по конкретным разделам справочных данных:

- команда Help⇨Compiler directives (<Alt+H&D>) — о директивах компилятора;
- команда Help⇨Procedures and functions (<Alt+H&O>) — о стандартных процедурах и функциях;
- команда Help⇨Reserved words (<Alt+H&R>) — о зарезервированных словах;
- команда Help⇨Standard units (<Alt+H&U>) — о стандартных модулях;
- команда Help⇨Borland Pascal Language (<Alt+H&L>) — о языковых средствах Паскаля;
- команда Help⇨Error messages (<Alt+H&E>) — о сообщениях об ошибках.

В ряде случаев результат нажатия клавиши <F1> имеет контекстную зависимость:

- на экране находится любое из окон помощи — открывается окно помощи с информацией о порядке пользования справочной системой;
- какое-либо из окон помощи на экране отсутствует, селектор находится в одном из меню — открывается окно помощи с информацией о соответствующем пункте меню;
- какое-либо из окон помощи на экране отсутствует, селектор находится в окне редактора — открывается окно помощи с информацией о порядке работы с редактором;
- при появлении сообщения об ошибке компиляции программы — открывается окно помощи с разъяснениями этой ошибки.

Справочные данные содержат множество примеров программ, которые помогают разобраться со свойствами того или иного стандартного средства языка Паскаль. Программу или любой иной произвольный текст из окна помощи можно скопировать в окно редактора следующим образом:

- выделите нужный текст в окне помощи или посредством мыши, или перемещая текстовый курсор при удерживаемой клавише <Shift>;
- скопируйте выделенный фрагмент в текстовый буфер обмена ИСП, нажав клавиши <Ctrl+Insert>;
- закройте окно помощи, нажав клавишу <Esc>;

- установите текстовый курсор в нужное место окна редактора;
- скопируйте фрагмент из текстового буфера обмена в окно редактора, нажав клавиши <Shift+Insert>.

2.2. Организация работы ИСП в Windows

2.2.1. Состав и режимы работы ИСП

Для установки комплекта программ Borland Pascal with Objects 7.0 фирмы Borland International рекомендуется использовать каталог `Soft`, который программисту собственноручно следует создать на диске `D:`. В этом каталоге могут находиться также и другие программные средства, ориентированные на работу под управлением MS DOS, но до сих пор не утратившие своей популярности. В частности, к ним относятся операционные оболочки Norton Commander и Volkov Commander.

В составе каталога `Soft` создается подкаталог `BP`, который и содержит все программные средства упомянутого комплекта.

В развернутом виде программные средства Borland Pascal with Objects 7.0 распределены по следующим основным подкаталогам:

- `...\BP\BIN` — содержит исполняемые файлы комплекта, в том числе файлы `TURBO.EXE` и `BP.EXE` интегрированных систем программирования Turbo Pascal и Borland Pascal соответственно;
- `...\BP\BGI` — содержит графические драйверы, обеспечивающие работу графической системы, а также файлы нестандартных шрифтов;
- `...\BP\UNITS` — содержит ряд стандартных модулей, в том числе модуль `GRAPH`, модуль `STRINGS` и многие другие.

Создание программ средствами комплекта Borland Pascal with Objects 7.0 может быть осуществлено для одного из трех основных режимов работы процессора: реального, защищенного и под управлением Windows. Соответственно этому потребности программирования обеспечиваются библиотечными файлами `TURBO.TPL`, `TPP.TPL` или `TPW.TPL`, расположенными в каталоге `...\BP\BIN`.

В учебных курсах по языку Паскаль наиболее популярно программирование для реального режима работы процессора. Соответствующий библиотечный файл `TURBO.TPL` содержит стандартные модули `SYSTEM`, `CRT`, `DOS`, `PRINTER` и `OVERLAY`. Дополнительно могут использоваться модули `GRAPH` и `STRINGS`, находящиеся в отдельных `TPU`-файлах.

Для извлечения того или иного модуля из TPL-файла может использоваться специализированная утилита TPUMOVER.EXE (каталог ...\BP\BIN). Например, для извлечения модуля SYSTEM соответствующая команда MS DOS имеет вид: TPUMOVER TURBO.TPL *SYSTEM.TPU, где символ "*" обозначает операцию извлечения.

2.2.2. Файлы и каталоги ИСП

Исторически ИСП не ориентирована на современные возможности файловой системы Windows. Поэтому необходимо знать специфику файлов, используемых и создаваемых или посредством самих Паскаль-программ, или текстовым редактором ИСП.

Основным внешним параметром файла, благодаря которому он обнаруживает себя в окружающем пространстве, является его имя. В рамках ИСП на имена файлов налагаются ограничения, известные по операционной системе MS DOS:

- имя файла составляют из двух частей — основной и расширения; чтобы отделить одну от другой, используют точку ".";
- для записи имени файла чаще всего используют латиницу, цифры, а также знак подчеркивания "_"; при этом строчные и прописные буквы не различаются;
- основная часть имени содержит от одного до восьми знаков; расширение рекомендуется составлять из трех знаков или выбирать стандартным.

Наиболее актуальны для нас стандартные расширения PAS, EXE и TPU, выбираемые по умолчанию. Первое из них использует текстовый редактор ИСП для хранения файлов, содержащих тексты Паскаль-программ. Второе и третье — использует компилятор ИСП, создавая файлы исполняемых программ и подключаемых модулей программиста.

Для текущего хранения файлов с текстами создаваемых программ непосредственно на рабочем логическом диске целесообразно выделить отдельный каталог, например, D:\User. Подкаталоги каталога User могут принадлежать разным программистам, например, D:\User\Ivanov. Дальнейшее разбиение на подкаталоги зависит от потребностей конкретного программиста и может соответствовать различным его проектам, например, D:\User\Ivanov\Example и т.д. Создавать каталоги можно обычными средствами Windows. Имена каталогов должны соответствовать тем же требованиям, что и имена файлов, за исключением расширений, которые для них не используются.

Доступ к образованной таким образом древовидной иерархической структуре каталогов вполне возможен с помощью приложения *Проводник* операционной системы Windows, а также операционных оболочек типа Norton Commander, FAR и др.

Но главное, что таким образом обеспечивается беспроблемный доступ к файлам и каталогам в пределах ИСП.

Учитывая существование различных дисков и деревьев каталогов, для идентификации файлов приходится использовать их расширенные имена. Расширенное имя файла образуют, присоединяя к обычному имени путь по ветвям-каталогам, начиная от корневого диска. Все составляющие расширенного имени отделяются друг от друга знаком обратной косой черты "\". Примеры путей, которые можно обнаружить в поле Адрес окна приложения Проводник, показывают, что расширенные имена могут быть весьма громоздкими. Поэтому в рамках ИСП активно используется понятие "текущий каталог". Оно удобно тем, что позволяет программисту оперировать только короткими именами файлов. А соответствующий диск и путь к текущему каталогу хранит сама ИСП.

Оперативная установка и изменение текущего каталога с одного на другой осуществляются в ИСП посредством выбора команды File⇨Change dir.... В качестве текущего в рамках ИСП настоятельно рекомендуется всегда использовать каталог хранения файлов с текстами создаваемых программ.

Отметим также, что расширение `PAS` удобно применять для любых текстовых файлов, содержащих произвольный текст, а не только для тех, которые включают тексты программ. Это объясняется тем, что текстовый редактор ИСП применяет его ко всем своим файлам по умолчанию.

Текстовые файлы, создаваемые редактором ИСП, частично "понимает" также и текстовый редактор Блокнот системы Windows. Единственная проблема связана здесь только со второй половиной таблицы ASCII (см. описание модуля `Printer` в разделе 2.5.2), кодировка которой совершенно не совпадает с системой кодов ANSI.

Чтение текстовых файлов ИСП текстовым редактором Блокнот:

- запустите текстовый редактор Блокнот и выберите команду Файл⇨Открыть…;
- в диалоговом окне Открыть в поле Папка: установите папку хранения нужного текстового файла;
- в поле Тип файлов: этого окна установите Все файлы;
- в поле Кодировка: диалогового окна установите ANSI;
- щелкнув левой кнопкой мыши на нужном файле, выделите его и щелкните на кнопке Открыть.

Создание текстовых файлов ИСП текстовым редактором Блокнот:

- запустите текстовый редактор Блокнот и наберите произвольный текст, используя знаки первой половины таблицы кодов ASCII (латиница, цифры и пр.);
- выберите команду Файл⇨Сохранить как…;

- в диалоговом окне Сохранить как в поле Папка: установите папку сохранения файла с подготовленным текстом;
- в поле Кодировка: диалогового окна установите ANSI;
- в поле Имя файла: этого окна укажите нужное имя с расширением PAS;
- щелкните на кнопке Сохранить.

После этого полученный файл можно открыть в текстовом редакторе ИСП обычным образом.

Приложением Проводник файлы с текстами Паскаль-программ отображаются значками стандартного вида. Вместе с тем может оказаться полезным предусмотреть для PAS-файлов специальный значок, который позволит значительно легче их обнаруживать. Для этого тип файлов с расширением PAS должен быть надлежащим образом зарегистрирован в Windows.

Упомянутые регистрация и выбор значка могут быть реализованы следующим образом.

- Запустите приложение Проводник и выберите команду Сервис⇨Свойства папки....
- В появившемся диалоговом окне Свойства папки щелкните на корешке вкладки Типы файлов.
- Щелкните на кнопке Создать, после чего в поле Расширение: появившегося диалогового окна Создание нового расширения введите значение PAS и щелкните на кнопке ОК.
- В поле Зарегистрированные типы файлов: диалогового окна Свойства папки отыщите и выделите запись, соответствующую расширению PAS.
- Щелкните на кнопке Дополнительно, в результате чего будет получено дополнительное диалоговое окно Изменение свойств типа файлов.
- В этом окне щелкните на кнопке Сменить значок... и в очередном диалоговом окне Смена значка подберите устраивающий вас значок для отображения PAS-файлов. При этом следует воспользоваться возможностью просмотра файлов со значками с помощью кнопки Обзор... (например, файла ...\Windows\system32\shell32.dll и других).
- Последовательно закройте окна Смена значка и Изменение свойств типа файлов выбором соответствующих кнопок ОК. Щелкните на кнопке Закрыть, закройте также и диалоговое окно Свойства папки.

Если же по каким-то причинам возникла необходимость исключить PAS-файлы из числа зарегистрированных, то для этого необходимо выполнить следующие действия.

- Запустите приложение Проводник и выберите команду Сервис⇨Свойства папки....
- В появившемся диалоговом окне Свойства папки щелкните на корешке вкладки Типы файлов.
- В поле Зарегистрированные типы файлов: диалогового окна Свойства папки отыщите и выделите запись, соответствующую расширению PAS.
- При необходимости сначала щелкните на кнопке Восстановить. Затем щелкните на кнопке Удалить. Если в результате появилось соответствующее окно предупреждения об отмене регистрации, то щелкните в нем на кнопке Да. Закройте окно Свойства папки, выбрав кнопку Закрыть.

Отметим, что в результате снятия PAS-файлов с регистрации для них отменяется ранее выбранный значок и автоматически устанавливается значок стандартного вида для файлов с неопределенным типом.

2.2.3. Запуск ИСП

- Запуск из командной строки MS DOS.

  Для получения окна Windows с командной строкой следует выбрать команду Главное меню⇨Все программы⇨Стандартные⇨Командная строка. Теперь для запуска ИСП Turbo Pascal возможен вариант с предварительным переходом в каталог ...\BP\BIN.

  Если, например, путь к этому каталогу имеет вид D:\SOFT\BP\BIN, то последовательность команд перехода может быть такой: D:, cd SOFT, cd BP, cd BIN. После этого остается командой TURBO запустить на выполнение файл TURBO.EXE.

  После завершения работы в ИСП окно Командная строка может быть закрыто обычным образом.

  Аналогично запускается и ИСП Borland Pascal.

- Запуск из операционной оболочки Norton Commander (или из аналогичной ей).

  В первую очередь должна быть запущена сама операционная оболочка.

  Теперь для запуска ИСП Turbo Pascal сначала следует в одной из панелей оболочки обычным образом зайти в каталог ...\BP\BIN. Затем нужно установить селектор на файл TURBO.EXE и нажать клавишу <Enter>.

Более удобным может оказаться иной вариант: сразу после перехода в каталог `...\BP\BIN` выполнить из командной строки оболочки команду `TURBO`.

Аналогично запускается и ИСП Borland Pascal.

- Запуск из среды Windows.

  Для запуска ИСП Turbo Pascal откройте приложение Проводник, перейдите в папку `...\BP\BIN`, отыщите в ней файл `TURBO.EXE` и дважды щелкните на нем мышкой. При этом допустимы и иные, предусмотренные в Windows варианты открытия файла.

  Аналогично запускается и ИСП Borland Pascal.

  Однако более рациональным все же следует считать запуск ИСП с помощью Главного меню.

  Выберите команду Главное меню⇨Выполнить.... В диалоговом окне Запуск программы нажмите кнопку Обзор.... Далее, используя средства появившегося диалогового окна Обзор, отыщите и выделите файл `TURBO.EXE`, после чего щелкните на кнопке Открыть. Теперь остается щелкнуть на кнопке ОК в диалоговом окне Запуск программы.

  Повторите аналогичные операции и для запуска ИСП Borland Pascal.

  В дальнейшем при запуске ИСП уже не нужно будет отыскивать ее с помощью диалогового окна Обзор. Команда запуска будет сохранена в соответствующем списке диалогового окна Запуск программы. Поэтому достаточно будет развернуть список его команд и выбрать нужную.

- Создание ярлыка ИСП.

  В любой папке Windows может быть создан ярлык ИСП, используя который можно ее запустить. Однако чаще всего ярлык ИСП располагают на Рабочем столе с целью быстрейшего к ней (среде) доступа. Наиболее популярны два способа создания ярлыков.

  Способ 1

  - Щелкните правой кнопкой на свободном месте Рабочего стола, в раскрывшемся контекстном меню выберите пункт Создать, а в выпавшем каскадном меню щелкните на пункте Ярлык для вызова средства Мастер создания ярлыков.
  - На этапе Создание ярлыка щелкните на кнопке Обзор..., после чего, используя средства диалогового окна Обзор папок, отыщите файл **TURBO.EXE** или **BP.EXE**, выделите его и щелкните на кнопке ОК, а затем — на кнопке Далее.

- На этапе Выбор названия программы измените, если необходимо, текст в поле Введите имя ярлыка: и щелкните на кнопке Далее.
- На этапе Выбор значка прокрутите их перечень, выделите нужный и щелкните на кнопке Готово.

Способ 2

- С помощью приложения Проводник откройте папку `...\BP\BIN`, отыщите в ней файл `TURBO.EXE` или `BP.EXE` и щелкните на нем правой кнопкой мыши.
- В появившемся контекстном меню щелкните на пункте Создать ярлык.
- Щелкните правой кнопкой мыши на образовавшемся ярлыке, после чего в появившемся контекстном меню щелкните на пункте Вырезать.
- Откройте доступ к Рабочему столу и щелкните правой кнопкой мыши по его свободному месту. В появившемся контекстном меню выберите пункт Вставить.
- На образовавшемся ярлыке щелкните правой кнопкой мыши, в появившемся контекстном меню выберите пункт Переименовать, после чего введите для ярлыка нужное название.

Если для имеющегося ярлыка возникла необходимость смены значка, то этого можно достичь с помощью следующих действий.

- Щелкните правой кнопкой мыши на имеющемся ярлыке и в полученном контекстном меню щелкните на пункте Свойства.
- В открывшемся диалоговом окне Свойства щелкните на корешке вкладки Программа.
- Щелкните на кнопке Сменить значок..., после чего средствами диалогового окна Смена значка найдите нужный файл со значками, выберите в нем походящий значок и щелкните на кнопке ОК. Также щелчком на кнопке ОК закройте и окно Свойства.

Отметим, что в качестве значка может быть использован любой из файлов с расширением `ICO`, находящихся в каталоге `...\BP\BIN`. Кроме того, огромное количество значков содержится в некоторых DLL-библиотеках, расположенных в папке `...\Windows\system32`. В частности, таковыми библиотеками являются файлы `HTICONS.DLL`, `MORICONS.DLL`, `PIFMGR.DLL`, `SHELL32.DLL` и другие.

- Автоматический запуск ИСП в соответствии со стандартной схемой открытия файлов, принятой в Windows.

В данном случае речь идет о том, что в рамках Windows любой документ зарегистрированного типа может быть открыт посредством воздействия на него одним из следующих способов:

- дважды щелкните мышью на имени файла с документом;
- щелкните правой кнопкой мыши на имени файла с документом, после чего в раскрывшемся контекстном меню выберите пункт Открыть;
- выделите файл с документом, после чего нажмите клавишу <Enter>.

При этом сначала автоматически запускается приложение, обрабатывающее файлы данного типа, а затем с его помощью открывается и сам файл.

Настройка Windows для распространения этой схемы на PAS-файлы с текстами программ, тип которых уже зарегистрирован, может быть реализована следующим образом.

- Запустите приложение Проводник и выберите команду Сервис⇨Свойства папки....
- В появившемся диалоговом окне Свойства папки щелкните на корешке вкладки Типы файлов.
- В поле Зарегистрированные типы файлов: диалогового окна Свойства папки отыщите и выделите запись, соответствующую расширению PAS.
- Щелкните на кнопке Изменить... диалогового окна Свойства папки, в результате чего будет получено окно с предупреждением Windows о том, что вам предстоит определить программу, с помощью которой обрабатываются файлы с расширением PAS.
- Установите переключатель на пункт Выбор программы из списка вручную и щелкните на кнопке ОК.
- В открывшемся диалоговом окне Выбор программы прокрутите список Программы, найдите и выделите в нем нужную программу, например, TURBO.EXE. Если же в этом списке нужной нам программы не оказалось, то щелкните на кнопке Обзор... и осуществите ее поиск в дополнительном диалоговом окне Открыть с помощью....
- В том же диалоговом окне Выбор программы установите отметку на пункте Использовать ее для всех файлов такого типа и щелкните на кнопке ОК. Щелкнув на кнопке Закрыть, закройте также диалоговое окно Свойства папки.

Таким образом образуется возможность запуска ИСП Turbo Pascal через посредство любого из PAS-файлов.

Если через некоторое время возникнет необходимость сменить ИСП Turbo Pascal на Borland Pascal, то для соответствующей перенастройки системы автоматического запуска ИСП окажется достаточным повторить те же операции.

2.2.4. Настройка ИСП

После запуска ИСП возможна работа в одном из двух режимов: в полноэкранном или в оконном. Переключение между ними достигается нажатием клавиш <Alt+Enter>.

Оконный режим работы ИСП соответствует практически всем соглашениям Windows и поэтому считается предпочтительным. Настройки некоторых характеристик интерфейса оконного режима работы (шрифт, цвет, размеры и пр.) сосредоточены в диалоговом окне Свойства, которое появляется при выборе команды Свойства из контекстного меню заголовка окна ИСП.

Ниже представлены основные настройки самой ИСП, которые должны быть выполнены для надежной ее работы под управлением операционной системы Windows.

- Выбрать команду Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует снять флажок с пункта 286 instructions.

  Благодаря этому обеспечивается переносимость EXE-файлов программ, поскольку в них в этом случае не будут использованы некоторые уникальные команды микропроцессора Intel 80286. Аналогичного эффекта для конкретной программы, не затрагивая компилятор в целом, можно добиться, используя директиву компиляции {$G-}.

- Выбрать команду Options⇨Linker... (<Alt+O&L>). При этом открывается диалоговое окно настройки работы компоновщика ИСП Linker, в котором для переключателя поля Map file следует установить вариант Off, а для переключателя поля Link buffer — вариант Memory.

  При этом запрещается создание карты распределения памяти, которая, как правило, оказывается нужной только при использовании внешнего отладчика. И, кроме того, для промежуточного хранения компонуемой программы выбирается оперативная память, что значительно ускоряет работу компоновщика. Только при компоновке больших программ нехватка оперативной памяти может вынудить использовать для этой цели диск.

- Выбрать команду Options⇨Debugger... (<Alt+O&B>). При этом будет открыто диалоговое окно Debugging/Browsing, в котором следует установить переключатель поля Display swapping в положение Smart.

  При этом устанавливается режим, когда обращение к окну выполнения программы Output осуществляться только при необходимости выполнить в программе очередной оператор ввода-вывода. Использовать режимы None (никогда) или Always (всегда) не следует.

- Выбрать команду Options⇨Directories... (<Alt+O&D>). При этом открывается диалоговое окно Directories, в котором следует очистить все поля указания функциональных каталогов ИСП.

  При этом ИСП ориентируется на то, что создаваемые EXE-файлы программ и TPU-файлы модулей будут размещаться в текущем каталоге ИСП. Кроме того, в случае, если программа использует включаемые PAS-файлы или внешние подпрограммы в виде OBJ-файлов, все они должны быть предварительно размещены в текущем каталоге. Все подключаемые к программе модули ИСП будет в первую очередь искать также в текущем каталоге. Поэтому при подключении к программе стандартных модулей (например, модуля GRAPH.TPU) необходимо или скопировать их в текущий каталог, или указать в поле Unit directories полный путь к каталогу . . .\BP\UNITS, в котором они содержатся.

  Для хранения создаваемых EXE-файлов программ и TPU-файлов модулей отдельно от текстов программ может быть предусмотрен иной каталог, путь к которому должен быть указан в поле EXE&TPU directory диалогового окна Directories. В этом случае тот же каталог следует указать и в поле Unit directories. При записи в одном и том же поле нескольких каталогов подряд их следует отделять друг от друга точкой с запятой.

- Выбрать команду Options⇨Environment⇨Preferences… (<Alt+O&E&P>), после чего в диалоговом окне Preferences установить флажки на пунктах Editor files, Environment и Desktop поля Auto save.

  Это соответственно обеспечивает следующие преимущества:

  - автоматическое сохранение на диске содержимого всех открытых окон редактора ИСП перед выполнением отлаживаемой программы; это необходимо на случай ее “зависания”, когда возможность обычного сохранения утрачивается;
  - автоматическое сохранение в файле <*имя*>.TP всех текущих параметров настройки ИСП при завершении работы с ней;
  - автоматическое сохранение в файле <*имя*>.DSK информации об открытых окнах редактора и о состоянии их содержимого при завершении работы с ИСП; это позволит начать очередной сеанс работы с ИСП точно из того же положения, что и в момент завершения.

- Выбрать команду Options⇨Environment⇨Editor… (<Alt+O&E&E>), после чего в диалоговом окне Editor Options снять флажок с пункта Create backup files.

  Благодаря этому будет отменено создание BAK-файлов, содержащих предыдущие варианты текстов программ на момент сохранения новых. Как показывает практика программирования, реальное значение этих файлов для обеспечения надежности несущественно, но места они занимают много.

- Выбрать команду Options⇨Environment⇨Mouse… (<Alt+O&E&M>), после чего в диалоговом окне Mouse options выполнить две следующие операции:
  - для ползунка Mouse double click подобрать положение, соответствующее наиболее удобному интервалу для двойного щелчка мышью;
  - установить флажок на пункте Reverse mouse buttons в том случае, если работа мышью выполняется левой рукой.
- Выбрать команду Options⇨Environment⇨Startup… (<Alt+O&E&S>), после чего в диалоговом окне Startup options установить флажок на пункте Load .TPL file.

  Благодаря этому обеспечивается использование стандартного модуля `SYSTEM` непосредственно из библиотечного `TPL`-файла. При появлении окна предупреждения Warning с сообщением о необходимости внести изменения в основную программу `BP(TURBO).EXE` необходимо щелкнуть на кнопке OK. Сделанные изменения вступают в силу после перезапуска ИСП.

Параметры настройки ИСП должны быть обязательно сохранены. Для этого есть две основные возможности:

- автоматическое использование стандартного файла конфигурации `BP(TURBO).TP`, находящегося в каталоге `...\BP\BIN`;
- целенаправленное использование личного файла конфигурации `<`*имя*`>.TP`, сохраняемого в том же каталоге, что и тексты программ, т.е. в текущем.

Первая возможность реализуется более оперативно и доставляет минимум хлопот. При этом, если в диалоговом окне Preferences (получаемого выбором команды Options⇨Environment⇨Preferences… или нажатием клавиш <Alt+O&E&P>) флажок на пункте Environment поля Auto save установлен, то сохранение параметров настройки вообще не предполагает выполнения каких-либо специальных действий. Если же упомянутый флажок не установлен, то для сохранения параметров настройки следует выбрать команду Options⇨Save (Alt+O&S). При этом в строке Save подчиненного меню вы обнаружите, что сохранение выполняется в стандартный файл конфигурации `BP(TURBO).TP`.

Второй вариант более надежен и удобен, особенно в том случае, если ИСП используется разными программистами или для реализации разных проектов. Личный файл конфигурации обычно создается перед началом работы над проектом следующим образом:

- любым из известных вам способов (например, средствами Windows) создайте каталог, предназначенный для хранения текстов программ проекта;
- запустите ИСП;
- закройте все окна предыдущего сеанса работы другого пользователя нажатием клавиш <Alt+W&O> (команда Window⇨Close all);

- каталог, в котором предполагается хранить тексты программ проекта, установите в качестве текущего средствами диалогового окна Change Directory, вызываемого нажатием клавиш <Alt+F&C> (команда File⇨Change dir...);
- выполните нужную вам настройку ИСП, в том числе обязательно выберите команду Options⇨Environment⇨Preferences…(<Alt+O&E&P>), и в диалоговом окне Preferences установите флажок на пункте Environment поля Auto save;
- выберите команду Options⇨Save as... (Alt+O&A), после чего в открывшемся диалоговом окне Save Options As в поле Options file name введите имя личного файла конфигурации и щелкните на кнопке OK.

В дальнейшем в вашем личном файле конфигурации <*имя*>.TP, находящемся в каталоге с текстами программ, все текущие настройки ИСП будут сохраняться автоматически. И кроме того, в том же каталоге с текстами программ будет автоматически сохраняться файл <*имя*>.DSK, содержащий всю информацию об открытых окнах редактора и о состоянии их содержимого при завершении работы с ИСП. Это позволит начать очередной сеанс работы с ИСП точно из того же положения, что и в момент завершения.

2.2.5. Основной порядок работы в ИСП

При создании новой программы или при внесении изменений в существующую (редактирование программы) рекомендуется следующая основная последовательность действий в ИСП.

- Запуск ИСП.
- Закрытие всех окон предыдущего сеанса работы другого пользователя нажатием клавиш <Alt+W&O> (команда Window⇨Close all). Чтобы закрыть единственное окно, достаточно нажать клавиши <Alt+F3> (команда Window⇨Close).
- Установка текущего каталога в диалоговом окне Change Directory, вызываемом нажатием клавиш <Alt+F&C> (команда File⇨Change dir...). Текущим каталогом должен быть тот, в котором размещаются файлы с текстами программ.
- Выполнение одного из трех возможных вариантов открытия окна текстового редактора ИСП.
  - Открытие пустого окна для новой впервые создаваемой программы нажатием клавиш <Alt+F&N> (команда File⇨New). Сразу же после этого должна быть нажата клавиша <F2> (команда File⇨Save as...) с целью вызова диалогового окна Save File As для присвоения имени файлу, в котором будет сохраняться текст новой программы.

- Загрузка в окно текстового редактора ИСП текста существующей программы с помощью средств диалогового окна Open a File, вызываемого нажатием клавиши <F3> (команда File⇨Open...), если необходимо внести в нее изменения, просто просмотреть или использовать ее фрагменты для какой-то другой программы.
- Выбор команды Options⇨Open... (Alt+O&O) в случае использования личного файла конфигурации. При этом открывается диалоговое окно Open Options, средствами которого следует выбрать и открыть нужный файл конфигурации *<имя>*.TP. Благодаря этому данный сеанс работы с ИСП будет продолжен точно из того же положения, что и в момент завершения предыдущего.

- Редактирование текста программы.
- Сохранение текста готовой программы в файле на диске нажатием клавиши <F2> (команды File⇨Save). Такое сохранение также рекомендуется делать периодически в процессе всего редактирования.
- Компиляция и запуск программы на выполнение нажатием клавиш <Ctrl+F9> (команда Run⇨Run).
- Просмотр полученных результатов в окне выполнения программы, получаемом посредством нажатия клавиш <Alt+F5> (команда Debug⇨User screen). Это окно закрывается при нажатии любой клавиши.
- Завершение работы с ИСП нажатием клавиш <Alt+X> (команда File⇨Exit).

2.2.6. Хранение текстов программ

Стандартные средства ИСП, с помощью которых осуществляется обращение к файлам с текстами программ на диске, таковы.

- Открытие нового пустого окна текстового редактора для вновь создаваемой программы посредством выбора команды File⇨New (<Alt+F&N>).
- Выбор нужного файла на диске и открытие окна редактора с текстом соответствующей программы. Все связанные с этим действия выполняются в диалоговом окне Open a File, вызываемом посредством выбора команды File⇨Open... (<F3>).
- Запись на диск (сохранение) текста редактируемой программы в том же файле, где он хранился до начала сеанса редактирования, достигается посредством выбора команды File⇨Save (<F2>).
- Сохранение текста редактируемой программы в новом файле. Используется для новых программ, создаваемых либо на пустом месте, либо на базе уже существующих. Все связанные с этим действия выполняются в диалоговом окне Save File As, вызываемом посредством выбора команды File⇨Save as... (<Alt+F&A>).

Однако современная практика программирования доказывает целесообразность долговременного хранения текстов нужных программ в виде единого файла приложения Word. Это существенно эффективнее, нежели иметь множество PAS-файлов. В таком сборнике значительно удобнее комментировать их и легче просматривать. К программам можно составить оглавление, чтобы упростить и ускорить поиск. И, наконец, можно избавиться от необходимости придумывать бесконечное количество имен файлов, не путаясь при этом.

Задача переноса текста нужной программы из окна открытого текстового файла сборника программ в окно редактора ИСП решается достаточно просто:

- запустите ИСП и установите оконный режим его работы, нажав клавиши <Alt+Enter>;
- в ИСП выберите команду Options⇨Environment⇨Editor…(<Alt+O&E&E>), после чего в диалоговом окне Editor Options снимите отметку с пункта Auto indent mode и щелкните на кнопке OK;
- откройте в ИСП пустое окно, выполнив команду File⇨New (Alt+F&N);
- откройте текстовый файл со сборником программ с помощью приложения Word, найдите текст нужной программы, выделите его обычным образом и скопируйте в буфер обмена, нажав клавиши <Ctrl+Insert>;
- щелкните правой кнопкой мыши на заголовке окна ИСП, в появившемся контекстном меню выберите пункт Изменить, а затем в появившемся каскадном меню щелкните левой кнопкой мыши на пункте Вставить;
- сохраните в ИСП текст программы обычным образом.

Чтобы осуществить перенос текста программы из ИСП в открытый документ приложения Word, необходимо поступить следующим образом:

- установите оконный режим работы ИСП, нажав клавиши <Alt+Enter>;
- щелкните правой кнопкой мыши на заголовке окна ИСП и в появившемся контекстном меню выберите пункт Изменить, после чего в появившемся каскадном меню щелкните левой кнопкой мыши на пункте Пометить; в результате в левой части главного меню ИСП появляется специальный курсор, а само окно переходит в режим Пометить;
- переместите специальный курсор в нужное место текста и, удерживая в нажатом состоянии клавишу <Shift>, выделите нужный его фрагмент; после чего выполните копирование нажатием клавиши <Enter>; в результате выделенный фрагмент текста попадает в буфер обмена; выделение нужного фрагмента текста можно выполнить также и с помощью перетаскивания указателя мыши, удерживая левую кнопку.

- активизируйте окно документа приложения Word, установите текстовый курсор в нужное место и нажмите клавиши <Shift+Insert>;
- выделите в документе вставленный текст программы и установите для него шрифт Courier New.

2.3. Текстовый редактор ИСП

2.3.1. Окно редактора

Окна редактора предназначены для создания и редактирования текстов программ.

Чтобы открыть новое пустое окно для вновь создаваемой программы, выполните команду File⇨New или нажмите клавиши <Alt+F&N>. При этом на верхней части рамки окна появляется надпись вида `NONAMExx.PAS`, где `xx` — условное обозначение номера от `00` до `99`. Это имя по умолчанию для создаваемого файла с текстом новой программы.

В окно редактора можно загрузить также текст программы из уже существующего файла для внесения в него изменений (для редактирования). Для этого используют средства диалогового окна Open a File, появляющегося или в результате выбора команды File⇨Open..., или после нажатия клавиши <F3>. В этом случае на верхней части рамки окна редактора видно имя загруженного файла.

В случае использования личного файла конфигурации `<`*имя*`>.TP` работа с редактором ИСП может быть продолжена точно из того же положения, в котором она была завершена в предыдущем сеансе работы. Для этого выбирают команду Options⇨Open... (Alt+O&O). Затем с помощью диалогового окна Open Options открывают нужный файл конфигурации.

В правом верхнем углу окна редактора отображается порядковый номер окна среди всех открытых в ИСП. В каждый данный момент редактирование возможно только в одном из нескольких открытых окон. Это окно считается активным или текущим. Оно выделяется двойной рамкой, чем и отличается от остальных окон, имеющих одинарную рамку. Кроме того, текущее окно всегда располагается поверх всех прочих.

В левом нижнем углу рамки текущего окна через двоеточие указаны два порядковых номера. Первый из них — порядковый номер строки, в которой находится текстовый курсор. Эта строка считается текущей. Второй из них — номер позиции, которую занимает курсор в текущей строке. Если редактируемый текст изменен, но не сохранен на диске, то в том же месте находится индикатор — “звездочка”.

В большей своей части справа и внизу рамка текущего окна представляют собой полосы прокрутки, ограниченные “кнопками-стрелками” и содержащие ползунок.

Управление окнами редактора с помощью клавиатуры.

- Поочередная смена одного текущего окна другим в прямом направлении достигается выбором команды Window⇨Next или нажатием клавиши <F6>. То же самое, но в обратном направлении — выбором команды Window⇨ Previous или нажатием клавиш <Shift+F6>.
- Целенаправленный выбор нужного окна и назначение его текущим осуществляется в диалоговом окне Window List, появляющемся или в результате выбора команды Window⇨List..., или после нажатия клавиш <Alt+0>. При этом необходимо сначала сделать выбор в списке, а затем щелкнуть на кнопке OK.
- Текущее окно редактора закрывается посредством выбора команды Window⇨Close или нажатием клавиш <Alt+F3>. Если закрываемое окно содержит несохраненный текст, то на экране появляется окно сообщений Information с предложением сначала выполнить сохранение "Save?". На это предложение следует дать ответ, щелкнув на одной из кнопок Yes — "Да, сохранить", No — "Нет, не сохранять" или Cancel — "Отменить закрытие". Закрыть все окна сразу можно посредством команды Window⇨ Close all или нажатием клавиш <Alt+W&O>.
- Целенаправленное закрытие нужного окна осуществляется в диалоговом окне Window List, появляющемся или в результате выбора команды Window⇨ List..., или после нажатия клавиш <Alt+0>. При этом необходимо сначала сделать выбор в списке окон, а затем щелкнуть на кнопке Delete.
- Изменение положения окна редактора в пределах окна ИСП достигается выбором команды Window⇨Size/Move или нажатием клавиш <Ctrl+F5>. При этом рамка окна меняет цвет, что является признаком возможности его перемещения. Нужное положение окна устанавливают постепенно с помощью клавиш управления курсором. Разовое перемещение в одно из крайних положений реализуется нажатием клавиш <Home>, <End>, <Page Up> или <Page Down>. Процесс изменения положения следует завершить, нажав клавишу <Enter>. Можно также отказаться от сделанных изменений, нажав клавишу <Esc>.
- Если открыто несколько окон редактора на экране, то их чаще всего располагают каскадом, т.е. одно за другим с небольшим смещением так, чтобы был виден край каждого из них. Для этого достаточно выбрать команду Window⇨Cascade или нажать клавиши <Alt+W&A>. При желании можно сделать все открытые окна видными одновременно, хоть и маленькими по размеру. Для этого достаточно выбрать команду Window⇨Tile или нажать клавиши <Alt+W&T>.

- Изменение размеров окна редактора в пределах окна ИСП достигается выбором команды Window⇨Size/Move или нажатием клавиш <Ctrl+F5>. При этом рамка окна меняет цвет, что является признаком возможности управления его размером. Нужные размеры окна устанавливают постепенно за счет перемещений правой и нижней части его рамки с помощью клавиш управления курсором при одновременном удержании клавиши <Shift>. Процесс следует завершить, нажав клавишу <Enter>. Можно также отказаться от сделанных изменений, нажав клавишу <Esc>. Разовое раскрытие окна редактора до максимальной величины и последующее возвращение его к исходным размерам достигается выбором команды Window⇨Zoom или нажатием клавиши <F5>.

Управление окнами редактора с помощью мыши.

- Окно приобретает статус текущего после щелчка левой кнопкой мыши на любой видимой его части.
- Текущее окно редактора закрывается щелчком мышью на кнопке "■" в левом верхнем его углу.
- Изменение положения окна достигается обычным перетаскиванием. Для этого нужно предварительно совместить указатель мыши с верхней частью рамки окна и удерживать левую ее кнопку в нажатом состоянии.
- Изменение размеров окна редактора достигается перетаскиванием нижнего правого угла его рамки при удержании левой кнопки мыши в нажатом состоянии.
- Разовое раскрытие окна редактора до максимальной величины и последующее возвращение его к исходным размерам достигается двойным щелчком мыши на верхней части его рамки.
- Посимвольное и построчное прокручивание текста в окне редактора с помощью кнопок со стрелками "◄", "►", "▼", "▲" на полосах прокрутки в соответствующую сторону.
- Плавная быстрая прокрутка текста в окне редактора в нужную сторону с помощью перетаскивания ползунков "■" на полосах прокрутки при удерживаемой левой кнопке мыши.
- Для быстрой постраничной прокрутки текста в окне редактора в нужную сторону щелкайте мышью по обе стороны ползунков "■" на полосах прокрутки.

2.3.2. Средства редактирования текста

Основное устройство редактирования — клавиатура. Нажатие любой символьной клавиши обязательно сопровождается изменениями в редактируемом тексте и контрольным выводом соответствующего знака на экран (эхо-повтор). Кроме этого, есть целая группа клавиш (например, функциональных <F1>...<F12> и управления курсором), нажатие которых в тексте никак не отображается.

После нажатия клавиши ее код автоматически помещается в так называемый буфер клавиатуры, откуда затем извлекается (уже программными средствами) для дальнейшей обработки. В некоторых случаях клавиши могут нажиматься быстрее, нежели программа успевает извлекать из буфера и обрабатывать соответствующие коды. По этой причине коды нажатых клавиш в буфере клавиатуры постепенно накапливаются. Таким образом, буфер клавиатуры предназначен для временного хранения кодов нажатых клавиш, ожидающих своей обработки. Объем буфера клавиатуры ограничен. С момента его заполнения начинает работать звуковой сигнал ("клавиатура пищит"), а лишние коды теряются.

Даже непрофессиональный взгляд на клавиатуру приводит к выводу о недостаточном количестве клавиш на ней. Действительно, в таблице ASCII почти вдвое больше кодов, чем клавиш. Известный выход из положения — использование клавиш-модификаторов <Shift>, <Ctrl> и <Alt>. Удерживая одну из клавиш-модификаторов и нажимая какую-либо клавишу, можно получить от нее совсем другой код. Однако теперь, наоборот, возможных кодов становится значительно больше, чем это допускает однобайтовое их представление. Именно поэтому аппаратно сделано так, что целый ряд клавиш генерируют при своем нажатии не однобайтовые, а двухбайтовые коды. Такие коды называются *расширенными*. Например, расширенные коды генерируются функциональными клавишами <F1>...<F12>, а также клавишами редактирования и управления курсором.

В первом байте двухбайтового кода, "снимаемого" с клавиши, всегда указывается знак `#0`. Иначе говоря, нулевой код всегда свидетельствует о нажатии клавиши, генерирующей расширенный код. Знак, следующий за `#0`, идентифицирует нажатую клавишу конкретно.

Ниже представлены основные операции редактирования текста.

- Перемещение курсора осуществляется с помощью курсорных клавиш, в том числе, клавиш со стрелками, клавиш <Home>, <End>, <PageUp> и <PageDown>, а также <Tab>:
  - влево и вправо на символ, вверх и вниз на строку — клавиши управления курсором;
  - на слово влево или вправо — соответствующие клавиши управления курсором одновременно с удержанием клавиши <Ctrl>;

- в начало или в конец строки — <Home> или <End> соответственно;
- в верхнюю или в нижнюю строку окна — <Ctrl+Home> или <Ctrl+End> соответственно;
- в начало или в конец всего текста — <Ctrl+PageUp> или <Ctrl+PageDown> соответственно;
- <Tab> — образование отступов в строке.

■ *Маркером* называется невидимая точка, устанавливаемая в программе с целью ускорения поиска этой точки в дальнейшем. На выполнение программы маркер никак не влияет. Всего в программе можно установить до десяти маркеров, пронумерованных от нуля до девяти. Установка нового маркера с тем же номером отменяет предыдущий. Работа с маркером:

- установить в программе маркер с порядковым номером `n` в текущей позиции курсора — <Ctrl+K&n>;
- искать в программе маркер с порядковым номером `n` и установить в эту точку курсор — <Ctrl+Q&n>.

■ Перемещения текста в окне редактора:

- на страницу (содержимое окна) вверх или вниз — <PageUp> или <PageDown> соответственно;
- прокрутка (просмотр) текста вверх или вниз — <Ctrl+W> или <Ctrl+Z> соответственно; при этом перемещение осуществляется вместе с курсором.

■ Вставка:

- смена режимов ввода текста "вставка/наложение" — <Insert>; при этом изменяется размер курсора;
- вставить пустую строку перед текущей (курсор должен находиться в начале строки), вставить пустую строку после текущей (курсор должен находиться в конце строки), рассечь строку в месте нахождения курсора — все это осуществляется нажатием клавиш <Ctrl+N>; аналогично действует клавиша <Enter>, но только во время режима ввода текста "наложение".

■ Удаление:

- удаление текущей строки — <Ctrl+Y>;
- удаление символов от курсора и до конца текущей строки — <Ctrl+Q&Y>;
- удаление символа слева от курсора или под курсором — <Backspace> или <Delete> соответственно;
- удаление слова справа от курсора — <Ctrl+T>; при этом удаляются также пробелы между словами;
- "склеивание" текущей строки с предыдущей (курсор должен находиться в начале строки) — <Backspace>;

- "склеивание" данной строки со следующей (курсор должен находиться в конце строки) – <Delete>.

■ *Блоком* называется непрерывный выделенный фрагмент текста. На экране блок отмечается цветом. Работа с блоком:

- пометить начало и конец блока – <Ctrl+K&B> и <Ctrl+K&K> соответственно; это можно сделать также и клавишами управления курсором, если одновременно удерживать в нажатом состоянии клавишу <Shift>;
- выделить одно слово – <Ctrl+K&T>;
- скопировать или переместить блок в место нахождения курсора – <Ctrl+K&C> или <Ctrl+K&V> соответственно;
- удалить блок – <Ctrl+K&Y>;
- прочитать блок с диска или записать блок на диск – <Ctrl+K&R> или <Ctrl+K&W> соответственно; в процессе выполнения на экране появляются диалоговые окна "`Read Block From File`" или "`Write Block To File`" для указания имени файла с блоком;
- снять или восстановить выделение блока цветом – <Ctrl+K&H>.

Максимальная ширина окна редактора составляет 78 знаков. Если продолжать ввод далее, то текст в окне будет смещаться влево.

Обычно в текстовом редакторе тот или иной знак вводят нажатием соответствующей клавиши. Но если для некоторых знаков "личные" клавиши не предусмотрены, то их все же можно ввести следующим образом:

■ удерживайте в нажатом состоянии клавишу <Alt> и одновременно набирайте на малой цифровой клавиатуре код нужного знака;

■ закончив набор, отпустите клавишу <Alt> – знак введен.

Описанным способом можно отобразить любой нужный знак не только в тексте программы, но и при вводе с клавиатуры исходных данных выполняемой программы.

В ряде случаев при отображении знаков в тексте программы бывает удобно использовать так называемое *смещение*, обозначаемое знаком крышки "^". Для изучения этого способа целесообразно иметь под руками таблицу кодов ASCII (см. описание модуля `Printer` в разделе 2.5.2). Рассмотрим верхнюю часть этой таблицы (строки с нулевой по седьмую). В свою очередь, в этой верхней части рассмотрим верхнюю и нижнюю половины (по четыре строки каждая). Существует простое правило для верхней части таблицы: если `S` – некоторый знак из одной ее половины, то `^S` – знак, находящийся в том же столбце таблицы, но в другой ее половине, и отстоящий от данного на четыре строки. Аналогичное правило действует и в нижней части таблицы (строки с номера `8` по номер `F`).

Например: комбинация ^} соответствует знаку "=", и наоборот, комбинация ^= соответствует символу "}"; ^д соответствует "ф", и наоборот, ^ф соответствует "д" и т.д. Благодаря этому с помощью прописных букв кириллицы можно обращаться к знакам псевдографики, например: `WriteLn(^А^Б^Б^Б^Щ)`. К сожалению, метод не различает прописные и строчные буквы латиницы. При вводе с клавиатуры исходных данных выполняемой программы знак крышки "^" соответствует клавише <Ctrl>, удерживаемой при одновременном нажатии того или иного знака. При этом следует помнить, что некоторые из таких сочетаний клавиш задействованы в системных целях, и поэтому ряд знаков получить описанным методом просто не удастся.

2.3.3. Настройка редактора

Стандартное классическое окно текстового редактора при работе ИСП в полноэкранном режиме предполагает наличие 25 строк по 80 знаков в каждой. Эти же характеристики сохраняются при использовании в программе операторов вывода на экран в текстовом режиме его работы.

Однако характеристики современных дисплеев позволяют увеличить количество строк без ущерба для удобства программирования. Для этого следует выбрать команду Options⇨Environment⇨Preferences… (<Alt+O&E&P>), после чего в диалоговом окне Preferences переключатель Screen sizes установить в положение 43/50 lines и щелкнуть на кнопке OK.

Целый ряд настроек редактора, имеющих важное значение в процессе создания программ, сосредоточен в диалоговом окне Editor Options.

Выберите команду Options⇨Environment⇨Editor…(<Alt+O&E&E>), после чего в диалоговом окне Editor Options выполните следующие операции:

- снимите флажок с пункта Auto indent mode, что обеспечит в дальнейшем нормальное копирование текстов программ из документов приложения Word за счет отказа от автоматически выполняемых отступов в программе; при этом каждая очередная строка программы будет начинаться строго от левого края окна редактора, независимо от начала предыдущей строки текста программы;
- установите флажок на пункте Insert mode, чтобы получить режим вставки знаков в процессе набора текста программы с самого начала работы редактора ИСП;
- отметьте пункт Use tab characters, чтобы зафиксировать положение позиций табуляции относительно окна текстового редактора; при этом обеспечивается независимость установки позиций табуляции клавишей <Tab> в данной строке от расположения знаков в предыдущей строке текста программы;

- установите флажок на пункте Backspace unindents, чтобы обеспечить ускоренную работу клавиши <Backspace> в начале строки текста программы;
- снимите флажок с пункта Cursor through tabs, чтобы обеспечить ускоренное движение курсора по позициям табуляции при его перемещении вдоль строки;
- в поле Tab size задайте длину табуляционного интервала при нажатии на клавишу <Tab>; для реализации отступов строк в тексте программы удобным оказывается значение, равное 3;
- установите флажок на пункте Group Undo, чтобы ускорить откат за счет группировки однотипных операций редактирования.

Традиционно практически во всех учебниках по Паскалю приводится перечень зарезервированных слов языка. Вместо этого в рамках редактора ИСП целесообразно использовать цветовое выделение зарезервированных слов (а также и других разновидностей лексем) непосредственно по ходу набора текста программы.

Чтобы воспользоваться этой возможностью, установите флажок на пункте Syntax highlight в диалоговом окне Editor Options, появляющемся после выбора команды Options⇨Environment⇨Editor… (<Alt+O&E&E>)

Выбрав команду Options⇨Environment⇨Colors… (<Alt+O&E&C>), откройте диалоговое окно Colors. В нем можно выбрать тот или иной вариант условий работы в редакторе ИСП в отношении цвета. Для этого диалоговое окно Colors содержит два меню: Group (группа) и Item (элемент). В меню Group выберите пункт Syntax (синтаксис), после чего в меню Item будет получен перечень элементов, каждый из которых может быть выделен своим уникальным цветом. При этом выбору подлежит один из шестнадцати цветов текста (Foreground) и один из восьми цветов фона (Background).

Рекомендуемый состав цветов для выделения разновидностей лексем представлен в табл. 2.1. Цвет фона во всех случаях рекомендуется выбирать синим.

Таблица 2.1. Цветовое выделение лексем

Элемент	*Смысловое значение*	*Цвет текста (Foreground)*
`Whitespace`	Курсор в свободной части окна	Желтый
`Comments`	Комментарий	Светло-серый
`Reserved Words`	Зарезервированные слова	Белый
`Identifiers`	Идентификаторы	Желтый
`Symbols`	Простые и составные символы	Светло-голубой
`Strings`	Строки	Светло-фиолетовый
`Numbers`	Числа	Светло-зеленый

Настройка редактора ИСП должна быть сохранена. При этом, если в диалоговом окне Preferences (получаемом выбором команды Options⇨Environment⇨ Preferences… или нажатием клавиш <Alt+O&E&P>) флажок на пункте Environment поля Auto save установлен, то сохранение параметров настройки будет выполнено автоматически при завершении работы с ИСП. Если же упомянутый флажок не установлен, то для сохранения параметров настройки следует выбрать команду Options⇨Save (Alt+O&S).

2.4. Отладка программы

2.4.1. Средства отладки ИСП

Использование средств отладки ИСП возможно при условии, что в процессе компиляции программы генерируется так называемая отладочная информация. *Отладочная информация* представляет собой совокупность таблиц, устанавливающих связь между операторами текста программы и кодами машинных команд, полученными в результате компиляции. Отладочная информация присоединяется к EXE-файлу программы или к TPU-файлу модуля.

Отладочная информация создается, если установлена директива компиляции {$D+}. Если, кроме того, установить директиву компиляции {$L+}, то в процессе отладки доступными будут не только переменные самой программы (глобальные), но и переменные, описанные в подпрограммах (локальные).

Создание отладочной информации может быть установлено для ИСП в целом посредством выбора команды Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует установить флажки на пунктах Debug information и Local symbols поля Debugging. Благодаря этому в процессе отладки оказываются доступными соответственно как переменные самой программы (глобальные), так и переменные, описанные в подпрограммах (локальные).

Кроме того, должна быть установлена возможность использования встроенного отладчика ИСП, что достигается выбором команды Options⇨Debugger... (<Alt+O&B>). При этом будет открыто диалоговое окно Debugging/Browsing, в котором следует установить флажок на пункте Integrated debugging/browsing поля Symbols. В противном случае при попытке начать процесс отладки в окне предупреждения Warning будет получено сообщение “No debug info for program entry point”.

А вот флажок с пункта Standalone debugging этого же поля следует снять, чтобы избежать увеличения EXE-файла программы за счет добавления информации, нужной для использования внешнего отладчика TD.EXE.

Отладка — специфический процесс выполнения программы, осуществляемый с приостановками.

Во время очередной приостановки программист получает возможность проконтролировать текущие значения переменных программы, сравнить их с ожидаемыми, а также изменить их. После этого процесс выполнения программы может быть продолжен до момента следующей приостановки. Строка текста программы, в которой был приостановлен процесс ее выполнения, отмечается треком альтернативного цвета. Наличие такого трека как раз и означает, что программа находится в режиме отладки.

Возможны следующие способы организации приостановок:

- приостановка выполнения программы в текущей строке;
- приостановка программы после выполнения операторов очередной строки ее текста;
- приостановка выполнения программы в предварительно установленных контрольных точках.

Отметим, что в режиме отладки выполнение программы всегда приостанавливается перед первым исполняемым оператором той строки, которая отмечена треком отладки.

Выполнение программы с приостановкой в текущей строке реализуется посредством выбора команды Go to cursor (<F4>).

Строка приостановки должна быть выбрана предварительно, т.е. в ней должен быть установлен курсор текстового редактора ИСП.

После первого выбора команды программа выполняется сначала и приостанавливается в текущей строке, которая при этом отмечается треком отладки. Выполнив необходимый контроль текущих значений переменных, можно переместить текстовый курсор к одной из следующих строк и повторить выбор этой же команды. После этого программа продолжает выполняться от места положения трека отладки.

Чтобы выполнить программу с приостановками после операторов каждой очередной строки ее текста, выберите либо команду Run⇨Step over (<F8>), либо команду Run⇨Trace into (<F7>).

При этом процесс выполнения программы имеет построчно-дискретный характер, т.е. идет от строки к строке с приостановками. Трек отладки, сопровождаемый текстовым курсором, отмечает каждую очередную строку текста программы.

При первоначальном выборе любой из этих команд трек отладки устанавливается перед первым выполняемым оператором программы, а при каждом очередном выборе выполняются все операторы текущей строки программы, после чего трек отладки перемещается к следующей строке.

Команды отличаются друг от друга тем, что при выборе первой из них Run⇨Step over (<F8>) строки текстов подпрограмм не отслеживаются. Если текущая строка содержит обращение к подпрограмме, то при выборе только второй из них Run⇨Trace into (<F7>) трек отладки перемещается к первому выполняемому оператору подпрограммы, после чего процесс продолжается обычным образом.

В зависимости от необходимости отслеживать строки тех или иных подпрограмм выбор обеих команд в процессе данного сеанса отладки можно чередовать.

Как обычно, в момент приостановки есть возможность проверить и проанализировать текущие значения переменных программы.

> Чтобы установить в программе очередную контрольную точку, следует сначала разместить курсор в соответствующей строке ее текста, после чего выбрать команду Debug⇨Add breakpoint... (<Alt+D&P>).

В результате открывается диалоговое окно Add Breakpoint, в котором предусмотрено отображение четырех основных параметров контрольной точки:

- наименование файла с текстом программы (File name);
- номер строки программы, в которой установлена данная контрольная точка (Line number);
- условие, при котором в данной контрольной точке выполняется останов программы (Condition); если условие не указано, то по умолчанию для него принимается значение `True` (программа останавливается всегда);
- количество пропусков данной контрольной точки, когда программа в ней не останавливается (Pass count); по умолчанию принимается значение, равное нулю (контрольная точка не пропускается и останов в ней выполняется всегда).

Изменив при необходимости параметры Condition и Pass count, щелкаем на кнопке OK, после чего контрольная точка оказывается установленной. Фон соответствующей строки текста программы при этом выделяется альтернативным цветом.

Отметим, что в качестве условия Condition может быть указано любое логическое выражение с использованием любых переменных программы. В процессе выполнения программы это логическое выражение вычисляется, и если оно оказалось истинным, то в соответствующей контрольной точке происходит останов.

Если необходимость изменения параметров Condition и Pass count отсутствует, и программиста удовлетворяют их значения по умолчанию, то для установки в тексте программы контрольной точки без открытия диалогового окна Add Breakpoint достаточно нажать клавиши <Ctrl+F8>.

> Выполнение программы с приостановкой в каждой очередной контрольной ее точке можно реализовать, выбрав либо команду Run⇨Run (<Ctrl+F9>), либо команду Go to cursor (<F4>).

В дальнейшем с помощью выбора команды Breakpoints (<Alt+D&B>) можно открыть диалоговое окно Breakpoints со списком контрольных точек и просмотреть и отредактировать параметры каждой из них.

Диалоговое окно Breakpoints является общим для всех программ, окна которых открыты в редакторе ИСП. В нем по каждой контрольной точке в виде таблицы представлены упомянутые выше ее четыре параметра: наименование файла с текстом программы (Breakpoint list), номер строки программы с контрольной точкой (Line #), условие останова программы (Condition) и количество пропусков данной контрольной точки (Pass).

Для управления контрольными точками отлаживаемой программы в диалоговом окне Breakpoints возможен выбор той или иной из них, а также предусмотрен ряд операций, выполняемых по щелчку на следующих кнопках.

- Кнопка Edit предназначена для вызова дополнительного диалогового окна Edit Breakpoint, средствами которого можно отредактировать все четыре параметра (Condition, Pass count, File name, Line number) выбранной контрольной точки. Завершив редактирование, следует щелкнуть на кнопке Modify (<Enter>). Отказаться от редактирования можно щелчком на кнопке Cancel (<Esc>). Если же щелкнуть на кнопке New, то можно получить новую контрольную точку с отредактированными параметрами.
- Кнопка Delete служит для удаления из текста программы выбранной контрольной точки.
- Кнопка View предназначена для перемещения курсора к выбранной контрольной точке в тексте программы.
- Кнопка Clear all используется для удаления из текста программы всех контрольных точек. При этом появляется дополнительное диалоговое окно

подтверждения Confirmation, в котором следует щелкнуть на кнопке Yes – "Да, удалить" или кнопке No – "Нет, не удалять".

Отказаться от текущего сеанса отладки программы можно с помощью команды Run⇨Program reset (<Ctrl+F2>). Снять контрольную точку в строке текста программы можно, установив в ней текстовый курсор и нажав клавиши <Ctrl+F8>.

При этом трек отладки программы снимается. В контрольной точке снимается выделение фона строки текста программы.

В момент приостановки процесса выполнения программы программисту доступны следующие средства контроля текущих значений переменных программы:

- окно Call stack с информацией о текущем состоянии программного стека;
- окно наблюдения Watches, в котором могут отображаться текущие значения заданных переменных или выражений;
- окно выполнения программы Output;
- окно оперативного контроля и модификации значений переменных отлаживаемой программы Evaluate and Modify.

Открытие окна Call stack с информацией о текущем состоянии программного стека осуществляется посредством команды Debug⇨Call stack (<Ctrl+F3>).

В процессе отладки программы в этом окне отображаются вызовы подпрограмм с конкретными значениями фактических параметров обращения.

Окно наблюдения за текущими значениями заданных переменных и выражений Watches можно открыть, выбрав команду Debug⇨Wath (<Alt+D&W>).

Предварительно переменную или выражение для наблюдения в окне Watches следует ввести через диалоговое окно Add Watch, открыв его с помощью команды Debug⇨Add watch... (<Ctrl+F7>).

Окно выполнения программы Output можно открыть, выбрав либо команду Debug⇨Output (<Alt+D&O>), либо команду Debug⇨User screen (<Alt+F5>).

Первый вариант Debug⇨Output (<Alt+D&O>) удобен возможностью одновременного расположения на экране и окна редактора, и окна выполнения программы. Второй вариант Debug⇨User screen (<Alt+F5>) позволяет получить окно выполнения программы в полноэкранном режиме.

Диалоговое окно оперативного контроля и модификации значений переменных отлаживаемой программы Evaluate and Modify можно открыть, выбрав команду Debug⇨Evaluate/modify... (<Ctrl+F4>).

В процессе отладки этим окном пользуются в моменты приостановки выполнения программы. В нем можно просмотреть текущие значения переменных, вычислить достаточно простые выражения. Диалоговое окно Evaluate and Modify особенно удобно тем, что с его помощью можно проконтролировать значения таких специфических переменных, как файловые, а также массивы, множества и др. Этим окном можно воспользоваться также как обычным калькулятором.

В поле Expression (выражение) диалогового окна Evaluate and Modify введите выражение и нажмите клавишу <Enter>, после чего в поле Result (результат) будет получено его значение. При этом возможны варианты:

- для выражений, использующих системные идентификаторы, наличие открытого окна редактора с текстом программы не обязательно;
- для выражений, использующих именованные константы, необходима предварительная компиляция соответствующей программы;
- для выражений, использующих идентификаторы переменных и типизированных констант, необходимо выполнение соответствующей программы.

Значение, получаемое в поле Result, может быть специфическим образом отформатировано. Для этого в поле Expression через запятую после выражения следует указать необходимый ключ. Чаще всего полезными оказываются следующие ключи:

- `h` — получение значения результата в виде шестнадцатеричного числа;
- `m` — получение результата в виде последовательности шестнадцатеричных значений отдельных байтов, начиная с младшего;
- `r` — получение списка полей записи с указанием их значений.

Средствами диалогового окна Evaluate and Modify можно искусственно изменить текущее значение той или иной переменной, после чего продолжить процесс отладки программы обычным образом. Для этого в поле Expression сначала следует указать нужную переменную, в поле New value — ее новое значение, а затем нажать кнопку Modify.

Для выхода из диалогового окна Evaluate and Modify следует использовать кнопку Cancel либо клавишу <Esc>.

2.4.2. Синтаксические ошибки

Синтаксические ошибки связаны с нарушением правил языка программирования. Их обнаружение, локализация и идентификация осуществляются компилятором ИСП на этапе трансляции программы. При этом каждая очередная обнаруженная синтаксическая ошибка приводит к прекращению компиляции программы и выводу соответствующего диагностического сообщения.

Компилятор, выявив в программе синтаксическую ошибку, устанавливает курсор текстового редактора ИСП на место ее обнаружения, а также выводит сообщение `Error:` с номером и наименованием ошибки.

- `1. Out of memory` — компилятору не хватает памяти.
- `2. Identifier expected` — ожидается идентификатор. Возможно, вы пытаетесь неправильно использовать зарезервированное слово.
- `3. Unknown identifier` — идентификатор не описан.
- `4. Duplicate identifier` — повторное описание данного идентификатора.
- `5. Syntax error` — обнаружена синтаксическая ошибка. Использован знак, недопустимый в данном контексте. Использован знак, отсутствующий в алфавите языка. Возможно, строка не ограничена апострофами.
- `6. Error in real constant` — ошибка записи константы вещественного типа.
- `7. Error in integer constant` — ошибка в константе целого типа. Например, в операторе `WriteLn(2147483648)` целое число `2147483648` выходит за пределы допустимого диапазона.
- `8. String constant exceeds line` — выход строки за допустимые границы. Возможно, не закрыт апостроф строковой константы.
- `10. Unexpected end of file` — неожидаемый конец. Возможно, несбалансированы операторные скобки `Begin` и `End` или не закрыт комментарий.
- `11. Line too long` — слишком длинная строка программы.
- `12. Type identifier expected` — ожидается тип идентификатора.
- `14. Invalid file name` — неправильное имя файла или несуществующий путь к нему.
- `15. File not found` — файл не найден.
- `17. Invalid compiler directive` — неправильная директива компилятора.
- `19. Undefined type in pointer definition` — неопределенный тип в определении указателя.

- 20. Variable identifier expected — ожидается идентификатор переменной, а не что-либо иное.
- 21. Error in type — ошибка в указании типа.
- 22. Structure too large — слишком велик размер структурного типа.
- 23. Set base type out of range — базовый тип множества выходит за допустимые границы.
- 25. Invalid string length — неправильная длина строки.
- 26. Type mismatch — несоответствие типов: переменной и выражения в операторе присваивания, фактического и формального параметров в обращении к подпрограмме, выражения для индексации массива, операндов в выражении.
- 27. Invalid subrange base type — неправильный базовый тип для интервального типа.
- 28. Lower bound greater than upper bound — в указании интервального типа нижняя граница больше верхней.
- 29. Ordinal type expected — ожидается порядковый тип.
- 30..32. ...constant expected — ожидается константа.
- 33. Pointer type identifier expected — ожидается идентификатор типа указателя.
- 34. Invalid function result type — недопустимый тип функции.
- 36, 37. Begin (End) expected — ожидается Begin (End).
- 38..40. Integer (ordinal, boolean) expression expected — данное выражение должно быть целого (порядкового, логического) типа.
- 41. Operand types do not match operator — тип операндов не соответствует типу знака операции.
- 42. Error in expression — ошибка в выражении.
- 43. Illegal assignment — неправильное употребление присваивания.
- 48. Code segment too large — размер сегмента машинного кода программы или модуля слишком большой.
- 49. Data segment too large — размер сегмента данных слишком большой.
- 50, 54, 55, 57, 58. Do (Of, Interface, Then, To or DownTo) expected — ожидается зарезервированное слово Do (Of, Interface, Then, To or DownTo).
- 59. Undefined forward — не определена указанная ранее подпрограмма.
- 60. Too many procedures — слишком много процедур.

- 61. Invalid typecast — неправильное преобразование типов.
- 62. Division by zero — деление на нуль.
- 63. Invalid file type — файл данного типа не соответствует процедуре обработки файлов.
- 64. Cannot Read or Write variables of type — нельзя вводить или выводить переменные этого типа.
- 65. Pointer variable expected — ожидается переменная-указатель.
- 66. String variable expected — ожидается строковая переменная.
- 67. String expression expected — ожидается выражение строкового типа.
- 68. Circular unit reference — недопустимая в интерфейсных частях двух модулей их ссылка друг на друга.
- 69. Unit name mismatch — несоответствие имен модулей.
- 70. Unit version mismatch — несоответствие версии модуля, т.е. модуль был изменен после его компиляции.
- 71. Internal stack overflow — переполнение внутреннего стека компилятора из-за чрезмерного уровня вложенности операторов.
- 72. Unit file format error — ошибка в формате файла модуля.
- 73. Implementation expected — ожидается раздел реализации.
- 74. Constant and case types do not match — константа и тип селектора в операторе Case несовместимы.
- 76. Constant out of range — константа выходит за границы допустимого диапазона.
- 78. Pointer expression expected — ожидается выражение типа указателя.
- 79. Integer or real expression expected — данное выражение должно быть типа Integer или Real.
- 84. Unit expected — ожидается зарезервированное слово Unit.
- 85..95. “;”, “:”, “,”, “(“, “)”, “=”, “:=”, “[“ or “(.“, “]” or “.)”, “.”, “..” expected — ожидается указанный знак препинания.
- 96. Too many variables — слишком много переменных.
- 97. Invalid FOR control variable — неправильный параметр в операторе For.
- 98. Integer variable expected — ожидается переменная целого типа.
- 100. String length mismatch — несоответствие длины строковой константы символьному массиву.

- 102. String constant expected — ожидается константа строкового типа.
- 103. Integer or real variable expected — ожидается переменная целого или вещественного типа.
- 104. Ordinal variable expected — ожидается переменная порядкового типа.
- 106. Character expression expected — ожидается выражение символьного типа.
- 108. Overflow in arithmetic operation — переполнение при выполнении арифметической операции. Например, результат сложения в операторе `WriteLn(2147483647+1)` выходит за границы типа `LongInt`.
- 109. No enclosing For, While or Repeat statement — процедура `Break` или `Continue` находится вне оператора цикла.
- 110. Debug information table overflow — переполнение таблицы отладочной информации.
- 112. Case constant out of range — константа оператора `Case` выходит за границы допустимого диапазона.
- 113. Error in statement — с данного символа оператор начинаться не может.
- 116. Must be in 8087 mode to compile this — компиляция возможна только в состоянии `{$N+}`.
- 117. Target address not found — оператор, соответствующий указанному адресу, не найден.
- 121. Invalid qualifier — неправильная квалификация переменной. Например, нельзя индексировать переменную, которая не является массивом, нельзя указать поле в переменной, не являющейся записью.
- 122. Invalid variable reference — ссылка на переменную, не указывающую на адрес в памяти.
- 123. Too many symbols — описано слишком много идентификаторов.
- 124. Statement part too large — размер операторной части превышает 24 Кбайт.
- 126. Files must be var parameters — файлы должны быть параметрами-переменными.
- 131. Header does not match previous definition — заголовок подпрограммы не соответствует предыдущему определению.
- 133. Cannot evaluate this expression — выражение вычислить невозможно.

- 134. Expression incorrectly terminated — выражение завершается неправильно.
- 135. Invalid format specifier — неправильная спецификация формата.
- 136. Invalid indirect reference — неправильная косвенная ссылка.
- 137. Structured variables are not allowed here — переменную структурированного типа здесь использовать нельзя.
- 138. Cannot evaluate without system unit — невозможно вычислить без модуля System.
- 139. Cannot access this symbol — доступ к данному идентификатору невозможен.
- 140. Invalid floating point operation — данная операция над вещественными величинами привела к переполнению или делению на нуль.
- 141. Cannot compile overlay to memory — оверлейную программу нельзя компилировать в память.
- 142. Pointer or procedural variable expected — ожидается переменная процедурного типа или переменная-указатель.
- 143. Invalid procedure or function reference — неправильное обращение к подпрограмме.
- 144. Cannot overlay this unit — этот модуль нельзя использовать в качестве оверлейного.
- 145. Too many nested scopes — слишком большой уровень вложенности.
- 146. File access denied — в доступе к файлу отказано.

2.4.3. Ошибки времени выполнения

Ошибки времени выполнения могут быть выявлены в процессе выполнения программы средствами операционной системы. Связаны они с нарушением ограничений языка программирования. Эти ошибки приводят к аварийному завершению работы программы и выводу на экран соответствующего диагностического сообщения.

Сообщение об ошибке времени выполнения имеет вид Runtime error nnn at xxxx : yyyy. Это значит: "Ошибка времени выполнения nnn по адресу xxxx : yyyy", где nnn — код ошибки, а xxxx : yyyy — адрес ошибки (сегмент : смещение) в загрузочном модуле.

- 1. Invalid function number — обращение к несуществующей функции DOS.

- `2. File not found` — файл не найден. Видимо, следует проверить имя, назначенное файловой переменной.
- `3. Path not found` — путь не найден. Видимо, следует проверить имя, назначенное файловой переменной.
- `4. Too many open files` — слишком много открытых файлов.
- `5. File access denied` — доступ к файлу невозможен. Например, в случае попытки записи в файл, не открытый для записи.
- `16. Cannot remove current directory` — текущий каталог удалить невозможно.
- `17. Cannot rename across drive` — при переименовании файла нельзя изменять имя диска.
- `18. No more files` — файлов с заданными характеристиками не найдено.
- `100. Disk read error` — попытка продолжить чтение из полностью прочитанного файла.
- `101. Disk write error` — попытка записи на заполненный диск.
- `102. File not assigned` — файловой переменной не назначено имя файла.
- `103. File not open` — попытка обращения к неоткрытому файлу.
- `104. File not open for input` — попытка обращения к файлу, не открытому для ввода.
- `105. File not open for output` — попытка обращения к файлу, не открытому для вывода.
- `106. Invalid numeric format` — введенное значение имеет неправильный числовой формат.
- `150. Disk is write protected` — диск защищен от записи.
- `151. Unknown unit` — неизвестный модуль.
- `152. Drive not ready` — неготовность дисковода.
- `153. Unknown command` — неопознанная команда.
- `154. CRC error in data` — ошибка контроля данных.
- `155. Bad drive requiest structure length` — указан неверный размер структуры при обращении к диску.
- `156. Disk seek error` — ошибка поиска позиции на диске.
- `157. Unknown media type` — неизвестный тип носителя.
- `158. Sector not found` — не найден сектор на диске.
- `159. Printer out of paper` — отсутствует бумага в принтере.

- `160. Device write fault` — ошибка записи на устройство.
- `161. Device read fault` — ошибка чтения с устройства.
- `162. Hardware failure` — аппаратная ошибка.
- `200. Division by zero` — деление на нуль.
- `201. Range check error` — выход значения за границы допустимого диапазона.
- `202. Stack overflow error` — переполнение стека.
- `203. Heap overflow error` — переполнение динамической памяти.
- `204. Invalid pointer operation` — некорректная операция с указателем.
- `205. Floating point overflow` — переполнение. При операции с вещественными числами возникает чересчур большое число.
- `206. Floating point underflow` — потеря порядка при операции с вещественными числами.
- `207. Invalid floating point operation` — некорректная операция с вещественным числом. Например, извлечение квадратного корня из отрицательного числа.
- `208. Overlay manager not installed` — не инициализирован оверлейный режим.
- `209. Overlay file read error` — ошибка чтения оверлейного файла.
- `215. Arithmetic overflow error` — выход результата целочисленной арифметической операции за допустимые границы, как, например, при выполнении последовательности операторов `x:=1; WriteLn(2147483647+x)`.

2.4.4. Тестирование подпрограмм с помощью таблиц исполнения

В основе достаточно эффективного метода тестирования не очень длинных подпрограмм лежит использование так называемых *таблиц исполнения*. Особенно полезно применение этого метода начинающими, поскольку при этом тестируется не только подпрограмма, но и сам программист, который начинает понимать, что же им все-таки написано.

Таблицы исполнения позволяют прослеживать процессы исполнения подпрограмм и таким образом проверять их правильность, а также изучать и анализировать их работу. Наиболее актуально их использование для циклических подпрограмм как более сложных по сравнению с линейными и разветвляющимися.

Таблица исполнения подпрограммы — прямоугольная.

Столбцы таблицы соответствуют переменным подпрограммы, а также условиям, проверяемым в процессе ее исполнения. При этом в таблицу нецелесообразно включать переменные, значения которых не будут изменяться. Чаще всего это касается тех параметров подпрограммы, которые являются ее исходными данными. Относительно условий следует помнить, что они могут быть неявными (например, условие завершения оператора арифметического цикла). Порядок расположения столбцов, вообще говоря, может быть произвольным. Однако, как мы увидим позже, от этого зависит размер и заполняемость таблицы.

В строках таблицы отображается последовательный процесс исполнения подпрограммы. В них записывают значения переменных и условий, получаемые в результате вычисления выражений и выполнения операторов подпрограммы.

Таблица исполнения подпрограммы заполняется для данного контрольного примера, который указывается в первой строке ее заголовка. Он содержит конкретные значения исходных данных подпрограммы, а также ожидаемые значения результатов, предварительно полученные "вручную". Если тестируемая подпрограмма правильна, то фактически полученные в таблице значения результатов должны совпасть с ожидаемыми результатами контрольного примера. При этом считается, что данная подпрограмма будет правильной для всего множества подобных наборов исходных данных.

С целью тщательного тестирования подпрограммы для разных контрольных примеров может быть построено несколько таких таблиц исполнения.

Во второй строке заголовка таблицы в отдельных столбцах перечисляются переменные и условия, вычисляемые в процессе исполнения. Таким образом, с помощью таблицы исполнения моделируется обращение к подпрограмме.

Для эффективной работы с таблицей исполнения необходимо досконально знать правила исполнения операторов языка Паскаль. Чтобы таблица исполнения подпрограммы была понятной и однозначной, при записи в нее значений переменных и условий следует неукоснительно придерживаться следующего правила: каждое очередное значение должно заноситься только справа относительно всех предыдущих в текущую строку или ниже в следующую строку. Иначе говоря, последовательность заполнения и анализа таблицы исполнения устанавливается строго в направлении "слева — направо — вниз".

Рассмотрим ряд примеров построения таблиц исполнения подпрограмм. Для удобства комментирования таблиц столбцы и строки в них проиндексированы строчными буквами латинского алфавита. При обычном построении таблиц этого можно не делать.

Пример 1. Построим таблицу исполнения подпрограммы вычисления наибольшего общего делителя натуральных чисел *M* и *N*.

```
Function NOD ( M,N: LongInt ): LongInt;
Begin
   While M<>N Do Begin
      If M > N Then M := M - N;
      If N > M Then N := N - M End;
   NOD := M
End;
```

Дано: M = 42, *N* = 70. *Результат: NOD* = 14.					
	M	*N*	*M* <> *N*	*M* > *N*	*N* > *M*
	a	*b*	*c*	*d*	*e*
a	42	70	True	False	True
b		28	True	True	
c	14				True
d		14	False		

Судя по таблице, фактическое значение результата совпало с ожидаемым. Следовательно, на данном контрольном примере подпрограмма работает правильно.

В подпрограмме NOD исходные данные M и N являются изменяемыми, поэтому они включены в таблицу (столбцы ***a*** и ***b***). Кроме того в таблицу включены все три проверяемых условия (столбцы ***c***, ***d*** и ***e***).

Начальные значения переменных M и N заданы (значения 42 и 70 в ячейках ***a-a*** и ***a-b*** соответственно). Первый оператор подпрограммы — оператор цикла с предусловием. Его выполнение начинается с проверки условия M<>N (значение 42<>70, равное True, в ячейке ***a-c***). Поскольку значение результата проверки условия M<>N истинно, то должен быть выполнен цикл.

Первый оператор в составе цикла — оператор ветвления If M>N Then M:=M-N. Его выполнение начинается с проверки условия M>N (значение 42>70, равное False, в ячейке ***a-d***). Поскольку значение результата проверки ложно, то оператор присваивания M:=M-N пропускается. Второй оператор в составе цикла — оператор ветвления If N>M Then N:=N-M. Его выполнение начинается с проверки условия N>M (значение 70>42, равное True, в ячейке ***a-e***). Так как проверенное условие оказалось истинным, то следует выполнить оператор присваивания N:=N-M (значение 70-42=28 в ячейке ***b-b***). Операторы в составе цикла выполнены.

В соответствии с правилом выполнения возвращаемся на проверку условия M<>N (значение 42<>28, равное True, в ячейке ***b-c***). Поскольку проверенное условие истинно, приступаем к выполнению цикла.

Проверяем условие M>N первого оператора ветвления (значение 42>28, равное True, в ячейке ***b-d***). Так как проверенное условие оказалось истинным, то должен быть выполнен оператор присваивания M:=M-N (значение 42-28=14 в ячейке ***c-a***). Проверяем условие N>M второго оператора ветвления (значение 28>14, равное True, в ячейке ***c-e***). Так как проверенное условие оказалось истинным, то должен быть выполнен оператор присваивания N:=N-M (значение 28-14=14 в ячейке ***d-b***). Операторы в составе цикла выполнены.

Возвращаемся к проверке условия M<>N (значение 14<>14, равное False, в ячейке ***d-c***). Поскольку проверенное условие оказалось ложным, то выполнение оператора цикла следует прекратить.

Второй оператор подпрограммы — оператор присваивания. С его помощью значение результата, равное 14, присваивается функции NOD. Исполнение подпрограммы закончено.

Пример 2. Построим таблицу исполнения подпрограммы вычисления суммы факториалов натуральных чисел от 1 до *N*.

```
Function Sum_Fact ( N: LongInt ): LongInt;
   Var i,F,S: LongInt;
Begin
   F := 1; S := 0;
   For i:=1 To N Do Begin
      F := F * i; S := S+F End;
   Sum_Fact := S
End;
```

Дано: N = 3. *Результат: Sum_Fact* = 1!+2!+3! = 9.				
	F	*S*	*i*	*i* <= *N*
	a	*b*	*c*	*d*
a	1	0	1	True
b	1	1	2	True
c	2	3	3	True
d	6	9	4	False

Судя по таблице, фактическое значение результата совпало с ожидаемым. Следовательно, на данном контрольном примере подпрограмма работает правильно.

В таблицу исполнения включены переменные подпрограммы i, F, S, а также неявно проверяемое условие продолжения выполнения оператора арифметического цикла i<=N. Сама переменная N в таблицу не включена, поскольку ее значение в процессе выполнения подпрограммы изменяться не будет.

Сначала выполняются два первых оператора присваивания (значение 1 в ячейке ***a-a*** и значение 0 в ячейке ***a-b***).

Очередной оператор — оператор арифметического цикла. Параметру i присваивается начальное значение (значение 1 в ячейке ***a-c***). Затем осуществляется проверка условия продолжения цикла i<=N (значение 1<=3, равное True, в ячейке ***a-d***).

Поскольку проверенное условие оказалось истинным, должен быть выполнен цикл. Первый оператор присваивания в составе цикла образует новое значение переменной F (значение 1 в ячейке ***b-a***) в результате перемножения переменных F (значение 1 в ячейке ***a-a***) и i (значение 1 в ячейке ***a-c***). Второй оператор присваивания в составе цикла образует новое значение переменной S (значение 1 в ячейке ***b-b***) в результате сложения переменных S (значение 0 в ячейке ***a-b***) и F (значение 1 в ячейке ***b-a***).

Далее значение параметра цикла i автоматически увеличивается на единицу (значение 2 в ячейке ***b-c***) и осуществляется возврат на проверку условия i<=N (значение 2<=3, равное True, в ячейке ***b-d***).

Снова выполняются операторы в составе цикла. В результате переменные F и S приобретают новые значения (значения 2 и 3 в ячейках ***c-a*** и ***c-b*** соответственно). Автоматически увеличивается значение параметра цикла (значение 3 в ячейке ***c-c***). Проверяется условие i<=N (значение 3<=3, равное True, в ячейке ***c-d***).

Операторы в составе цикла выполняются третий раз подряд. Получает новое значение переменная F (значение 6 в ячейке ***d-a***), а также переменная S (значение 9 в ячейке ***d-b***). После увеличения значения параметра цикла (значение 4 в ячейке ***d-c***) и проверки условия i<=N (значение 4<=3, равное False, в ячейке ***d-d***) выясняется, что выполнение оператора арифметического цикла можно прекратить.

Последний оператор подпрограммы — оператор присваивания. С его помощью значение результата, равное 9, присваивается функции Sum_Fact. Исполнение подпрограммы закончено.

Пример 3. Построим таблицу исполнения подпрограммы, определяющей, какое наименьшее количество четных натуральных чисел, начиная с 2, нужно взять, чтобы произведение их квадратов превысило некоторое предельное число *M*. Подпрограмма определяет также значение этого произведения.

```
Procedure Product ( M: LongInt; Var P: LongInt; Var K: Integer );
   Var A: Integer;
Begin
   K := 0; P := 1; A := 2;
   Repeat
      K := K+1; P := P * A * A; A := A+2
   Until P > M
End;
```

Дано: M = 2000. *Результат: P* = 4 *16 *36 = 2304, *K* = 3.				
	K	*P*	*A*	*P* > *M*
	a	*b*	*c*	*d*
a	0	1	2	
b	1	4	4	False
c	2	64	6	False
d	3	2304	8	True

Судя по таблице, фактическое значение результата совпало с ожидаемым. Следовательно, на данном контрольном примере подпрограмма работает правильно.

В таблицу исполнения включены переменные подпрограммы K, P, A, а также условие окончания выполнения оператора цикла с постусловием P>M. В то же время в таблицу не включена переменная M, значение которой в процессе исполнения подпрограммы не изменяется.

Сначала выполняются три первых оператора присваивания (значения 0, 1 и 2 в ячейках ***a-a***, ***a-b*** и ***a-c*** соответственно).

Очередной оператор — оператор цикла с постусловием.

Сначала должны быть выполнены операторы из состава цикла. Первый оператор присваивания увеличивает на единицу значение переменной K (значение 1 в ячейке ***b-a***). Второй оператор присваивания образует новое значение переменной P (значение 4 в ячейке ***b-b***) посредством умножения предыдущего значения переменной P (значение 1 в ячейке ***a-b***) на квадрат значения переменной A (значение 2 в ячейке ***a-c***). Третий оператор присваивания образует новое значение переменной A (значение 4 в ячейке ***b-c***) посредством увеличения на два предыдущего значения той же переменной (значение 2 в ячейке ***a-c***).

Далее осуществляется проверка условия окончания цикла P>M (значение 4>200, равное False, в ячейке ***b-d***).

Снова выполняются операторы из состава цикла. В результате переменные K, P и A приобретают новые значения (значение 1+1=2 в ячейке ***c-a***, значение 4*4*4=64 в ячейке ***c-b*** и значение 4+2=6 в ячейке ***c-c***). Далее проверяется условие P>M (значение 64>2000, равное False, в ячейке ***c-d***).

Операторы из состава цикла выполняются третий раз подряд. Получают новые значения переменная K (значение 2+1=3 в ячейке ***d-a***), переменная P (значение 64*6*6=2304 в ячейке ***d-b***), а также переменная A (значение 6+2=8 в ячейке ***d-c***).

Последующая проверка условия P>M (значение 2304>2000, равное True, в ячейке ***d-d***) показывает, что выполнение оператора цикла с постусловием можно прекратить.

Следующий пример построения таблицы исполнения для той же подпрограммы показывает, что может случиться, если порядок расположения столбцов в таблице выбрать другим.

Дано: M = 2000. *Результат:* P = 4 * 16 * 36 = 2304, K = 3.				
	P	K	A	P > M
	a	*b*	*c*	*d*
a		0		
b	1		2	
c		1		
d	4		4	False
e		2		
f	64		6	False
g		3		
h	2304		8	True

Естественно, порядок следования и общее число значений в таблице не изменились, но зато изменилось их общее расположение в поле таблицы. Изменился также размер таблицы. Данный пример показывает, что порядок расположения столбцов в таблице может быть удачным или неудачным.

Пример 4. Построим таблицу исполнения подпрограммы, выполняющей замену значений всех элементов массива X, отличающихся от заданного Z1, другим заданным значением Z2. Массив X — одномерный, содержит *N* чисел.

```
Type
   P = Array[0..10] Of Integer;
Procedure Replace ( N: Byte; Z1,Z2: Integer; Var X: P );
   Var j: Byte;
Begin
   For j:=1 To N Do
      If X[j]<>Z1 Then X[j] := Z2
End;
```

Содержанием подпрограммы `Replace` является просмотр всех элементов одномерного массива `X` с первого по `N`-й включительно с использованием арифметического цикла (полная обработка массива). В процессе обработки массива значения некоторых его элементов заменяются другими. При этом параметр цикла `j` имеет смысл порядкового номера элемента. Сам массив имеет тип `P`, элемент `X[0]` не используется.

Дано: N=4, X=<5,7,0,7>, Z1=7, Z2=100. *Результат: X*=<100,7,100,7>.			
j	*j*<= N	X[*j*]	X[*j*] <> Z1
1	True	5	True
		100	
2	True	7	False
3	True	0	True
		100	
4	True	7	False
5	False		

Судя по таблице, фактическое значение результата совпало с ожидаемым. Следовательно, на данном контрольном примере подпрограмма работает правильно.

В таблицу исполнения включены переменные подпрограммы `j` и `X[j]`, неявное условие продолжения выполнения оператора арифметического цикла `j<=N`, а также условие замены значений `X[j]<>Z1`. В то же время, в таблицу не включены переменные `N`, `Z1` и `Z2`, значения которых в процессе исполнения подпрограммы не изменяются.

Подпрограмма `Replace` содержит всего лишь один оператор арифметического цикла. Применяем правило его выполнения и сопоставляем получающиеся значения с теми, что встречаются при продвижении вдоль первой строки таблицы.

- Параметру цикла `j` присваивается начальное значение, равное `1`. Именно это значение содержится в первой ячейке строки.
- Выполняется проверка условия `j`≤`N`. При `j=1` и `N=4` это условие истинно, что и указано во второй ячейке строки.
- Так как проверенное условие истинно, то выполняется цикл, содержащий оператор ветвления. Чтобы выполнить этот оператор, следует сначала извлечь значение элемента массива `X[j]`. При `j=1` элемент `X[j]=5`, что мы и видим в третьей ячейке строки.
- Далее проверяется условие `X[j]<>Z1` оператора ветвления. При `X[j]=5` и `Z1=7` это условие истинно, что и отмечено в четвертой ячейке первой строки. Таким образом, первая строка таблицы исполнения исчерпана.
- Поскольку условие оператора ветвления истинно, то должен быть выполнен оператор присваивания, указанный после слова `Then`. В результате этого элемент массива `X[j]`, соответствующий `j=1`, принимает значение `100`. Это значение вносится в третью ячейку второй строки таблицы исполнения.

Далее, в соответствии с правилом выполнения арифметического цикла автоматически должно увеличиваться на единицу значение параметра цикла j. Однако это значение мы не можем внести в первую ячейку второй строки таблицы. В противном случае это значило бы продвижение вдоль строки влево. Поэтому мы должны считать и вторую строку исчерпанной.

Продвигаемся вдоль третьей строки таблицы.

- Параметр цикла j увеличиваем на единицу: j=2. Проверяем условие продолжения цикла j<=N: оно истинно при j=2 и N=4. Извлекаем второй элемент массива: X[j]=7 при j=2.
- Условие оператора ветвления X[j]<>Z1 при X[j]=7 и Z1=7 ложно (значение False в четвертой ячейке третьей строки). Поэтому оператор присваивания, указанный после слова Then, пропускается.

Третья строка таблицы исполнения исчерпана. Продвигаемся вдоль четвертой строки.

- Увеличение значения параметра цикла на единицу: j=3. Проверка условия продолжения цикла j≤N: оно истинно (3≤4=True). Извлечение третьего элемента массива: X[j]=0 при j=3.
- Проверка условия оператора ветвления X[j]<>Z1: при X[j]=0 и Z1=7 получаем значение True. Четвертая строка таблицы исчерпана.
- Выполнение оператора присваивания, указанного после слова Then оператора ветвления: элемент массива X[j], соответствующий j=3, принимает значение 100. Это значение вносится в третью ячейку пятой строки таблицы, после чего и эта строка исчерпывается.

Продвигаемся вдоль шестой строки таблицы исполнения.

- Параметр цикла j увеличиваем на единицу: j=4. Проверяем условие продолжения цикла j≤N: оно истинно при j=4 и N=4. Извлекаем четвертый элемент массива: X[j]=7 при j=4.
- Условие оператора ветвления X[j]<>Z1 при X[j]=7 и Z1=7 ложно (значение False в четвертой ячейке шестой строки). Поэтому оператор присваивания, указанный после слова Then, пропускается.

Шестая строка таблицы исполнения исчерпана. Продвигаемся вдоль седьмой строки.

- Значение параметра цикла j увеличивается на единицу: j=5. Это значение содержится в первой ячейке седьмой строки.
- Выполняется проверка условия j≤N. При j=5 и N=4 это условие ложно, что и указано во второй ячейке седьмой строки.

- Поскольку условие продолжения цикла оказалось ложным, то оператор цикла, а вместе с ним и подпрограмма, оказываются выполненными полностью.

Результатом исполнения алгоритма являются измененные значения некоторых элементов массива X. Прочитать их можно из третьего столбца таблицы исполнения. Делаем это сверху вниз. Сначала при j=1 элемент X[j]=5, но затем он изменяется: X[j]=100. При j=2 элемент X[j]=7. Сначала при j=3 элемент X[j]=0, но затем он изменяется: X[j]=100. При j=4 элемент X[j]=7. Окончательно получаем X=<100,7,100,7>, что совпадает с ожидаемым результатом.

Пример 5. Построим таблицу исполнения подпрограммы, заменяющей значения элементов массива *X* в режиме частичной обработки. При этом только некоторые элементы, отличающиеся от заданного *Z*1, заменяются другим заданным значением *Z*2. Массив *X* — одномерный, содержит *N* чисел.

```
Type
   P = Array[0..10] Of Integer;
Procedure Replace_Part ( a,b,h: Byte; Z1,Z2: Integer; Var X: P );
   Var j: Byte;
Begin

   j := a;
   While j <= b Do Begin
      If X[j]<>Z1 Then X[j] := Z2; j := j+h End;
End;
```

Параметры подпрограммы a,b,h: Byte определяют частичную обработку элементов массива X: P. Обработка выполняется, начиная с элемента с порядковым номером a и заканчивая элементом с порядковым номером b. Элементы обрабатываются не все подряд, а с шагом h по порядковому номеру.

Дано: N=10, X=<5,7,4,6,7,4,6,5,1,3>, *a*=3, *b*=8, *h*=2, Z1=7, Z2=0. *Результат:* X=<5,7,0,6,7,4,0,5,1,3>.			
j	*j* <= *b*	X[*j*]	X[*j*] <> Z1
3	True	4	True
		0	
5	True	7	False
7	True	6	True
		0	
9	False		

Для получения результата обработки, как и прежде, анализируем третий столбец таблицы исполнения. Обработке подвергались элементы массива с порядковыми номерами 3, 5 и 7. Изменены были 3-й и 7-й. Остальные элементы остались без изменения. Результат обработки X=<5,7,0,6,7,4,0,5,1,3> совпадает с ожидаемым.

2.5. Технология работы с модулями

2.5.1. Модули программиста

Модулем `Unit` называется оформляемая по определенным правилам и автономно компилируемая совокупность описаний типов, констант, переменных и подпрограмм. В результате компиляции образуется файл машинного кода модуля. Этот файл не является исполняемым, и воспользоваться им как самостоятельной единицей невозможно. Его можно только подключить к той или иной программе, считающейся первичной, в качестве вспомогательного.

Условия работы с программами и модулями несколько отличаются при использовании разных версий среды программирования: `TURBO.EXE` (Turbo Pascal) или `BP.EXE` (Borland Pascal).

При работе в ИСП Turbo Pascal следует обратить внимание на выбор места размещения машинного кода, образуемого в результате компиляции. Есть возможность поместить его либо в память (Memory), либо на диск (Disk). Для этого необходимо выполнить команду Compile⇨Destination (<Alt+C&D>). Первый вариант (Memory) используется при отладке программы, когда достаточно убедиться в ее правильности и работоспособности. После этого машинный код, образованный в памяти, сразу же утрачивается. Второй вариант (Disk) используется при необходимости образовать машинный код длительного хранения и неоднократного дальнейшего использования.

Таким образом, есть возможность выбрать место размещения машинного кода для первичной программы. Для модулей такого выбора нет. Результаты компиляции модулей должны размещаться на диске обязательно, чтобы ими могла воспользоваться компилируемая первичная программа. Этот вариант для модулей реализуется автоматически.

В отличие от ИСП Turbo Pascal, результаты компиляции в ИСП Borland Pascal всегда размещаются на диске.

Каждый модуль и первичная программа, которая эти модули использует, создаются в отдельных окнах ИСП. При этом могут возникать определенные технологические неудобства, связанные, например, с необходимостью частых переходов между окнами. На этот случай в ИСП предусмотрены средства, частично упрощающие работу с комплексом в составе первичной программы и отдельных модулей.

Для обеспечения комплексной работы следует выполнить команду Compile⇨ Primary file... (<Alt+C&P>), с помощью которой можно назначить первичную программу. При этом в поле Primary program file диалогового окна Primary File следует указать имя файла с расширением `PAS`, содержащего текст первичной программы. Теперь компилироваться в первую очередь будет всегда именно эта программа, независимо от того, что находится в текущем окне текстового редактора

ИСП. Так будет во всех случаях компиляции при выборе команд Run⇨Run (<Ctrl+F9>), Compile⇨Make (<F9>) и Compile⇨Build (<Alt+C&B>).

Теперь предположим, что некая программа определена в качестве Primary program file, но возникла необходимость компиляции совсем другой программы, находящейся в текущем окне текстового редактора ИСП. В этом случае первичная программа будет проигнорирована, если воспользоваться командой компиляции Compile⇨Compile (<Alt+F9>).

И наконец, если программа не использует модули, то имеющееся назначение первичной программы только мешает. В этом случае целесообразно избавиться от нее, выбрав команду Compile⇨Clear primary file (<Alt+C&L>). В результате, независимо от вида команды компиляции, всегда будет компилироваться только та программа, которая находится в текущем окне редактора ИСП.

Различия разновидностей команд компиляции описаны ниже.

- Команда Run⇨Run (<Ctrl+F9>). Выполняется компиляция и запуск на выполнение программы, текст которой находится в текущем окне текстового редактора ИСП. При этом в отношении компиляции действуют те же правила, что и при выполнении команды компиляции Compile⇨Make (<F9>). Попытка выполнить рассматриваемую команду в отношении модуля приводит к появлению сообщения об ошибке "`Cannot run a unit`".
- Команда компиляции Compile⇨Compile (<Alt+F9>). Компилируется та программная единица, текст которой находится в текущем окне редактора ИСП. Если компилировалась первичная программа, то результирующий машинный код записывается в файл с расширением `EXE`. Если компилировался модуль программиста, то результирующий машинный код записывается в файл с расширением `TPU`. В обоих случаях имя созданного файла совпадает с именем файла, содержащего исходный текст программной единицы. Модули, подключаемые к компилируемой программной единице, должны быть откомпилированы и размещены на диске заранее.
- Команда компиляции Compile⇨Make (<F9>). Если имя первичного файла не было определено, то эта команда выполняется точно так же, как и предыдущая. В противном случае, независимо от содержимого текущего окна редактора, выполняется компиляция первичного файла. В процессе компиляции выполняется проверка всех подключенных к программе модулей, и те из них, которые были изменены, перекомпилируются.
- Команда компиляции Compile⇨Build (<Alt+C&B>). Эта команда отличается от предыдущей тем, что перекомпилируются все без исключения модули, подключаемые к данной первичной программе, независимо от того, вносились в них изменения или нет.

При работе с модулями важно правильно ориентироваться в системе каталогов ИСП. Все их разновидности можно видеть в диалоговом окне Directories, которое появляется в результате выполнения команды Options⇨Directories... (Alt+O&D). Однако чаще всего целесообразно использовать простейшую систему хранения, которая состоит в том, что все файлы, имеющие отношение к данной программе, собираются в один и тот же каталог. При этом могут быть полезными нижеследующие рекомендации.

- Поле EXE&TPU directory диалогового окна Directories следует очистить. При этом создаваемые в результате компиляции файлы с расширениями `EXE` и `TPU` будут помещаться в каталог, являющийся текущим. Разумеется, это имеет смысл только в том случае, если, приступая к работе с ИСП, вы назначаете текущим (команда File⇨Change dir…) каталог, содержащий исходные тексты компилируемых программных единиц.
- В поле Unit directories диалогового окна Directories следует указать перечень каталогов, в которых будет осуществляться поиск файлов с модулями, которые должны подключаться к компилируемой первичной программе. Если таких каталогов несколько, то их отделяют друг от друга знаком '`;`'. В частности, чтобы обеспечить доступ первичной программы к стандартным модулям `Strings` и `Graph`, в поле Unit directories следует указать полный путь к каталогу `...BP\UNITS`.

После всех вышеуказанных настроек каталогов необходимо выбрать команду Options⇨Save (Alt+O&S). Благодаря этому все параметры работы ICП, установленные во время настройки, будут сохранены в файле конфигурации.

2.5.2. Стандартные модули

В ряде случаев работа со стандартными модулями требует выполнения специальных условий. Некоторые из них рассматриваются ниже.

2.5.2.1. Модуль Crt

Для управления клавиатурой, дисплеем, звуком используется модуль `Crt`. К сожалению, его подпрограммы способны выполняться не на всех типах процессоров. Если при попытке выполнить в программе процедуру очистки экрана `ClrScr` будет получено сообщение об ошибке "`Error 200: Division by zero`" — деление на нуль, значит, используемый персональный компьютер "не походит" для модуля `Crt`. В этом случае есть смысл попробовать поискать и установить другую версию системной библиотеки `TURBO.TPL`.

2.5.2.2. Модуль Strings

Для работы в программе с ASCIIZ-строками необходимо подключить к ней модуль `Strings`. Следует указать соответствующую директиву `Uses`, выбрать команду Options⇨Directories... (Alt+O&D) и в поле Unit directories прописать полный путь к каталогу `...BP\UNITS`, в котором содержатся стандартные модули ИСП. Полный набор операций с ASCIIZ-строками допустим при условии, что установлен расширенный синтаксис. Для его установки в программе можно воспользоваться директивой компиляции `{$X+}`, а для компилятора в целом необходимо выбрать команду Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует отметить пункт Extended syntax поля Syntax options.

2.5.2.3. Модуль Graph

Для работы в графическом режиме к программе необходимо подключить модуль `Graph`. Непосредственно перед использованием графических процедур этого модуля следует инициализировать графический режим работы экрана. Это достигается с помощью процедуры `InitGraph(`*`Driver`*`, `*`Mode`*`, `*`Path`*`)`, которая вызывает и подключает нужный графический драйвер, представляющий собой файл с расширением `BGI`. Вероятнее всего, на персональном компьютере удастся воспользоваться графическим драйвером `EGAVGA.BGI`, который стандартно хранит ИСП в каталоге `...BP\BGI`. Параметры обращения к этой процедуре таковы.

- *`Driver`* — переменная, определяющая тип драйвера, выбор которого зависит от видеосистемы персонального компьютера. Чтобы воспользоваться имеющимся графическим драйвером `EGAVGA.BGI`, достаточно указать режим автоопределения. Для этого переменной *`Driver`* предварительно присваивается значение `Detect`.
- *`Mode`* – переменная, определяющая режим работы выбранного драйвера. Если был указан режим автоопределения, то значение этой переменной автоматически будет установлено соответствующим наибольшей разрешающей способности экрана.
- Третий параметр представляет собой константу типа `String`, с помощью которой следует указать полный путь к каталогу, в котором хранится нужный графический драйвер. Если на месте этого параметра указана пустая строка, то поиск драйвера будет осуществляться в текущем каталоге.

Таким образом, чтобы обеспечить работоспособность графической программы, достаточно, во-первых, в качестве текущего установить каталог с этой программой, а во-вторых, скопировать в тот же каталог файл графического драйвера `EGAVGA.BGI`.

Результат работы процедуры InitGraph рекомендуется проверять. Для этого используется стандартная функция GraphResult, значением которой всегда является код результата выполнения последней графической операции (в данном случае — инициализации графического режима). Если указанный код отличен от нуля, то это является свидетельством ошибки. В частности, если GraphResult=-3, то это значит, что не был найден файл графического драйвера.

В качестве примера рассматривается программа Diagram, предназначенная для построения графика заданной функции.

```
{$B+,D+,E+,I+,L+,N+,Q+,R+,X-}
Program Diagram;
   Uses Crt,Graph;
   Const
      NHmax=500;
   Var
      Driver,Mode,Error: Integer;
      A,B: Real;
      XH,YH: Array[1..NHmax] Of Real;
      LH: Array[1..NHmax] Of Boolean;
      minX,maxX: Real;
      minY,maxY: Real;
      MX,MY: Real;
      Ox,Oy: Real;
      RHX,RHY: Integer;
      XP,YP: Integer;
      NH: Integer;
      PRX,PRY: Real;
      InitialMode: Integer;
      InitialTextAttr: Byte;
   Procedure IDH;
   Begin
      TextBackground(Blue); TextColor(White); ClrScr;
      Repeat
         Write('Укажите промежуток построения графика: ');
         ReadLn(A,B);
      Until A<B;
      Repeat
         Write('Укажите количество точек: ');
         ReadLn(NH)
      Until (NH>1) And (NH<=NHmax);
      Repeat
         Write('Укажите размер графика по x и y в
↳процентах: ');
         ReadLn(PRX,PRY)
      Until (PRX>=10) And (PRX<=100) And (PRY>=10) And (PRY<=100);
      InitialMode := LastMode; InitialTextAttr := TextAttr
   End;
   Procedure OKX;
      Var j: Integer; h,x: Real;
   Begin
      h := (B-A)/(NH-1); x := A;
      For j:=1 To NH Do
```

```
      Begin XH[j] := x; x := x+h End
End;
Procedure f(x: Real; Var y: Real; Var z: Boolean);
Begin
   z := x<>0;
   If z Then y := x*x * Sin(1/x)
End;
Procedure OKY;
   Var j: Integer;
Begin
   For j:=1 To NH Do f(XH[j],YH[j],LH[j])
End;
Procedure IHR;
Begin
   Driver := Detect;
   InitGraph(Driver,Mode,'');
   Error := GraphResult;
   If Error<>0 Then
      Begin WriteLn('Ошибка: ',GraphErrorMsg(Error));
      ReadLn; Halt End
End;
Procedure PHE;
Begin
   RHX := Round(GetMaxX/100*PRX); RHY := Round(
                                         GetMaxY/100*PRY);
   XP := (GetMaxX-RHX) Div 2; YP := (GetMaxY-RHY) Div 2
End;
Procedure MXY;
   Var j,k: Integer;
Begin
   minX := XH[1]; maxX := XH[NH];
   MX := (maxX-minX)/RHX;
   k := 0; j := 1;
   While j<=NH Do Begin
      If LH[j] Then Begin k := j; j := NH End;
      j := j+1 End;
   If k = 0 Then Begin
      CloseGraph; TextMode(InitialMode);
      TextAttr := InitialTextAttr; ClrScr;
      WriteLn('Нет точек для графика'); ReadLn; Halt End;
   minY := YH[k]; maxY := YH[k];
   For j:=k+1 To NH Do Begin
      If Not LH[j] Then Continue;
      If YH[j]<minY Then minY := YH[j];
      If YH[j]>maxY Then maxY := YH[j] End;
   If maxY=minY Then
      If maxY>0 Then Begin maxY:=2*maxY;
                           minY:=-minY End Else
         If maxY<0 Then Begin maxY:=-maxY;
                              minY:=2*minY End Else
            Begin maxY:=1; minY:=-1 End;
   MY := (maxY-minY)/RHY;
End;
Procedure PKH;
   Var j: Integer;
```

```
   Begin
      For j:=1 To NH Do Begin
         If Not LH[j] Then Continue;
         XH[j] := XP+(XH[j]-minX)/MX;
         YH[j] := YP+RHY - (YH[j]-minY)/MY End;
      Ox := XP+(0-minX)/MX;
      Oy := YP+RHY - (0-minY)/MY
   End;
   Procedure MPH;
      Var sminX,smaxX,sminY,smaxY,s: String;
   Begin
      SetBkColor(Cyan); SetFillStyle(SolidFill,White);
      Bar(XP,YP,XP+RHX,YP+RHY);
      SetColor(Blue); SetLineStyle(SolidLn,0,ThickWidth);
      Rectangle(XP,YP,XP+RHX,YP+RHY);
      SetColor(Red);
      Str(minX:0:3,sminX); Str(maxX:0:3,smaxX);
      Str(minY:0:3,sminY); Str(maxY:0:3,smaxY);
      SetTextJustify(CenterText,TopText);
      SetTextStyle(DefaultFont,HorizDir,1);
      s := 'Диапазон по x: от '+sminX+' до '+smaxX;
      OutTextXY(XP+RHX Div 2,YP+RHY+5,s);
      SetTextJustify(RightText,CenterText);
      SetTextStyle(DefaultFont,VertDir,1);
      s := 'Диапазон по y: от '+sminY+' до '+smaxY;
      OutTextXY(XP-5,YP+RHY Div 2,s)
   End;
   Procedure PTH;
      Var j: Integer; x,y: Integer;
   Begin
      x := Round(Ox); y := Round(Oy);
      SetColor(LightGray); SetLineStyle(SolidLn,0,NormWidth);
      Line(XP,y,XP+RHX,y); Line(x,YP,x,YP+RHY);
      For j:=1 To NH Do Begin
         If Not LH[j] Then Continue;
         x := Round(XH[j]); y := Round(YH[j]);
         PutPixel(x,y,DarkGray) End;
   End;
Begin
   IDH;
   OKX; OKY;
   IHR;
   PHE;
   MXY;
   PKH;
   MPH;
   PTH;
   ReadLn; CloseGraph
End.
```

- Описательная часть программы `Diagram` содержит описания глобальных переменных, определяющих все необходимые величины, средства и параметры построения графика заданной функции.
  - `Const NHmax=500` — именованная константа, определяющая максимально допустимое количество точек, которое может быть выведено на экран при построении графика;
  - `Var Driver, Mode, Error: Integer` — параметры, используемые при инициализации графического режима;
  - `Var A,B: Real` — переменные, определяющие заданный промежуток [*A*, *B*] по оси *x*, на котором должно быть выполнено построение графика заданной функции;
  - `Var XH, YH: Array[1..NHmax] Of Real` — массивы координат точек графика по осям *x* и *y*;
  - `Var LH: Array[1..NHmax] Of Boolean` — массив логических значений, определяющих существование точки графика для данного значения координаты *x*;
  - `Var minX, maxX: Real` — наименьшее и наибольшее значения по координате *x*;
  - `Var minY, maxY: Real` — наименьшее и наибольшее значения по координате *y*;
  - `Var MX, MY: Real` — масштабы по осям *x* и *y* (число линейных единиц на пиксель);
  - `Var Ox, Oy: Real` — положение на экране вертикальной и горизонтальной осей координат, выраженное в пикселях;
  - `Var RHX, RHY: Integer` — экранные размеры поля расположения графика по осям *x* и *y*, выраженные в пикселях;
  - `Var XP, YP: Integer` — экранные координаты левого верхнего угла поля расположения графика;
  - `Var NH: Integer` — фактически заданное количество точек для построения графика;
  - `Var PRX, PRY: Real` — задаваемые размеры поля расположения графика по осям *x* и *y*, выраженные в процентах по отношению к размерам экрана;
  - `Var InitialMode: Integer` — переменная, сохраняющая параметры исходного текстового режима при переходе к графическому режиму работы экрана;
  - `Var InitialTextAttr: Byte` — переменная, сохраняющая текстовый атрибут (цвета фона и знаков для текстового режима).

- Описательная часть программы `Diagram` содержит также описания процедур без параметров, в каждой из которых реализован отдельный функционально законченный этап на пути построения графика заданной функции:
  - `Procedure IDH` — предназначена для ввода исходных данных построения графика;
  - `Procedure OKX` — предназначена для вычисления значений массива координат точек графика по оси *x*;
  - `Procedure OKY` — предназначена для вычисления значений массива координат точек графика по оси *y*;
  - `Procedure IHR` — предназначена для инициализации графического режима;
  - `Procedure PHE` — определение характеристик расположения поля графика на экране, в том числе его размеров по горизонтали и вертикали, а также координат левого верхнего его угла;
  - `Procedure MXY` — вычисление значений масштабов по осям *x* и *y*;
  - `Procedure PKH` — предназначена для пересчета значений координат точек графика и положения осей в пиксели;
  - `Procedure MPH` — предназначена для графического оформления поля построения графика (цвет, рамка, текст);
  - `Procedure PTH` — построение осей и вывод точек графика на экран.
- Исполняемая часть программы `Diagram` содержит обращения ко всем перечисленным выше подпрограммам в том же порядке. При этом работа подпрограмм ввода исходных данных (процедура `IDH`) и вычисления массивов координат точек графика (процедуры `OKX` и `OKY`) осуществляется в текстовом режиме. Графический режим инициализируется (процедура `IHR`) в тот момент, когда возникает потребность в использовании процедур, функций и констант модуля `Graph`. Исполняемая часть программы `Diagram` содержит также оператор `ReadLn`, обеспечивающий задержку графического изображения на экране до момента нажатия клавиши <Enter>.
- Описательная часть программы `Diagram` содержит процедуру `Procedure f(x: Real; Var y: Real; Var z: Boolean)`, с помощью которой задается функция для построения графика. За одно обращение к этой процедуре можно вычислить значение функции `y` для заданного значения аргумента `x`. При этом логический флажок `z` оказывается равным `True`, если значение `y` для заданного `x` вычислить удалось, и равным `False` — в противном случае.

Пример 1. Если необходимо построить график непрерывной функции $y = x \cdot \ln\left(x^2 + 1\right)$, то процедура должна быть оформлена следующим образом:

```
Procedure f(x: Real; Var y: Real; Var z: Boolean);
Begin
   z := True;
   y := x*Ln(x*x+1)
End;
```

Пример 2. Если необходимо построить график разрывной функции $y = x^2 \cdot \sin\frac{1}{x}$, то процедура должна быть оформлена следующим образом:

```
Procedure f(x: Real; Var y: Real; Var z: Boolean);
Begin
   z := x <> 0;
   If z Then y := x*x * Sin(1/x)
End;
```

Пример 3. Если необходимо построить график разрывной функции $y = \frac{1}{\sin\ x} + \lg(\cos\ x)$, то процедура должна быть оформлена следующим образом:

```
Procedure f( x: Real; Var y: Real; Var z: Boolean );
Begin
   z := (Sin(x)<>0) And (Cos(x)>0);
   If z Then y := 1/Sin(x)+Ln(Cos(x))/Ln(10)
End;
```

Примечание. Более подробные сведения о методике программирования вычисления разрывных функций содержатся в главе 3, п. 3.7.1.

- В подпрограммах используются следующие стандартные процедуры, функции, переменные и константы модуля `Crt`:
  - `Procedure TextBackground(Color: Byte);`

    Устанавливает цвет фона для выводимого на экран текста. В качестве параметра `Color` могут использоваться константы из нижеследующей таблицы в пределах от `0` до `7`.
  - `Procedure TextColor(Color: Byte);`

    Устанавливает цвет знаков для выводимого на экран текста. В качестве параметра `Color` могут использоваться константы из табл. 2.2.

Таблица 2.2. Константы цвета текстового режима

Значение константы	*Наименование константы*	*Цвет*
0	Black	черный
1	Blue	синий
2	Green	зеленый
3	Cyan	бирюзовый
4	Red	красный
5	Magenta	фиолетовый
6	Brown	коричневый
7	LightGray	светло-серый
8	DarcGray	темно-серый
9	LightBlue	голубой
10	LightGreen	светло-зеленый
11	LightCyan	светло-бирюзовый
12	LightRed	светло-красный
13	LightMagenta	светло-фиолетовый
14	Yellow	желтый
15	White	белый

- `Procedure ClrScr;`

  Очищает текущее окно экрана и помещает текстовый курсор в его левый верхний угол. Технически это осуществляется заполнением всех знаковых позиций пробелами с цветом фона.

- `Var LastMode: Word;`

  Переменная, сохраняющая номер текущего текстового видеорежима при каждом его изменении.

- `Var TextAttr: Byte;`

  Переменная, сохраняющая так называемый *текстовый атрибут*, т.е. значения цветовых данных, установленных посредством обращения к процедурам `TextBackground` и `TextColor`. Нужные цвета можно устанавливать также прямым присваиванием этой переменной. Для этого используется выражение в виде суммы константы цвета знака и константы цвета фона, умноженной на `16`. Например, оператор присваивания `TextAttr := White+16 * Blue` устанавливает белый цвет для знаков на синем фоне. Отметим, что данный способ позволяет устанавливать в качестве цвета фона любые цвета из табл. 2.2, а не только из первой ее половины.

- `Procedure TextMode(Mode: Word);`

  Устанавливает заданный текстовый режим. Рекомендуется использовать для восстановления исходного текстового режима при выходе из режима графического. С этой целью следует предварительно запоминать исходный текстовый режим с помощью переменной `LastMode`.

■ В подпрограммах используются следующие стандартные процедуры, функции и константы модуля `Graph`:

- `Procedure InitGraph(Var Driver: Integer; Var Mode: Integer; Path: String);`

  Назначение этой процедуры — инициализация графического режима работы экрана (подробности см. выше).

- `Function GraphResult: Integer;`

  Значением функции является код результата выполнения последней графической операции (код завершения). Если указанный код отличен от нуля, то это является свидетельством ошибки. Поскольку значение функции сбрасывается сразу же после обращения к ней, следует предусмотреть сохранение кода завершения во вспомогательной переменной, которую затем и обрабатывать.

- `Function GraphErrorMsg(ErrorCode: Integer): String;`

  Значением функции является строка с сообщением об ошибке, соответствующим заданному ее коду `ErrorCode`.

- `Function GetMaxX: Integer; Function GetMaxY: Integer;`

  Значениями функций являются порядковые номера крайнего справа столбца и самой нижней строки (разрешающая способность экрана в пикселях по `X` и `Y` соответственно) для установленного графического режима. Следует помнить, что нумерация столбцов и строк начинается с нуля.

- `Procedure CloseGraph;`

  Завершает графический режим работы программы. При этом восстанавливается текстовый режим, а также высвобождается память, занятая под графический буфер, графические драйверы и шрифты.

- `Procedure SetBkColor(Color: Word);`

  Устанавливает цвет фона для графических построений на экране. В качестве параметра `Color` могут использоваться константы из табл. 2.2.

- `Procedure SetFillStyle(Pattern: Word; Color: Word);`

  Устанавливает стиль (образец) и цвет закраски для тех графических объектов, которые используют закраску при своем выводе на экран.

В качестве параметра Color могут использоваться константы из табл. 2.2. Параметр Pattern указывает номер образца закраски из числа представленных в табл. 2.3.

Таблица 2.3. Образцы закраски

Значение константы	*Наименование константы*	*Характеристика образца*
0	EmptyFill	Сплошная закраска цветом фона
1	SolidFill	Сплошная закраска заданным цветом
2	LineFill	Заполнение жирными непрерывными горизонтальными линиями
3	LtSlashFill	Заполнение тонкими наклонными линиями "/"
4	SlashFill	Заполнение жирными наклонными линиями "/"
5	BkSlashFill	Заполнение жирными наклонными линиями "\"
6	LtBkSlashFill	Густое заполнение наклонными линиями "\"
7	HatchFill	Заполнение сеткой из горизонтальных и вертикальных тонких линий
8	XhatchFill	Заполнение сеткой из наклонных тонких линий
9	InterLeaveFill	Густое заполнение тонкими горизонтальными штриховыми линиями
10	WideDotFill	Заполнение пунктиром
11	CloseDotFill	Густое заполнение пунктиром
12	UserFill	Заполнение образцом программиста

- `Procedure SetFillPattern(Pattern: FillPatternType; Color Word);`

  Регистрирует определенный программистом образец и цвет закраски. В качестве параметра Color могут использоваться константы из табл. 2.2. Параметр `Pattern: FillPatternType` представляет собой битовый образ закраски в виде массива типа `FillPatternType = Array[1..8] Of Byte`. Например, константа `Const Pattern: FillPatternType=($00,$66,$66,$18,$18,$66,$66,$00)` определяет образец закраски в виде "звездочек". Этот образец со светло-красным цветом закраски регистрируется обращением к процедуре

SetFillPattern(Pattern, LightRed). Теперь, как обычно, остается установить его, обратившись к процедуре SetFillStyle(UserFill, Color), где в данном случае значение параметра Color может быть произвольным.

- Procedure Bar(x1,y1,x2,y2: Integer);

  Рисует закрашенный прямоугольник, для которого (x1,y1) и (x2,y2) — экранные координаты двух его противоположных углов. Для закраски используются стиль и цвет, предварительно установленные обращением к процедуре SetFillStyle.

- Procedure SetColor(Color: Word);

  Устанавливает цвет рисования для графических построений на экране. В качестве параметра Color могут использоваться константы из табл. 2.2.

- Procedure SetLineStyle(LineStyle: Word; Pattern: Word; Thickness: Word);

  Устанавливает стиль (образец) и толщину линии рисования для графических построений на экране. Для толщины линии Thickness может быть выбрано одно из двух значений: NormWidth (значение равно 1) — обычная линия или ThickWidth (значение равно 3) — жирная линия. Параметр Pattern учитывается только при LineStyle=UserBitLn и представляет собой битовый образец линии, созданной программистом, читать который следует с конца. Например, если Pattern=$2724, то линия будет построена из точек и тире азбуки Морзе, соответствующих букве "Ф" (ти-ти-тааа-ти). Параметр LineStyle указывает номер образца линии из числа представленных в табл. 2.4.

Таблица 2.4. Стили линий

Значение константы	*Наименование константы*	*Характеристика стиля*
0	SolidLn	Сплошная линия
1	DottedLn	Пунктирная линия
2	CenterLn	Штрихпунктирная линия
3	DashedLn	Штриховая линия
4	UserBitLn	Образец программиста

- Procedure Rectangle(x1,y1,x2,y2: Integer);

  Рисует прямоугольник, для которого (x1,y1) и (x2,y2) — экранные координаты двух его противоположных углов. При рисовании используются стиль и цвет линии, предварительно установленные обращениями к процедурам SetLineStyle и SetColor соответственно.

- `Procedure SetTextJustify(Horizontal, Vertical: Word);`

  Устанавливает способ выравнивания выводимой на экран горизонтальной строки текста по горизонтали и вертикали относительно точки вывода. Для выравнивания по горизонтали (параметр `Horizontal`) возможны три варианта: `LeftText` (значение `0`) — точка вывода слева от текста, `CenterText` (значение `1`) — точка вывода по центру текста, `RightText` (значение `2`) — точка вывода справа от текста. Для выравнивания по вертикали (параметр `Vertical`) также возможны три варианта: `BottomText` (значение `0`) — точка вывода под строкой текста, `CenterText` (значение `1`) — точка вывода по центру высоты строки текста, `TopText` (значение `2`) — точка вывода над строкой текста.

  Отметим, что для вертикальной строки текста смысл этих параметров несколько изменяется. Для выравнивания по горизонтали (параметр `Horizontal`) возможны два варианта: `LeftText` или `RightText` (значение `0` или `2`) — точка вывода справа от текста, `CenterText` (значение `1`) — точка вывода по центру текста. Для выравнивания по вертикали (параметр `Vertical`) возможны три варианта: `BottomText` (значение `0`) — точка вывода под столбцом текста, `CenterText` (значение `1`) — точка вывода по центру высоты столбца текста, `TopText` (значение `2`) — точка вывода над столбцом текста.

- `Procedure SetTextStyle(Font: Word; Direction: Word; CharSize: Word);`

  Устанавливает разновидность используемого шрифта `Font`, направление вывода текста `Direction` и коэффициент увеличения размера знаков `CharSize`.

  Из всех разновидностей шрифтов оказываются доступными один точечный шрифт размером 8×8, а также несколько штриховых шрифтов. Точечный шрифт используется по умолчанию. Штриховые шрифты хранятся в файлах со стандартными расширениями `CHR` в каталоге `...BP\BGI`. Так же, как и в случае с графическим драйвером `EGAVGA.BGI`, чтобы обеспечить работу графической программы с нужным штриховым шрифтом, достаточно, во-первых, в качестве текущего установить каталог с этой программой, а во-вторых, скопировать в тот же каталог файл со шрифтом. Разновидности шрифтов представлены в нижеследующей таблице.

  Параметр `Direction` может принимать одно из двух возможных значений: `HorizDir` (значение `0`) — направление вывода текста слева направо и `VertDir` (значение `1`) — направление вывода текста снизу вверх.

Значение параметра CharSize=1 для точечного шрифта обеспечивает размещение знаков в прямоугольной области экрана размером 8×8 пикселей. Значение CharSize=2 — в прямоугольной области размером 16×16 пикселей и т.д. Аналогичным образом действует этот параметр и на любой штриховой шрифт, однако исходный размер при этом зависит от его разновидности (табл. 2.5).

Таблица 2.5. Разновидности шрифтов

Значение константы	*Наименование константы*	*Характеристика шрифта*	*Имя файла*
0	DefaultFont	точечный 8×8	—
1	TriplexFont	штриховой	TRIP.CHR
2	SmallFont	штриховой	LITT.CHR
3	SansSerifFont	штриховой	SANS.CHR
4	GothicFont	штриховой	GOTH.CHR
5	—	штриховой	SCRI.CHR
6	—	штриховой	SIMP.CHR
7	—	штриховой	TSCR.CHR
8	—	штриховой	LCOM.CHR
9	—	штриховой	EURO.CHR

- `Procedure OutTextXY(X,Y: Integer; TextString: String);`

  Выводит строку текста TextString на экран в точку с координатами (X,Y). Процедура OutTextXY использует направление вывода, разновидность шрифта и его размер, установленные процедурой SetTextStyle. По умолчанию без вызова этой процедуры используются значения параметров Font=DefaultFont, Direction= HorizDir и CharSize=1. Расположение выведенной строки текста относительно точки вывода определяется процедурой SetTextJustify. По умолчанию без вызова этой процедуры используются значения параметров Horizontal=LeftText и Vertical=TopText.

- `Procedure Line(x1,y1,x2,y2: Integer);`

  Рисует прямую линию, соединяющую точки с координатами (x1,y1) и (x2,y2). При рисовании используются стиль и цвет линии, предварительно установленные обращениями к процедурам SetLineStyle и SetColor соответственно.

- `Procedure PutPixel(X,Y: Integer; Pixel: Word);`

  Отображает пиксель цветом `Pixel` в точке экрана с координатами (`X,Y`). В качестве значений цвета могут использоваться константы из табл. 2.2.

■ Основные особенности процедур программы `Diagram` таковы.

- Процедура `IDH`, предназначенная для ввода исходных данных построения графика, имеет следующий вид.

```
Procedure IDH;
Begin
   TextBackground(Blue); TextColor(White); ClrScr;
   Repeat
      Write('Укажите промежуток построения графика: ');
      ReadLn(A,B);
   Until A<B;
   Repeat
      Write('Укажите количество точек: ');
      ReadLn(NH)
   Until (NH>1) And (NH<=NHmax);
   Repeat
      Write(
       'Укажите размер графика по x и y в процентах: ');
      ReadLn(PRX,PRY)
   Until (PRX>=10) And (PRX<=100) And (PRY>=10)
                                      And (PRY<=100);
   InitialMode := LastMode; InitialTextAttr := TextAttr
End;
```

  Ввод исходных данных осуществляется в текстовом режиме. При этом знаки выводимого текста имеют белый цвет `TextColor(White)` на синем фоне `TextBackground(Blue)`. Использование оператора `ClrScr` после применения `TextBackground` обеспечивает один и тот же фон для всего экрана, а не только в позициях вывода.

  При вводе значений концов промежутка построения графика `A` и `B` соблюдается условие `A<B`, чем, во-первых, гарантируется ненулевое значение его ширины, а во-вторых, обеспечивается правильный порядок указания концов промежутка. Для количества точек построения графика `NH` следует указать значение, во-первых, удовлетворяющее условию `NH>1`, поскольку для графика нужны, как минимум, две точки, а во-вторых, удовлетворяющее условию `NH<=NHmax`, чтобы не выйти за пределы отведенной памяти. Необходимый размер графика по горизонтали и вертикали указывается в процентах по отношению к размерам экрана (`PRX` и `PRY` соответственно). При этом оба значения не могут быть меньше 10% и, естественно, не могут превышать 100%.

  С помощью операторов `InitialMode:=LastMode` и `InitialText Attr:=TextAttr` переменными `InitialMode` и `InitialTextAttr`

запоминаются исходные значения текстового режима и текстового атрибута, чтобы была возможность восстановить их при выходе из графического режима.

- Процедура OKX, предназначенная для вычисления значений массива координат точек графика по оси *x*, имеет следующий вид.

```
Procedure OKX;
   Var j: Integer; h,x: Real;
Begin
   h := (B-A)/(NH-1); x := A;
   For j:=1 To NH Do
      Begin XH[j] := x; x := x+h End
End;
```

Шаг по оси *x* вычисляется по формуле $h = \frac{B-A}{NH-1}$. Первоначальное значение x совпадает с началом промежутка построения графика A. Затем в цикле для каждой из NH точек в массив координат XH заносится очередное значение x, каждый раз увеличиваемое на величину шага h.

- Процедура OKY, предназначенная для вычисления значений массива координат точек графика по оси *y*, имеет следующий вид.

```
Procedure OKY;
   Var j: Integer;
Begin
   For j:=1 To NH Do f(XH[j],YH[j],LH[j])
End;
```

В цикле по j для каждой из NH точек осуществляется обращение к процедуре f(x:Real; Var y:Real; Var z:Boolean) с фактическими параметрами XH[j], YH[j] и LH[j] соответственно. При этом для каждого очередного значения аргумента x=XH[j], если возможно, вычисляется значение функции YH[j]=y, а также определяется логическое значение LH[j]=z, указывающее, удалось или нет вычислить значение функции.

- Процедура IHR, предназначенная для инициализации графического режима, имеет следующий вид.

```
Procedure IHR;
Begin
   Driver := Detect;
   InitGraph(Driver,Mode,'');
   Error := GraphResult;
   If Error<>0 Then
      Begin WriteLn('Ошибка: ',GraphErrorMsg(Error));
      ReadLn; Halt End
End;
```

При инициализации графического режима `InitGraph(Driver,Mode,'')` используется режим автоопределения графического драйвера, для чего выполнено предварительное присваивание `Driver:=Detect`. При этом для переменной `Mode`, определяющей режим работы выбранного драйвера, автоматически устанавливается значение, соответствующее наибольшей разрешающей способности экрана. Третий параметр обращения к `InitGraph` представляет собой пустую строку, что указывает на необходимость поиска файла графического драйвера в текущем каталоге.

Результат работы процедуры `InitGraph` фиксируется присваиванием `Error:=GraphResult`. Затем значение переменной `Error` анализируется, и, если оно отлично от нуля, на экран выводится сообщение об ошибке, получаемое как результат обращения к функции `GraphErrorMsg(Error)`. Сообщение об ошибке задерживается на экране с помощью оператора `ReadLn` до момента нажатия клавиши <Enter>, после чего работа программы завершается по оператору `Halt`.

- Процедура `PHE`, предназначенная для определения характеристик расположения поля графика на экране, имеет следующий вид.

```
Procedure PHE;
Begin
   RHX := Round(GetMaxX/100*PRX);
   RHY := Round(GetMaxY/100*PRY);
   XP := (GetMaxX-RHX) Div 2; YP := (GetMaxY-RHY) Div 2
End;
```

Определению подлежат размеры поля построения графика по горизонтали и вертикали (`RHX` и `RHY` соответственно), а также координаты левого верхнего его угла (`XP`, `YP`). Размеры поля графика определяются в соответствии с заданными их значениями в процентах от максимальных размеров экрана `GetMaxX` и `GetMaxY`. Чтобы получить значения в пикселях, выполняется округление. Координаты левого верхнего угла поля графика определяются так, чтобы в результате поле графика попало в центральную часть экрана.

- Процедура `MXY`, предназначенная для вычисления значений масштабов по осям *x* и *y*, имеет следующий вид.

```
Procedure MXY;
   Var j,k: Integer;
Begin
   minX := XH[1]; maxX := XH[NH];
   MX := (maxX-minX)/RHX;
   k := 0; j := 1;
   While j<=NH Do Begin
      If LH[j] Then Begin k := j; j := NH End;
      j := j+1 End;
```

```
   If k = 0 Then Begin
      CloseGraph; TextMode(InitialMode);
      TextAttr := InitialTextAttr; ClrScr;
          WriteLn('Нет точек для графика');
          ReadLn; Halt End;
   minY := YH[k]; maxY := YH[k];
   For j:=k+1 To NH Do Begin
      If Not LH[j] Then Continue;
      If YH[j]<minY Then minY := YH[j];
      If YH[j]>maxY Then maxY := YH[j] End;
   If maxY=minY Then
      If maxY>0 Then Begin maxY:=2*maxY;
                           minY:=-minY End Else
         If maxY<0 Then Begin maxY:=-maxY;
                              minY:=2*minY End Else
            Begin maxY:=1; minY:=-1 End;
   MY := (maxY-minY)/RHY;
End;
```

Масштабы по осям *x* и *y* представляют собой соответствующие количества линейных единиц на один экранный пиксель.

Масштаб `MX` по оси *x* определяется как результат деления максимальной разности значений по координате *x* на экранный размер поля графика по горизонтали `RHX`. При этом учитывается, что наименьшее значение по координате `minX` находится в первой ячейке массива `XH`, а наибольшее `maxX` — в последней.

Далее в массиве `YH` осуществляется поиск первого из тех значений функции, которые удалось вычислить с помощью процедуры `OKY`. Для этого используются логические значения из массива `LH`. Порядковый номер найденного значения фиксируется с помощью переменной `k`.

Если процедуре `OKY` не удалось вычислить ни одного значения функции, т.е. `k=0`, то с помощью процедуры `CloseGraph` графический режим работы программы завершается, восстанавливается исходный текстовый режим `TextMode(InitialMode)`, восстанавливается значение текстового атрибута `TextAttr:=InitialTextAttr`, а после очистки экрана выводится сообщение об отсутствии точек для построения графика. Это сообщение задерживается на экране с помощью оператора `ReadLn` до момента нажатия клавиши <Enter>, после чего осуществляется завершение работы программы по оператору `Halt`.

Если значения функции процедурой `OKY` были вычислены, т.е. `k<>0`, то, начиная с точки с порядковым номером `k`, в массиве `YH` осуществляется поиск наименьшего `minY` и наибольшего `maxY` значения функции *y*. Поиск ведется с учетом логических значений массива `LH`, т.е. только среди тех значений функции, которые были вычислены процедурой `OKY`.

После завершения поиска обрабатывается исключительный случай, когда maxY=minY, т.е. график функции представляет собой горизонтальную прямую. При этом значения minY и maxY устанавливаются принудительно.

Масштаб MY по оси *y* определяется как результат деления максимальной разности значений по координате *y* на экранный размер поля графика по вертикали RHY.

- Процедура PKH, предназначенная для пересчета значений координат точек графика и положения осей в пиксели, имеет следующий вид.

```
Procedure PKH;
   Var j: Integer;
Begin
   For j:=1 To NH Do Begin
      If Not LH[j] Then Continue;
      XH[j] := XP+(XH[j]-minX)/MX;
      YH[j] := YP+RHY - (YH[j]-minY)/MY End;
   Ox := XP+(0-minX)/MX;
   Oy := YP+RHY - (0-minY)/MY
End;
```

Чтобы пересчитать в пиксели значение координаты *x*, необходимо вычесть из него наименьшее значение minX, эту разность разделить на значение масштаба MX, после чего сместить полученное значение на величину XP (экранная координата левого верхнего угла поля графика по горизонтали).

Чтобы пересчитать в пиксели значение координаты *y*, необходимо вычесть из него наименьшее значение minY, а эту разность разделить на значение масштаба MY. Теперь необходимо учесть, что экранный нуль по координате *y* находится в левом верхнем углу поля построения графика. Поэтому, чтобы перевернуть ось, необходимо вычесть ранее полученную величину из максимального значения по экранной координате *y*, т.е. из RHY. Остается сместить полученное значение на величину YP (экранная координата левого верхнего угла поля графика по вертикали).

Указанные преобразования выполняются для координат всех тех точек с первой по NH, для которых удалось вычислить значение функции.

Аналогичным образом пересчитываются в пиксели и положения осей координат. При этом учитывается, что вертикальная ось имеет нулевое значение своей координаты по оси *x*, а горизонтальная ось имеет нулевое значение своей координаты по оси *y*.

- Процедура MPH, предназначенная для графического оформления поля построения графика, имеет следующий вид.

```
Procedure MPH;
   Var sminX,smaxX,sminY,smaxY,s: String;
Begin
   SetBkColor(Cyan); SetFillStyle(SolidFill,White);
   Bar(XP,YP,XP+RHX,YP+RHY);
   SetColor(Blue); SetLineStyle(SolidLn,0,ThickWidth);
   Rectangle(XP,YP,XP+RHX,YP+RHY);
   SetColor(Red);
   Str(minX:0:3,sminX); Str(maxX:0:3,smaxX);
   Str(minY:0:3,sminY); Str(maxY:0:3,smaxY);
   SetTextJustify(CenterText,TopText);
   SetTextStyle(DefaultFont,HorizDir,1);
   s := 'Диапазон по x: от '+sminX+' до '+smaxX;
   OutTextXY(XP+RHX Div 2,YP+RHY+5,s);
   SetTextJustify(RightText,CenterText);
   SetTextStyle(DefaultFont,VertDir,1);
   s := 'Диапазон по y: от '+sminY+' до '+smaxY;
   OutTextXY(XP-5,YP+RHY Div 2,s)
End;
```

В качестве оформления в процедуре MPH предусмотрены следующие элементы. Процедурой SetBkColor(Cyan) на экране устанавливается бирюзовый цвет фона. Для непосредственного вывода точек графика процедурой Bar(XP,YP,XP+RHX,YP+RHY) вырисовывается прямоугольник поля построения графика со сплошной закраской белым цветом SetFillStyle(SolidFill,White). Поле построения графика имеет прямоугольную рамку Rectangle(XP,YP,XP+RHX,YP+RHY), изображенную сплошной жирной линией SetLineStyle(SolidLn, 0,ThickWidth) синего цвета SetColor(Blue). Все надписи для поля построения графика выполняются красным цветом SetColor(Red). Для надписей используются граничные значения по осям minX, maxX, minY и maxY, предварительно преобразованные в строки sminX, smaxX, sminY и smaxY соответственно с помощью процедуры Str. Для горизонтальной надписи точка вывода устанавливается по центру и над строкой текста SetTextJustify(CenterText,TopText). Для горизонтальной надписи устанавливается точечный шрифт с единичным значением коэффициента увеличения его размера SetTextStyle(DefaultFont,HorizDir,1). Вывод горизонтальной надписи s осуществляется в точку, которая находится по центру ширины поля построения графика на 5 пикселей ниже его нижнего края OutTextXY(XP+RHX Div 2, YP+RHY+5,s). Для вертикальной надписи точка вывода устанавливается справа и по центру высоты столбца текста SetTextJustify(RightText,CenterText). Для вертикальной надписи устанавливается точечный шрифт с единичным значением коэффициента увеличения его размера SetTextStyle(DefaultFont,VertDir,1).

Вывод вертикальной надписи s осуществляется в точку, которая находится по центру высоты поля построения графика на 5 пикселей левее его левого края OutTextXY(XP-5,YP+RHY Div 2,s).

- Процедура PTH, предназначенная для построения осей и вывода точек графика на экран, имеет следующий вид.

```
Procedure PTH;
   Var j: Integer; x,y: Integer;
Begin
   x := Round(Ox); y := Round(Oy);
   SetColor(LightGray);
   SetLineStyle(SolidLn,0,NormWidth);
   Line(XP,y,XP+RHX,y); Line(x,YP,x,YP+RHY);
   For j:=1 To NH Do Begin
      If Not LH[j] Then Continue;
      x := Round(XH[j]); y := Round(YH[j]);
      PutPixel(x,y,DarkGray) End;
End;
```

В процедуре PTH во всех случаях непосредственных графических построений используется предварительное округление координат с помощью функции Round. Для построения осей графика используется сплошная линия нормальной толщины SetLineStyle(SolidLn,0, NormWidth) светло-серого цвета SetColor(LightGray). Горизонтальная ось графика прорисовывается в пределах от левого до правого края поля построения графика Line(XP,y,XP+RHX,y) на высоте, которая соответствует нулевому значению координаты *y*. Вертикальная ось графика прорисовывается в пределах от верхнего до нижнего края поля построения графика Line(x,YP,x,YP+RHY) с горизонтальным смещением, которое соответствует нулевому значению координаты *x*. Отметим, что в зависимости от промежутка построения графика и от конкретных значений функции оси координат могут либо находиться за пределами поля построения графика, либо быть вовсе не видны. Вывод на экран всех NH точек графика (XH[j], YH[j]) с учетом признака их существования, определяемого логическим значением LH[j], осуществляется темно-серым цветом с помощью процедуры PutPixel(x,y, DarkGray).

2.5.2.4. Модуль Overlay

Средства модуля Overlay позволяют строить программу в виде совокупности оверлейных модулей, использующих одну и ту же область оперативной памяти (оверлейный буфер). Оверлейные модули могут замещать друг друга, если в одном из них необходимость отпадает, а в другом — появляется.

Ниже представлен пример оформления оверлейной структуры двух модулей, которые постоянно находятся на диске, а в оверлейный буфер загружаются попеременно.

```
{$F+,O+}
Program Example_Overlay;
Uses Overlay,OV1,OV2;
   {$O OV1}
   {$O OV2}
Begin
   OvrInit('Example.Ovr');
   WriteLn('Код завершения инициализации: ',OvrResult);
   Repeat
      Write1; ReadLn;
      Write2; ReadLn;
      WriteLn('Количество загрузок: ',OvrLoadCount)
   Until False;
End.
```

```
{$F+,O+}
Unit OV1;
Interface
   Procedure Write1;
Implementation
   Procedure Write1;
   Begin
      WriteLn('Работает OV1')
   End;
End.
```

```
{$F+,O+}
Unit OV2;
Interface
   Procedure Write2;
Implementation
   Procedure Write2;
   Begin
      WriteLn('Работает OV2')
   End;
End.
```

Правила оформления таковы:

- для главной программы `Example_Overlay` и обоих оверлейных модулей используются директивы компиляции `{$F+}` — дальняя модель вызова подпрограмм, обеспечивающая межсегментные связи при обращении к подпрограммам, и `{$O+}` — разрешение использования оверлейной структуры;
- в главной программе непосредственно за директивой `Uses`, в которой указываются вообще все подключаемые модули, следует также отдельный перечень всех оверлейных модулей;
- сама главная программа должна начинаться с обращения к процедуре `OvrInit`, инициализирующей оверлейный файл, содержащий все оверлейные модули, имя которого указывается в качестве параметра;
- имя оверлейного файла `Example` должно совпадать с именем файла главной программы и иметь расширение `Ovr`.

В главной программе с демонстрационной целью выводится значение переменной `OvrResult`, нулевое значение которой свидетельствует об успешной инициализации оверлейного файла. Кроме того, после обращений к процедурам `Write1` и `Write2`, находящимся в оверлейных модулях, выводится значение системной переменной `OvrLoadCount` (счетчик количества загрузок оверлейных модулей из оверлейного файла в оверлейный буфер).

Отметим, что компиляция с расчетом на дальнюю модель вызова подпрограмм и разрешение использования оверлейной структуры могут быть установлены также для ИСП в целом с помощью команды Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует отметить пункты Force far calls и Overlays allowed поля Code generation.

2.5.2.5. Модуль Printer

Работу программы с печатающим устройством обеспечивает модуль `Printer`. Он содержит стандартное имя `Lst`, с помощью которого и осуществляется обращение к принтеру в обычном операторе вывода. Эта схема не срабатывает, если принтер подсоединен к порту USB персонального компьютера. Поэтому приходится использовать промежуточную печать в текстовый файл.

В качестве примера ниже представлена программа, которая выводит таблицу кодов ASCII в текстовый файл.

```
{$B+,D+,E+,I+,L+,N+,Q+,R+,X-}
Program Table_ASCII;
Const
   N: String[16]='0123456789ABCDEF';
Var
   F: Text;
   i,j: Byte;
   c: Char;
   s: String;
Begin
   Assign(F,'ASCII.txt'); Rewrite(F);                    {1}
   Write(F,'   ¦ ');                                     {2}
   For j:=0 To 15 Do Write(F,N[j+1],'  ');               {3}
   WriteLn(F);                                           {4}
   FillChar(s,51,#196); s[0]:=#50; s[3]:=#197;           {5}
   WriteLn(F,s);                                         {6}
   WriteLn(F,'   ¦');                                    {7}
   Write(F,'0 ¦ ');                                      {8}
   For j:=0 To 15 Do Begin                               {9}
      If j In [9,10,13] Then c:=' ' Else c:=Chr(j);     {10}
      Write(F,c,'  ') End;                              {11}
   WriteLn(F);                                          {12}
   For i:=1 To 15 Do Begin                              {13}
      WriteLn(F,'   ¦');                               {14}
      Write(F,N[i+1],' ¦ ');                            {15}
```

```
      For j:=0 To 15 Do Write(F,Chr(i*16+j),'  ');     {16}
      WriteLn(F) End;                                  {17}
   Close(F)                                            {18}
End.
```

С помощью полученной таблицы легко сопоставить знак с его кодом и наоборот. Два шестнадцатеричных номера строки и столбца таблицы, на пересечении которых находится данный знак, образуют его двухразрядный код, который можно преобразовать в десятичную систему счисления известным способом.

Краткое описание программы.

- Строка {1}. В текущем каталоге создается текстовый файл ASCII.txt для вывода таблицы кодов ASCII.
- Строки {2}-{4}. Вывод шапки таблицы, содержащей номера шестнадцати ее столбцов. В качестве номеров столбцов используются шестнадцатеричные цифры от 0 до F. Сами цифры содержатся в типизированной текстовой константе N. В начале строки выводится фрагмент вертикальной линии таблицы.
- Строки {5}-{6}. Формирование и вывод горизонтальной линии ниже шапки. Сначала с помощью процедуры FillChar образуется последовательность s из 51 знака с кодом 196 (горизонтальный отрезок). Последовательность s преобразуется в строку s посредством указания в качестве нулевого байта символа с кодом 50. Третий байт этой строки заменяется на крестообразный символ с кодом 197 для отображения пересечения с вертикальной линией таблицы.
- Строка {7}. Вывод пустой строки таблицы ниже горизонтальной линии. В составе строки имеется фрагмент вертикальной линии.
- Строки {8}-{12}. Вывод строки с порядковым номером нуль. Необходимость отдельной ее обработки обусловлена тем, что только она содержит знаки, выводить которые нежелательно. Сначала выводится порядковый номер 0 вместе с фрагментом вертикальной линии. Затем организуется цикл для всех 16 элементов нулевой строки таблицы. Каждый код j преобразуется в соответствующий знак c с помощью функции Chr. Из их числа исключаются коды 9, 10 и 13, которые преобразуются в обычный пробел. Знаки с указанными кодами соответствуют операциям табуляции, перевода строки и возврата каретки, что портит внешний вид таблицы. Знак c и пробелы после него выводятся. Все это завершается выводом конца строки.
- Строки {13}-{17}. Организация цикла для всех остальных 15 строк, с первой по пятнадцатую. Выше каждой из них выводится пустая строка с фрагментом вертикальной линии. Для каждой i-й строки выводится ее порядковый номер, выбираемый из константы N, а также фрагмент верти-

кальной линии. Все 16 кодов строки вычисляются по формуле `i*16+j`, преобразуются в знак функцией `Chr` и последовательно выводятся с сопутствующими пробелами. После этого выводится конец строки.

- Строка `{18}`. Файл закрывается.

Полученный текстовый файл `ASCII.TXT` не может быть распечатан обычным образом (например, в Windows с помощью стандартной программы Блокнот). Это объясняется тем, что Windows использует кодировку знаков, отличающуюся от той, которая используется в Паскале. Поэтому для вывода на принтер необходимо сначала преобразовать его содержимое в формат документа текстового процессора MS Word. Основные действия при этом практически те же, что и при сохранении текстов Паскаль-программ в виде документов MS Word.

- Запустите текстовый процессор MS Word, откройте пустое окно (кнопка Создать на панели инструментов Стандартная) и сохраните документа (кнопка Сохранить на панели инструментов Стандартная) под именем "Таблица кодов ASCII".
- Запустите ИСП и откажитесь от полноэкранного режима, переведя ИСП в оконный режим Windows. Для этого достаточно нажать клавиши <Alt+Enter>.
- Откройте файл `ASCII.TXT` в окне редактора ИСП. Поскольку система по умолчанию отображает только те файлы, которые имеют расширение `PAS`, то, выполняя функцию File⇨Open... (<F3>), нужно будет в поле Name диалогового окна Open a File своевременно изменить шаблон на `*.TXT`.
- Щелкните правой кнопкой мыши по заголовку окна и в появившемся контекстном меню выберите пункт Изменить, после чего в появившемся каскадном меню щелкните левой кнопкой мыши на пункте Пометить. Выделите таблицу кодов, после чего скопируйте ее в буфер обмена, нажав клавишу <Enter>.
- Перейдите в окно текстового процессора Word, установите текстовый курсор в нужное положение и вставьте скопированный фрагмент текста в документ "Таблица кодов ASCII", щелкнув на кнопке Вставить панели инструментов Стандартная. Если за один раз скопировать всю таблицу кодов в документ не удается, то повторите предыдущие действия для очередных фрагментов таблицы.
- Выделите в документе полностью скопированную таблицу (команда Правка⇨ Выделить все) и в поле Шрифт панели инструментов Форматирование выберите для нее моноширинный шрифт Courier New. Теперь документ можно сохранить окончательно и вывести его на печать (кнопка Печать панели инструментов Стандартная).

2.6. Директивы компиляции

Чтобы обеспечить независимость от конкретной настройки ИСП, рекомендуется практически всегда использовать для своих программ предваряющий набор директив компиляции.

Директивы компиляции используются для придания исполняемой программе определенных свойств и особенностей. Удобнее всего их разместить непосредственно перед текстом программы. В этом случае полный набор директив компиляции может быть получен автоматически посредством нажатия клавиш <Ctrl+O&O>. Далее он может быть нужным образом отредактирован и приведен, например, к виду `{$B+,I+,Q+,R+,X-}`.

Директивы компиляции имеют вид комментария, обрамленного фигурными скобками "{}". Непосредственно за открывающей скобкой должен следовать символ "$". Такой комментарий воспринимается компилятором как указание обеспечить соответствующие свойства компилируемой программы.

Большинство директив состоят из двух знаков. Первый знак — буква латинского алфавита, которая имеет определенный смысл. Вторым знаком такой директивы является "+" или "-", обозначающий действие или отмену действия данной директивы.

Смысл основных из них таков:

- `{$B+}` — установка полной схемы вычислений логических выражений;
- `{$D+}` — создание отладочной информации в процессе компиляции;
- `{$E+}` — разрешение эмуляции; наличие сопроцессора определяется автоматически; он используется при его обнаружении, в противном же случае его работа эмулируется;
- `{$F+}` — установка дальней модели вызова подпрограмм;
- `{$I+}` — установка автоматического контроля правильности выполнения операций ввода-вывода;
- `{$L+}` — создание отладочной информации не только для глобальных, но и для локальных переменных;
- `{$N+}` — ориентирование компилятора на выполнение операций с вещественными числами с помощью либо сопроцессора (если он есть), либо эмуляции (если сопроцессор отсутствует);
- `{$Q+}` — установка автоматического контроля переполнения при выполнении арифметических операций;
- `{$R+}` — установка автоматического контроля выхода за допустимые границы, предусмотренные типами;
- `{$S+}` — установка автоматического контроля переполнения стека;

- {$V+} — установка автоматического контроля типов строк, передаваемых в качестве фактических параметров подпрограммы;
- {$X+} — разрешение расширенного синтаксиса, в том числе разрешение обращения к функциям как к процедурам.

2.6.1. Вычисление логических выражений

Во многих случаях результат вычисления логического выражения оказывается предопределенным, даже если вычисления не доведены до конца. Например, при A=1 вычисление логического выражения (A>0) Or (B=5) можно прекратить сразу после вычисления первого отношения (A>0). Поскольку оно истинно, и выполняться должна логическая операция Or, то результат будет истинным, независимо от значения второго отношения (B<>5). Аналогичный пример может быть указан и для логической операции And. Вычисление выражения (A<0) And (B=5) также может быть прекращено на первом отношении. Поскольку оно ложно, то ложным будет и результат в целом. Очевидно, что учитывать возможность появления предопределенного результата полезно, поскольку при этом уменьшается количество вычислений.

Таким образом, принципиально возможны две схемы вычисления логических выражений. При вычислении логического выражения по полной схеме процесс осуществляется формально, т.е. выражение вычисляется полностью от начала до конца и возможность появления предопределенного результата не учитывается. И наоборот, при вычислении логического выражения по сокращенной схеме так же, как и в вышеприведенных примерах, учитывается возможность появления предопределенного результата. Это значит, что в определенный момент процесс вычислений может быть прекращен, если в его продолжении нет фактической необходимости.

Установка сокращенной схемы вычисления логических выражений достигается с помощью указания в начале программы директивы компиляции {$B-}. При указании директивы компиляции {$B+} для данной программы устанавливается полная схема вычисления логических выражений.

Однако в ряде случаев использование сокращенной схемы вычисления логических выражений может привести к ошибкам. В качестве примера рассмотрим вычисление логического выражения (A<0) And F(A,B) при A=1, где в качестве второй составляющей используется обращение к некоторой логической функции F(A:Integer; Var B:Integer): Boolean. Предположим, что эта функция вычисляет не только собственное логическое значение, но также и значение параметра B. Очевидно, что попытка учесть предопределенность результата и прекратить вычисление выражения на первом же ложном отношении (A<0) приведет к тому, что обращение к функции F(A,B) не состоится, и переменная B окажется неопределенной.

Во избежание возможных ошибок пользоваться сокращенной схемой вычислений {$B-} рекомендуется не в целом для программы, а на отдельных ее участках.

Полная схема вычисления логических выражений может быть установлена для ИСП в целом с помощью выбора команды Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует отметить пункт Complete boolean eval поля Syntax options.

2.6.2. Создание отладочной информации

В процессе создания программы целесообразно пользоваться всеми средствами отладки программы, которые доступны нам в рамках ИСП (см. раздел 2.4.1). Чтобы эти возможности могли быть реализованы, в процессе компиляции должна генерироваться отладочная информация.

Отладочная информация создается, если установлена директива компиляции {$D+}. Если, кроме того, установить директиву компиляции {$L+}, то в процессе отладки будут доступными не только переменные самой программы (глобальные), но и переменные, описанные в подпрограммах (локальные).

Создание отладочной информации может быть установлено для ИСП в целом с помощью команды Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует отметить пункты Debug information и Local symbols поля Debugging.

2.6.3. Настройка компилятора на вещественные типы

Чтобы иметь возможность использовать при программировании любые вещественные типы, следует правильно настроить процесс компиляции программы. При этом необходимо помнить, что числа типа Real могут беспрепятственно использоваться на персональных компьютерах любых типов. Работа со всеми остальными вещественными типами может быть реализована по одной из двух возможных схем:

- аппаратно, с помощью специального математического сопроцессора;
- программно, посредством эмуляции (воспроизведения) работы математического сопроцессора.

Настройку процесса компиляции программы на использование вещественных типов определяет директива компиляции {$N±} — использование установленного сопроцессора или его программная эмуляция, а также директива {$E±} — использование эмуляции. Возможные сочетания значений этих директив представлены в табл. 2.6.

Таблица 2.6. Настройка компилятора на вещественные типы

Значения директив	*Смысл сочетания директив*
{$N-,E-} {$N-,E+}	Вообще исключается возможность использования программой математического сопроцессора или подпрограмм его эмуляции. В программе доступен только тип Real
{$N+,E+}	Компилятор ориентируется на выполнение операций с вещественными числами с помощью или математического сопроцессора (если он установлен), или подпрограмм эмуляции (если сопроцессор отсутствует). При этом наличие сопроцессора определяется автоматически. Он будет использоваться в случае его обнаружения, иначе его работа будет эмулироваться
{$N+,E-}	Компилятор ориентируется на обязательное использование математического сопроцессора. Эмуляция запрещена. В этом случае программа не может быть выполнена, если математический сопроцессор отсутствует

Соответствующая настройка на вещественный тип может быть установлена для ИСП в целом с помощью команды Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует отметить пункты 8087/80287 и Emulation поля Numeric processing.

2.6.4. Контроль операций ввода-вывода

Программист имеет возможность подключить к своей программе средства, которые будут автоматически контролировать правильность выполнения операций ввода-вывода данных. Для этого он должен использовать директиву компиляции {$I+}.

При использовании противоположной по смыслу директивы {$I-} программист должен сам организовать в своей программе анализ успешности выполнения операций ввода-вывода. Такой анализ может быть осуществлен с помощью стандартной целочисленной функции IOResult: Integer. Ее значением является код ошибки последней из выполненных операций ввода-вывода. Нулевое же значение кода свидетельствует об отсутствии ошибок.

Если автоматический контроль отключен, то в программе к функции IOResult следует обратиться обязательно. В противном случае, если произойдет ошибка ввода-вывода, то все последующие операции ввода-вывода программы будут просто проигнорированы. Отметим также, что результативным может быть лишь однократное обращение к этой функции, поскольку ее значение при этом теряется.

Таблица 2.7. Ошибки ввода-вывода

Код ошибки	*Сообщение об ошибке*	*Содержание ошибки*
100	Disk read error	Ошибка чтения с диска
101	Disk write error	Ошибка записи на диск
102	File not assigned	Файл не назначен
103	File not open	Файл не открыт
104	File not open for input	Файл не открыт для ввода
105	File not open for output	Файл не открыт для вывода
106	Invalid numeric format	Неправильный числовой формат

Ниже представлен пример простой программы, в которой выполняется проверка введенного значения на его соответствие целочисленному числовому формату.

```
{$B+,D+,E+,F+,I+,L+,N+,Q+,R+,X-}
   Var k: Byte; IOR: Integer;
Begin
   {$I-}
   Write('Введите целое число: '); ReadLn(k);
   IOR := IOResult;

   {$I+}
   If IOR<>0 Then WriteLn('Ошибка. IOR = ',IOR)
             Else WriteLn('Нет ошибки. IOR = ',IOR);
   If IOR=0 Then WriteLn('Число k = ',k);
   ReadLn
End.
```

Автоматический контроль правильности выполнения операций ввода-вывода может быть установлен для ИСП в целом с помощью команды Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует отметить пункт I/O checking поля Runtime errors.

2.6.5. Контроль ситуаций переполнения и выхода за границы

В процессе выполнения арифметических операций могут возникать ситуации переполнения, т.е. выхода значений выражений за пределы возможных для размещения в памяти.

При выполнении операций присваивания могут возникать ситуации выхода значений переменных и индексов за допустимые границы, предусмотренные их типами.

При использовании директив компиляции {$Q+} и {$R+} над указанными ошибочными ситуациями устанавливается контроль, выполняемый операционной системой во время работы программы. При этом переполнение и выход за границы приводят к завершению выполнения программы и выдаче соответствующего сообщения об ошибке.

Следует помнить, что при использовании директивы {$Q-} для целочисленных операций во многих случаях есть риск получить неправильный результат, не подозревая об этом. Например:

- при `A: Integer=200` для выражения `A*190` в качестве результата получаем значение `-27536`;
- при `x: Byte=1` в результате выполнения оператора `WriteLn(2147483647+x)` выводится число `-2147483648` и т.п.

Контроль ситуаций переполнения и выхода значений переменных и индексов за допустимые границы может быть установлен для ИСП в целом с помощью команды Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует отметить пункты Range checking и Overflow checking поля Runtime errors.

2.6.6. Расширенный синтаксис

В ИСП предусмотрены некоторые элементы так называемого расширенного синтаксиса. К их числу относится возможность обращения к функциям не только в составе выражений, но и в виде отдельного оператора, т.е. как к процедурам. Для этого следует использовать директиву компиляции {$X+}.

Разумеется, если записать обращение к функции в виде отдельного оператора, то само значение функции будет при этом теряться. Поэтому делать это имеет смысл в тех случаях, когда среди параметров функции есть еще дополнительные параметры-переменные, которые могут передавать значения других ее результатов.

Указанная возможность не относится к стандартным функциям.

Кроме того, установка расширенного синтаксиса необходима при обработке в программе ASCIIZ-строк.

Возможность использования расширенного синтаксиса может быть установлена для ИСП в целом с помощью команды Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует отметить пункт Extended syntax поля Syntax options.

2.6.7. Работа со стеком

Весь механизм вызова подпрограмм построен на использовании стека. Особенно активная работа со стеком ведется при рекурсивных обращениях.

Определенное влияние на стек программист имеет возможность оказывать непосредственно из ИСП. Выберите команду Options⇨Memory sizes… (<Alt+O&M>) и в соответствующем диалоговом окне Memory sizes вы обнаружите размер стека (Stack size) для реального режима работы процессора (Real Target), который должен быть указан в пределах от 1024 до 65520 байт. Кроме того, размер стека может регулироваться и непосредственно в программе с помощью директивы компиляции {$M 65520,0,655360}, в которой первое число должно указывать размер стека в указанных выше пределах. Например, минимальный размер стека в 1024 байт устанавливается директивой {$M 1024,0,655360}.

Ограниченность размеров стека приводит к необходимости контроля его заполняемости в процессе работы программы во избежание его переполнения.

Упомянутый контроль может быть установлен с помощью директивы компиляции {$S+}, в результате действия которой при переполнении стека работа программы прерывается и появляется сообщение об ошибке "Error 202: Stack overflow error".

Если снять контроль переполнения стека, установив директиву компиляции {$S-}, то, независимо от установленного размера стека, ничего плохого не происходит, пока заполненная его часть не превышает 65520 байт. При превышении указанного граничного значения процессор персонального компьютера обнаруживает недопустимую инструкцию, после чего программа автоматически снимается с выполнения.

Контроль переполнения стека может быть установлен для ИСП в целом посредством выбора команды Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует установить флажок на пункте Stack checking поля Runtime errors.

2.6.8. Дальняя модель вызова подпрограмм

Архитектурой персонального компьютера предусмотрено разбиение его оперативной памяти на сегменты, размером 64 Кбайт каждый. При этом могут использоваться как короткие адреса (в пределах сегмента), так и полные адреса (с указанием номера сегмента). Соответственно могут использоваться и две модели обращения к подпрограммам: ближняя (Near) и дальняя (Far). Для ближней модели адресация выполняется в пределах сегмента памяти. Дальняя модель вызова подпрограмм обеспечивает межсегментные связи.

Компиляция с расчетом на дальнюю модель может потребоваться, например, в программах, использующих подпрограммы с фактическими параметрами процедурных типов. В этих случаях использование дальней модели вызова подпрограмм необходимо для установления правильных связей функций, выступающих в качестве фактических параметров, с вызывающими подпрограммами.

Установка дальней модели для тех процедур и функций, которым она требуется, осуществляется с помощью стандартной директивы `Far`. Эти директивы должны указываться непосредственно после заголовков процедур и функций. Дальняя модель вызова подпрограмм может быть установлена также для всей программы в целом. Для этого перед началом программы должна указываться директива компиляции `{$F+}`. В этом случае указание директив `Far` не требуется.

Рассмотренный механизм не распространяется на стандартные процедуры и функции Паскаля.

Компиляция с расчетом на дальнюю модель вызова подпрограмм необходима также при построении программ с оверлейной структурой, при вызове одних программ из других с помощью оператора `Exec` и пр.

Дальняя модель вызова подпрограмм для ИСП устанавливается с помощью команды Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует отметить пункт Force far calls поля Code generation.

2.6.9. Контроль типов строк

С помощью директивы компиляции `{$V+}` для данной программы может быть установлен автоматический контроль типов строк, передаваемых в качестве фактических параметров подпрограммы.

В общем случае между типами фактических и формальных параметров-переменных должно быть полное совпадение. Исключения из этого правила таковы:

- имена типов фактических и формальных параметров-переменных могут быть различными, но только в том случае, если это различные имена для одного и того же типа;
- если тип формального параметра-переменной стандартный строковый, то совпадение типов для фактических параметров не обязательно.

Типы формальных и фактических параметров-значений считаются совместимыми в любых случаях.

С помощью директивы компиляции `{$V-}` автоматический контроль типов строк, передаваемых в качестве фактических параметров подпрограмм, можно отменить. В этом случае формальные и фактические параметры-переменные стандартных и нестандартных типов считаются совместимыми всегда.

Автоматический контроль типов строк, передаваемых в качестве фактических параметров подпрограмм, может быть установлена для ИСП в целом с помощью команды Options⇨Compiler... (<Alt+O&C>). При этом открывается диалоговое окно настройки компилятора Compiler Options, в котором следует отметить пункт Strict var-strings поля Syntax options. Однако на практике серьезного значения этот контроль не имеет, и его рекомендуется снимать.

2.7. Обработка кода завершения программы

Пусть некоторая программа разработчика завершает свою работу по оператору `Halt[(Code: Byte)]`, после чего осуществляется выход в среду, из которой программа запускалась. Необязательный параметр `Code` представляет собой выражение, значение которого является кодом завершения программы. Различные значения кода завершения могут предусматриваться программистом для той или иной аварийной ситуации. Значение кода воспринимается системной переменной `ERRORLEVEL`. Сразу после выполнения программы значение этой переменной может быть проанализировано средствами операционной системы и сделан соответствующий вывод о причине завершения работы программы.

Поскольку значение системной переменной `ERRORLEVEL` может быть получено непосредственно для предыдущей выполненной программы, то сделать это не удастся, если программа будет выполняться из ИСП. Действительно, после выполнения программы и выхода из ИСП в операционную систему можно получить код завершения работы только самой ИСП.

Таким образом, с целью анализа системной переменной `ERRORLEVEL` следует создать `EXE`-файл для исходной Паскаль-программы, а уж затем, после ее выполнения, можно будет получить реальную возможность проанализировать и код ее завершения.

Для того чтобы исполняемый файл Паскаль-программы был создан в том же каталоге, где хранится и ее исходный `PAS`-файл, предварительно нужно обеспечить следующее:

- сделать каталог с исходной программой текущим с помощью средств диалогового окна Change Directory, выбрав команду File⇨Change dir... (<Alt+F&C>);
- выполнить команду Compile⇨Target... (<Alt+C&T>) и в диалоговом окне Target выбрать реальный режим Real mode Application;
- выбрать команду Options⇨Directories... (Alt+O&D) и в диалоговом окне Directories очистить строку EXE&TPU directory;
- нажатием клавиш <Alt+F9> создать `EXE`-файл программы; имя `EXE`-файла будет совпадать с именем файла самой программы.

Наиболее удобно получать информацию о значении кода завершения программы в том случае, если и запуск самой программы, и анализ значения переменной `ERRORLEVEL` осуществляются с помощью специально созданного командного файла. Рассмотрим все необходимые действия на примере.

На базе подпрограммы `V.1.5` специально подготовим программу `V_01_05`, вычисляющую действительные корни квадратного уравнения $ax^2+bx+c=0$. Эта задача удобна тем, что при ее решении возможны две ошибочные ситуации:

- коэффициент *a* не должен быть равным нулю, иначе уравнение нельзя считать квадратным; в противном случае программа завершает свою работу по оператору `Halt(1)` с кодом завершения 1;
- дискриминант уравнения не должен быть отрицательным, иначе уравнение действительных корней не имеет; в противном случае программа завершает свою работу по оператору `Halt(2)` с кодом завершения 2;

```
{$B+,D+,E+,I+,L+,N+,Q+,R+,X-}
Program V_01_05;
   Procedure Square_Equation ( a,b,c: Real; Var x1,x2: Real;
                               Var z: Boolean);
      Var D: Real;
   Begin
      D := b*b - 4*a*c;
      D := Int(D*1E10)/1E10;
      z := D >= 0; If Not z Then Exit;
      x1 := ( -b+Sqrt(D) )/(2*a);
      x2 := -b/a - x1
   End;
   Var a,b,c,x1,x2: Real; z: Boolean;
Begin
   Write('Введите коэффициенты a,b,c: ');
   ReadLn(a,b,c);
   Square_Equation(a,b,c,x1,x2,z);
   If z Then WriteLn('x1=',x1:0:4,', x2=',x2:0:4)
        Else WriteLn('Действительные корни отсутствуют');
   ReadLn
End.
```

При нормальном завершении программы код ее завершения всегда равен нулю автоматически.

Сохраним программу `V_01_05` в файле `V_01_05.PAS`, откомпилируем ее и получим в текущем каталоге исполняемый файл `V_01_05.EXE`. Теперь создадим для работы с этой программой командный файл `V_01_05.BAT`. Сделать это можно в текстовом редакторе ИСП. Откроем новое пустое окно, выберем команду File⇨Save as, указав имя `V_01_05.BAT`, после чего введем и сохраним текст следующего командного файла:

```
@echo off
V_01_05.exe
if ERRORLEVEL 2 goto 2
if ERRORLEVEL 1 goto 1
echo Нормальное завершение программы
goto final
:2
echo Ошибка 2: Дискриминант уравнения отрицателен
goto final
:1
echo Ошибка 1: Введенные числа не могут быть коэффициентами
```

```
:final
pause
exit
```

Командный файл V_01_05.BAT позволяет собрать в один общий пакет целую последовательность команд операционной системы и тем самым делает возможным анализ кода завершения программы V_01_05.EXE сразу после ее выполнения. Рассмотрим смысл команд данного командного файла.

- Каждая команда командного файла перед своим выполнением выводится на экран. Но если применить команду echo off, то все последующие команды будут выполняться без предварительного вывода. Чтобы на экран не выводилась и сама команда echo off, ее используют с префиксом "@", т.е. в виде @echo off. Таким образом, экран не загромождается излишней выводимой информацией.
- По команде V_01_05.EXE выполняется сама программа решения квадратного уравнения.
- Команда вида "if <*условие*> <*команда*>" используется для проверки элемента <*условие*>, в зависимости от которого должна выполняться или не выполняться <*команда*>. Если <*условие*> истинно, то <*команда*> выполняется, в противном случае — пропускается.
- Одним из возможных элементов <*условие*> может быть "ERRORLEVEL <*число*>". Это <*условие*> считается истинным, если код завершения предыдущей выполненной программы оказался не меньше указанного элемента <*число*>. Именно поэтому во избежание ошибок проверки значений системной переменной ERRORLEVEL следует располагать в порядке убывания значений элемента <*число*>.
- Порядком выполнения команд можно управлять с помощью команд перехода. Команда перехода имеет вид "goto <*метка*>", где <*метка*> представляет собой последовательность произвольных символов (кроме пробела) в отдельной строке. Признаком элемента <*метка*> является двоеточие перед ней. В результате выполнения команды перехода дальнейшее выполнение командного файла продолжается со строки, следующей за элементом <*метка*>.
- С помощью команды "echo <*сообщение*>" на экран можно вывести любое нужное <*сообщение*>.
- Команда "if ERRORLEVEL 2 goto 2" читается так: "если код завершения больше или равен 2, то перейти на метку 2". Аналогично читается и команда "if ERRORLEVEL 1 goto 1". По меткам 1 и 2 в командном файле выполняется вывод на экран сообщений о соответствующих ошибках. Таким образом, если при выполнении командного файла не осуществится

переход ни по метке 1, ни по метке 2, следовательно, код ее завершения равен нулю, и можно вывести на экран сообщение о нормальном завершении программы, после чего перейти на метку final для окончания работы.

- Окончание работы командного файла заключается в том, что сначала необходимо сделать паузу в работе командного файла для того, чтобы можно было прочитать с экрана результаты вычисления корней квадратного уравнения либо сообщения об ошибках. Приостановка выполнения командного файла до нажатия любой клавиши реализуется с помощью команды pause.

Для того чтобы выполнить командный файл V_01_05.BAT, необходимо осуществить временный выход в среду операционной системы, выбрав команду File⇨DOS shell (<Alt+F&D>). В появившейся командной строке, указывающей на текущий каталог, следует напечатать команду V_01_05.BAT и нажать клавишу <Enter>, после чего по запросу программы ввести исходные данные, ознакомиться с ее сообщениями и нажать любую клавишу. На этом работа с программой заканчивается и можно вернуться в среду ИСП. Для этого, как известно, следует выполнить из командной строки операционной системы команду EXIT. Однако для большего удобства эта команда включена в состав командного файла, поэтому возврат в среду ИСП осуществляется автоматически после ознакомления с сообщениями программы и нажатия любой клавиши.