Лабораторная работа № 2

Подготовка программ к выполнению в среде *Free Pascal*

Цель работы: освоить средства, предоставляемые ИС Free Pascal, для компиляции и компоновки программы. Научиться по сообщениям компилятора определять место и причины появления синтаксических ошибок в программе.

1. Теоретическая часть

1.1. Средства ИС для компиляции программ

После того как текст программы набран в редакторе ИС или загружен в редактор с диска, можно приступать к подготовке программы для выполнения.

В традиционной схеме разработки программы такая подготовка осуществляется в два этапа: сначала программа переводится на машинный язык, в результате чего получается так называемый объектный модуль, а затем вместе с добавляемыми к ней стандартными подпрограммами и, возможно, другими объектными модулями компонуется в исполняемую программу.

Перевод программы на машинный язык называется *трансляцией*. Если трансляция выполняется с языка высокого уровня, то ее называют *компиляцией*, а если с машинно-ориентированного языка, то *ассемблированием*. Нас же будет интересовать только компиляция, поскольку Паскаль относится к языкам высокого уровня. Таким образом, если программа написана на Паскале, то при традиционном подходе для формирования исполняемой программы требуется два этапа: *компиляция* и *компоновка*.

Разбиение подготовки исполняемой программы на два этапа позволяет разрабатывать сложные программы по частям, компонуя законченные части в одно целое. Однако даже если программа не делится на части, для получения исполняемой программы всегда требуется выполнение двух упомянутых выше этапов, что существенно замедляет разработку небольших программ.

В ИС *Free Pascal* средства для получения объектных модулей отсутствуют, хотя включение в программу объектных модулей, созданных вне ИС, допускается. Вместо ориентации только на объектные модули в системе Free Pascal введено собственное понятие модуля, что сохраняет возможность разрабатывать сложные программы по частям. В отличие от традиционного подхода программа на языке *Free Pascal* всегда компилируется в готовую к выполнению программу. И лишь при компиляции модуля получается нечто похожее на объектный модуль, но в формате *PPU (Pascal Unit).* Таким образом, два этапа для получения исполняемой программы требуются лишь при добавлении в нее вновь

разработанного модуля. Однако и в данном случае ИС предоставляет удобные средства, позволяя выполнить компиляцию модулей, используемых в программе, и последующую компиляцию программы за одну команду.

Настройки средств ИС для создания исполняемой программы (компилятора, компоновщика и др.) находятся в меню *Options*. Меню *Options* состоит из двенадцати команд и имеет следующий вид:

Mode - открывает диалоговое окно *Switches Mode*, в котором программист может выбрать опции, управляющие работой интегрированного отладчика.

Compiler... - открывает диалоговое окно *Compiler Options*, в котором программист может выбрать опции, управляющие генерацией выходного кода, обнаружением ошибок на этапе выполнения, уровнем детальности отладочной информации.

Memory sizes... - открывает диалоговое окно *Memory Sizes*, в котором программист может установить для программы размеры оперативной памяти, отведенные под стек и под динамические переменные.

Linker... - открывает диалоговое окно *Linker*, в котором выполняются установки, управляющие работой редактора связей.

Debugger... - открывает диалоговое окно *Debugger*, в котором выполняются установки, управляющие работой интегрированного отладчика.

Directories... - открывает диалоговое окно *Directories*, в котором указываются папки, где находятся различные группы файлов, необходимые для выполнения компиляции.

Browser... - открывает диалоговое окно *Browser Options*, в котором программист может установить различные опции, управляющие работой браузера.

Tools... - открывает диалоговое окно *Tools*, в котором программист может добавить или удалить из меню *Tools* команды запуска программ, а также выполнить настройку этих программ.

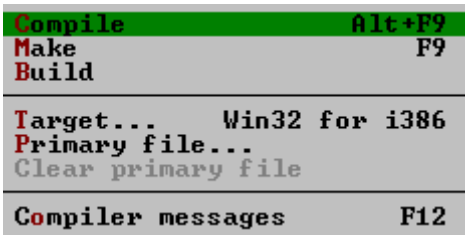
Environment - открывает окно, содержащее меню из шести команд, которые управляют внешним видом ИС и ее опциями, принятыми по умолчанию.

Open... - открывает диалоговое окно *Open Options*, в котором пользователь может восстановить установки ИС, сохраненные командой Save меню Options в файле с расширением .FP.

Save - сохраняет в файле установки опций, сделанные в диалоговых окнах меню *Search,* командой *Primary File* меню *Compile,* а также все установки, выполненные в меню *Options.*

***Save* as...** - открывает диалоговое окно *Save Options,* в котором указываются имена папки и файла, где будут сохранены текущие установки ИС.

Возможности ИС *Free Pascal* по компиляции программ лучше всего рассматривать, обратившись к подменю *Compile* главного меню. Вид этого меню показан на рисунке ниже.



Compile - компиляция файла, находящегося в активном окне редактирования.

Make - условная компиляция многомодульной программы с созданием *.EXE* файла. Если со времени последней компиляции были внесены изменения в некоторые модули, то при выполнении *Make* перекомпилируются только измененные и зависящие от них модули.

Build - безусловная компиляция многомодульной программы с созданием *.EXE* файла. Выполняется перекомпиляция всех модулей программы, независимо от того, вносились в них изменения со времени последней компиляции или нет.

Target... - выбор в диалоговом окне *Target* целевой платформы для приложения. Возможны варианты: *Real mode Application* (приложение реального режима), *Protected mode Application* (приложение защищенного режима), *Windows Application* (*Windows*-приложение).

Primary file... - открывает диалоговое окно для указания главного файла компилируемой программы для выполнения команд *Make* и *Build.*

Clear primary file - отмена указания главного компилируемого файла.

Compiler messages - открывает диалоговое окно, содержащее информацию о скомпилированном файле.

Из названий пунктов меню *Compile* можно видеть, что команды ИС *Compile*, *Make* и *Build* - это три возможных пути для компиляции программ, состоящих из нескольких файлов. Результат работы команд *Make* и *Build* зависит от порядка внесения изменений в тексты программ, компилируемых совместно, а также от состояния опции *Primary File* в этом меню.

1.1.1. Команда Compile

Команда запускает компиляцию текущего файла, находящегося в данный момент в окне редактирования *Edit*. При успешном завершении компиляции, что возможно лишь в случае, когда компилятор не обнаружит синтаксических ошибок, в нижней строке этого окна выдается сообщение:

“Compile successful : Press any key”

(Перевод: “Компиляция успешна: Нажмите любую клавишу".)

Если же в программе имеются синтаксические ошибки, то компилятор, обнаружив первую из них, прекращает свою работу. После этого *Free Паскаль* активизирует редактор, и его курсор устанавливается на то место в программе, где прервалась компиляция. Сообщение об ошибке на английском языке высвечивается в верхней строке окна редактора.

Так, в программе на рис. 2.1 пропущено имя “*y*” после запятой. В верхней строке окна редактора выведено довольно точное описание ошибки, которое может быть переведено как “*Ожидается имя*”. Курсор указывает на то место, где действительно должно находиться имя, а на самом деле стоит двоеточие.

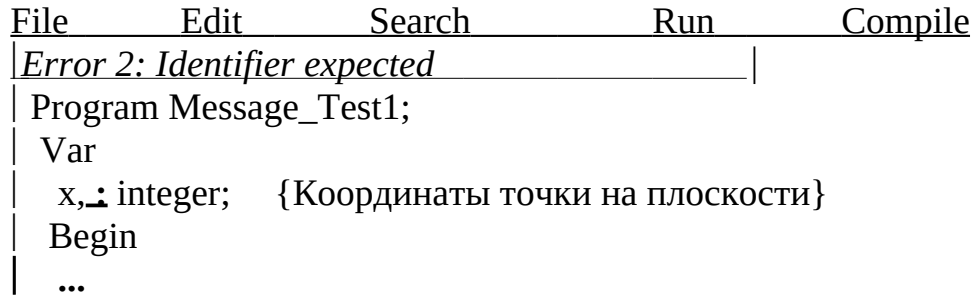


Рис. 2.1. Пример сообщения компилятора об ошибке в программе

Поскольку компилятор прекращает свою работу, как только обнаруживается ошибка, то после исправления указанной им ошибки может встретиться еще одна ошибка, расположенная ниже по тексту. Таким образом, для устранения всех синтаксических ошибок из текста программы может потребоваться несколько попыток ее компиляции. Поэтому удобнее подавать команду *Compile* не из меню, а с помощью горячих клавиш *Alt+F9.*

1.1.2. Команда избирательной компиляции *Make*

Если программа состоит из модулей и исходные тексты модулей доступны ИС, то следует перекомпилировать только те модули, в которые были внесены изменения, а прочие подключить уже в откомпилированном виде. Именно такой режим компиляции задает команда *Make*. При ее подаче ИС проверяет все файлы модулей, составляющие программу, и если эти файлы изменены после последней компиляции, то они будут перекомпилированы.

Эта команда используется при работе с трудоемкими сложными программами, на полную компиляцию которых затрачивается много времени.

1.1.3. Команда общей компиляции *Build*

Команда *Build* производит компиляцию всех доступных системе текстов, составляющих программу, независимо от того, были они скорректированы после компиляции или нет.

1.1.4. Команда назначения первого файла *Primary file*

Этот пункт значительно упрощает работу с множеством файлов. Используя его, можно указывать системе на главный файл в многофайловой программе. При этом любая команда компиляции будет обрабатывать

именно этот файл, а не тот, что загружен в текущий момент в редактор. Файл, который ранее находился в редакторе, после компиляции восстановится в нем.

Выбор файла для этого пункта осуществляется аналогично выбору файлов в пункте *File/Load.* Прежде чем закончить работу с одним программным проектом и приступить к другому, нужно удалить старое имя из пункта. Для этого выбирается описываемый пункт и нажимается клавиша пробела или комбинация клавиш *Ctrl-Y.* Имя *Primary File* стирается из окна выбора файла.

1.1.5. Команда Clear primary file

Выбор этого пункта отменяет сделанный ранее выбор *Primary file.* После этого команды *Compile* и *Build* будут использовать файл из активного окна редактирования.

1.1.6. Команда получения общего состояния Information

Выбрав этот пункт, можно открыть на экране новое окно, содержащее различную информацию о текущем состоянии программы: количество строк программы, размер кода, размер данных, размер стека и т.д.

1.2. Исправление синтаксических ошибок в программе по сообщениям компилятора

Чтобы исправить ошибку, указанную компилятором, необходимо определить действительное место ошибки в тексте программы, а затем причину, вызвавшую ошибку. Перечисленные действия зависят от вида ошибки.

1.2.1. Классификация сообщений компилятора

Ошибки, выявляемые компилятором, можно разделить на следующие группы:

1. Нарушение структуры конструкций языка, например пропуск обязательного зарезервированного слова внутри конструкции и т.п. Сообщения о таких ошибках, как правило, начинаются со слов типа “ожидается” или “неожиданный”.
2. Использование вне комментариев символов, не принадлежащих алфавиту языка, например русских букв, на что выдается одно из сообщений об ошибке: “недопустимый символ” или “синтаксическая ошибка”, а в комментариях к сообщению указывается на недопустимость данного символа.
3. Превышение ресурсных возможностей компилятора системы *Free Pascal*. В сообщениях о таких ошибках обязательно присутствует одно из слов: “слишком”, “превышен” и т.п.
4. Нарушение ограничений языка *Free Pascal*. Сообщения о таких ошибках часто напоминают сообщения об ошибках предыдущей группы, но в них упоминаются параметры не всей программы (например, количество переменных), а только конкретной конструкции (например, имени, строки и т.п.).

5. Нарушение ограничений и требований файловой системы ОС. Сообщения, относящиеся к этой группе, даются в терминах ОС.

В лабораторной работе рассматриваются только ошибки первой группы. Их выявление требует лишь элементарных представлений о работе компилятора и умений использовать описание грамматики языка *Free Pascal*. В то же время знание самого языка *Free Pascal* не является обязательным, поскольку действия по распознаванию ошибок первой группы носят чисто формальный характер.

1.2.2. Определение места ошибки в программе

Получив от компилятора сообщение об ошибке, следует учитывать, что курсор в этот момент указывает на место, где прервалась компиляция, но не на позицию текста, где допущена ошибка. Объясняется это тем, что компилятор, находясь в некоторой позиции текста, пытается найти так называемый *значащий* символ, к которым в программе на Паскале относятся либо так называемые терминальные символы (запятая, двоеточие и т.п.), либо символы заданного класса, например цифры, буквы и т.п., из которых строятся имена, числовые константы и т.д. При этом он пропускает все *незначащие* символы типа пробела, табуляции, перехода на начало новой строки и т.п. Если требуемый значащий символ пропущен, то компилятор в поисках его может уйти от первоначальной позиции довольно далеко. Наконец, найдя первый значащий символ (тот символ, который не является разделителем), компилятор сравнивает его с искомым или анализирует, к тому ли классу он относится, и, лишь обнаружив несоответствие, выдает сообщение об ошибке, отмечая последнюю проанализированную позицию текста курсором.

Например, во фрагменте программы, приведенном на рис. 2.2, пропущена точка с запятой после слова *Real*. Ошибка допущена в *15*-й позиции *4*-й строки. Однако после этого места первый значащий символ встречается только во второй позиции *12*-й строки. Поэтому курсор будет указывать на первую букву слова *Procedure*. В такой ситуации, чтобы найти действительное место ошибки, необходимо проследить в обратном направлении от указанной позиции путь, проделанный компилятором. Т.е. ошибку следует искать непосредственно перед первым значащим символом, который предшествует позиции, указываемой курсором в окне *Edit*.

Номер позиции

Номер строки	1 2 3 4 5 12345678901234567890123456789012345678901234567890
1	Program Message_Test2;
2	Var
3	x0, y0, { Начальные координаты тела ---}
4	x, y : Real; { Текущие координаты тела ------}
5	{----------}
6	{ }
7	{ Процедура определения текущих координат }

```
 8  │          {  тела, брошенного под углом к горизонту            }
 9  │          {                                                     }
10  │          {-------------------------------------------------------------}
11  │       Procedure Current_Body_Coord (x0, y0 : Real;
12  │            . . .
```

Рис. 2.2. Пример программы, содержащей синтаксическую ошибку

Так, в рассматриваемом примере сначала анализируется *1*-я позиция *12*-й строки. Затем необходимо перейти в конец *11*-й строки и, найдя здесь закрывающую скобку комментария и двигаясь к началу строки, найти парную ей открывающую скобку. Далее следует пропустить начальный пробел, перейти в конец *10*-й строки и т. д. Пропустив так все строки с *11*-й по *6*-ю, необходимо затем пропустить пустую *5*-ю строку и перейти в конец *4*-й. Пропустив комментарий в этой строке и предшествующий ему пробел, можно, наконец, найти позицию с номером *16*, после которой компилятор ожидал встретить точку с запятой.

Позиция, указываемая курсором, может оказаться и действительным местом ошибки, если в ней находится первый значащий символ текста программы либо ей предшествует значащий символ. Например, в ситуации, приведенной на рис. 1, перед курсором находится значащий символ (запятая), следовательно, пропущенное имя должно начинаться действительно с той позиции, на которую указывает курсор.

1.2.3. Определение причины ошибки

После того как место ошибки найдено, необходимо определить причину возникновения этой ошибки. В простейшем случае объяснение причины дается в сообщении компилятора, т.е. действительно пропущены указанный символ или слово, как это было в рассмотренных примерах. Однако сообщение компилятора может не соответствовать истинной причине ошибки, что демонстрируется на рис. 2.3.

```
File  Edit  Search  Run  Compile
│Error 36: BEGIN  expected                    │
│ Programm Message_Test3;
│  Begin
│  End.
```

Рис. 2.3. Пример неверного указания причины ошибки

В этом примере ошибка допущена в слове “*Programm*”, тогда как компилятор сообщает, что в отмеченной позиции ожидается слово "*BEGIN*". Такое расхождение объясняется наличием необязательных компонент в конструкциях языка *Free Pascal*.

Действительно, компилятор начинает синтаксический анализ текста программы с правила, определяющего понятие <*программа*>. Из синтаксической диаграммы для этого понятия следует, что программа может начинаться со слова *PROGRAM*. Не найдя это слово, поскольку в

рассматриваемом примере оно записано с двумя "*m*", компилятор обнаруживает, что *PROGRAM* может отсутствовать, и пропускает необязательный фрагмент правила. Далее таким же образом пропускается фрагмент правила, начинающийся со слова *USES*. Наконец, компилятор находит ссылку на диаграмму “блок”. Перейдя к правилу, определяющему понятие <блок>, компилятор последовательно пропускает его необязательные альтернативы, начинающиеся со слов *LABLE, CONST, TYPE, VAR, PROCEDURE и FUNCTION*. Далее он наталкивается на обязательное слово *BEGIN* и обращается к тексту программы. Не обнаружив, начиная с анализируемой позиции, слово *BEGIN*, компилятор и сообщает об этом.

Таким образом, получив от компилятора сообщение об ошибке и уточнив ее местонахождение, необходимо найти правило, описывающее конструкцию, в которой допущена ошибка, и выделить в нем фрагмент, относящийся к данной части текста программы. Так, в последнем примере на рис. 2.3 будет выделен фрагмент правила, начинающийся со слова *PROGRAM*. Если компонента конструкции языка, описываемая данным фрагментом правила, является обязательной, как это было в примерах на рис. 2.2 и 2.3, то ее отсутствие и является причиной ошибки. Если же компонента необязательна, но на самом деле присутствует и компилятор указывает именно на нее, тогда как сообщение об ошибке отсылает к другому фрагменту правила, то следует еще раз проверить правильность записи этой компоненты конструкции, используя правила грамматики. Так, если в примере на рис. 2.3 не обращать внимание на неправильное сообщение компилятора и сопоставить фрагмент правила грамматики, начинающийся со слова *PROGRAM*, с текстом программы, то можно сразу же обнаружить ошибку в слове “*Programm*”.

К рассматриваемой группе ошибок относятся также ошибки, вызывающие сообщение компилятора: “*Unexpected end of file*”, которое переводится как “(*Обнаружен) неожиданный конец файла*”.

Указанное сообщение может появляться либо, как показано на рис. 2.4, в случае, когда, не завершив набор текста программы, делается попытка ее откомпилировать, либо при компиляции программ, на первый взгляд, вполне завершенных (рис. 2.5).

```
File  Edit  Search  Run  Compile
Error 10: Unexpected end of file
 Program Message_Test4;
  Var
   x : Real;
```

Рис. 2.4. Пример компиляции незавершенной программы

```
File  Edit  Search  Run  Compile
Error 10: Unexpected end of file
 Program Message_Test5;
  Var
```

```
x : Real;
Begin
End
...
```

Рис. 2.5. Пример компиляции, на первый взгляд, завершенной программы

Однако в обоих случаях в конце текста программы пропущена обязательная компонента конструкции <*блок*>. Так, в программе на рис. 2.4 отсутствует последовательность “*Begin End*.”, возможно, с текстом между *Begin* и *End*, а в программе на рис. 2.5 пропущена точка после *End* программы. Компилятор, пытаясь найти ожидаемую компоненту, продвигается по программе, но обнаруживает конец ее текста и фиксирует ошибку.

Таким образом, при появлении сообщения о неожиданном обнаружении конца файла необходимо проверить наличие парных *Begin* и *End* программы, а также завершающей точки после последнего *End*. Если же все эти компоненты присутствуют, как показано на рис. 2.6, а компилятор выдает сообщение об *ошибке 10*, то следует проверить, не попадает ли какая-либо из обязательных компонент блока в комментарий.

```
File  Edit  Search  Run  Compile
Error 10: Unexpected end of file
Program Message_Test6;
 Var
  x : Real;
 Begin
 { Ввод координаты }
  Write (‘ x= ‘); Readln (x);
  { Проверка правильности ввода
  Writeln (‘     x= ‘,x);
 End.
```

Рис. 2.6. Пример компиляции программы с "незакрытым комментарием"

Так, на рис. 2.6 “не закрыт” комментарий в одной из строк, предшествующих *End* программы. Поэтому *End* и следующая за ним точка попали в комментарий и были пропущены компилятором вместе со всем текстом этого комментария.

2. Практическая часть

2.1. Порядок выполнения работы

1. Получить у преподавателя и изучить следующие материалы (или аналогичные):

- по средствам описания языков (файлы Средства описания языков.doc и Средства описания языков.txt);

- по аналитическому метаязыку для описания языка Паскаль (файл Аналитический язык.doc или Аналитический язык.txt);

- по графическому метаязыку для описания языка Паскаль (файл Графический язык.doc или Графический язык.txt);

- по примерам сообщений об ошибках, выдаваемых компилятором Паскаля (Сообщения об ошибках.doc и Сообщения об ошибках.txt).

2. Получить у преподавателя вариант (2 задания) программы для поиска в ней ошибок (файл Варианты.doc или Варианты.txt). Ошибки надо найти без запуска компилятора, а визуально с использованием документов, перечисленных в пункте 1 порядка выполнения работы.

3. Найти первую ошибку в каждом из заданий варианта, обосновать (с использованием аналитического языка) место и причину ошибки и занести результаты в отчет (на бумаге). Показать преподавателю.

4. Запустить среду *Free Pascal*.

5. Вырезать из файла Варианты.txt или Варианты.doc вариант (2 задания по каждому варианту), указанный преподавателем, и загрузить его, используя пункт основного меню *File/Open*, или набрать текст нужного варианта программы, используя пункт меню *File/New*.

6. Переписать в отчет текст программы, находящейся в окне *Edit*.

7. Перейти в пункт *Compile/Compile (Alt+F9)*. Нажать *Enter*. Компилируется программа, которая загружена в данный момент в окно редактирования. Если в программе нет ошибок, то после успешной компиляции на экране появятся информация о программе и строка: *Compile successful: Press any key*. В противном случае в верхней строке окна редактирования появляется сообщение об ошибке.

8. Зафиксировать это сообщение в отчете, перевести его, либо используя свои знания английского языка, либо найдя перевод по номеру ошибки в документе "Сообщения и коды ошибок, генерируемые компилятором" для *Free Pascal*. Определить действительное место ошибки в тексте программы (отметьте его в тексте программы), затем установить причину возникновения ошибки. Записать свои пояснения в отчет.

9. Внести исправления в программу. Убедиться в том, что ошибка исправлена, повторив компиляцию.

10. Исправить все ошибки в программе, последовательно выполняя пп. 5-7.

11. Перейти в пункт *Compile/Information*. После выполнения этой команды на экране появляется окно, содержащее подробную информацию о программе. Записать результаты выполнения этой команды в отчет. Для удаления окна нажать любую клавишу.

12. Перейти в пункт *Compile/Compile.* Нажать клавишу *Enter*. Эта команда создает программу, готовую к запуску на выполнение и называемую загрузочным модулем. Аналогичные действия осуществляет любая из команд *Make* или *Build*. В соответствии с п. *10* загрузочный модуль будет создан на диске.

13. Убедиться, что имя вашего файла с расширением *EXE* есть в каталоге. В отчете сделать вывод о том, как формируется имя файла для загрузочного модуля.

14. Перейти в пункт *File/Quit.* Этот пункт осуществляет выход из среды *Free Pascal*. Эти же действия можно выполнить из режима редактирования с помощью «горячих клавиш» *Alt-X*.

2.2. Требования к содержанию отчета

Отчет о лабораторной работе должен включать:

1. Конспект теоретической части.
2. Текст программы с указанием места ошибки (первой) для каждого из вариантов.
3. Образцы экранов с сообщениями компилятора об ошибках.
4. Объяснение причины и места ошибки для каждого из вариантов:
   а) истинной ошибки;
   б) ошибки, на которую указывает компилятор.

Объяснение необходимо привести с использованием аналитического и графического языков описания синтаксиса *Free Pascal*.

2.3. Контрольные вопросы

1. В чем заключается подготовка программы к выполнению?
2. Какие способы компиляции предусмотрены в ИС *Free Pascal* и в чем их различия?
3. Куда помещается результат компиляции?
4. Как создать выполнимую программу на диске и убедиться в ее создании?
5. Что может помешать записи результата компиляции на диск?
6. Какое сообщение выдает компилятор в случае успешного завершения компиляции?
7. Как получить информацию о текущем состоянии программы?
8. Как реагирует компилятор на обнаруженную ошибку?
9. Как определить действительное место ошибки в программе, получив сообщение компилятора об ошибке (см. конец п.1.2.2)?
10. Как формально (по синтаксической диаграмме) можно (нужно) находить ошибки в тексте программы?
11. Чем в каждом из заданий (см. п. 2 по порядку выполнения) *Вариант 1* отличается от *Варианта 2*? В каком из двух случаев (*Вариант 1* или *Вариант 2*) и почему удается при ошибке в программе зайти в диаграмму и при движении по ней заметить ошибку (соответственно и компилятор сможет точно определить место и причину ошибки), в каком случае не удается при ошибке в программе зайти в диаграмму, например по причине того, что пропущено ключевое слово, с которого может начинаться раздел или секция раздела (соответственно и компилятор не сможет точно определить место и причину ошибки)?

12. В каком случае курсор после выдачи сообщения компилятором указывает на место, где действительно допущена ошибка (см. конец п.1.2.2)?

13. Чем отличаются ситуации, приведенные на рис. 2.1 и 2.2?

14. В каком случае сообщение компилятора типа “ожидается...” правильно объясняет причину ошибки и почему (см. начало п.1.2.3)?

15. Почему сообщение компилятора типа “ожидается...” может не соответствовать действительной причине ошибки (см. начало п.1.2.3, пример на рис.2.3)?

16. Каковы возможные причины выдачи сообщения типа “недопустимый символ” или “синтаксическая ошибка” и на какую позицию при этом указывает курсор (см. документ с описанием графического метаязыка или аналитического метаязыка)?

17. Каковы возможные причины выдачи сообщения “Error 10: Unexpected end of file” и на какую позицию при этом указывает курсор (см. п.1.2.3, примеры на рис. 2.4, 2.5, 2.6)?

18. Как исправить ошибку в программе примера на рис. 2.5 (см. п.1.2.3)?

19. Как исправить ошибку в программе примера на рис. 2.6 (см. п.1.2.3)?

20. Что сообщит компилятор, если в программе примера на рис. 2.6 с позиции, указываемой курсором, добавить недостающие End и снова выполнить компиляцию?