Лекция 2 Основы отладки программного обеспечения

ОТЛАДКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Отладка программы - один их самых сложных этапов разработки программного обеспечения, требующий глубокого знания:

- специфики управления используемыми техническими средствами,
- операционной системы,
- среды и языка программирования,
- реализуемых процессов,
- природы и специфики различных ошибок,
- методик отладки и соответствующих программных средств.

Процесс отладки включает:

- действия, направленные на выявление ошибок (тестирование);
- диагностику и локализацию ошибок (определение характера ошибок и их местонахождение);
- внесение исправлений в программу с целью устранения ошибок.

Классификация ошибок

Рассмотрим процесс локализации и исправления ошибок, обнаруженных при тестировании программного обеспечения. *Локализацией* называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса. Для исправления ошибки необходимо определить ее причину, т. е. определить оператор или фрагмент, содержащие ошибку. Причины ошибок могут быть как очевидны, так и очень глубоко скрыты.

В целом сложность отладки обусловлена следующими причинами:

- требует от программиста глубоких знаний специфики управления используемыми техническими средствами, операционной системы, среды и языка программирования, реализуемых процессов, природы и специфики различных ошибок, методик отладки и соответствующих программных средств;
- психологически дискомфортна, так как необходимо искать собственные ошибки и, как правило, в условиях ограниченного времени;
- возможно взаимовлияние ошибок в разных частях программы, например, за счет затирания области памяти одного модуля другим из-за ошибок адресации;
- отсутствуют четко сформулированные методики отладки.

Рис. 1. Классификация ошибок по этапу обработки программы

В соответствии с этапом обработки, на котором проявляются ошибки, различают (рис. 1):

синтаксические ошибки — ошибки, фиксируемые компилятором (транслятором, интерпретатором) при выполнении синтаксического и частично семантического анализа программы;

ошибки компоновки — ошибки, обнаруженные компоновщиком (редактором связей) при объединении модулей программы;

ошибки выполнения — ошибки, обнаруженные операционной системой, аппаратными средствами или пользователем при выполнении программы,

Синтаксические ошибки. Синтаксические ошибки относят к группе самых простых, так как синтаксис языка, как правило, строго формализован, и ошибки сопровождаются развернутым комментарием с указанием ее местоположения. Определение причин таких ошибок, как правило, труда не составляет, и даже при нечетком знании правил языка за несколько прогонов удается удалить все ошибки данного типа.

Следует иметь в виду, что чем лучше формализованы правила синтаксиса языка, тем больше ошибок из общего количества может обнаружить компилятор и, соответственно, меньше ошибок будет обнаруживаться на следующих этапах, в связи с этим говорят о языках программирования с защищенным синтаксисом и с незащищенным синтаксисом. К первым, безусловно, можно отнести Pascal, имеющий очень простой и четко определенный синтаксис, хорошо проверяемый при компиляции программы, ко вторым - Си со всеми его модификациями. Чего стоит хотя бы возможность выполнения присваивания в условном операторе в Си, например:

If (c=n) x=0; /*

в данном случае не проверятся равенство *c* и *n,* а выполняется присваивание с значения *n,* после чего результат операции сравнивается с нулем, если программист хотел выполнить не присваивание, а сравнение, то эта ошибка будет обнаружена только на этапе выполнения при получении результатов, отличающихся от ожидаемых */

Ошибки компоновки. Ошибки компоновки, как следует из названия, связаны с проблемами, обнаруженными при разрешении внешних ссылок. Например, предусмотрено обращение к подпрограмме другого модуля, а при объединении модулей данная подпрограмма не найдена или не стыкуются списки параметров. В большинстве случаев ошибки такого рода также удается быстро локализовать и устранить.

Ошибки выполнения. К самой непредсказуемой группе относятся ошибки выполнения. Прежде всего, они могут иметь разную природу, и соответственно по-разному проявляться. Часть ошибок обнаруживается и документируется операционной системой. Выделяют четыре способа проявления таких ошибок:

• появление сообщения об ошибке, зафиксированной схемами контроля выполнения машинных команд, например, переполнении разрядной сетки, ситуации «деление па ноль», нарушении адресации и т. п.;

• появление сообщения об ошибке, обнаруженной операционной системой, например, нарушении зашиты памяти, попытке записи на устройства, защищенные от записи, отсутствии файла с заданным именем и т. п.;

• «зависание» компьютера, как простое, когда удается завершить программу бел перезагрузки операционной системы, так и «тяжелое», когда для продолжения работы необходима перезагрузка;

• несовпадение полученных результатов с ожидаемыми.

Причины ошибок выполнения очень разнообразны, а потому и локализация может оказаться крайне сложной. Все возможные причины ошибок можно разделить на следующие группы:

• неверное определение исходных данных,

• логические ошибки,

• накопление погрешностей результатов вычислений (рис.2). Неверное определение исходных данных происходит, если возникают любые ошибки при выполнении операций ввода-вывода: ошибки передачи, ошибки преобразования, ошибки перезаписи и ошибки данных. Причем использование специальных технических средств и программирование с защитой от ошибок позволяет обнаружить и предотвратить только часть этих ошибок, о чем безусловно не следует забывать.

Логические ошибки имеют разную природу. Так они могут следовать из ошибок, допущенных при проектировании, например, при выборе методов, разработке алгоритмов или определении структуры классов, а могут быть непосредственно внесены при кодировании модуля. К последней группе относят:

• *ошибки некорректного использования переменных,* например, неудачный выбор типов данных, использование переменных до их инициализации, использование индексов, выходящих за границы определения массивов, нарушения соответствия типов данных при использовании явного или неявного переопределения типа данных, расположенных в памяти при использовании не типизированных переменных, открытых массивов, объединений, динамической памяти, адресной арифметики и т. д.

• *ошибки вычислений,* например, некорректные вычисления над неарифметическими переменными, некорректное использование целочисленной арифметики, некорректное преобразование типов данных в процессе вычислений, ошибки, связанные с незнанием приоритетов выполнения операций для арифметических и логических выражений, и т. п.;

• *ошибки межмодульного интерфейса,* например, игнорирование системных соглашений, нарушение типов и последовательности при передаче параметров, несоблюдение единства единиц измерения формальных и фактических параметров, нарушение области действия локальных и глобальных переменных;

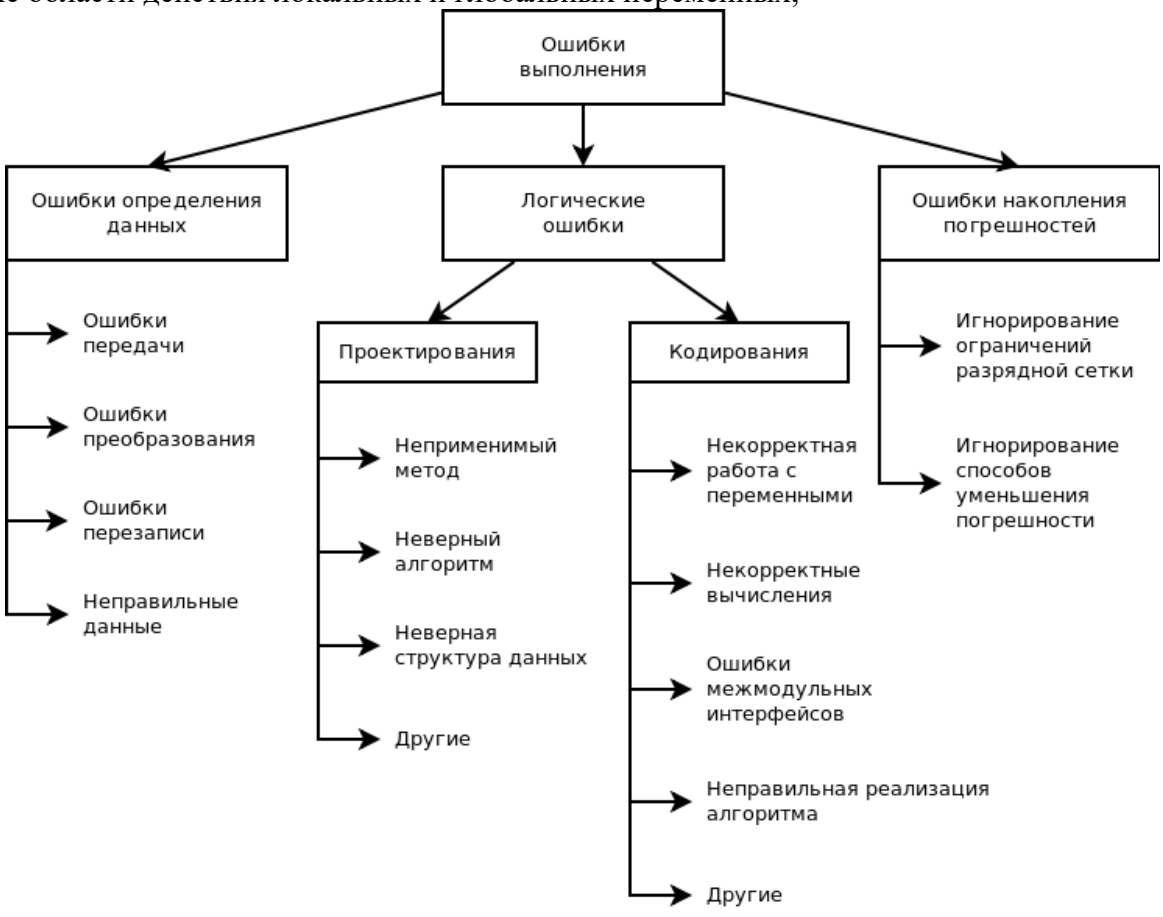

Рис.2. Классификация ошибок этапа выполнения по возможным причинам

• другие *ошибки кодирования,* например, неправильная реализация логики программы при кодировании, игнорирование особенностей или ограничений конкретного языка программирования.

Накопление погрешностей результатов числовых вычислений возникает, например, при некорректном отбрасывании дробных цифр чисел, некорректном использовании приближенных методов вычислений, игнорировании ограничения разрядной сетки представления вещественных чисел в ЭВМ и т. п.

Все указанные выше причины возникновения ошибок следует иметь в виду в процессе отладки. Кроме того, сложность отладки увеличивается также вследствие влияния следующих факторов:

• опосредованного проявления ошибок;

• возможности взаимовлияния ошибок;

• возможности получения внешне одинаковых проявлений разных ошибок;

• отсутствия повторяемости проявлений некоторых ошибок от запуска к запуску - так называемые стохастические ошибки;

• возможности устранения внешних проявлений ошибок в исследуемой ситуации при внесении некоторых изменений в программу, например, при включении в программу диагностических фрагментов может аннулироваться или измениться внешнее проявление ошибок;

• написания отдельных частей программы разными программистами.

Принципы локализации ошибок:

·Большинство ошибок обнаруживается вообще без запуска программы - просто внимательным просматриванием текста.

·Если отладка зашла в тупик и обнаружить ошибку не удается, лучше отложить программу. Когда глаз "замылен", эффективность работы упорно стремится к нулю.

·Чрезвычайно удобные вспомогательные средства — это отладочные механизмы среды разработки: трассировка, промежуточный контроль значений. Можно использовать даже дамп памяти, но такие радикальные действия нужны крайне редко.

·Экспериментирования типа "а что будет, если изменить плюс на минус" - нужно избегать всеми силами. Обычно это не дает результатов, а только больше запутывает процесс отладки, да еще и добавляет новые ошибки.

Принципы исправления ошибок еще больше похожи на законы Мерфи:

·Там, где найдена одна ошибка, возможно, есть и другие.

·Вероятность, что ошибка найдена правильно, никогда не равна ста процентам.

·Наша задача - найти саму ошибку, а не ее симптом.

Это утверждение хочется пояснить. Если программа упорно выдает результат 0,1 вместо эталонного нуля, простым округлением вопрос не решить. Если результат получается отрицательным вместо эталонного положительного, бесполезно брать его по модулю - мы получим вместо решения задачи ерунду с подгонкой.

·Исправляя одну ошибку, очень легко внести в программу еще парочку. "Наведенные" ошибки - настоящий бич отладки.

Методы отладки программного обеспечения

Отладка программы в любом случае предполагает обдумывание и логическое осмысление всей имеющейся информации об ошибке. Большинство ошибок можно посредством тщательного анализа текстов программ и результатов обнаружить по косвенным признакам тестирования без получения дополнительной информации. При этом используют различные методы:

• ручного тестирования;
• индукции;
• дедукции;
• обратного прослеживания.

Метод ручного тестирования. Это - самый простой и естественный способ данной группы. При обнаружении ошибки необходимо выполнить тестируемую программу вручную, используя тестовый набор, при работе с которым была обнаружена ошибка.

Метод очень эффективен, но не применим для больших программ, программ со сложными вычислениями и в тех случаях, когда ошибка связана с неверным представлением программиста о выполнении некоторых операций. Данный метод часто используют как составную часть других методов отладки.

Метод индукции. Метод основан на тщательном анализе симптомов ошибки, которые могут проявляться как неверные результаты вычислений или как сообщение об ошибке. Если компьютер просто «зависает», то фрагмент проявления ошибки вычисляют, исходя из последних полученных результатов и действий пользователя. Полученную таким образом информацию организуют и тщательно изучают, просматривая соответствующий фрагмент программы. В результате этих действий выдвигают гипотезы об ошибках, каждую из которых проверяют. Если гипотеза верна, то детализируют информацию об ошибке, иначе - выдвигают другую гипотезу. Последовательность выполнения отладки методом индукции показана на рис. 3 в виде схемы алгоритма. Самый ответственный этап - выявление симптомов ошибки. Организуя данные об ошибке, целесообразно записать все, что известно о ее проявлениях, причем фиксируют, как ситуации, в которых фрагмент с ошибкой выполняется нормально, так и ситуации, в которых ошибка проявляется. Если в результате изучения данных никаких гипотез не появляется, то необходима дополнительная информация об ошибке. Дополнительную информацию можно получить, например, в результате выполнения схожих тестов.

Рис. 3. Схема процесса отладки методом индукции

В процессе доказательства пытаются выяснить, все ли проявления ошибки объясняет данная гипотеза, если не все, то либо гипотеза не верна, либо ошибок несколько.

Метод дедукции. По методу дедукции вначале формируют множество причин, которые могли бы вызвать данное проявление ошибки. Затем анализируя причины, исключают те, которые противоречат имеющимся данным. Если все причины исключены, то следует выполнить дополнительное тестирование исследуемого фрагмента. В противном случае наиболее вероятную гипотезу пытаются доказать. Если гипотеза объясняет полученные признаки ошибки, то ошибка найдена, иначе - проверяют следующую причину (рис. 4).

Метод обратного прослеживания. Для небольших программ эффективно применение метода обратного прослеживания. Начинают с точки вывода неправильного результата. Для этой точки строится гипотеза о значениях основных переменных, которые могли бы привести к получению имеющегося результата. Далее, исходя из этой гипотезы, делают предположения о значениях переменных в предыдущей точке. Процесс продолжают, пока не обнаружат причину ошибки.

Рис. 4. Схема процесса отладки методом дедукции

Методы и средства получения дополнительной информации

Для получения дополнительной информации об ошибке можно выполнить добавочные тесты или использовать специальные методы и средства:

- отладочный вывод;
- интегрированные средства отладки;
- независимые отладчики.

Отладочный вывод. • Вывод текущего состояния программы с помощью расположенных в критических точках программы операторов вывода — на экран, принтер, громкоговоритель или в файл. Вывод отладочных сведений в файл называется журналированием. Метод требует включения в программу дополнительного отладочного вывода в *узловых* точках. Узловыми считают точки алгоритма, в которых основные переменные программы меняют свои значения. Например, отладочный вывод следует предусмотреть до и после завершения цикла изменения некоторого массива значений. (Если отладочный вывод предусмотреть в цикле, то будет выведено слишком много значений, в которых, как правило, сложно разбираться.) При этом предполагается, что, выполнив анализ выведенных значений, программист уточнит момент, когда были получены неправильные значения, и сможет сделать вывод о причине ошибки.

Данный метод не очень эффективен и в настоящее время практически не используется, так как в сложных случаях в процессе отладки может потребоваться вывод большого количества — «трассы» значений многих переменных, которые выводятся при каждом изменении. Кроме того, внесение в программы дополнительных операторов может привести к изменению проявления ошибки, что нежелательно, хотя и позволяет сделать определенный вывод о ее природе.

Примечание. Ошибки, исчезающие при включении в программу или удалению ид нее каких-либо «безобидных» операторов, как правило, связаны с "затиранием" памяти. В результате добавления или удаления операторов область затирания может

сместиться в другое место, и ошибка либо перестанет проявляться, либо будет проявляться по-другому.

Интегрированные средства отладки. Большинство современных сред программирования (Delphi, Lazarus, Builder C++, Visual Studio и т. д.) включают средства отладки, которые обеспечивают максимально эффективную отладку, они позволяют:

• выполнять программу по шагам, причем как с заходом в подпрограммы, так и выполняя их целиком;

• предусматривать точки останова;

• выполнять программу до оператора, укачанного курсором;

• отображать содержимое любых переменных при пошаговом выполнении;

• отслеживать поток сообщений и т. п.

На рис. 5 показан вид программы в момент перехода. В режим пошагового выполнения по достижении точки останова в Delphi. В этот момент программист имеет возможность посмотреть значения интересующих его переменных.

Применять интегрированные средства в рамках среды достаточно просто, используют разные приемы в зависимости от проявлений ошибки. Если получено сообщение об ошибке, то сначала уточняют, при выполнении какого оператора программы оно получено. Для этого устанавливают точку останова в начало фрагмента, в котором проявляется ошибка, и выполняют операторы в пошаговом режиме до проявления ошибки.

Аналогично поступают при «зависании» компьютера.

Если получены неправильные результаты, то локализовать ошибку обычно существенно сложнее. В этом случае сначала определяют фрагмент, при выполнении которого получаются неправильные результаты. Для этого последовательно проверяют интересующие значения в узловых точках. Обнаружив значения, отличающиеся от ожидаемых, по шагам трассируют соответствующий фрагмент до выявления оператора, выполнение которого дает неверный результат.

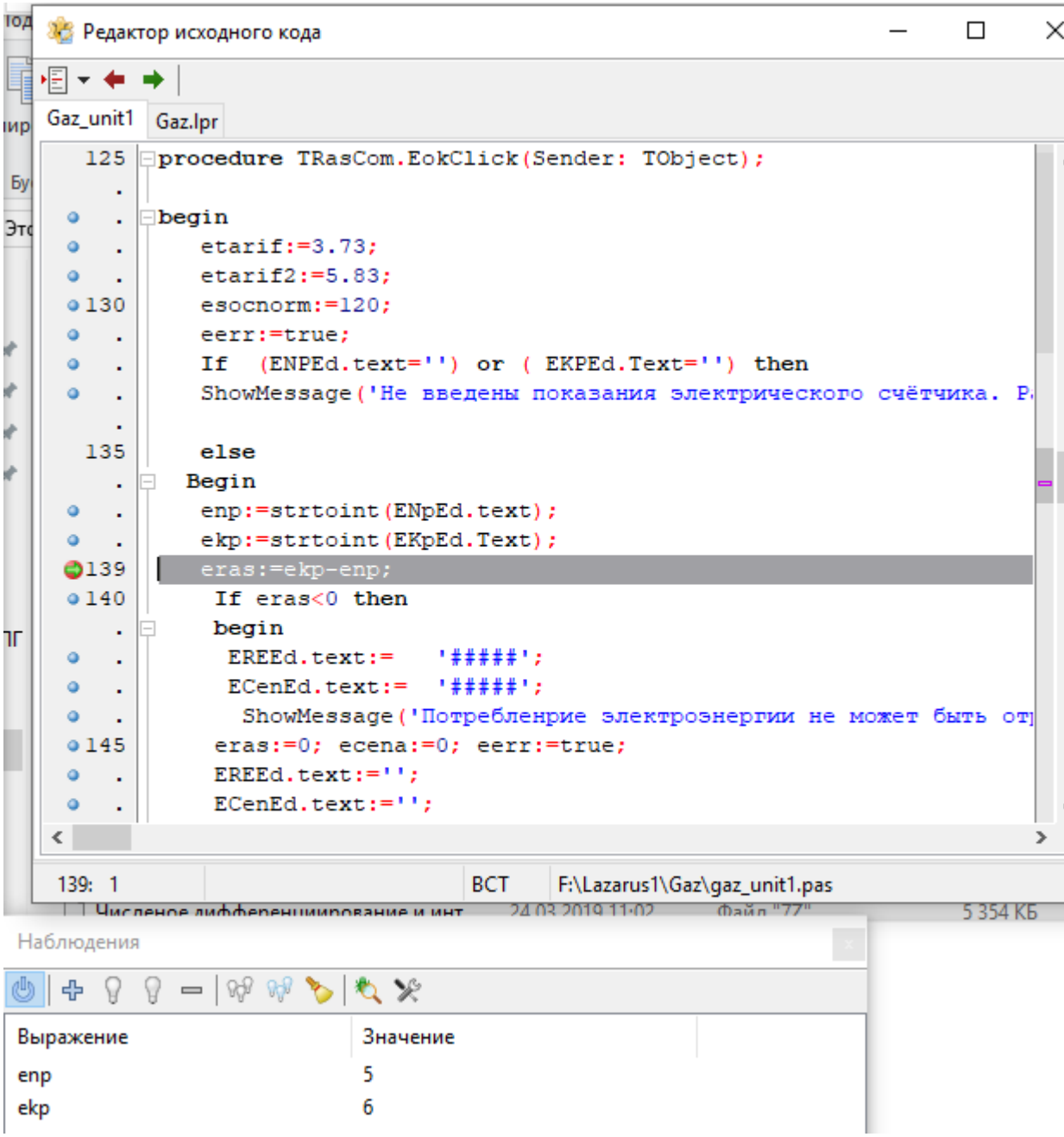

Рис. 5. Переход в режим пошагового выполнения по достижении точки останова в Lazarus (в рамке указано значение переменной «под курсором»)

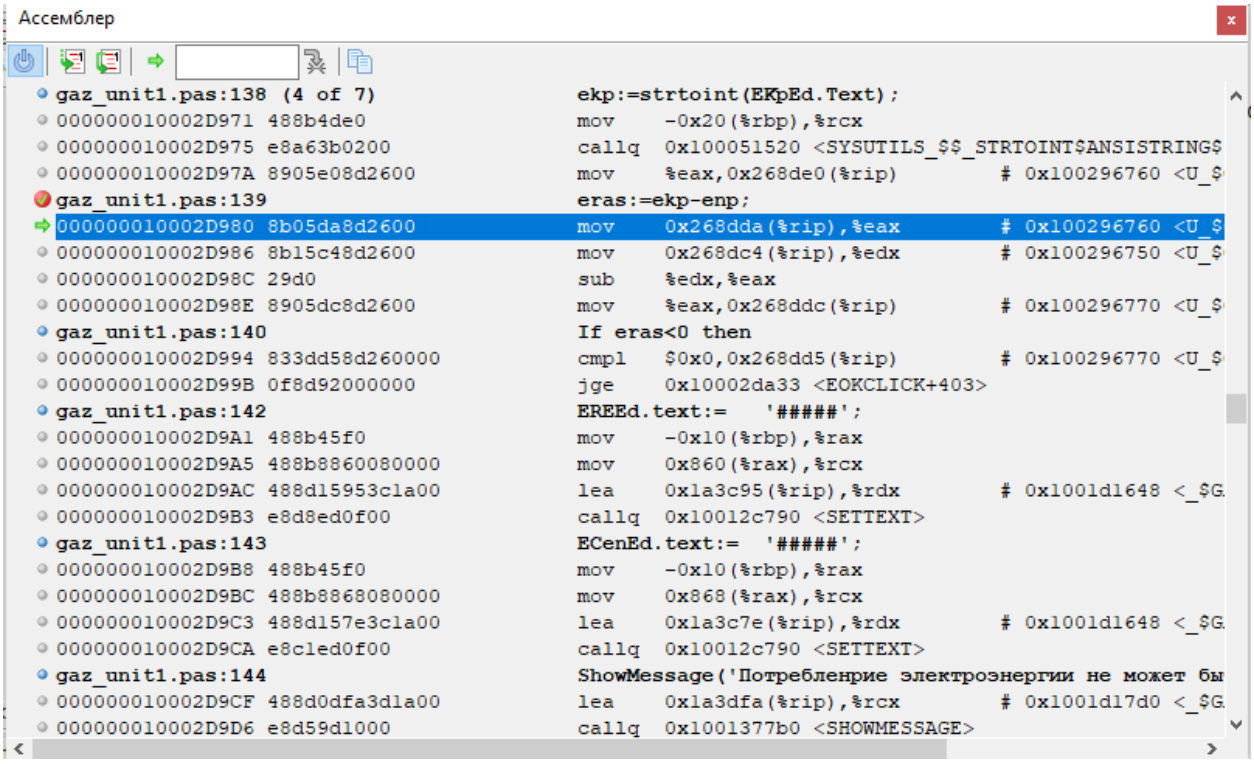

Рис. 6. Вид экрана при отладке программы в окне «Ассемблер»

Для уточнения природы ошибки возможен анализ машинных кодов, флагов и представления программы и значений памяти в 16-ричном виде (рис. 6).

При этом для проверки гипотез также можно использовать интегрированные средства отладки.

Отладка с использованием независимых отладчиков. При отладке программ иногда используют специальные программы - отладчики, которые позволяют выполнить любой фрагмент программы в пошаговом режиме и

проверить содержимое интересующих программиста переменных. Как правило такие отладчики позволяют отлаживать программу только в машинных командах, представленных в 16-ричном коде.

Общая методика отладки программного обеспечения

Суммируя все сказанное выше, можно предложить следующую методику отладки программного обеспечения, написанного на универсальных языках программирования для выполнения в современных операционных системах :

1 этап - изучение проявления ошибки - если выдано какое-либо сообщение или выданы неправильные или неполные результаты, то необходимо их изучить и попытаться понять, какая ошибка могла так проявиться. При этом используют индуктивные и дедуктивные методы отладки. В результате выдвигают версии о характере ошибки, которые необходимо проверить. Для этого можно применить методы и средства получения дополнительной информации об ошибке.

Если ошибка не найдена или система просто «зависла», переходят ко второму этапу.

2 этап - локализация ошибки - определение конкретного фрагмента, при выполнении которого произошло отклонение от предполагаемого вычислительного процесса. Локализация может выполняться:

• путем отсечения частей программы, причем, если при отсечении некоторой части программы ошибка пропадает, то это может означать как то, что ошибка связана с этой частью, так и то. что внесенное изменение изменило проявление ошибки;

• с использованием отладочных средств, позволяющих выполнить интересующих нас фрагмент программы в пошаговом режиме и получить дополнительную информацию о месте проявления и характере ошибки, например, уточнить содержимое указанных переменных.

При этом если были получены неправильные результаты, то в пошаговом режиме проверяют ключевые точки процесса формирования данного результата.

Как подчеркивалось выше, ошибка не обязательно допущена в том месте, где она проявилась, если в конкретном случае это так, то переходят к следующему этапу,

3 этап - определение причины ошибки - изучение результатов второго этапа и формирование версий возможных причин ошибки. Эти версии необходимо проверить, возможно, используя отладочные средства для просмотра последовательности операторов или значений переменных,

4 этап - исправление ошибки - внесение соответствующих изменений во все операторы, совместное выполнение которых привело к ошибке.

5 этап - повторное тестирование - повторение всех тестов с начала, так как при исправлении обнаруженных ошибок часто вносят в программу новые.

Следует иметь в виду, что процесс отладки можно существенно упростить, если следовать основным рекомендациям структурного подхода к программированию:

• программу наращивать «сверху-вниз», от интерфейса к обрабатывающим подпрограммам, тестируя ее по ходу добавления подпрограмм;

• выводить пользователю вводимые им данные для контроля и проверять их на допустимость сразу после ввода;

• предусматривать вывод основных данных во всех узловых точках алгоритма (ветвлениях, вызовах подпрограмм).

Кроме того, следует более тщательно проверять фрагменты программного обеспечения, где уже были обнаружены ошибки, так как вероятность ошибок в этих местах по статистике выше. Это вызвано следующими причинами. Во-первых, ошибки чаще допускают в сложных местах или в тех случаях, если спецификации на реализуемые операции недостаточно проработаны. Во-вторых, ошибки могут быть результатом того, что программист устал, отвлекся или плохо себя чувствует. В-третьих, как уже упоминалось выше, ошибки часто появляются в результате исправления уже найденных ошибок.

Возвращаясь к рис. 2. можно отметить, что проще всего обычно искать ошибки определения данных и ошибки накопления погрешностей: их причины локализованы в месте проявления. Логические ошибки искать существенно сложнее. Причем обнаружение ошибок проектирования требует возврата на предыдущие этапы и внесения соответствующих изменений в проект. Ошибки кодирования бывают как простые, например, использование неинициализированной переменной, так и очень сложные, например, ошибки, связанные с затиранием памяти.

Затиранием памяти называют ошибки, приводящие к тому, что в результате записи некоторой информации не на свое место в оперативной памяти, затираются фрагменты данных или даже команд программы. Ошибки подобного рода обычно вызывают появление сообщения об ошибке. Поэтому определить фрагмент, при выполнении которого ошибка проявляется, несложно, А вот определение фрагмента программы, который затирает память - сложная задача, причем, чем длиннее программа, тем сложнее искать ошибки такого рода. Именно в этом случае часто прибегают к удалению из программы частей, хотя это и не обеспечивает однозначного ответа на вопрос, в какой из частей программы находится ошибка. Эффективнее попытаться определить операторы, которые записывают данные в память не по имени, а по адресу, и

последовательно их проверить. Особое внимание при этом следует обращать на корректное распределение памяти под данные.

Автономное тестирование модуля целесообразно осуществлять в четыре последовательно выполняемых шага.

Шаг 1. На основании спецификации отлаживаемого модуля подготовьте тесты для каждой возможности и каждой ситуации, для каждой границы областей допустимых значений всех входных данных, для каждой области изменения данных, для каждой области недопустимых значений всех входных данных и каждого недопустимого условия.

Шаг 2. Проверьте текст модуля, чтобы убедиться, что каждое направление любого разветвления будет пройдено хотя бы на одном тесте. Добавьте недостающие тесты.

Шаг 3. Проверьте текст модуля, чтобы убедиться, что для каждого цикла существуют тесты, обеспечивающие, по крайней мере, три следующие ситуации: тело цикла не выполняется ни разу, тело цикла выполняется один раз и тело цикла выполняется максимальное число раз. Добавьте недостающие тесты.

Шаг 4. Проверьте текст модуля, чтобы убедиться, что существуют тесты, проверяющие чувствительность к отдельным особым значениям входных данных. Добавьте недостающие тесты.

Советы отладчику

1) Проверяйте тщательнее: ошибка скорее всего находится не в том месте, в котором кажется.

2) Часто оказывается легче выделить те места программы, ошибок в которых нет, а затем уже искать в остальных.

3) Тщательнее следить за объявлениями констант, типов и переменных, входными данными.

4) При последовательной разработке приходится особенно аккуратно писать драйверы и заглушки - они сами могут быть источником ошибок.

5) Анализировать код, начиная с самых простых вариантов. Чаще всего встречаются ошибки:

значения входных аргументов принимаются не в том порядке,

переменная не проинициализирована,

при повторном прохождении модуля, переменная повторно не инициализируется,

вместо предполагаемого полного копирования структуры данных, копируется только верхний уровень (например, вместо создания новой динамической переменной и присваивания ей нужного значения, адрес тупо копируется из уже существующей переменной),

скобки в сложном выражении расставлены неправильно.

6) При упорной длительной отладке глаз "замыливается". Хороший прием - обратиться за помощью к другому лицу, чтобы не повторять ошибочных рассуждений. Правда, частенько остается проблемой убедить это другое лицо помочь вам.

7) Ошибка, скорее всего окажется вашей и будет находиться в тексте программы. Гораздо реже она оказывается:

·в компиляторе,

·операционной системе,

·аппаратной части,

·электропроводке в здании и т.д.

Но если вы совершенно уверены, что в программе ошибок нет, просмотрите стандартные модули, к которым она обращается, выясните, не менялась ли версия среды разработки, в конце концов, просто перегрузите компьютер - некоторые проблемы (особенно в DOS-средах, запускаемых из-под Windows) возникают из-за некорректной работы с памятью.

8) Убедитесь, что исходный текст программы соответствует скомпилированному объектному коду (текст может быть изменен, а запускаемый модуль, который вы тестируете - скомпилирован еще из старого варианта).

9) Навязчивый поиск одной ошибки почти всегда непродуктивен. Не получается - отложите задачу, возьмитесь за написание следующего модуля, на худой конец займитесь документированием.

10) Старайтесь не жалеть времени, чтобы понять причину ошибки. Это поможет вам:

·исправить программу,

·обнаружить другие ошибки того же типа,

·не делать их в дальнейшем.

11) Если вы уже знаете симптомы ошибки, иногда полезно не исправлять ее сразу, а на фоне известного поведения программы поискать другие ляпы.

12) Самые трудно обнаруживаемые ошибки - наведенные, то есть те, что были внесены в код при исправлении других.