Федеральное государственное бюджетное образовательное учреждение высшего
профессионального образования
«Вологодский государственный университет»
Кафедра автоматики и вычислительной техники

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания
к выполнению лабораторной работы

ОТЛАДКА ПРОГРАММ

Вологда 2014

1. Цель работы

Цель работы*:* изучение возможностей предоставляемых встроенными средствами Free Pascal для обнаружения и исправления ошибок и основных методов отладки программ.

2. Основные теоретические положения

2.1. Основные понятия

Отладка – это процесс поиска и исправления ошибок в программе, препятствующих корректной работе программы. Отладка программы является одним из наиболее важных и трудоемких этапов разработки. Трудоемкость и эффективность отладки напрямую зависит от способа отладки и от средств языка программирования. Существует ряд простых вещей, которые могут облегчить отладку Вашей программы. Для большинства случаев справедливо то, что не следует располагать на одной строке программы более одного оператора. Так как выполнение программы в процессе отладки происходит строка за строкой, это требование будет обеспечивать выполнение не более одного оператора каждый раз. В то же время допускаются и случаи, когда можно разместить на строке несколько операторов. Если есть список операторов, которые нужно выполнить, но которые фактически не имеют отношения к отладке, то Вы можете произвольно располагать их на одной или двух строках так, чтобы их можно было быстрее пройти при пошаговом выполнении. В конце концов, лучшей отладкой является профилактическая отладка. Хорошо разработанная, ясно написанная программа может иметь не только немного ошибок, но и будет облегчать для Вас трассировку и фиксацию местоположения этих ошибок. Существует несколько основных положений, о которых следует помнить при составлении программы: программируйте с постепенным наращиванием. При возможности кодируйте и отлаживайте программу небольшими секциями. Прорабатывайте каждую секцию до конца прежде чем переходить к следующей; разбивайте программу на части: модули, процедуры, функции. Избегайте построения функций, размер которых больше 25-30 строк, в противном случае разбивайте их на несколько меньших по размеру функций; старайтесь передавать информацию только через параметры, вместо использования глобальных переменных внутри процедур и функций. Это поможет Вам избежать побочных явлений и облегчит отладку программы, так как Вы сможете легко прослеживать всю информацию, входящую и выходящую из заданной процедуры или функции; не торопитесь. Сосредоточьте действия на том, чтобы программа работала правильно, прежде чем предпринимать шаги по ускорению ее работы.

2.2. Классификация ошибок

Существует различные типы ошибок:

1) Ошибки этапа **компиляции**: ошибки этапа компиляции или синтаксические ошибки происходят, когда исходный код нарушает правила синтаксиса языка;

2) Ошибки этапа компоновки: возникают при обращении к компонентам системы;

3) Ошибки этапа выполнения: ошибки этапа выполнения или семантические ошибки происходят, когда после компиляции полной программы, при ее выполнении делается что-то недопустимое, хотя программа содержит допустимые операторы. Например, программа может пытаться выполнить деление на ноль или открыть для ввода несуществующий файл.

4) Логические ошибки – это ошибки проектирования и реализации программы. То есть, все операторы допустимы, но делают не то, что предполагалось. Эти ошибки часто трудно отследить, поскольку IDE не может найти их автоматически, как синтаксические и семантические ошибки. Логические ошибки приводят к некорректному или непредвиденному значению переменных, неправильному виду графических изображений или невыполнению кода, когда это ожидается.

2.3 Методы отладки

Иногда причина непредвиденных действий программы достаточно очевидна. Но есть трудноуловимые ошибки и ошибки, вызываемые взаимодействием различных частей программы. В этих случаях необходимо интерактивное выполнение программы, во время которого производится наблюдение за значениями определенных переменных или выражений. Может потребоваться остановка программы при достижении определенного места так, чтобы просмотреть, как она проработала этот кусок. Или хочется прервать выполнение и изменить значения некоторых переменных, изменить определенный режим или проследить за реакцией программы. Желательно сделать это в режиме, когда возможно быстрое редактирование, перекомпилирование и повторное выполнение программы. Такое управляемое выполнение – ключевой элемент отладки.

Для облегчения локализации ошибок и отладки программы используют

- отладочный вывод и компиляцию ее с отладочной информацией.
- пошаговое выполнение и контроль за содержимым переменных или ячеек памяти;
- программные инструменты – отладчики.

2.4 Отладчики

Основные средства отладки достаточно консервативны. Еще на ЭВМ первого поколения программисты набирали на пульте адрес команды, на которой автоматическое выполнение программы прекращалось, и появлялась возможность просмотреть содержимое машинных регистров и ячеек оперативной памяти. Вторым магическим средством был перевод компьютера в пошаговый режим работы, в котором очередное нажатие кнопки <ПУСК> приводило к выполнению следующей команды программы. На некоторых ЭВМ была предусмотрена возможность останова работы программы в момент записи данных в ячейку с указанным адресом. Сегодня кодами машинных команд пользуются очень редкие профессионалы, да и те предпочитают более продвинутые средства вроде услуг ассемблера. Большинство пользователей работает с алгоритмическими языками высокого уровня. Однако старинные средства отладки сохранились в несколько модернизированном виде. Так называемые *точки останова* (breakpoints) теперь задаются не по адресам машинных команд, а на входе во фрагмент исполняемой программы, соответствующий началу указанной строки исходного текста. Вместо просмотра содержимого ячеек оперативной памяти появилась возможность вывода на дисплей значений переменных и даже формул, заданных в обычном для программиста виде. Контролируемые таким образом выражения в англоязычной литературе обозначают терминами *watches*. Вместо одноактного режима выполнения программы по одной машинной команде появилась возможность "простукивать" исходную программу по одной строке. Конечно, каждое из этих новшеств обрастает дополнительными деталями, но, в целом, процесс развития средств отладки сохранился на первобытном уровне.

Отладчики бывают ***независимые*** и ***встроенные*** в среду программирования. Интегрированный отладчик работает очень просто. Ему не требуются специальные инструкции в Вашем коде, он не увеличивает размер Вашего .EXE файла и не требует перекомпиляции для создания отдельного .EXE файла после окончания отладки. Если Ваша программа разделена на ряд модулей, исходный код каждого из них автоматически загружается в редактор при трассировке. Если Вы используете оверлеи, отладчик автоматически обрабатывает их внутри IDE, которая выполняет переключения между компилятором, редактором и отладчиком.

Возможности :

1) пошаговое выполнение программы с заходом в процедуры и без захода;
2) исполнение до курсора;
3) установка контрольных точек, в том числе условных и со счетчиком;

4) наблюдение за значениями переменных;
5) вычисление выражений и изменение значений переменных во время приостанова программы;
6) наблюдение за состоянием стека вызванных подпрограмм;
7) динамическое управление отладочными средствами.

2.5. Средства отладки среды Free Pascal

Интегрированный отладчик **Free Pascal** имеет все описанные выше возможности и даже более того. Он представляет собой встроенную часть интегрированной усовершенствованной среды **Free** Pascal (IDE). Основные средства отладки в среде FP IDE сосредоточены в меню **Run** и **Debug** (рис. 2.1). Команды меню Run обеспечивают управление способом выполнения программы; меню Debug – управление контрольными точками, наблюдение за значениями переменных и вызовами подпрограмм, вычисление выражений и переменных.

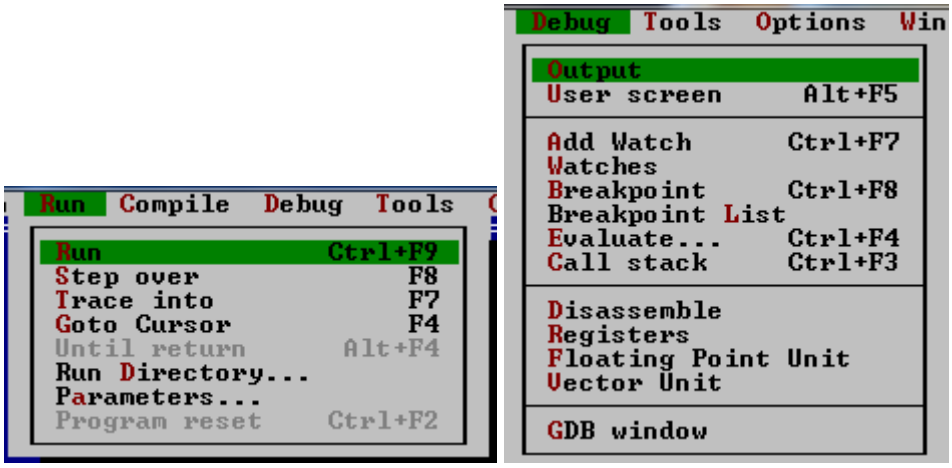

Рис. 2.1. Отладочные команды системы в среде IDE Free Pascal

2.4. Использование точек останова

Точки останова прерывают выполнение программы, когда достигается одна из установленных точек останова. В этот момент управление передается IDE, после чего выполнение программы может быть продолжено.

Для набора точки останова в текущей строке исходной программы можно выполнить команду Debug -> Breakpoint или нажать комбинацию клавиш <Ctrl>+ +<F8>. Окно, в котором хранится информация о точке останова, приведено на рис. 2.4.1

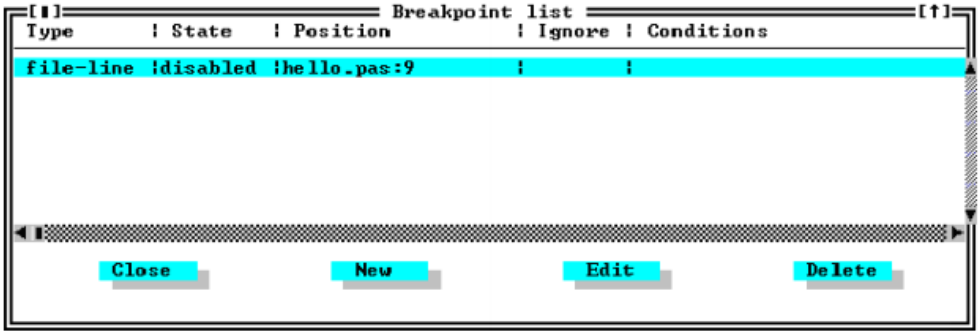

Рис. 2.4.1Окно со списком точек останова

Список точек останова можно увидеть, выполнив команду Debug->Break-point List. В этом окне можно инициировать следующие операции:
New — отобразить свойства точки останова при наборе новой точки;
Edit — показать свойства точки останова для изменения ее свойств;
Delete — удалить точку останова.

При наборе новой точки или изменении свойств ранее установленной точки используются поля окна, приведенного на рис. 2.4.2.

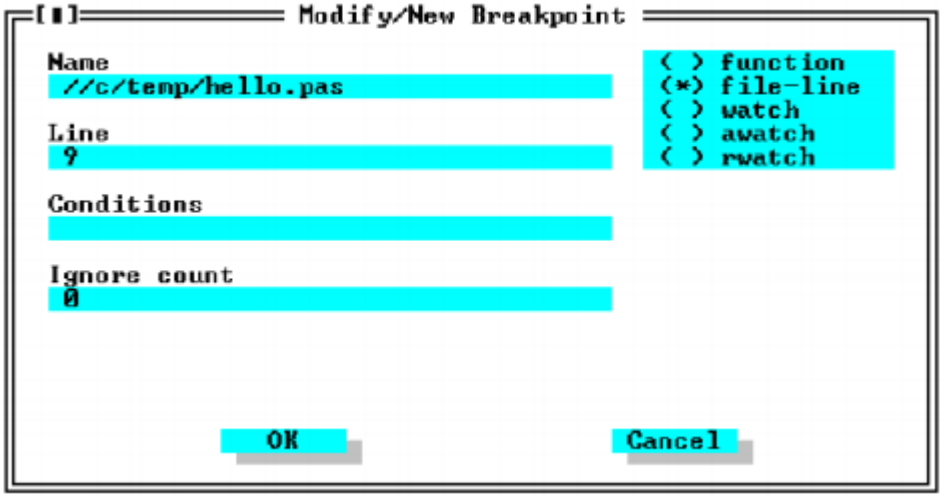

Рис. 2.4.2. Окно для набора новых точек останова и модификации старых

Вы можете установить тип точки, выбрав из списка типов нужную строку:

function — точка останова в функции. Выполнение программы останавливается при вызове функции с указанным именем;

file-line — точка останова в строке исходной программы. Программа останавливается при попадании на указанную строку в файле с заданным именем;

watch — точка останова выражения. Вы можете ввести выражение, и программа остановится при изменении значения этого выражения;

awatch (access watch) — точка останова выражения. Можно ввести выражение, которое является ссылкой на ячейку памяти, и программа остановится при любом обращении к этой ячейке (при записи или чтении);

rwatch (read watch) — точка останова выражения. Останов происходит при чтении из указанной ячейки.

В окне Modify/New Breakpoint есть также поля:

Name — имя функции или файла, в котором должен произойти останов;

Line — номер строки, в которой должен произойти останов (только для точек останова типа file-line);

Conditions — здесь указывается условие, при вычислении которого для останова получится значение True. Формула условия должна быть набрана прописными буквами;

Ignore count — количество прохождений через заданную точку без останова. После достижения указанного значения происходит останов.

2.5. Контролируемые выражения

Контролируемые выражения могут использоваться, если программа компилируется с отладочной информацией. Их значения вычисляются IDE и отображаются в отдельном окне. Когда происходит останов программы (например, в точке останова), текущие значения контролируемых выражений можно увидеть в этом окне.

Набор новых контролируемых выражений производится по команде Debug -> Add Watch или с помощью нажатия комбинации клавиш < Ctrl>+<F7>. При этом появляется диалоговое окно, в котором видны и все ранее набранные выражения. Так как IDE использует GDB, то формулы должны набираться прописными буквами.

Список всех контролируемых выражений и их текущие значения отображаются в окне, открывающемся по команде Debug ->Watches. По нажатию клавиши <Enter> или <Пробел> отображается текущее значение подсвеченного выражения.

2.6. Стек обращений

Стек обращений позволяет проследить динамику выполнения программы. В нем в обратном порядке отображены названия вызванных и еще не завершенных к моменту останова процедур. Для просмотра стека надо выполнить команду Debug ->Call stack. Здесь будут показаны имена и адреса всех активных процедур. Если им переданы параметры, то их значения тоже отображаются. При нажатии клавиши Пробел> на выделенной строке этого окна в поле редактора подсвечивается соответствующая строка исходного файла.

3. Практическая работа

3.1. Установка программы (если ее нет на ПК, можно на съемных носителях)**:**

Запустите файл **fpc-2.6.0.i386-win32** из папки лабораторной работы. Устанавливаем и запускаем программу Free Pascal IDE.

3.2. Ознакомление со средствами отладки среды Free pascal

3.2.1. Пошаговый прогон: какая разница между F4, F7 и F8?

Основной смысл использования встроенного отладчика состоит в управляемом выполнении. Отслеживая выполнение каждой инструкции, Вы можете легко определить, какая часть Вашей программы вызывает проблемы. В отладчике предусмотрено пять основных механизмов управления выполнением программы, которые позволяют вам:

1) Выполнять инструкции по шагам
2) Трассировать инструкции
3) Выполнять программу до заданной точки
4) Находить определенную точку
5) Выполнять сброс программы

1) Пошаговое выполнение программы

Само по себе выполнение программы по шагам может быть недостаточно полезным, разве что поможет найти то место, где что-то происходит совершенно неверно. Но управляемое выполнение дает Вам возможность проверять состояние программы и ее данных, например, отслеживать вывод программы и ее переменные.

Пошаговое выполнение – это простейший способ выполнения программы по элементарным фрагментам. Наименьшим выполняемым элементом (элементарным фрагментом, шагом) в отладчике является **строка**, а не оператор. (Все выполнение в отладчике, включая выполнение по шагам, трассировку и останов, основывается на строках.) Строка, которую отладчик выполнит на следующем шаге (строка выполнения) выводится цветом, отличным от обычного (курсор выполнения). Пошаговое выполнение инициируется выбором команды **Run → Step Over** или нажатием клавиши **F8.** После выполнения указанной строки, включая любые вызываемые на ней процедуры или функции, курсор выполнения переходит на следующую строку. Если в одной строке программы содержится несколько операторов Паскаля, эти операторы не могут быть отлажены индивидуально. С другой стороны, с целью отладки оператор можно разбить на несколько строк, каждая из которых будет выполняться за один шаг.

Например, отлаживаем программу *StepTest*:

```
program StepTest;
   function Negate(X: Integer): Integer;
  begin
    Negate := -X;
  end;

 var
  I: Integer;
begin
  for I := 1 to 10 do Writeln(Negate(I));
end.
```

После нажатия клавиши F8 выполняется оператор begin и строка выполнения перемещается вниз до оператора for на следующей строке. Второе нажатие F8 вызывает выполнение всего цикла for; на экран пользователя выводятся числа от –1 до –10, а строка выполнения перемещается к end.

Обратите внимание, что а) хотя функция Negate и вызывается 10 раз, строка выполнения никогда на нее не перемещается (режим **Step Over** позволяет отладчику не показывать детали любых вызовов для отдельной строки); б) цикл for выполняется сразу весь, поэтому Вы не сможете видеть изменения в ходе выполнения цикла. Чтобы видеть подробности выполнения цикла, внесем в пример следующее простое изменение:

```
begin
  for I := 1 to 10 do
    WriteLn(Negate(I));
end.
```

Новая программа будет в точности эквивалентна предыдущей версии (оператор Паскаля может занимать несколько строк). Но оператор WriteLn теперь находится на отдельной строке и отладчик обрабатывает его отдельно: при нажатии F8 строка выполнения будет 10 раз возвращаться на WriteLn.

2) Трассировка программы

Трассировка программы инициируется выбором команды **Run → Trace Into** или нажатием клавиши **F7**. Различие между командами Trace Into (F7) и Step Over (F8) в том, что при использовании F7 осуществляется трассировка внутри процедур и функций, в то время как использование F8 приведет к обходу вызовов подпрограмм. Эти команды имеют особое значение при выполнении оператора begin основной программы, если программа использует модули, имеющие раздел инициализации. В этом случае, использование F7 приведет к трассировке раздела инициализации каждого модуля, что позволяет увидеть, что инициализируется в каждом модуле. При использовании F8 эти разделы не будут трассироваться, и курсор выполнения переходит на следующую строку после begin.

Выполните трассировку программы *StepTest*. Убедитесь, что нажатие клавиши F7 трассирует вызов функции Negate – строка выполнения перемещается на оператор begin в блоке функции. Обратите внимание на различие в трассировке в зависимости от способа записи цикла основной программы (в одну или две строки). При трассировке формат программы влияет на поведение строки выполнения меньше, чем при пошаговом выполнении. Если код сформатирован как в первоначальном варианте приведенного выше примера, то трассировка оператора for приводит к выполнению 10 раз функции Negate. Если разбить оператор for на две строки, то трассировка оператора end функции возвращает строку выполнения: ту строку основной программы, которая будет выполняться следующей. Первые девять раз это снова будет вызов функции. В десятый раз строка выполнения перемещается на оператор end программы.

Если в программе используются объекты, отладчик ведет себя аналогично своему поведению в случае обычных процедур/функций. Пошаговое выполнение метода интерпретирует метод как один шаг, возвращая управление к отладчику после того как метод завершает выполнение. Трассировка метода загружает и выводит на экран код метода и трассирует его операторы.

3) Выполнение программы до заданной точки

Чтобы быстро добраться до той точки, с которой планируется начать выполнение по шагам, можно воспользоваться командой **Run → Go to Cursor** или нажатием клавиши **F4**, установив предварительно курсор на определенную строку в отлаживаемой программе. Команда Go to Cursor также позволяет выполнять большой фрагмент программы, проходить циклы или другие утомительные участки программы. Применить команду можно, как в начале сеанса отладки, так и после трассировки или пошагового выполнения части программы.

Существуют три случая, когда команда Go to Cursor (F4) не будет выполнять программу до отмеченной курсором строки:

– если курсор установлен между двумя выполняемыми строками; например, на пустой строке или строке с комментариями. В этом случае программа будет выполняться до следующей строки с оператором, который может быть выполнен;

– если курсор расположен вне процедурного блока, например, на операторе объявления переменной или операторе program: отладчик выведет сообщение "no code generated for this line" (для этой строки код не генерируется);

– если курсор расположен на строке, которая никогда не выполняется. Например, строка располагается выше курсора выполнения (предполагается, что вы находитесь не в цикле) или строка является частью else - условного оператора, когда выражение if имеет

значение true. В этом случае программа будет выполняться до конца (как по команде Run/Run (Ctrl-F9)) или до точки прерывания.

Внимание: С использованием этой команды связана одна особенность: если Вы хотите, чтобы программа выполнилась до определенной строки, и устанавливаете курсор внутри модуля (Unit), то этого не произойдет, Вы просто получите ошибку Cannot Run a Unit , и этим все закончится, т.к. IDE понимает это действие, как приказ запустить модуль, чего делать нельзя. Требуется объяснить IDE, чего Вы от нее хотите примерно так: «Запусти основную программу, и только потом выполни все, до текущего положения курсора». Для этого надо зайти в меню **Compile → Primary File**, указать системе основной файл (НЕ модуль) Вашего приложения, и только после этого установить курсор внутрь модуля, и нажать **F4**… Казалось бы, что поменялось? А вот что: теперь IDE точно знает - основным файлом приложения является тот, который был установлен, как **Primary File**, следовательно, совершенно нет необходимости запускать МОДУЛЬ, достаточно запустить основной файл, и остановиться тогда, когда выполнение дойдет до нужной строки в модуле.

4) Точки останова

Точка останова – это обозначенная в коде программы позиция, в которой желательно прекратить выполнение программы и вернуть управление отладчику. После установки точек прерывания, выполнение программы инициируют командой Run (Ctrl-F9). Можно также использовать команды Trace Into, Step Over или Go to Cursor (F7, F8 или F4). Когда достигнута одна из установленных точек останова, программа прерывается, управление передается IDE, после чего выполнение программы может быть продолжено. Основное отличие от команды GoTo Cursor состоит в том, что точек останова можно задать несколько (до 16 активных точек прерывания) и можно задать точки останова, которые будут срабатывать не при каждом их достижении. Точка прерывания не высвечивается, когда на ней находится курсор выполнения. Следует отметить, что точки прерывания существуют только во время сеанса отладки; они не сохраняются в файле .EXE, если программа компилируется на диск.

Чтобы *задать* точку прерывания, устанавливаем курсор на нужную строку (это не должны быть пустая строка, комментарии, директивы компиляции; объявления констант, типов, меток, переменных; заголовки программы, модуля, процедуры или функции) и выполняем команду **Debug → Breakpoint (Ctrl-F8)**. При этом текущая строка окрасится в красный цвет. Повторное нажатие той же комбинации клавиш *отменит* точку останова в текущей строке, и ее цвет станет обычным.

Список всех **точек останова**, установленных к текущему моменту, можно увидеть в окне, которое появляется по команде **Breakpoint List** из меню **Debug** (рис. 2.4.1). Для каждой точки указаны Type, State, Position, Path. В этом окне можно инициировать следующие операции (попробуйте поэкспериментировать самостоятельно):

New – задать свойства новой точки останова;

Edit – показать свойства точки останова для их изменения;

Toggle (Переключить) – нажатие этой кнопки меняет статус точки останова на противоположный с Enabled на Disabled и наоборот. Это позволяет, не удаляя точку из списка, сделать ее пассивной (Disabled) – теперь останов на ней не произойдет;

Delete – удалить точку останова.

При **наборе новой точки или изменении** свойств ранее установленной точки используются поля окна Modify/New Breakpoint, приведенного на рис. 2.4.2.:

Name – имя функции или файла, в котором должен произойти останов;

Line – номер строки, в которой должен произойти останов (только для точек останова типа file-line);

Conditions – здесь указывается условие, при вычислении которого для останова получится значение True. Формула условия должна быть набрана прописными буквами;

Ignore count – количество прохождений через заданную точку без останова. После достижения указанного значения происходит останов;

Type – установка типа точки. выбирается из списка типов:

– function – точка останова в функции. Выполнение программы останавливается при вызове функции с указанным именем;

– file-line – точка останова в строке исходной программы. Программа останавливается при попадании на указанную строку в файле с заданным именем;

– watch – точка останова выражения. Вы можете ввести выражение, и программа остановится при изменении значения этого выражения;

– awatch (access watch) – точка останова выражения. Можно ввести выражение, которое является ссылкой на ячейку памяти, и программа остановится при любом обращении к этой ячейке (при записи или чтении);

– rwatch (read watch) – точка останова выражения. Останов происходит при чтении из указанной ячейки

– address – .

Представьте себе ситуацию, когда вы установили точку останова на внутренней строке цикла, который должен выполняться сотни раз. Не устать бы, заставляя программу продолжаться после каждого останова. Ошибки в циклах чаще всего возникают либо при первом, либо при последнем прохождении цикла. Дополнительно можно проконтролировать условие досрочного завершения цикла.

5) Выполнять сброс программы. Прерывание

Вы можете также прерваться в любой точке Вашей программы, нажав клавишу **Ctrl-Break.** Произойдет остановка на следующей строке исходной программы, как если бы в этой строке была установлена точка прерывания.

3.2.2. Запустите файл из папки лабораторная работа *primer 1* с кодом задачи:

Дано 100 целых чисел от 1 до 50. Определить сколько среди них чисел Фибоначчи и сколько чисел, первая значищая цифра в дисятичной записи которых является 1 или 2.

Начнем отладку программы с возможности прерывания автоматического режима выполнения программы. По команде Run -> Run (клавишный аналог — <Ctrl>+<F9>) как на рисунке 3.2.2. программа начнет выполняться в автоматическом режиме, и при достаточно объемной выдаче результатов на экран мы ничего путного увидеть не успеем. Конечно, имеется возможность подменить стандартный вывод записью в файл, запуская программу с параметром командной строки:

>prog.exe >1.txt

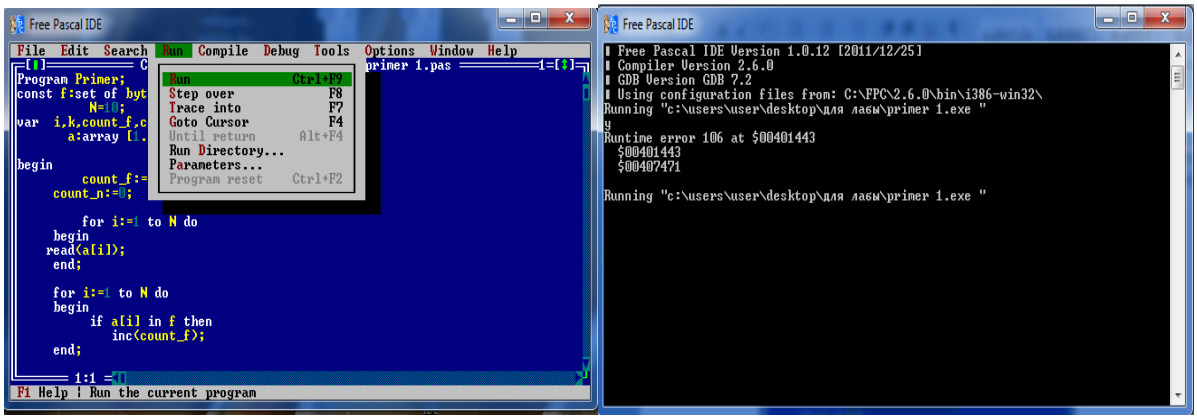

Рис. 3.2.2. Выполнение программы в автоматическом режиме.

В этом случае мы сможем прочитать из файла с именем 1.txt все результаты, выданные программой, но обнаружить причины неправильной работы программы таким способом мы не сможем. Желательно остановить работу программы в подозрительной точке и проанализировать значения наиболее важных переменных. А потом продолжить автоматическую работу до следующей точки останова. Для назначения *первой* точки останова можно воспользоваться командой Goto Cursor (клавишный аналог — <F4>) из меню Run. 3.2.3. Предварительно нужно установить курсор в поле редактора в ту строку, перед выполнением которой мы хотим прервать работу программы.

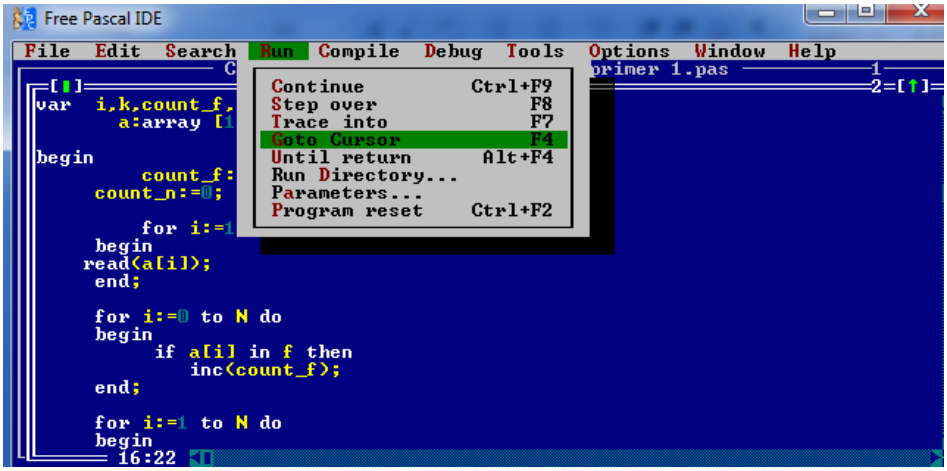

Рис. 3.2.3. Команда Goto Cursor.

В результате увидим ошибку обращение к нулевому элементу массива

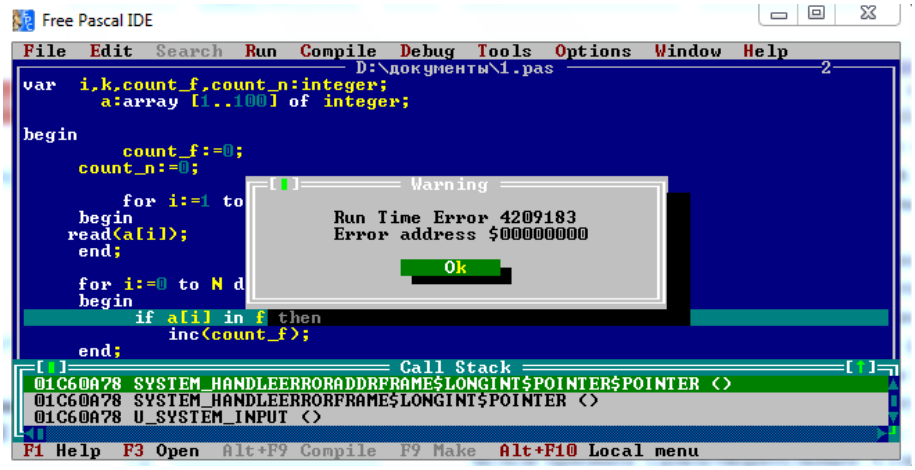

Рис. 3.2.4.Ошибка найденная командой Goto Cursor.

После выхода на первый останов и просмотра нужных данных у нас появляются следующие возможности — продолжить работу в автоматическом режиме или перейти на пошаговое выполнение программы. В первом случае можно воспользоваться командой Continue из меню Run (клавишный аналог — <Ctrl>+<F9>) (Рис. 3.2.5.), но тогда дальнейшее выполнение программы будет до конца только автоматическим. А можно опять перевести курсор на строку следующего останова и снова воспользоваться командой Goto Cursor. Конечно, такой выбор следующей точки останова нельзя признать удовлетворительным. Существует и более разумный подход, но о нем — далее.

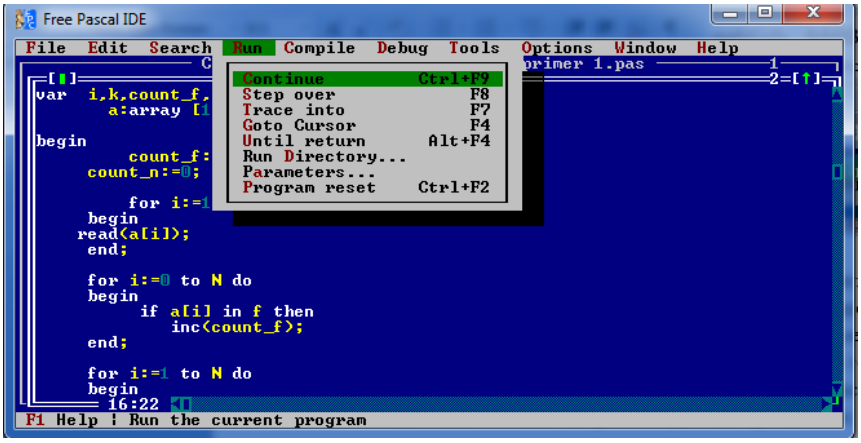

Рис. 3.2.5. Команда Continue.

После выхода в точку останова можно продолжить работу программы в пошаговом режиме. Для этой цели предназначены клавишные команды <F8> (аналог команды Step over) и <F7> (аналог команды Trace into). Каждое нажатие одной из этих клавиш приводит к выполнению очередной строки в исходной программе. Если выполняемая строка содержала обращение к процедуре, то по нажатию клавиши <F8> процедура будет выполнена в автоматическом режиме, а после возврата из процедуры сохранится пошаговый режим выполнения программы. Такой процесс целесообразен, когда вы уверены в правильности работы процедуры. Если у вас появились сомнения, то и процедуру разумно "простучать" в пошаговом режиме. В этом случае надо воспользоваться клавишей <F7>. Наконец, существует ситуация, когда, выполнив несколько шагов в процедуре, вам захотелось заставить ее доработать в автоматическом режиме. На этот случай вам предлагается команда Until return (клавишный аналог — <Alt>+<F4>).

Более рациональный способ задания точек останова заключается в том, что вы устанавливаете их заранее или добавляете при выходе в ту или иную точку программы. Для того чтобы сделать текущую строку точкой останова, достаточно выполнить клавишную команду <Ctrl>+<F8> (аналог команды Breakpoint из меню Debug). При этом текущая строка окрасится в красный цвет. Повторное нажатие той же комбинации клавиш отменит точку останова в текущей строке, и ее цвет станет нормальным. За один присест можно набрать несколько точек останова.

Попробуем это сделать на примере задачи MINimym, для этого запустим файл *min.pas* (из папки лабораторной работы)

Задача: Нахождение минимального числа из введенных.

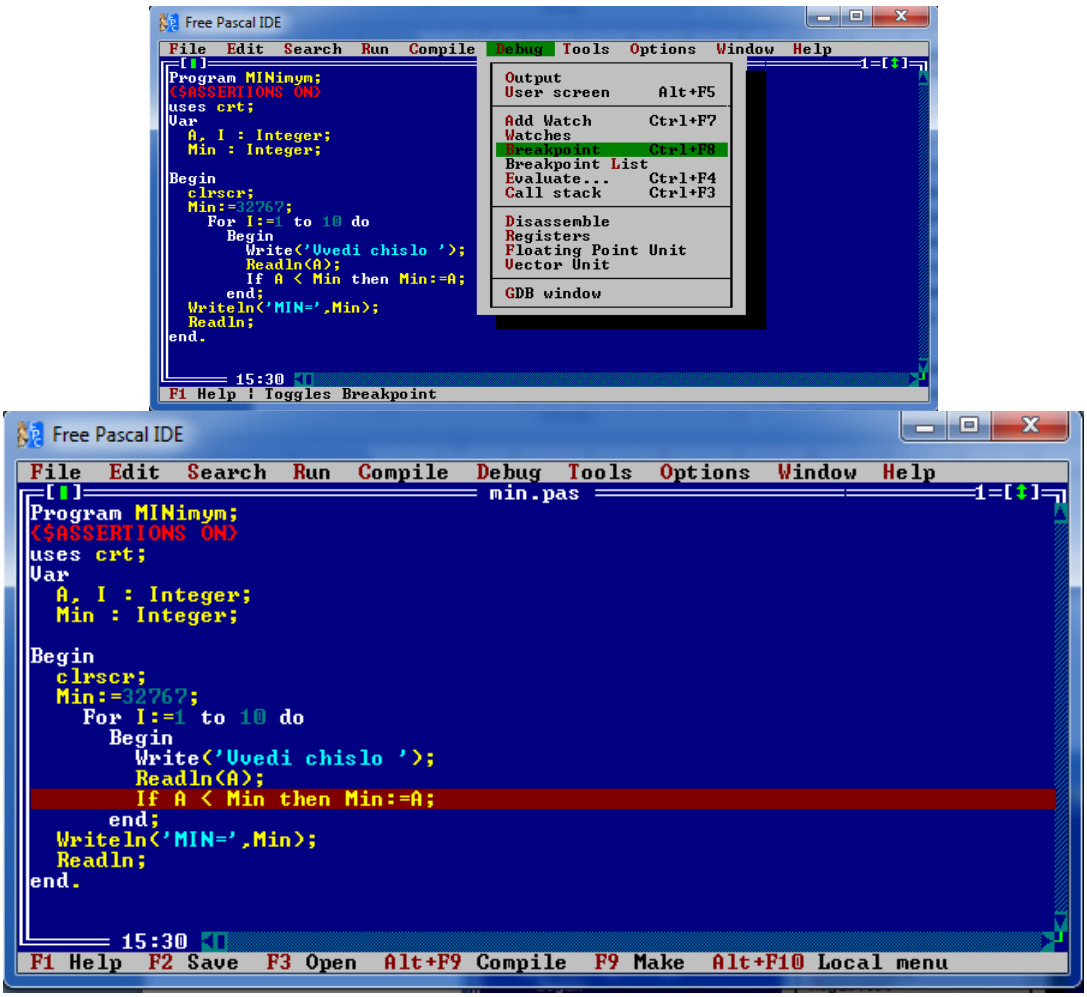

Рис. 3.2.6. Команда Breakpoint.

Список всех точек останова, установленных к текущему моменту, можно увидеть в окне, которое появляется по команде Breakpoint List из меню Debug (рис. 3.2.7)

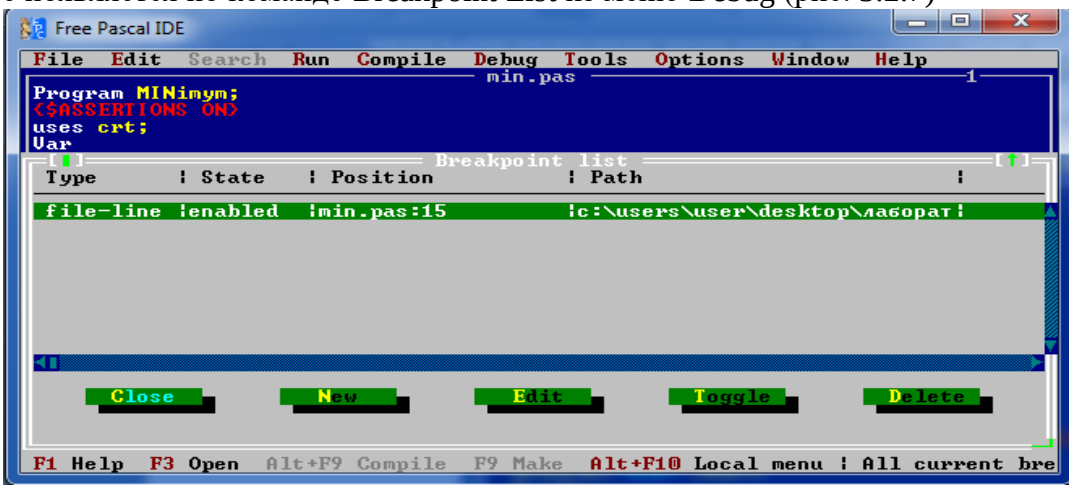

Рис. 3.2.7. Команда Breakpoint List.

Новые точки останова можно добавлять, редактируя строки в этом окне. Вторая колонка позволяет, не удаляя точку останова из списка, сделать ее пассивной (Disabled) — теперь останов на ней не произойдет. Для этой цели используется кнопка Toggle (Переключить). Повторное нажатие этой кнопки меняет статус точки останова на противоположный. Полное удаление точки останова осуществляется с помощью кнопки Delete. (Попробуйте про экспериментировать самостоятельно).

Нажатие кнопки Edit открывает диалоговое окно (рис. 3.2.8), в котором можно оговорить дополнительные условия останова в той или иной точке. Точка останова, установленная обычным способом (<Ctrl>+<F8>), заставляет компьютер прервать выполнение программы при каждом попадании в начало соответствующей строки исходной программы. Представьте себе ситуацию, когда вы установили точку останова на внутренней строке цикла, который должен выполняться сотни раз. Не устать бы, заставляя программу продолжаться после каждого останова. Ошибки в циклах чаще всего возникают либо при первом, либо при последнем прохождении цикла. Дополнительно можно проконтролировать условие досрочного завершения цикла.

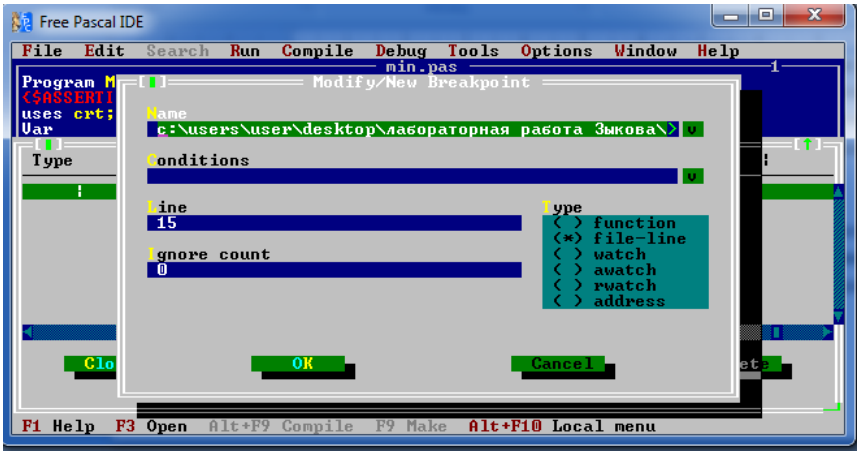

Рис 3.2.8 Дополнительные условия останова

Поэтому условия фактического останова в данной точке программы можно дополнить некоторыми проверками. К таким возможностям относятся:

- задание номера прохождения через данную строку (номер набирается в нижней строке диалогового окна);
- задание логического условия, при выполнении которого должен произойти останов.

Логическое условие останова записывается в строке, над которой расположена надпись Conditions. Формат записи логического выражения такой же, каким вы пользуетесь в операторе if. С помощью этого средства вы можете установить ситуацию, когда, например, в данном операторе в переменную *A* записывается недопустимое (с точки зрения алгоритма) значение: *A* > *Min* (рис. 3.2.9)

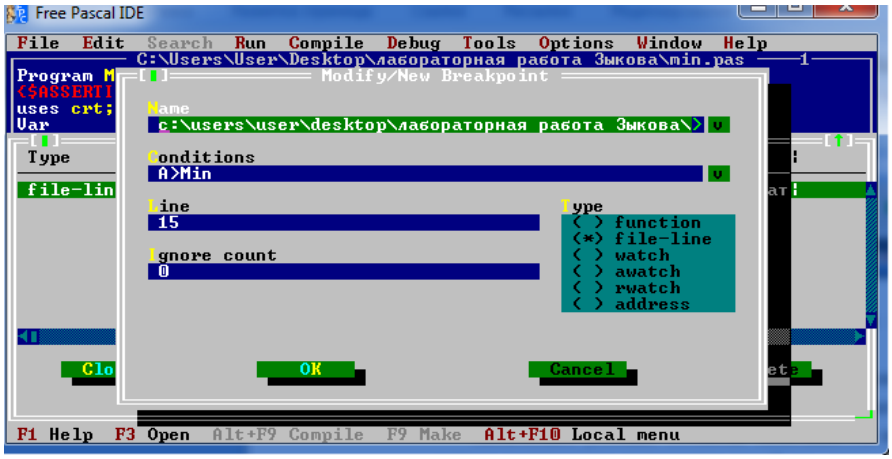

Рис 3.2.9 Логическое условие останова

Запускаем программу вводим заначения

Рис 3.2.10 Запуск программы с вводом значений

После этого программа закрывается так как первый раз min=32000 a=4 a<min не сработает, потом min=4, a=5 a>min сработает.

Нулевое значение счетчика проходов через заданную точку останова означает, что останов должен произойти при каждом попадании на заданную строку программы (именно этой ситуации соответствует надпись на рис. 3.2.9 — Ignore count).

Если в точке останова одновременно заданы дополнительные условия останова по счетчику и по истинности логического выражения, то для фактического останова должны выполниться оба условия.

Наличие в программе нескольких точек останова позволяет перемещаться между ними в автоматическом режиме, что повышает производительность отладки.

Теперь о возможностях просмотра значений переменных и выражений в точках останова. Первая из них заключается в том, что предварительно с помощью команды Add Watch из меню Debug мы набираем нужные имена и формулы в открывающемся окне (рис. 3.2.11).

В момент останова с помощью команды Watches из меню Debug в нижней части экрана открывается окно, в котором представлены все набранные выражения и значения тех из них, которые можно было сосчитать в этой точке.

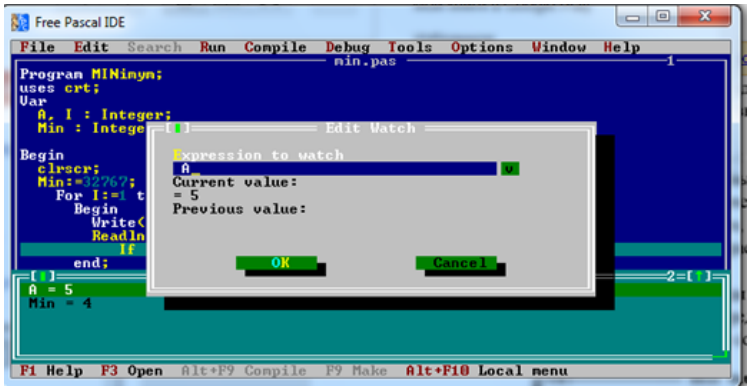

Рис 3.2.11 Команда Add Watch

Вторая возможность оценить значение выражения, не включенного в список Watches, предоставляется командой Evaluate (Вычислить). Соответствующее диалоговое окно представлено на рис. 3.2.12.

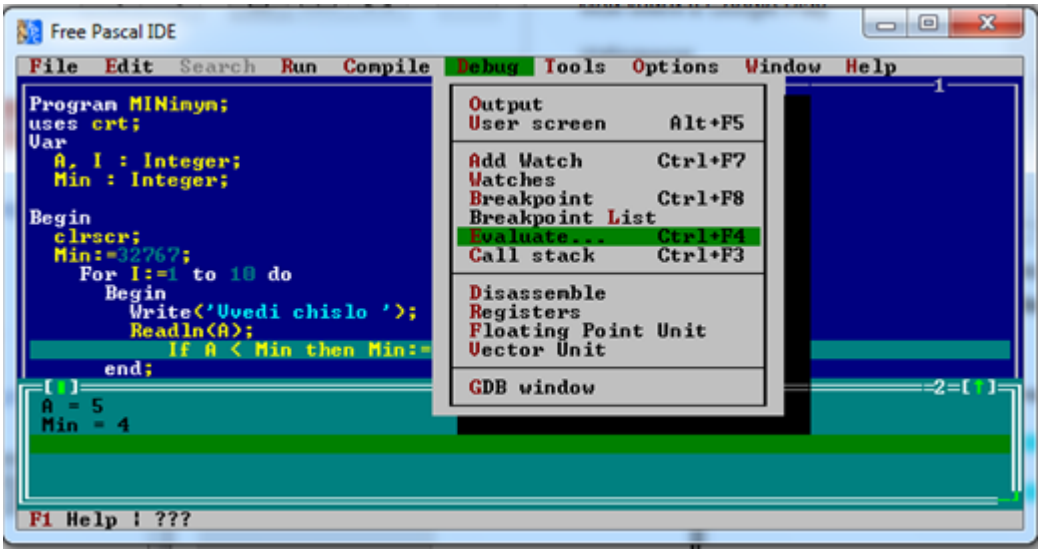

Рис 3.2.12 Команда Evaluate

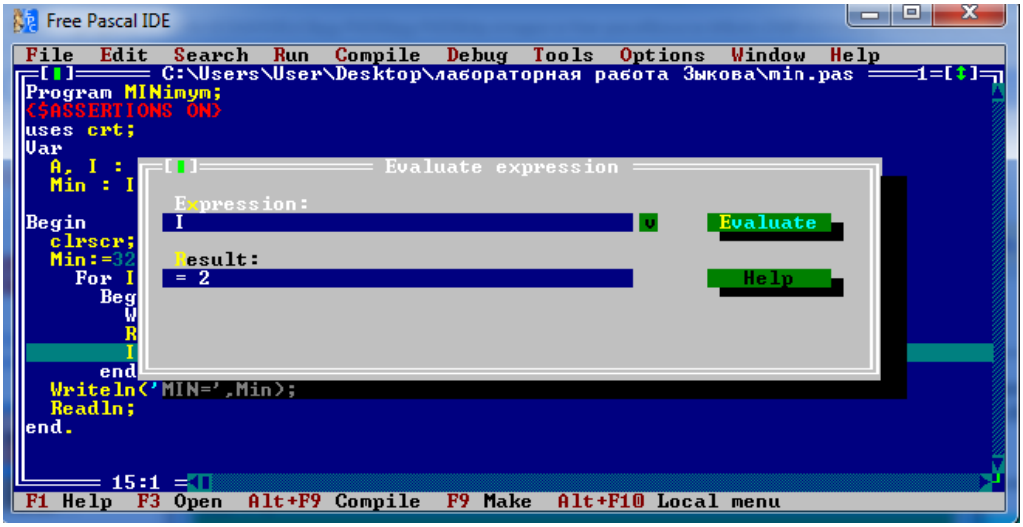

Рис 3.2.13 Команда Evaluate поле Expression

Операнды выражения, набираемого в поле Expression, должны быть доступны в текущей точке останова.

Дополнительным средством отладки является использование системной процедуры Assert (от англ. *assert* — утверждать), допускающей один из двух следующих форматов вызова:
Assert (условное_выражвние);
Assert(условное_выражение, сообщение);

Если в момент выполнения процедуры Assert условное выражение принимает значение True, то работа процедуры завершается без каких-либо дополнительных действий и выполнение программы продолжается. В случае нарушения заданного условия процедура Assert прекращает выполнение программы и выдает либо системное сообщение, либо сообщение, предусмотренное вторым аргументом в обращении. Процедура Assert функционально дублирует действия, возникающие в точке останова, нагруженной дополнительным логическим условием. Разница только в том, что точка останова переводит программу в режим ручного изучения возникшей ситуации. Использование процедуры Assert не приводит к таким задержкам и позволяет включить в окно вывода программы дополнительное смысловое сообщение по поводу происшедшего события. Кроме того, несколько одинаковых обращений к процедуре Assert можно включить в конце каждого участка программы, подозреваемого в нарушении условий правильного функционирования алгоритма.

Добавим процедуру Assert как показано на рисунке Рис 3.2.14:

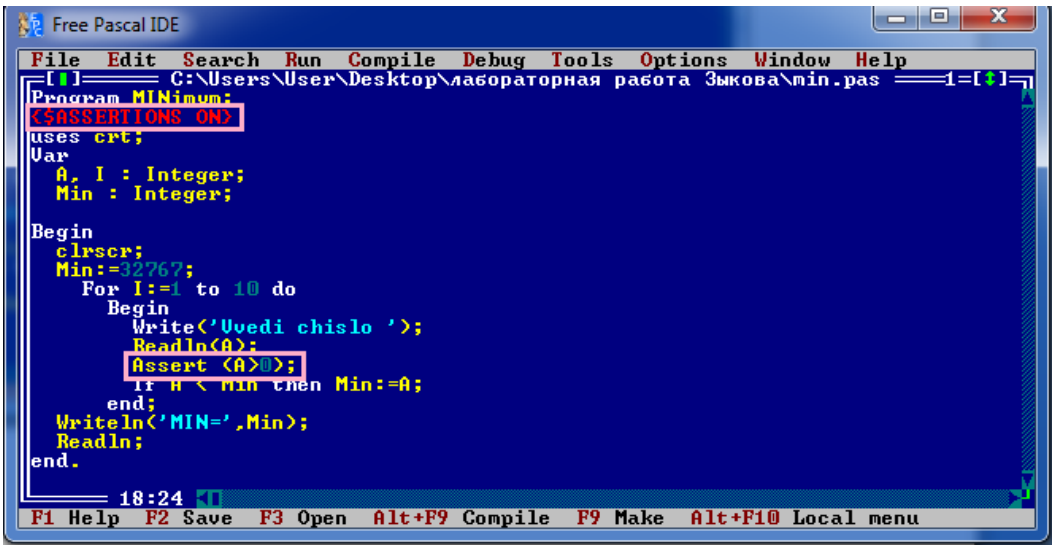

Рис 3.2.14 Процедуру Assert

Запускаем программу вводим отрицательное число и видим сообщение

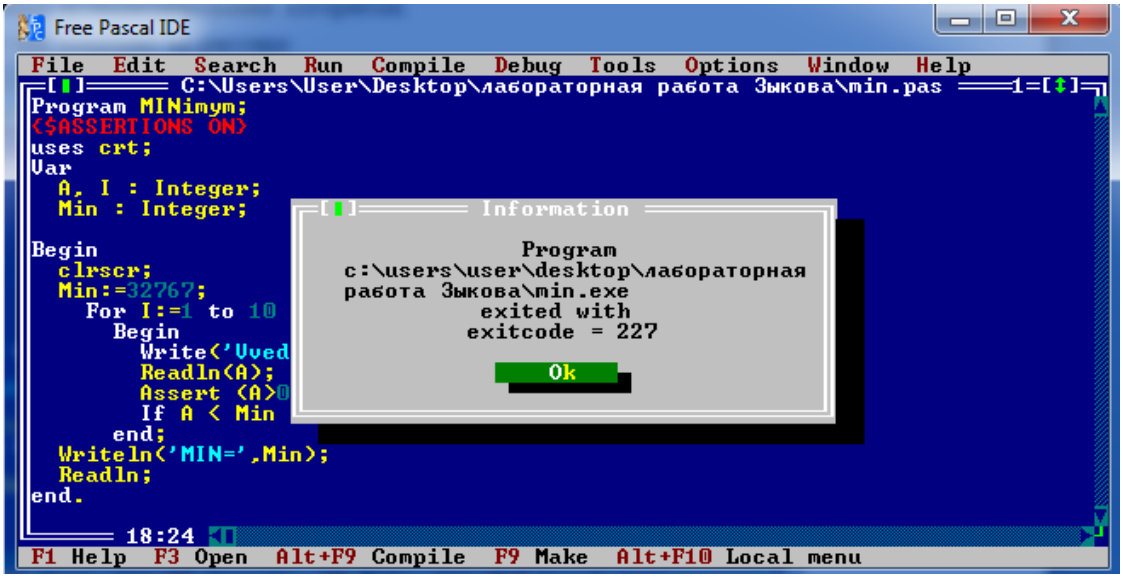

Рис 3.2.15Сообщение процедуры Assert

4. Самостоятельная работа:

Задача: Найти сумму первых n членов ряда y = 1 + x/2 + x2/3 + +x3/4+..., при |x|<1.(файл с кодом программы sam.pas)

Переменные:
n – количество членов ряда;
x – переменная ряда;
z – вспомогательная переменная;
i – переменная цикла;
y – сумма ряда.
Алгоритм решения задачи:

1. вводим количество членов ряда n и переменную X;
2. в цикле порождаем очередной член ряда и прибавляем его к сумме y;
3. выводим результат.

Задание:

1. С помощью изученного материала произвести отладку программы.
2. Использовать все возможности отладки Free Pascal.
3. Сделать Scrn выполнения отладки.

5. Порядок выполнения работы

1. Ознакомиться с теоретической частью.

2. Запустить Free Pascal. Ознакомиться со средствами, предоставляемыми ее для отладки программ.

3. Изучить инструменты.

4.Выполнить практические задания.

5. После исправлений каждую предоставленную программу проверить на наличие ошибок с помощью инструмента CodeGuard – чтоб проверить: не были ли внесены новые ошибки и не пропущены ли какие-либо существовавшие.

6. Составить отчет по выполненной работе.

6. Требования к отчету

Отчет должен содержать:

- ✓ Цель работы
- ✓ Результаты проверки
- ✓ Для каждой ошибки указать ее вид, ее описание (причину), подробно как она была найдена, какой метод отладки использовался, как была исправлена, фрагмент исправленного кода с комментариями (изменения выделить шрифтом)
- ✓ Выводы