Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем
Системное программирование

Дмитриева Дарья Алексеевна

Разработка отладчика для языка РуСи

Бакалаврская работа

Научный руководитель:
д. ф.-м. н., профессор Терехов А. Н.

Рецензент:
инженер ООО Ланит-Терком Смирнов К. К.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems
Software engineering

Dmitrieva Darya

Development of debugger for RuC

Graduation Thesis

Scientific supervisor:
professor Andrey Terekhov

Reviewer:
engineer, OOO Lanit-Tercom Kirill Smirnov

Saint-Petersburg
2017

Оглавление

Введение

В современном мире компьютерные технологии занимают очень большое место. Обучаться работе с ними дети начинают с младших классов. Причем некоторые школы хотят дать знания не просто на уровне пользователя персонального компьютера, а гораздо глубже, обучая основам программирования. Стандартные инструменты и языки могут не подходить для первоначального знакомства с программированием по ряду причин. Во-первых, большинство известных языков программирования используют английский язык, который школьники если и знают, то совсем не на том уровне, чтобы понимать каждую команду и слово, что сильно осложняет восприятие даже простых программ и их смысла. Особенно это важно на начальных этапах, когда обучаемый еще не успел запомнить наизусть операторы. Во-вторых, сложные высокоуровневые языки не всегда точно определяют ошибки в коде. В существующих компиляторах фактические места ошибок зачастую могут не совпадать с указанными, а сообщения шаблонны и не объясняют суть ошибки. А для начинающего программиста порой очень сложно понять, где именно он написал код неправильно. Или же наоборот, некоторые ошибки никак не отслеживаются компилятором и человек получает неправильный результат, но не знает даже места, где он ошибся. В-третьих, стандартные языки предназначены, в первую очередь, для промышленного программирования, в следствие чего содержат большое количество средств возможностей, которые не только не нужны на начальном этапе обучения, но и мешают ему.

К сожалению, в России достаточно мало средств для обучения программированию именно с нуля, еще и в младшем школьном возрасте. Особенно остра нехватка средств для обучения именно традиционному составлению программ в виде текстов. Существуют такие среды как КуМИР и ЛогоМиры, но они нацелены больше на работу обучаемого с графической информацией, что не является хорошей и полной основой для продолжения обучения программированию и легкому переходу на более серьезные языки.

А.Н. Тереховым был разработан язык РуСи и его транслятор, решающий все вышеперечисленные проблемы[8]. Его основой является язык C[7][3], но в очень упрощенном варианте[9]. Оставлены несколько основных типов (целочисленный, дробный, символьный), убрана сложная арифметика указателей и некоторые другие возможности языка, не нужные начинающим программистам. Также на РуСи для всех ключевых слов есть варианты как на английском языке они совпадают с классическим C), так и на русском. Из-за упрощения возможностей и подробного анализа этот транслятор может различать более различных 100 ошибок в коде с подробным описанием и точным определением их места.

Также для языка РуСи был реализован транслятор в код виртуальной машины и интерпретатор, исполняющий машинный код. С РуСи можно работать как в консоли, так и в специально разработанной для него интегрированной среде разработки (IDE)[10].

Для помощи в поиске несинтаксических ошибок в коде у многих IDE есть режим отладки, который может быть полезен для начинающих программистов. С его помощью можно увидеть, в какой последовательности выполняются операторы ветвления или отследить изменения всех переменных. Иногда начинающий программист может даже не догадываться, в какой момент программа начала исполняться не как он задумывал, а с помощью точек останова можно определить место ошибки с точностью до оператора.

В трансляторе и редакторе РуСи не было режима отладки, но он является важным инструментом разработчика, возможности которого можно использовать с самого начала обучения программированию.

1. Постановка задачи

Целью данной работы является доработка транслятора и IDE РуСи для добавления возможности отладки программ. Для достижения этой цели в рамках работы были сформулированы следующие задачи.

- Изучение архитектуры и компонентов транслятора РуСи.
- Разработка архитектуры отладчика.
- Реализация отладчика.
- Тестирование доработанного транслятора и редактора РуСи.

2. Обзор

2.1. Виды возможных прерываний программы

2.1.1. Стандартные точки останова

Точки останова ставятся на строку кода для прерывания исполнения программы, когда она доходит до исполнения оператора в этой строке. При этом большинство отладчиков поддерживают возможности просмотра значений переменных, стека вызова функций и другой информации об исполняемой программе. Точки останова являются основным и самым используемым инструментом отладчика, поэтому были добавлены в отладчик РуСи в первую очередь.

2.1.2. Трассировка

Режим трассировки подразумевает прерывания программы в каждой строке кода. То есть его можно реализовать с помощью постановки точек останова на всех строках. В рамках выпускной квалификационной работы этот режим не был реализован, но его легко добавить при необходимости.

2.1.3. Условные точки останова

Для постановки условных точек останова пользователь вводит какое-либо условие, и прерывание происходит при его срабатывании. Данный инструмент достаточно специфичен и сложен в использовании, поэтому его не стали добавлять в отладчик РуСи, предназначенный в первую очередь для начинающих программистов.

2.1.4. Остановки по доступу к переменной

Для этого типа прерывания пользователем сначала выбирается переменная для отслеживания и режим, при котором он хочет сделать остановку (по чтению из переменной или записи в нее), и, как только

при исполнении программы она получает доступ к выбранной переменной в нужном режиме, происходит прерывание. Из всех IDE для высокоуровневых языков программирования этот вид прерываний в явном виде реализован только в отладчике РуСи.

2.2. Стандартная реализация отладчика

Так как большинство современных трансляторов используют двухуровневую систему трансляции (из исходного кода в объектный или промежуточный, а из него – в исполняемый)[6], отладчик тоже является двухуровневым, то есть он должен предоставлять пользователю информацию об отлаживаемой программе в терминах исходного языка, в то время как реальное исполнение программы происходит в кодах виртуальной машины. Поэтому, как правило, отдельно реализуется интерфейс для отладки исполняемого кода, а уже на его основе строится система связи между исходным кодом и объектным и отладчик исходного кода.

Промежуточный код представляет из себя набор объектов, каждый из которых отвечает за определенный оператор исходного кода. Для связи между ними в объект добавляется новое поле с координатами исходного кода. Благодаря этому, внешний отладчик будет знать, на каком объкте промежуточного кода надо остановиться. При трансляции промежуточного кода в исполняемый для координат каждого объекта сопоставляются координаты команд исполняемого кода (обычно, это просто их адреса в памяти). После этого отладчик сможет понимать какой координате объектного кода соответствует текущая исполняемая команда.

Отладчик исполняемого кода обычно предоставляет следующий интерфейс [4] из стандартных функций и оповещений: загрузки модуля, что означает начало исполнения программы, один шаг исполнения программы (одного оператора), приостановка программы, просмотр стека вызовов и значений переменных и другие менее важные функции.

Внешний отладчик с помощью событий отслеживает пошаговое ис-

полнение программы и, в случае необходимости, запрашивает прерывание исполнения кода.

Особенность РуСи заключается в том, что и редактор, и транслятор (в том числе перевод в исполняемый код и виртуальную машину для него) написала одна и та же команда разработчиков, причем транслятору не надо быть кроссплатформенным, так как продукт является профилированным, и для него не будет много разных IDE для которых надо поддерживать гибкость и абстрактность, поэтому не имело смысла делить отладчик на две совсем независимые части со сложными интерфейсами взаимодействия.

2.3. Описание RuC

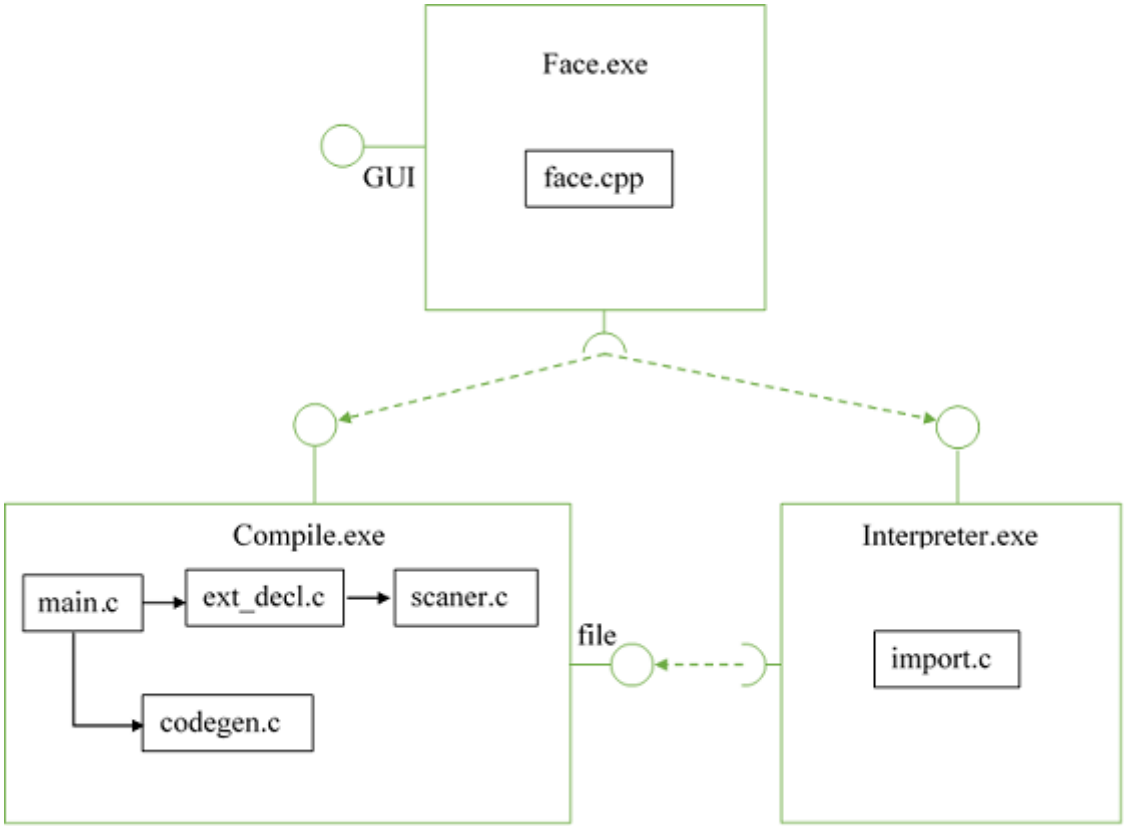

Рис. 1: Транслятор РуСи (UML-диаграмма компонентов)

На рисунке 1 представлена UML-диаграмма компонентов транслятора и редактора РуСи. Компонент compile.exe включает в себя лексический анализатор (scaner.c), синтаксический анализатор с видозависи-

мым анализом (extdecl.c), который строит объектный код в виде дерева разбора и кодогенератор (codegen.c), переводящий это дерево в исполняемый код. Компонент interpreter.exe содержит только интерпретатор кода виртуальной машины (import.c). Компонент face.exe является интерфейсом для общения с пользователем (интегрированной средой разработки) и запускает остальные компоненты по мере необходимости.

2.3.1. Синтаксический и видозависимый анализ

На вход синтаксическому анализатору подается текст программы в терминах исходного языка РуСи, написанной пользователем. Он преобразует его в объектный код, представленный в виде стандартного дерева разбора, которое хранится в табличном виде, строит таблицы всех переменных и функций (есть отдельные таблицы для всех представлений и значений идентификаторов, функций и сложных типов данных), а также проводит видозависимый анализ. После этого он переводит дерево в код виртуальной машины, который и будет исполняться интерпретатором. Все результаты записываются в специальный файл.

2.3.2. Интерпретатор

Интерпретатор берет код виртуальной машины (он хранится в отдельной от данных таблице и представляет собой последовательный набор чисел, зарезервированных константами под определенные операции, и обращений к переменным в памяти через локальное или глобальное смещение) и все нужные дополнительные данные о переменных и функциях из файла, сгенерированного синтаксическим анализатором. Перед началом исполнения ищется точка входа (все функции связаны между собой в цепочкой и только у первой исполняемой стоит вызов на нее саму же) и с нее запускается пошаговое исполнение команд.

2.3.3. Интегрированная среда разработки

Интегрированная среда разработки представляет из себя удобный пользователю интерфейс для написания, редактирования, сохранения,

компилирования и исполнения программ. При нажатии на кнопку или горячую клавишу компиляции она запускает compile.exe, который генерирует исполняемый код из исходного, а по нажатии на кнопку или горячую клавишу исполнения запускает import.exe, исполняющий код ранее сгенерированный синтаксическим анализатором. Причем коммуникации между редактором и интерпретатором осуществляются через так называемый водопровод (pipeline).

3. Архитектура отладчика

3.1. Функциональная часть отладчика

3.1.1. Постановка и удаление точек останова

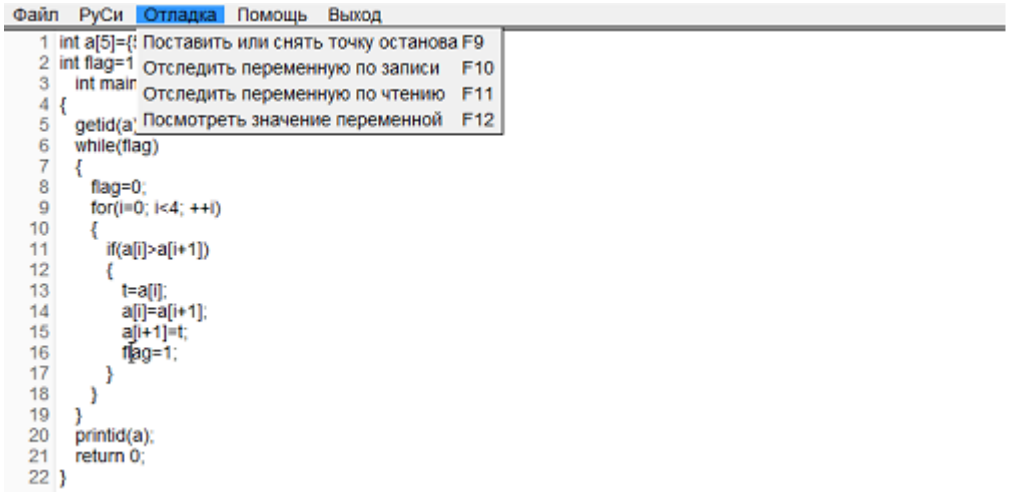

Рис. 2: Новые функции в меню IDE РуСи

Для добавления точек останова в меню пользователя был добавлен новый пункт (пример представлен на рисунках 2 и 3), при нажатии на который или на клавишу F9, строка, где в данный момент стоит курсор, помечается и заносится в список для прерывания.

3.1.2. Выбор переменных для отслеживания

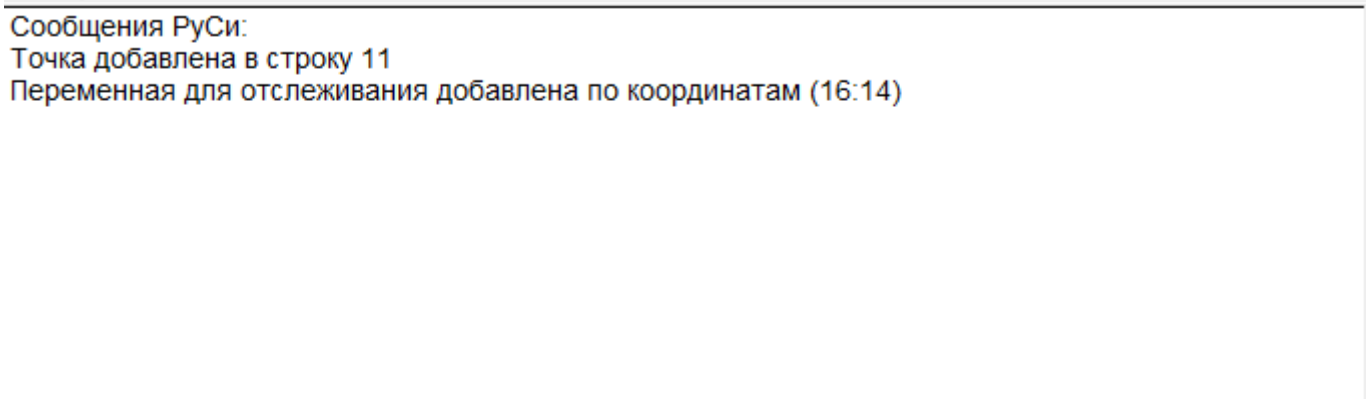

Рис. 3: Добавление точек останова и переменных для отслеживания

Для добавления переменных, которые надо отслеживать, в меню добавлены два пункта (пример представлен на рисунках 2 и 3) – отдельные для отслеживания переменных по записи и по чтению. При добавлении координат переменной в список РуСи выводит соответствующее уведомление.

3.1.3. Остановка исполнения программы

Остановка программы будет происходить в случае исполнения кода из строки, на которой стоит точка останова, или по изменению переменной, выбранной для отслеживания. В первом случае IDE выводит номер строки, на которой произошла остановка и значения переменных, а во втором случае – переменную, из-за которой произошло прерывание, и ее значение.

3.2. Сравнительная таблица возможностей отладчика IDE РуСи и других IDE

Таблица 1: Сравнительная таблица возможностей отладчика IDE РуСи и других IDE

	Точки останова	Трассировка	Условные точки останова	Прерывания по изменению переменной
VS Ultimate 2012	+	+	+	-
Code::Blocks	+	+	+	*
IDE РуСи	+	+	-	+

* – сама среда Code::Blocks не поддерживает такую функцию, но ее поддерживает совместимый с ней отладчик GNU Debugger[2].

Так как РуСи является точным подмножеством языка C, для сравнения были взяты среды разработки, поддерживающие данный язык – Visual Studio Ultimate 2012 (VS 2012)[5] и Code::Blocks[1]. Первая среда

является большим коммерческим продуктом для промышленной разработки с множеством дополнительных функций и расширений, вторая – свободная и бесплатная, поддерживает переносимый отладчик GNU Debugger.

В представленной таблице пятая колонка прерываний по изменению переменной отвечает за возможность отслеживания именно переменной как ее видит пользователь, а не какого-либо количества байт в памяти по определенному адресу.

3.3. Взаимодействия между компонентами транслятора для отладчика

На рисунке 4 представлена UML-диаграмма последовательностей объектов транслятора с учетом действий отладчика. Все действия можно разделить на 4 группы – действия при компиляции, действия при запуске, действия по постановке точек и действия при установке переменных для отслеживания.

Первая группа действий происходит, когда пользователь хочет из своего исходного кода получить код виртуальной машины. При переводе сначала генерируется дерево, каждому узлу которого отладчик сопоставляет номер строки, соответствующий исходному оператору. Также при трансляции из дерева в коды виртуальной машины номера строк передаются сгенерированным по узлам кодам. Причем одному узлу дерева могут соответствовать несколько команд исполняемого кода и для всех них координата будет такая же как для исходного узла дерева. Для всех переменных отладчик составляет двумерную таблицу, соответствующую разметке исходного кода, в которой запоминает все вхождения всех идентификаторов, чтобы по координатам можно было определить, какую переменную хочет отслеживать пользователь.

Вторая группа действий происходит, когда запускает исполнение уже сгенерированного кода виртуальной машины. Для отладчика перед началом исполнения передаются строки, на которые поставлены точки останова и координаты мест, где находятся переменные для от-

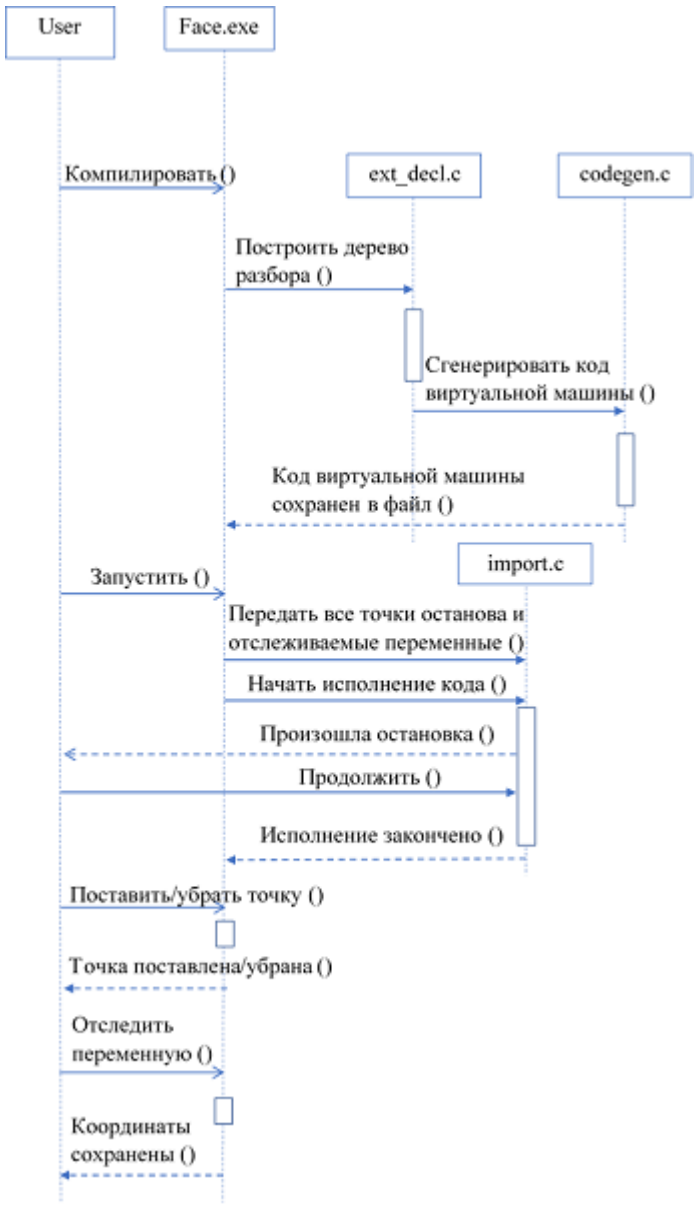

Рис. 4: Все возможные действия внутри транслятора для отладчика (UML-диаграмма последовательностей)

слеживания. По построенной в предыдущем действии таблице отладчик определяет, какие именно переменные пользователь хочет отслеживать и на каких строках остановиться. Во время работы виртуальной машины он отслеживает координаты исполняемых команд и используемые переменные. Если в какой-либо момент исполнение попадает на строку, где стоит точка останова, или использует отслеживаемую переменную, происходит прерывание исполнения кода, пока пользователь не разрешит продолжить.

Для третьей группы действия при постановке или снятию точки останова соответствующая строка добавляется или удаляется из таблицы точек останова.

Аналогичные действия происходят для четвертой группы, но добавляется или удаляется не только строка, но и положение курсора в ней.

4. Реализация

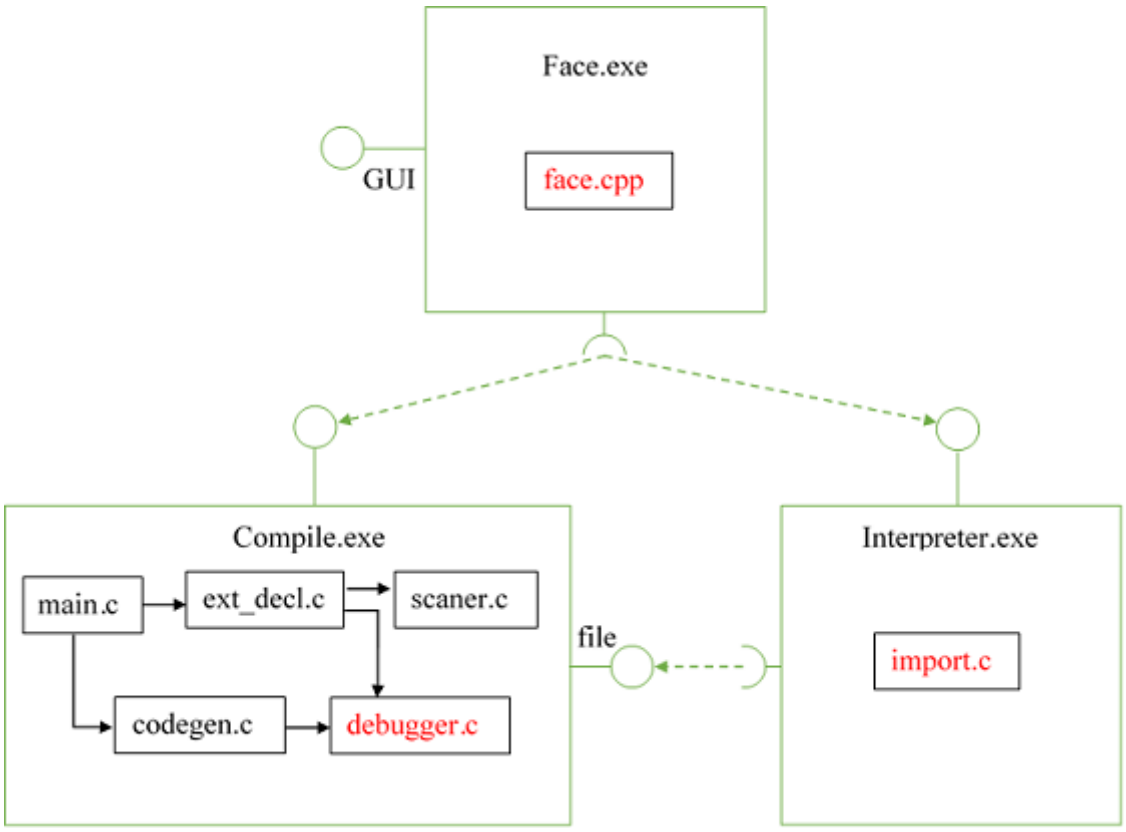

Рис. 5: Отладчик в трансляторе РуСи (UML-диаграмма компонентов)

Отладчик состоит из трех независимых компонентов, обозначенных красным цветом на рисунке 5, но он не делится на два независимых внешних и внутренних отладчика как это принято по стандартам.

Первый компонент – debugger.c отвечает за привязку строк исходного текста к кодам виртуальной машины и местоположениям всех вхождений переменных. Он содержит в себе все функции для добавления номеров строк к узлам дерева, передачи этих данных командам виртуальной машины, добавления координат переменным, разметки блочной структуры.

Второй компонент встроен в face.cpp и отвечает за общение с пользователем, то есть предоставляет интерфейс по постановке и снятию точек останова, выбору переменных для отслеживания и передачу действий пользователя интерпретатору.

Третий компонент является частью интерпретатора (interpreter.c) и

осуществляет сами прерывания в момент исполнения программы. Также он отправляет сообщения через водопровод о типе прерывания и его координаты и переводит виртуальную машину в режим ожидания продолжения исполнения.

4.1. Доработка процедуры парсинга с учетом положения операторов и переменных

Привязка текста для операторов

Так как промежуточное представление кода хоть и является деревом, но хранится не в объектном виде, а в табличном, была добавлена таблица, параллельная таблице tree (пример вырезки из дерева представлен на рисунке 6), где хранится дерево разбора, и для каждого добавляемого узла в дерево во вторую таблицу по тому же индексу добавляется номер строки, которая содержит лексему этого узла.

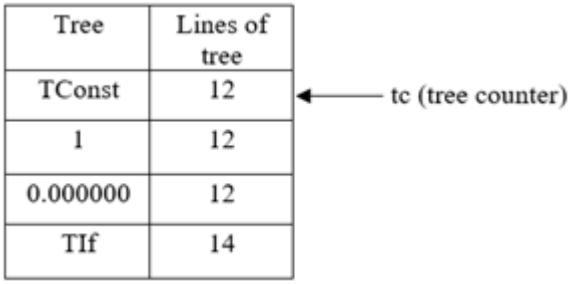

Рис. 6: Соответствие узлов дерева и строк, к которым они относятся

Причем некоторые операторы могут быть растянуты на несколько строк (пример представлен на рисунке 7, где оператор if и его условие находятся на 7 и 8 строках). Их строкой было решено считать строку начала оператора, чтобы не нарушать целостность оператора и останавливаться на нем ровно один раз, даже если точки останова будут стоять на всех строках «растянутого» оператора. Отдельный случай представляют из себя присваивания и арифметические и логические выражения, которые могут быть как отдельными операторами, так и частью ”растянутых”. Для них был введен особый режим, который слабее режима ”растянутого” оператора. То есть, если присваивание находится внутри

объявления for (значит, будет включен режим "растянутого" оператора), то оно будет относится к нему и той строке кода, где начинается описание цикла, а если оно будет стоять отдельно, то будет воспринято как самостоятельный оператор с координатой, соответствующей именно его началу.

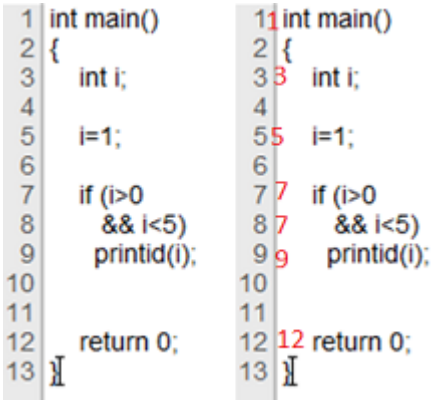

Рис. 7: Пример исходного кода в редакторе и положения его операторов

Чтобы понимать, на каком операторе поставлена точка останова, есть таблица reallinesofcode, индексами которой являются реальные строки, а значениями – номера строк, к которым на самом деле относятся операторы строки по данному индексу. В примере на рисунке серым цветом обозначены индексы, а красным – их значения. Таким образом, для реальной строки 8 все ее операторы относятся к 7, так как именно на ней начинается if.

Привязка координат для каждого вхождения переменной

В трансляторе РуСи есть таблица identab, в которой хранятся все встречающиеся идентификаторы и их смещение. Но эти смещения уникальны только внутри блока, поэтому, чтобы однозначно идентифицировать переменную, встреченную в тексте программы, надо хранить ее смещение и номер блока.

Для отслеживания блоков был реализован новый узел дерева TCuRBlock N, где N – номер текущего блока. Она вставляется в дерево в начала нового блока и его конце, чтобы восстановить предыдущий блок из стека.

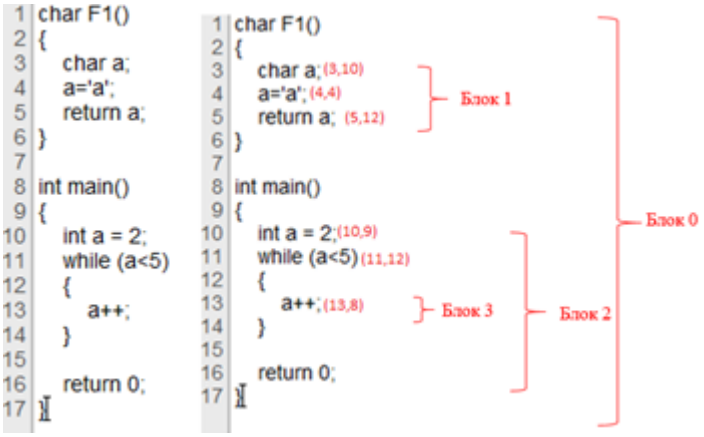

Рис. 8: Пример исходного кода в редакторе и положения его переменных с учетов блоков

Вне функций блок считается под номером 0, далее идет порядковая нумерация по мере нахождения нового блока как показано на рисунке 8.

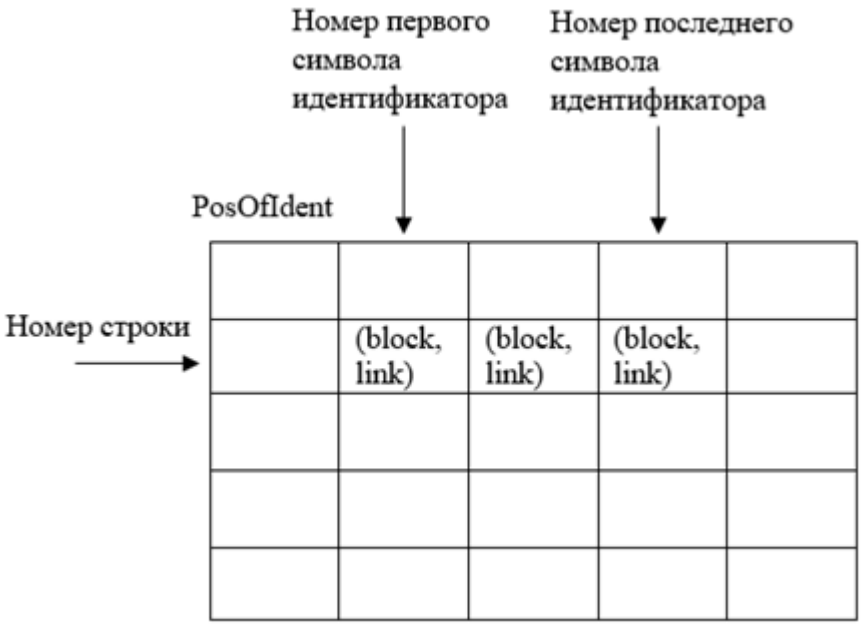

Рис. 9: Пример таблицы положений всех вхождений переменных

Для хранения всех вхождений переменных в исходном коде используется двумерная таблица posofident. При добавлении в нее нового вхождения переменной в ячейки строки, где она находится, с положения ее первого символа до положения последнего вставляется структура вида (block, link), где block – номер текущего блока, а link – ссылка на

identab. В примере на рисунке 9 переменная встретилась во 2 строке со 2 символа и содержала 3 символа.

4.2. Доработка генератора кода виртуальной машины

Отслеживание блочной структуры в коде виртуальной машины

При генерации кода виртуальной машины из дерева разбора для вершины TCurBlock N была добавлена команда CURBLOCK N, которая позволит отслеживать текущий блок при интерпретации программы. При выходе из блока вставляется команда ENDBLOCK, что означает выход в предыдущий блок.

Передача координат операторов из дерева разбора в коды виртуальной машины

Каждый узел дерева переводится в какую-либо команду виртуальной машины. Также и номер строки этого узла будет передаваться в таблицу, теперь уже параллельную кодам виртуальной машины, по индексу новой команды. Некоторые узлы переводятся в несколько команд, тогда хотя бы одной из этих команд передастся номер строки узла, а у остальных номера строк останутся нулями, которые будут игнорироваться в процессе исполнения кода виртуальной машины.

4.3. Дополнение процесса взаимодействий между интерпретатором и IDE

Обработка постановки точки останова

При постановке точки останова, она будет храниться в специальной таблице, индексом которой является строка, на которой точка поставлена. На одну строку нельзя ставить больше одной точки останова.

Только после компиляции программы можно точно определить, на какие операторы действительно поставлены точки останова с помощью

таблицы reallinesofcode и записать их в новую таблицу realbp. Например, если поставить 3 точки на разные строки одного «растянутого» оператора, все точки будут считаться как одна на одном операторе в первой строке этого оператора.

Если остановка невозможна на строке, указанной пользователем (например, в строке только ””, то есть не происходит никаких реальных действий), то точка будет поставлена на ближайшую возможную для остановки строку.

Обработка выбора переменной для отслеживания

При выборе переменной для отслеживания, ее координаты также будут храниться необработанными до компиляции программы.

После компиляции для каждого положения переменной с помощью таблицы posofident будет определен блок и ссылка на identab, которые будут помещены в таблицу текущих отслеживаемых переменных.

Остановка исполнения

В интерпретаторе отслеживается строка предыдущего оператора. Если на новую строку попали в первый раз и она есть в таблице realbp, то надо сделать прерывание исполнения программы, во время которого интерпретатор находится в режиме ожидания продолжения исполнения. Остальные попадания на эту строку игнорируются, если интерпретатор с нее ни разу не сошел.

Для остановки по изменению переменной в интерпретатор были добавлены два параллельных режима – чтения и записи. Для команды LOAD включается режим чтения, для команд простого присваивания – режим записи, для смежных команд (например, ++, += и тому подобных) активируются оба режима. Если в одном из режимов происходит доступ к переменной, то, зная ее смещение и текущий блок, можно определить, находится ли она в таблице текущих отслеживаемых переменных и остановить исполнение программы в случае надобности. Для отслеживания блочной структуры был создан стек блоков, который контролируется командами CURBLOCK и ENDBLOCK. Полезным

побочным эффектом этого стека стала возможность отслеживать вызовы функций.

5. Тестирование

Тестирование проводилось отдельно для точек останова и отдельно для отслеживания переменных по разным методикам. Для проверки использовался набор тестов регрессионного тестирования, написанный специально для РуСи.

Для тестирования точек остановок применялись следующие методики.

- Проверка для всех возможных конструкций языка, таких как объявления и вызовы функций, декларации переменных (простых типов, массивов и структур), присваивания (со всеми возможными арифметическими операциями и использованием стандартных и пользовательских функций), операторы ветвления (if и switch), циклы (for, while, do-while), принудительные переходы (goto, break, continue, return). Далее были выполнены проверки для их всевозможных комбинаций. Особое внимание уделялось операторам, которые могут находиться на нескольких строках, так как все точки останова, поставленные на эти строки, должны считаться за одну.

- Проверка всех возможных режимов. Самый простой режим – однострочный, для операторов состоящих из одного слова. Второй режим – режим арифметического выражения, он может содержать в себе присваивания и декларации переменных. Третий режим – режим сложного оператора, такого как for или if, который является самым сильным и может в себе содержать операторы и операции из первых двух, причем они не должны рассматриваться как отдельные и относиться к строке внешнего оператора, чтобы не было лишних остановок.

Тестирование отслеживания переменных было разделено на 2 этапа – проверка работы отслеживания блочной структуры и проверка самих остановок. Для первого этапа были проверены блоки функций и тела операторов (причем они делились на два вида – те, которые определены в явном виде и обособлены фигурными скобками и состоящие из одного

оператора). На втором этапе были проверены переменные, объявленные и использующиеся в одном блоке, объявленные и использующиеся в разных блоках, а также переменные, имена которых состоят из разного количества символов.

Заключение

В ходе данной работы были получены следующие результаты.

- Разработана архитектура отладчика, включающая в себя режим прерывания по точке останова и по изменению выбранной для отслеживания переменной.
- Реализованы изменения в трансляторе и IDE РуСи для работы отладчика. Добавлен новый компонент в синтаксический анализатор, а также модифицированы интерпретатор и IDE.
- Было проведено "ручное" тестирование по разработанной с учетом функциональных возможностей отладчика методике.

Список литературы

[1] Code::Blocks. Официальный сайт продукта. — URL: http://www.codeblocks.org/ (online; accessed: 18.05.2017).

[2] GDB: The GNU Project Debugger. Официальный сайт продукта. — URL: https://www.gnu.org/software/gdb/ (online; accessed: 18.05.2017).

[3] Jeff Lee. C grammar. — 1985. — URL: https://www.lysator.liu.se/c/ANSI-C-grammar-y.html.

[4] Microsoft. Basic Debugging // Microsoft Developer Network. — URL: https://msdn.microsoft.com/en-us/library/windows/desktop/ms679276(v=vs.85).aspx (online; accessed: 18.05.2017).

[5] Microsoft. Using Breakpoints // Microsoft Developer Network. — URL: https://msdn.microsoft.com/en-us/library/5557y8b4.aspx (online; accessed: 18.05.2017).

[6] Карташев М.С. Двухуровневая схема отладки / Под ред. Терехов А. Системное программирование. Выпуск 1. — 2004.

[7] Керниган Брайан У., Ритчи Деннис М. Язык программирования Си. / Под ред. Керниган Б. — 3-е изд. — СПб.: Невский Диалект, 2001.

[8] Терехов А. Н. Отечественные инструментальные средства обучения программированию. Материалы VI международной конференции Информационные технологии для новой школы. — 2015.

[9] Терехов А. Н. Инструментальное средство обучения программированию и технике трансляции. Компьютерные инструменты в образовании, №1. — 2017.

[10] Терехов А. Н., Терехов М. А. Платформа РуСи для обучения и создания высоконадежных программных систем. Языки програм-

мирования и компиляторы – 2017, Труды конференции, Ростов-на-Дону,. — 2017.