Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Системное программирование

Овсянникова Василиса Павловна

Создание системы регрессионного тестирования для транслятора RuC

Бакалаврская работа

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

Рецензент:
руководитель группы разработки ЗАО «Ланит-Терком» Тиунова А. Е.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Information Systems Administration and Mathematical Support

Software Engineering

Vasilisa Ovsiannikova

Creating regression testing system for RuC translator

Graduation Thesis

Scientific supervisor:
professor Andrey Terekhov

Reviewer:
team lead at Lanit-Tercom CJSC Anna Tiunova

Saint-Petersburg
2016

Оглавление

Введение

Сейчас происходит компьютеризация всех областей деятельности человека, и вполне вероятно, что вскоре умение программировать станет таким же обыденным навыком, как умение читать или писать. Поэтому задача эффективного обучения школьников программированию является очень актуальной.

Учиться программировать трудно, а использование классических учебных языков, таких как Pascal или BASIC, нередко создаёт дополнительную проблему, так как все ключевые слова в этих языках программирования на английском. Для учащихся младших и средних классов языковой барьер может стать серьёзным препятствием - английский на данном этапе они знают плохо, а некоторые так и вовсе могут изучать другой иностранный язык.

Выявив эту проблему, А.Н.Терехов разработал компилятор языка С в коды виртуальной машины с русскими сообщениями, русскими ключевыми словами и русскими идентификаторами и интерпретатор этой машины. Транслятор получил название «РуСи».

На данный момент транслятор RuC реализует часть функционала языка С, достаточную для написания учебных программ, и продолжает развиваться. Усовершенствованием «РуСи» занимается группа студентов под руководством Андрея Николаевича. Однако за всё время своего существования RuC ни разу не подвергался серьёзному всестороннему тестированию, что на нынешнем этапе его развития просто необходимо.

Регрессионное тестирование - это тестирование, направленное на выявление ошибок в уже протестированных участках кода. При изменении программного продукта всегда есть вероятность, что модификации затронут реализованный ранее функционал и негативно скажутся на корректности его работы. Системы регрессионного тестирования отслеживают подобные случаи и сообщают о найденных проблемах,

исключая таким образом деградацию качества работы продукта при росте функциональности.

Система регрессионного тестирования представляется удобным инструментом проверки качества в случае разработки транслятора. Если создать набор тестов, охватывающий все реализованные в RuC конструкции языка C, то их успешное прохождение будет гарантировать стабильную работу транслятора после внесения в него изменений.

1. Цель и постановка задачи

Цель данной работы - создание системы регрессионного тестирования для транслятора RuC.

Задача данной работы:

- Разработать среду для тестирования транслятора RuC;
- Изучить конструкции языка C, реализованные в трансляторе RuC и планирующиеся к реализации;
- Создать набор тестов, охватывающий все изученные в предыдущем этапе конструкции.

2. Существующие решения

Существует множество готовых инструментов для регрессионного тестирования.

IBM Rational Functional Tester[3] предоставляет функции автоматического тестирования для функционального, регрессионного тестирования, тестирования графических пользовательских интерфейсов и тестирования, ориентированного на данные. Для данной работы этот инструмент не подходит, во-первых, из-за громоздкости (необходимо потратить время на изучение функционала) и, во-вторых, из-за цены, несоразмерной поставленной задаче.

У компании IBM есть также IBM Rational Test Workbench[4] с возможностью тридцать дней пользоваться им бесплатно, но и это решение отвергается ввиду своей объёмности. К тому же неизвестно, подойдёт ли оно для тестирования транслятора.

Существует система автоматизированного тестирования TestComplete[5], предоставляющая в том числе инструмент для регрессионного тестирования. Однако и она чересчур массивна для поставленной задачи.

Одна из главных сложностей регрессионного тестирования - то, что в больших системах на проверку всего функционала уходит куча времени, а ошибок находится не так много, около 6-15% [1] из всех найденных багов приходится на регрессионные тесты. Поэтому предлагаются различные методы выделения из всей массы тестов тех, которые имеет смысл прогонять, на которые могли повлиять внесённые в программный продукт изменения. Однако транслятор RuC вовсе не огромная многомодульная система, конструкций в языке C не так много, чтобы базовые тесты на них прогонялись по несколько суток.

Таким образом, в результате проведённого обзора можно сделать вывод, что уже существующие решения по тем или иным причинам для

данной работы не подходят и самым рациональным вариантом является создание собственной узко направленной системы тестирования конкретно для транслятора RuC и ручное написание набора тестов.

3. Реализация

3.1. Разработка системы тестирования

На языке C был разработан программный модуль для регрессионного тестирования транслятора RuC. Он работает таким образом:

- Транслирует каждый тест из директории;
- Печатает в файл выходные данные;
- Сравнивает результаты тестов с ожидаемыми;
- Если находит несоответствие - выводит название теста с ошибкой.

Для реализации тестирования были внесены изменения в исходный код транслятора: если раньше при ошибке компиляции он вылетал с кодом ошибки, то теперь заканчивает свою работу, чтобы можно было отметить тест как неудачный и продолжить смотреть оставшиеся тесты в директории.

3.2. Разработка тестов

В ходе работы были изучены реализованные и планирующиеся к реализации в трансляторе конструкции языка C, в связи с чем были созданы следующие тесты:

%.c - проверяет работу операции %, нахождения целочисленного остатка от деления, в случаях статических и динамических переменных, в случаях наличия остатка и остатка, равного нулю, в случае, когда первый аргумент больше второго, и в обратном.

%=.c - проверяет работу операции %=, нахождения целочисленного остатка от деления и присваивания первому аргументу результата, в случаях статических и динамических переменных, в случаях наличия остатка и остатка, равного нулю, в случае, когда первый аргумент больше второго, и в обратном.

-.c - проверяет работу операции -, нахождения разности, в случаях статических и динамических переменных, в случаях целочисленных типов и типов с плавающей точкой, в случае, когда первый аргумент больше второго, и в обратном.

-=.c - проверяет работу операции -=, нахождения разности и присваивания первому аргументу результата, в случаях статических и динамических переменных, в случаях целочисленных типов и типов с плавающей точкой, в случае, когда первый аргумент больше второго, и в обратном.

++--.c - проверяет работу операций ++, инкремента, и --, декремента, в случаях статических и динамических переменных, в случаях целочисленных типов и типов с плавающей точкой, в случаях префиксного и суффиксного использования.

+.c - проверяет работу операции +, нахождения суммы, в случаях статических и динамических переменных, в случаях целочисленных типов и типов с плавающей точкой.

+=.c - проверяет работу операции +=, нахождения суммы и присваивания первому аргументу результата, в случаях статических и динамических переменных, в случаях целочисленных типов и типов с плавающей точкой.

=.c - проверяет работу операции =, присваивания, в случаях присваивания константного значения и выражения.

b=.c - проверяет работу оператора >=, проверки, меньше ли второй аргумент первого, в случаях статических и динамических переменных, в случаях целочисленных типов, типов с плавающей точкой и символьных типов, в случае равенства аргументов, в случае, когда первый аргумент больше второго, и в обратном.

bigger.c - проверяет работу оператора >, проверки, больше ли первый аргумент второго, в случаях статических и динамических переменных, в случаях целочисленных типов, типов с плавающей точкой и символьных типов, в случае равенства аргументов, в случае, когда первый аргумент больше второго, и в обратном.

cc.c - проверяет работу операторов сдвига > > и < <.

cc=.c - проверяет работу операций сдвига и присваивания первому аргументу результата, > >= и < <=.

declaration.c - проверяет работу механизма объявления переменных и массивов в случаях присваивания значения при объявлении и присваивания значения после объявления, в случаях конечных и бесконечных массивов, в случаях присваиваивания константного значения и выражения логического и арифметического.

div.c - проверяет работу операции /, деления, в случаях статических и динамических переменных, в случаях целочисленных типов и типов с плавающей точкой.

div=.c - проверяет работу операции /=, деления и присваивания первому аргументу результата, в случаях статических и динамических переменных, в случаях целочисленных типов и типов с плавающей точкой.

dowhile.c - проверяет работу конструкции dowhile в случаях конечного количества шагов цикла до выполнения условия выхода из цикла, изначально выполненного условия выхода из цикла, недостижимого условия выхода из цикла и выхода с помощью break.

enum.c - проверяет работу типа enum в случаях присвоения численных значений автоматически и самостоятельно, в том числе в случае соответствия одного числа нескольким элементам одного перечня.

expressions.c - проверяет работу с выражениями: правильность порядка вычислений в скобочных выражениях и корректность работы с элементами массива, чей порядковый номер является выражением.

for.c - проверяет работу цикла for в случаях наличия и отсутствия каждого из трёх условий цикла, в случаях нулевого, конечного и бесконечного числа шагов цикла, в случаях наличия и отсутствия операторных скобок в теле цикла.

if.c - проверяет работу конструкции if в случаях наличия и отсутствия else, в случаях выполнения и невыполнения условия, в случаях наличия и отсутствия операторных скобок.

jumpstatement.c - проверяет работу операторов перехода break, continue в случаях выполнения и невыполнения условий перехода, проверяет работу оператора return в функциях, возвращающих и не возвращающих результат.

label.c - проверяет работу оператора перехода на метку goto с помощью релизованного ею цикла.

logic.c - проверяет работу логических операций: & (И), | (ИЛИ) и ^ (исключающее ИЛИ) в случаях истинности и ложности утверждений, а также работу операций && и ||, проверяющих единовременное выполнение всех (&&) или хотя бы одного (||) из перечня условий.

logic=.c - проверяет работу логических операций с присваиванием: &= (И), |= (ИЛИ) и $\hat{=}$ (исключающее ИЛИ) в случаях истинности и ложности утверждений.

math.c - проверяет работу математических функций: abs (взятие модуля) в случае положительного и отрицательного значения аргумента, sqrt (извлечение квадратного корня), exp (значение экспоненты в

степени аргумента), log (натуральный логарифм), log10 (десятичный логарифм), cos, sin, asin (вычисление арксинуса).

sm=.c - проверяет работу оператора <=, проверки, больше ли второй аргумент первого, в случаях статических и динамических переменных, в случаях целочисленных типов, типов с плавающей точкой и символьных типов, в случае равенства аргументов, в случае, когда первый аргумент больше второго, и в обратном.

smaller.c - проверяет работу оператора <, проверки, меньше ли первый аргумент второго, в случаях статических и динамических переменных, в случаях целочисленных типов, типов с плавающей точкой и символьных типов, в случае равенства аргументов, в случае, когда первый аргумент больше второго, и в обратном.

star.c - проверяет работу операции *, умножения, в случаях статических и динамических переменных, в случаях целочисленных типов и типов с плавающей точкой, в случаях положительных и отрицательных аргументов.

star=.c - проверяет работу операции *=, умножения аргументов и присваивания первому аргументу результата, в случаях статических и динамических переменных, в случаях целочисленных типов и типов с плавающей точкой, в случаях положительных и отрицательных аргументов.

struct.c - проверяет работу структур в случае объявления элемента типа структуры при объявлении структуры вне функции main, в случае объявления элемента типа структуры в теле функции main, в случае объявления безымянной структуры.

switch.c - проверяет работу конструкции switch в случаях наличия и отсутствия default, в случаях наличия и отсутствия break.

ternar.c - проверяет работу тернарного оператора (условие) ? (значение, если условие выполнено) : (значение, если условие не выполнено).

typedef.c - проверяет работу механизма переопределения ключевых слов и констант typedef.

types().c - проверяет корректность приведения типов float и int друг к другу.

types.c - проверяет корректность вывода и определения типов int, float, void и char.

unar.c - проверяет работу оператора (отрицание), а также проверки равенства == и != в случаях истинности и ложности утверждений.

while.c - проверяет работу цикла while в случаях нулевого, конечного и бесконечного числа шагов цикла, а также работу continue и break внутри этого цикла.

Все тесты выполнены таким образом, что сначала выводится результат выполнения теста, а потом ожидаемый результат.

Для создания тестов использовались исходный код транслятора и стандарт языка C[2].

4. Результаты

В полученном модуле для тестирования транслятора RuC был запущен созданный в ходе работы набор тестов, что позволило выявить несколько ошибок в исходном коде транслятора, например, в операции %=. К тому же ошибки в трансляторе находились и исправлялись в процессе работы над тестировочной средой.

В будущем разработанная система поможет поддерживать стабильность работы транслятора при внесении в него изменений.

Заключение

В рамках данной работы были получены следующие результаты:

- Разработана среда для тестирования транслятора RuC;
- Изучены конструкции языка C, реализованные в трансляторе RuC и планирующиеся к реализации;
- Создан набор тестов, охватывающий изученные конструкции.

Список литературы

[1] Bolton Michael. Things Could Get Worse: Ideas About Regression Testing // Develop Sense. — 2013. — URL: http://www.developsense.com/presentations/2013-05-STAREast-Regression.pdf (online; accessed: 14.05.2016).

[2] Degener Jutta. ANSI C Yacc grammar // Lysator, the Academic Computer Society. — 1995. — URL: http://www.lysator.liu.se/c/ANSI-C-grammar-y.html (online; accessed: 12.12.2015).

[3] IBM. IBM Rational Functional Tester // IBM.

[4] IBM. IBM Rational Test Workbench // IBM.

[5] SmartBear. TestComplete Platform // SmartBear.