Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _Информатика и системы управления_

КАФЕДРА _Программное обеспечение ЭВМ и информационные технологии» (ИУ7)_

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

Компилятор Smalltalk

Студент **ИУ7-21М** **Игнатьев А.И.**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **Ступников А.А.**
(Подпись, дата) (И.О.Фамилия)

2022 г.

Оглавление

РЕФЕРАТ

Расчетно-пояснительная записка 35 с., 2 рис., 10 источников, 5 прил.

Цель данной работы – разработать компилятор языка Smalltalk для платформы LLVM.

Задачи работы:

- Проанализировать способы и алгоритмы построения компилятора;
- Выбрать инструменты для разработки компилятора;
- Спроектировать компилятор;
- Реализовать компилятор.

В первой части работы проанализированы алгоритмы и инструменты, используемые при разработке компиляторов, описаны особенности языка Smalltalk и платформы LLVM. Во второй части описаны способы реализации составляющих компилятора. В третьей части описаны сборка и использование разработанного компилятора.

Введение

На данный момент большинство программ пишутся на языках высокого уровня. Они удобны тем, что абстрагируют программиста от многих базовых условностей и позволяют писать код быстрее и чище, а также во многих случаях не заботиться о совместимость с платформой. Однако, чтобы код, написанный на этих языках, работал, его нужно перевести в код, воспринимаемый платформой, на которой программа будет работать. Для этого используются компиляторы.

Компилятор – комплексная программа, которая анализирует код, написанный на некотором языке программирования, и преобразует его в полностью эквивалентный код на целевом языке. Целевым языком в данном случае может быть языком ассемблера, байткодом или другим высокоуровневым языком. При этом компилятор должен обнаруживать ошибки в коде и сообщать о них.

Целью данной работы является разработка компилятора языка Smalltalk для платформы LLVM.

1. Аналитический раздел

В данном разделе описана общая архитектура компилятора, особенности языка Smalltalk и платформы LLVM.

1.1. Архитектура компилятора

Компилятор – это программа, которая считывает текст программы, написанной на исходном языке, и транслирует его в эквивалентный текст на целевом языке. Этот процесс делится на стадии анализа и синтеза. Анализ разбирает текст программы на составные части и строит ее грамматическую структуру, в результате чего обнаруживаются ошибки и строится промежуточное представление программы. Синтез строит программу на целевом языке на основе полученного промежуточного представления. Анализ и синтез называют фронтендом и бэкендом компилятора [1].

Процесс компиляции состоит из фаз. Рисунок 1 изображает разложение компилятора на фазы. Основных фаз 4: лексический анализ, синтаксический анализ, семантический анализ и генерация кода [1].

1.1.1. Лексический анализ

Лексический анализ заключается в чтении потока символов, составляющих исходную программу, и преобразовании этих символов в значащие последовательности – лексемы. Для каждой лексемы анализатор составляет токен. В простых случаях понятия лексемы и токена синонимичны, однако иногда лексемы дополнительно разбиваются на классы [1].

Правила разбиения текста программы на лексемы могут быть заданы с помощью грамматики языка. Существуют и другие способы их задания, например, регулярные выражения, однако на практике чаще используются грамматики. Грамматика задает определенный набор лексем, которые могут встретиться в тексте программы.

В процессе лексического анализа обнаруживаются простейшие ошибки, такие как недопустимые символы или неправильные последовательности [1].

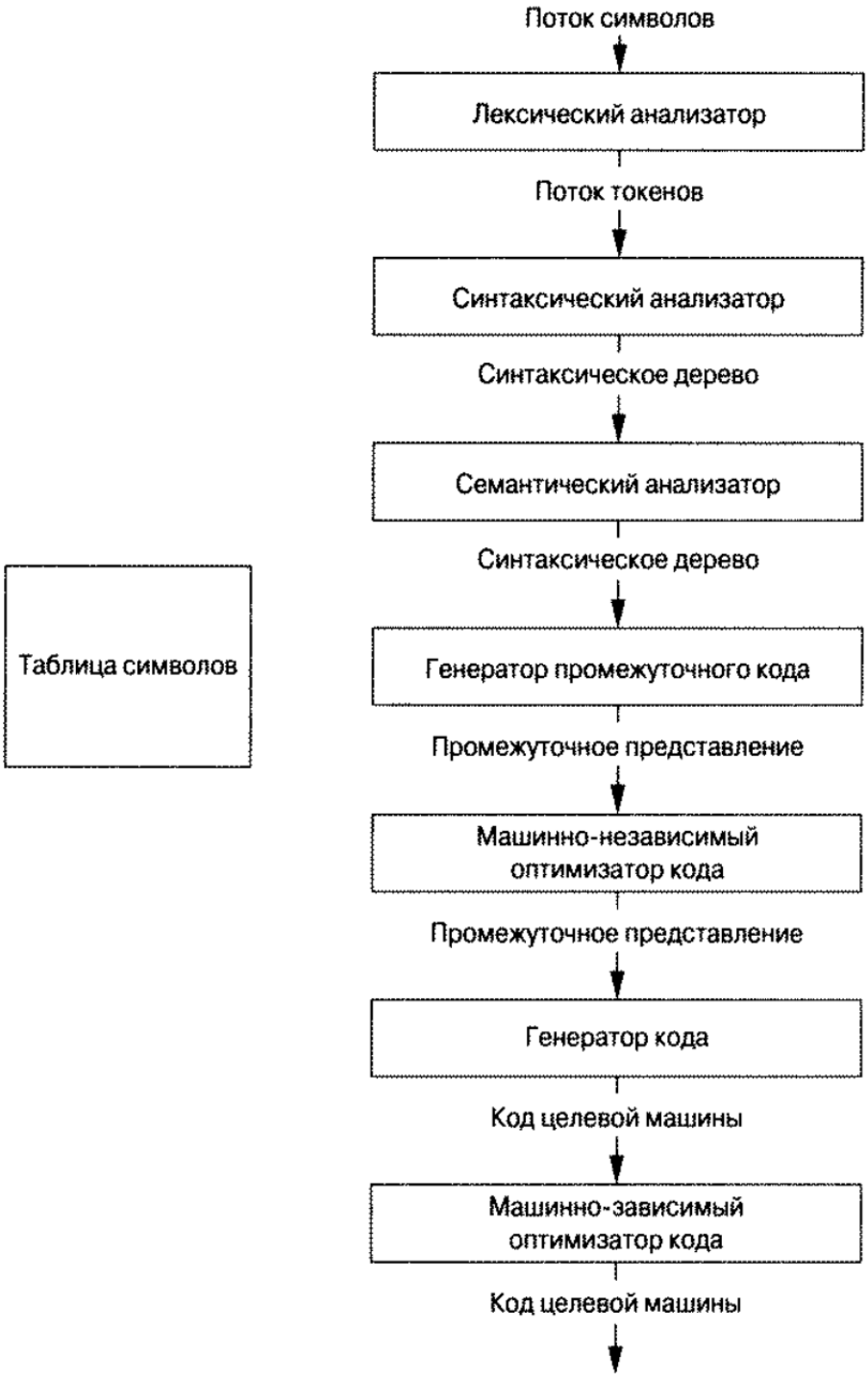

Рисунок 1. Фазы компиляции

Лексический анализ осуществляется программой (или модулем) – лексическим анализатором. Анализатор принимает на вход поток символов

программы и выдает на выходе поток лексем, сопоставленных с токенами. Поток токенов затем подается на вход синтаксического анализатора.

1.1.2. Синтаксический анализ

При синтаксическом анализе входной поток лексем преобразуется в древовидное представление в соответствии с грамматикой языка. Полученное дерево называется деревом разбора. Вершины такого дерева представляют собой операции, а дочерние узлы – аргументы операции [2].

Алгоритмы синтаксического анализа можно разделить на два основных типа: алгоритмы восходящего и нисходящего анализа.

При нисходящем анализе входная цепочка разбирается, начиная со стартового символа грамматики и заканчивая требуемой последовательностью токенов. Нисходящий анализ применим к LL контекстно-свободным грамматикам. LL означает, что разбор входящей цепочки производится слева направо и строится левый вывод.

При восходящем анализе дерево строится от листьев к корню. Сначала распознаются низкоуровневые структуры, затем объединяющие их более высокоуровневые, и так до корня дерева. Одним из примеров восходящего анализа является алгоритм «перенос/свертка». Это алгоритм применяется для LR-грамматик. LR означает, что они читают входящий поток слева направо и строят правый вывод. Синтаксические анализаторы для таких грамматик называются LR-анализаторами.

Поскольку многие языки программирования могут быть представлены LR-грамматикой, при создании компиляторов часто используются LR-анализаторы. В зависимости от способа создания таблицы анализа такие анализаторы можно разделить на SLR-анализаторы (простые LR-анализаторы), LALR-анализаторы (LR-анализаторы с предпросмотром) и канонические LR-анализаторы. На данный момент, как правило, используются LALR-анализаторы [1].

1.1.3. Семантический анализ

На этапе семантического анализа построенное синтаксическим анализатором дерево разбора обрабатывается с целью интерпретации распознанных конструкций и их связей друг с другом. На данном этапе также формируется связь переменных и соответствующих им данных.

Семантический анализ позволяет перестроить дерево разбора с учетом смысла конструкций, а также распознать такие ошибки, как:

- Несоответствие типов;
- Необъявленные переменные;
- Множественные объявления переменной;
- Отсутствие переменной в области видимости;
- Несоответствие параметров.

В результате семантического анализа дерево приводится к виду, пригодному для генерации кода [3].

1.1.4. Генерация кода

Генерация кода осуществляется путем нисхождения по полученному на предыдущих стадиях дереву разбора. Для каждого узла дерева генерируется соответствующий операции узла код на языке целевой платформы. При этом выделенные в ходе анализа кода программы данные связываются с именами переменных. При выполнении генерации кода предполагается, что вход генератора не содержит ошибок. Результатом генерации является код, пригодный для исполнения на целевой платформе (ассемблерный код, байткод и т.д.) [1].

1.2. Средства лексического и синтаксического анализа

Лексический и синтаксический анализаторы разрабатываются на основе грамматики языка. Разрабатывать анализаторы можно вручную, однако

существуют инструменты для автоматического создания анализаторов путем генерации кода анализаторов по предоставленной грамматике в том или ином формате.

1.2.1. Генераторы лексических анализаторов

Наиболее популярными инструментами для генерации лексических анализаторов являются ANTLR и lex с его вариацией flex.

Lex – это стандартный генератор в среде Unix. Он получает на вход специальный lex-файл, и по содержащейся в нем информации о языке генерирует код лексического анализатора на языке Си. Lex-файл имеет следующий формат:

```
Определения
%%
правила
%%
подпрограммы, составленные
пользователем
```

В результате своей работы lex генерирует файл lex.yy.c, содержащий код анализатора [4].

Flex (fast lexical analyzer) – аналог lex в системах на базе пакетов GNU. По функционалу и формату входа и выхода полностью аналогичен lex.

ANTLR – инструмент для генерации парсеров по описанию грамматики языка. На вход генератора подается грамматика языка в специальном формате, и по этой грамматике создаются файлы, содержащие лексический и синтаксический анализатор, а также файлы с простейшими инструментами обхода деревьев разбора, которые можно дополнять самостоятельно. ANTLR способен генерировать анализаторы на многих языках, включая C++, Java, JS/TS и другие [5].

1.2.2. Генераторы синтаксических анализаторов

Для генерации синтаксических анализаторов используются такие инструменты, как Bison, Coco/R, yacc и описанный в предыдущем разделе ANTLR.

Yacc и bison – генераторы парсеров, которые конвертируют поданную на вход грамматику в код парсера на языке Си. Yacc – стандартный генератор синтаксических парсеров в Unix системах, bison – аналогичный генератор для GNU систем. Как правило, используются в связке с lex или flex [6].

Coco/R – генератор лексических и синтаксических анализаторов, работающий с грамматиками, описанными в расширенной форма Бэкуса-Наура. Лексические анализаторы работают по принципу конечных автоматов, а синтаксические анализаторы используют рекурсивный спуск. Присутствует поддержка языков C++, C#, Java и других языков [7].

1.3. Язык Smalltalk

Smalltalk – объектно-ориентированный язык программирования с динамической типизацией, в основе которого лежит принцип посылки сообщений. Основные идеи языка включают в себя следующее:

- Всё – объекты, взаимодействие которых осуществляется через посылку сообщений. У объектов есть методы и состояние. Любому объекту можно отправить любое сообщение. При получении объектом сообщения выбирается выбирает подходящий метод для его обработки, и, если такого метода нет, вызывается специальный метод для обработки неопознанных сообщений. Объект сам определяет, как обработать сообщение.
- Все доступно для изменения, в том числе исполняемая среда и, в некоторых реализациях, синтаксис языка. Это происходит в процессе исполнения программы без остановки или перезапуска.
- Динамическая типизация делает код лаконичнее.

К особенностям языка относится также то, что управляющие конструкции (ветвление, циклы и т.д.) не являются частью языка и встроены в объекты в виде сообщений. Частью языка являются присваивание, посылка сообщения (возможно, с данными), возвращение объекта из метода и конструкции объявления переменных и литералов [8].

Простейший пример кода на Smalltalk:

```
Transcript show: 'Hello, world!'
```

Пример показывает, как посылается сообщение `show: 'Hello, world!'` объекту `Transcript`. Для его обработки будет вызван метод show: объекта Transcript. Это же пример показывает основной синтаксис посылки сообщения:

<получатель сообщения> <пробел> <сообщение>

На данный момент язык имеет множество реализаций. Для данной работы взята грамматика реализации Redline Smalltalk. Грамматика представлена в приложении А.

1.4. LLVM

LLVM – проект программной инфраструктуры для создания компиляторов. В основе инфраструктуры лежит платформонезависимая система кодирования машинных инструкций – байткод LLVM IR (Intermediate Representation). LLVM может создавать байткод для множества платформ, включая ARM, x86, x86-64, GPU от AMD и Nvidia и другие. В проекте есть генераторы кода для множества языков, а для компиляции LLVM IR в код платформы используется clang. В состав LLVM входит также интерпретатор LLVM IR, способный исполнять код без компиляции в код платформы, хотя скорость исполнения в данном случае будет ниже [9].

В данной работе реализуется генерация кода LLVM IR с помощью API языка разрабатываемого компилятора.

2. Конструкторский раздел

В данном разделе описаны способы реализации составляющих компилятора.

2.1. Лексический и синтаксический анализаторы

Лексический и синтаксический анализатор в данной работе генерируются с помощью ANTLR. На вход генератора поступает грамматика языка в формате ANTLRv4 (файлы *.g4). Проект Redline Smalltalk содержит грамматику в нужном формате.

В результате работы создаются файлы, содержащие классы лексера и парсера, а также вспомогательные файлы и классы для их работы. Помимо этого, генерируются шаблоны классов для обхода дерева разбора, которое получается в результате работы парсера.

На вход лексера подается текст программы, преобразованный в поток символов. На выходе получается поток токенов, который затем подается на вход парсера. Поток токена можно формировать вручную, реализуя соответствующий интерфейс. Результатом работы парсера является дерево разбора.

Ошибки, возникающие в ходе работы лексера и парсера, выводятся в стандартный поток вывода.

2.2. Дерево разбора и семантический анализ

Дерево разбора можно обходить двумя способами: паттерн listener и паттерн visitor. Listener автоматически обходит дерево в глубину и вызывает обработчики соответствующих событий при входе и выходе из узла дерева. В коде обработчиков осуществляется анализ соответствующих узлов и выполнение связанных с ними действий. Visitor позволяет более гибко обходить дерево и решать, какие узлы и в каком порядке нужно посетить. Для каждого узла реализуется метод посещения, который при необходимости

вызывается из метода посещения родительского узла. Обход начинается с вызова метода посещения корневого узла.

Язык Smalltalk обладает рядом особенностей, поэтому в данной работе были произведены семантический анализ и перестройка дерева разбора, чтобы оно было более удобно для предстоящей генерации кода. В ходе анализа дерево упрощается и приводится ближе к структуре языка. На этом же этапе имена переменных связываются с узлами дерева, отвечающими за присвоенные в эти переменные объекты, устанавливается их тип, обнаруживаются ошибки, связанные с отсутствием объявления переменной.

Интерфейсы узлов результирующего AST, а также его пример в формате JSON приведены в приложениях Б и В.

2.3. Генерация кода

Генерация кода происходит после окончания формирования AST. Узлы дерева содержат метод codegen, который генерирует код LLVM IR в переданном ему контексте. Ввиду неполноты разрабатываемого компилятора и отсутствия описаний методов стандартных объектов Smalltalk в работе созданы вспомогательные объекты методов, которые также содержат метод codegen, создающий код этих методов.

В процессе генерации кода AST обходится в глубину. При посещении узла определяется его тип и вызывается метод codegen. Если узел представляет собой вызов метода, вызывается метод codegen у объекта метода. Контекст генератора сохраняет стек базовых блоков и объект текущей функции: новые инструкции добавляются к текущему блоку, а новые блоки создаются в текущей функции. Глобальные переменные объявляются при посещении узлов с объявлением и также сохраняются в контексте для использования в операциях с ними.

При генерации кода используется библиотека llvm-bindings. Она представляет собой TypeScript-версию LLVM IR API для C++ и предоставляет

аналогичны набор методов генерации конструкций LLVM IR, оптимизированный для использования в TypeScript [10].

Примеры кода методов codegen для генерации статического массива (класс STLiteralArray) и генерации метода ‘to:do:’ приведены в приложении Г.

Сгенерированный код LLVM IR верифицируется встроенными методами API. Отдельно верифицируются все функции и модуль целиком. В процессе верификации выявляются ошибки в написании кода компилятора, приводящие к генерации неверного кода LLVM IR.

По окончании генерации и верификации код записывается в файл, который является результатом работы всего компилятора.

3. Технологический раздел

В данном разделе описаны детали сборки и развертывания разработанного компилятора, а также приведен пример его работы.

3.1. Сборка компилятора

Компилятор написан на языке TypeScript и работает на платформе Node.js. Перед работой с ним в операционной системе Windows 10 нужно установить cmake, скачать или собрать LLVM и выполнить предварительные шаги по установке пакета llvm-bindings [10]. После этого установить зависимости командой npm i.

Сборка компилятора осуществляется командой tsc. В результате будет сгенерирован набор js файлов, содержащих код компонентов компилятора.

3.2. Запуск компилятора

Для запуска компилятора нужно запустить файл index.js в среде node. Для этого используется команда:

```
node index -i input.st -o output.ll
```

Для быстрой сборки и запуска компилятора можно использовать команду:

```
npm run start -- -i input.st -o output.ll
```

Флаг -i указывает имя входного файла, флаг -o – выходной файл с кодом LLVM IR.

Результирующий файл затем можно скомпилировать в код платформы посредством clang или запустить в интерпретаторе lli.

3.3. Пример использования

В качестве примера рассматривается код на языке Smalltalk, который выполняет сортировку массива чисел методом пузырька с выводом результата в консоль:

```
|item swap itemCount hasChanged fact|
```

```
item := #(1, 5, -3, 7, -32, 8, 4, 90, 2, -4, 2, 0, -7).

swap :=
    [:indexOne :indexTwo |
      |temp|
      temp := item at: indexOne.
      item at: indexOne put: (item at: indexTwo).
      item at: indexTwo put: temp
      ].

itemCount := item size.
[hasChanged := false.
itemCount := itemCount - 1.
1 to: itemCount do:
      [:index |
      (item at: index) > (item at: index + 1) ifTrue:
            [swap value: index value: index + 1.
            hasChanged := true]].
hasChanged] whileTrue.

1 to: item size do:
    [:index |
    (item at: index) printfInt.
].
```

Результатом компиляции представленной программы в LLVM IR будет код, приведенный в приложении Д. Каждый блок кода, не присвоенный в переменную, генерируется в анонимную функцию. Переменные сделаны глобальными в связи с необходимостью реализации контекста и доступа к переменным из контекста.

Запуск скомпилированного кода с помощью интерпретатора lli производится командой:

```
path/to/llvm/bin/lli.exe output.ll
```

Рисунок 2 демонстрирует результат работы скомпилированной программы.

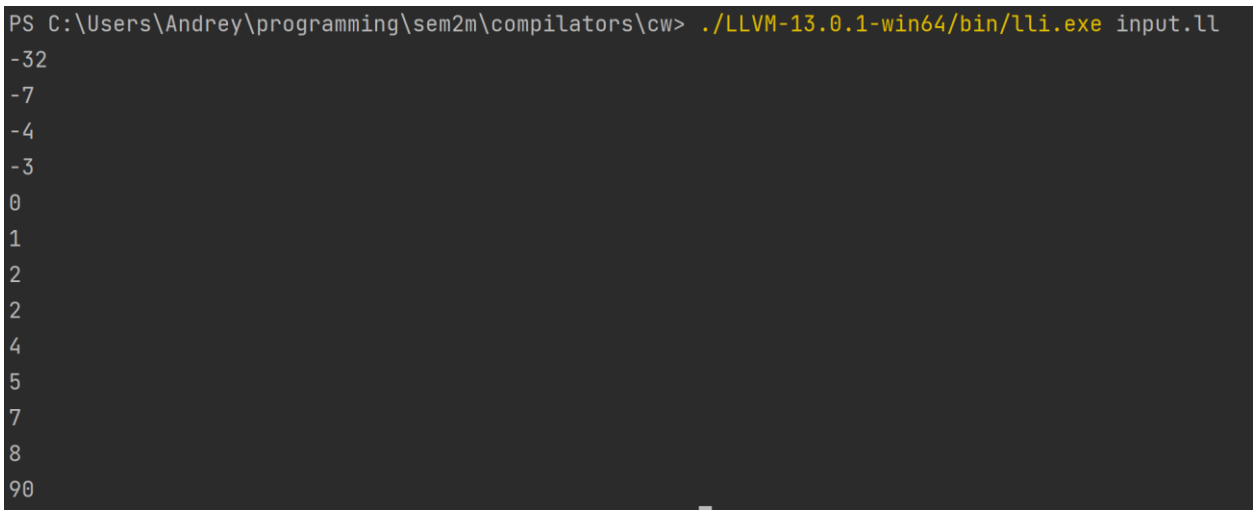

Рисунок 2. Результат работы программы

Заключение

В данной работе рассмотрены основные части компилятора, алгоритмы и способы их реализации. Были рассмотрены инструменты генерации лексических и синтаксических анализаторов, спроектирована структура AST для реализации стадий семантического анализа и генерации кода.

По результатам работы был разработан компилятор языка Smalltalk для платформы LLVM. В ходе работы были сгенерированы лексический и синтаксический анализаторы по грамматике языка с помощью инструмента ANTLR. Получаемое при работе синтаксического анализатора дерево разбора проходит через семантический анализатор и перестраивается для более удобной генерации кода. После построения дерева по нему генерируется код целевой платформы с использованием пакета llvm-bindings для языка TypeScript.

Список литературы

[1] А. Ахо, Р. Сети и Д. Ульман, Компиляторы. Принципы, технологии, инструменты, М.: Издательский дом "Вильямс", 2008.

[2] А. Ахо и Д. Ульман, Теория синтаксического анализа, перевода и компиляции, т. 1, Москва: Издательство "Мир", 1978.

[3] «Дизайн компилятора - семантический анализ,» 28 06 2018. [В Интернете]. Available: https://coderlessons.com/tutorials/akademicheskii/izuchite-dizain-kompiliatora/dizain-kompiliatora-semanticheskii-analiz. [Дата обращения: 05 06 2022].

[4] «Генератор программ лексического анализа lex,» [В Интернете]. Available: http://lib.ru/MAN/DEMOS210/lex.txt. [Дата обращения: 05 06 2022].

[5] «About The ANTLR Parser Generator,» [В Интернете]. Available: https://www.antlr.org/about.html. [Дата обращения: 05 06 2022].

[6] «Bison - GNU Project - Free Software Foundation,» [В Интернете]. Available: https://www.gnu.org/software/bison/. [Дата обращения: 05 06 2022].

[7] «The Compiler Generator Coco/R,» [В Интернете]. Available: https://ssw.jku.at/Research/Projects/Coco/. [Дата обращения: 05 06 2022].

[8] Р. Себеста, Основные концепции языков программирования, М., 2001.

[9] «The LLVM Compiler Infrastructure Project,» [В Интернете]. Available: https://llvm.org/. [Дата обращения: 05 06 2022].

[10] ApsarasX, «LLVM Bindings for Node.js/JavaScript/TypeScript,» [В Интернете]. Available: https://github.com/ApsarasX/llvm-bindings. [Дата обращения: 05 06 2022].

Приложение А

```
grammar Smalltalk;

script : sequence ws EOF;
sequence : temps? ws statements? ws;
ws : (SEPARATOR | COMMENT)*;
temps : ws PIPE (ws IDENTIFIER)+ ws PIPE;
statements : answer ws # StatementAnswer
           | expressions ws PERIOD ws answer # StatementExpressionsAnswer
           | expressions PERIOD? ws # StatementExpressions
           ;
answer : CARROT ws expression ws PERIOD?;
expression : assignment | cascade | keywordSend | binarySend | primitive;
expressions : expression expressionList*;
expressionList : PERIOD ws expression;
cascade : (keywordSend | binarySend) (ws SEMI_COLON ws message)+;
message : binaryMessage | unaryMessage | keywordMessage;
assignment : variable ws ASSIGNMENT ws expression;
variable : IDENTIFIER;
binarySend : unarySend binaryTail?;
unarySend : operand ws unaryTail?;
keywordSend : binarySend keywordMessage;
keywordMessage : ws (keywordPair ws)+;
keywordPair : KEYWORD ws binarySend ws;
operand : literal | reference | subexpression;
subexpression : OPEN_PAREN ws expression ws CLOSE_PAREN;
literal : runtimeLiteral | parsetimeLiteral;
runtimeLiteral : dynamicDictionary | dynamicArray | block;
block : BLOCK_START blockParamList? ws (PIPE ws)? sequence BLOCK_END;
blockParamList : (ws BLOCK_PARAM)+;
dynamicDictionary : DYNDICT_START ws expressions? ws DYNARR_END;
dynamicArray : DYNARR_START ws expressions? ws DYNARR_END;
parsetimeLiteral : charConstant | pseudoVariable | number | literalArray |string | symbol;
number : numberExp | hex | stFloat | stInteger;
numberExp : (stFloat | stInteger) EXP stInteger;
charConstant : CHARACTER_CONSTANT;
hex : MINUS? HEX HEXDIGIT+;
stInteger : MINUS? DIGIT+;
stFloat : MINUS? DIGIT+ PERIOD DIGIT+;
pseudoVariable : RESERVED_WORD;
string : STRING;
symbol : HASH bareSymbol;
primitive : LT ws KEYWORD ws DIGIT+ ws GT;
bareSymbol : (IDENTIFIER | BINARY_SELECTOR) | KEYWORD+ | string | PIPE+;
literalArray : LITARR_START literalArrayRest;
literalArrayRest : (ws (parsetimeLiteral | bareLiteralArray | bareSymbol))* ws CLOSE_PAREN;
bareLiteralArray : OPEN_PAREN literalArrayRest;
unaryTail : unaryMessage ws unaryTail? ws;
unaryMessage : ws unarySelector;
unarySelector : IDENTIFIER;
keywords : KEYWORD+;
reference : variable;
binaryTail : binaryMessage binaryTail?;
binaryMessage : ws BINARY_SELECTOR ws (unarySend | operand);

SEPARATOR : [ \t\r\n];
STRING : '\'' (.)*? '\'';
COMMENT : '"' (.)*? '"';
BLOCK_START : '[';
BLOCK_END : ']';
CLOSE_PAREN : ')';
OPEN_PAREN : '(';
PIPE : '|';
PERIOD : '.';
SEMI_COLON : ';';
BINARY_SELECTOR : ('\\' | '+' | '*' | '/' | '=' | '>' | '<' | ',' | '@' | '%' | '~' | PIPE | '&' |
'-' | '?')+;
LT : '<';
GT : '>';
MINUS : '-';
RESERVED_WORD : 'nil' | 'true' | 'false' | 'self' | 'super';
IDENTIFIER : [a-zA-Z]+[a-zA-Z0-9_]*;
CARROT : '^';
COLON : ':';
ASSIGNMENT : ':=';
HASH : '#';
DOLLAR : '$';
```

```
EXP : 'e';
HEX : '16r';
LITARR_START : '#(';
DYNDICT_START : '#{';
DYNARR_END : '}';
DYNARR_START : '{';
DIGIT : [0-9];
HEXDIGIT : [0-9a-fA-F];
KEYWORD : IDENTIFIER COLON;
BLOCK_PARAM : COLON IDENTIFIER;
CHARACTER_CONSTANT : DOLLAR .;
```

Приложение Б

```
export interface ISTNode {
    isEmpty: boolean;
    getClassName: string;
    varName?: string;
}

export class STNode implements ISTNode {
    public isEmpty = false;
    public getClassName = 'Node';
    public varName?: string;
}

export class STEmptyNode extends STNode {
    public isEmpty = true;
    public getClassName = 'EmptyNode';
}

export class STObjectNode extends STNode {
    public obj?: STClass;
    public getClassName = 'Value';
}

export class STScript extends STNode {
    public startingSequence?: STSequence;

    public getClassName = 'Script';
}

export class STBlock extends STObjectNode {
    public blockParamList?: STBlockParams;

    public sequence?: STSequence;

    public getClassName = 'Block';

    public codegen(generator: CodeGenerator, varName?: string): llvm.Value {/* */}
}

export class STSequence extends STObjectNode {
    public temps?: STTemps;
    public statements?: STStatements;

    public getClassName = 'Sequence';

    public codegen(generator: CodeGenerator): llvm.Value {/* */}
}

export class STTemps extends STNode {
    public identifiers?: string[];

    public getClassName = 'Temps';

    public codegen(generator: CodeGenerator): llvm.Value {/* */}
}

export class STBlockParams extends STTemps {
    public getClassName = 'BlockParams';
}

export class STStatements extends STObjectNode {
    public answer?: never;
    public expressions?: STMessage[];
```

```
    public getClassName = 'Statements';
}

export class STExpression extends STObjectNode {
    public message?: STMessage;

    public getClassName = 'Expression';
}

export class STVariable extends STObjectNode {
    public varName = '';
    public ref: STObjectNode = new STEmptyNode();

    public getClassName = 'Variable';

    public constructor(name?: string) {
        super();

        this.varName = name ?? '';
    }

    public codegen(generator: CodeGenerator): llvm.Value {/* */}
}

export class STMessage extends STObjectNode {
    public receiver?: STMessage;
    public message?: STMethod;
    public chainedMessage?: STMessage;

    public getClassName = 'Message';

    public codegen(generator: CodeGenerator, receiver?: llvm.Value): llvm.Value {/* */}
}

export class STBinaryMessage extends STMessage {
    public operation?: string;
    public secondOperand?: STMessage;

    public getClassName = 'BinaryMessage';

    public codegen(generator: CodeGenerator, receiver?: llvm.Value): llvm.Value {/* */}
}

export class STKeywordMessage extends STMessage {
    public arguments: STMessage[] = [];

    public getClassName = 'KeywordMessage';

    public codegen(generator: CodeGenerator, receiver?: llvm.Value): llvm.Value {/* */}
}

export class STLiteralArray extends STObjectNode {
    public elements?: STNode[];
    public obj = stArrayClass;

    public getClassName = 'LiteralArray';

    public codegen(generator: CodeGenerator, varName?: string): llvm.Value {/* */}
}

export class STInteger extends STObjectNode {
    public obj = stIntegerClass;
    public value?: number;

    public getClassName = 'Integer';

    public codegen(generator: CodeGenerator): llvm.Value {/* */}
}
```

Приложение В

```
{
  "isEmpty": false,
  "getClassName": "Script",
  "startingSequence": {
    "isEmpty": false,
    "getClassName": "Sequence",
```

```
"temps": {
  "isEmpty": false,
  "getClassName": "Temps",
  "identifiers": [
    "item",
    "swap",
    "itemCount",
    "hasChanged",
    "fact"
  ]
},
"statements": {
  "isEmpty": false,
  "getClassName": "Statements",
  "expressions": [
    {
      "isEmpty": false,
      "getClassName": "Message",
      "receiver": {
        "isEmpty": false,
        "getClassName": "LiteralArray",
        "obj": { },
        "elements": [
          {
            "isEmpty": false,
            "getClassName": "Integer",
            "obj": { },
            "value": 1
          },
          {
            "isEmpty": false,
            "getClassName": "Integer",
            "obj": { },
            "value": 5
          },
          {
            "isEmpty": false,
            "getClassName": "Integer",
            "obj": { },
            "value": -3
          },
          {
            "isEmpty": false,
            "getClassName": "Integer",
            "obj": { },
            "value": -7
          },
          {
            "isEmpty": false,
            "getClassName": "Integer",
            "obj": { },
            "value": -32
          },
          {
            "isEmpty": false,
            "getClassName": "Integer",
            "obj": { },
            "value": 8
          },
          {
            "isEmpty": false,
            "getClassName": "Integer",
            "obj": { },
            "value": -4
          },
          {
            "isEmpty": false,
            "getClassName": "Integer",
            "obj": { },
            "value": 90
          },
          {
            "isEmpty": false,
            "getClassName": "Integer",
            "obj": { },
            "value": 2
          },
          {
            "isEmpty": false,
            "getClassName": "Integer",
```

```
        "obj": { },
        "value": 4
      },
      {
        "isEmpty": false,
        "getClassName": "Integer",
        "obj": { },
        "value": 2
      },
      {
        "isEmpty": false,
        "getClassName": "Integer",
        "obj": { },
        "value": 0
      },
      {
        "isEmpty": false,
        "getClassName": "Integer",
        "obj": { },
        "value": 7
      }
    ]
  },
  "varName": "item"
},
{
  "isEmpty": false,
  "getClassName": "Message",
  "receiver": {
    "isEmpty": false,
    "getClassName": "Block",
    "blockParamList": {
      "isEmpty": false,
      "getClassName": "BlockParams",
      "identifiers": [
        "indexOne",
        "indexTwo"
      ]
    },
    "sequence": {
      "isEmpty": false,
      "getClassName": "Sequence",
      "temps": {
        "isEmpty": false,
        "getClassName": "Temps",
        "identifiers": [
          "temp"
        ]
      },
      "statements": {
        "isEmpty": false,
        "getClassName": "Statements",
        "expressions": [
          {
            "isEmpty": false,
            "getClassName": "Message",
            "receiver": {
              "isEmpty": false,
              "getClassName": "Variable",
              "varName": "item",
              "ref": { }
            },
            "chainedMessage": {
              "isEmpty": false,
              "getClassName": "KeywordMessage",
              "arguments": [
                {
                  "isEmpty": false,
                  "getClassName": "Message",
                  "receiver": {
                    "isEmpty": false,
                    "getClassName": "Variable",
                    "varName": "indexOne",
                    "ref": { }
                  }
                }
              ],
              "message": {
                "name": "at:"
              }
```

```
  },
  "varName": "temp"
},
{
  "isEmpty": false,
  "getClassName": "Message",
  "receiver": {
    "isEmpty": false,
    "getClassName": "Variable",
    "varName": "item",
    "ref": { }
  },
  "chainedMessage": {
    "isEmpty": false,
    "getClassName": "KeywordMessage",
    "arguments": [
      {
        "isEmpty": false,
        "getClassName": "Message",
        "receiver": {
          "isEmpty": false,
          "getClassName": "Variable",
          "varName": "indexOne",
          "ref": { }
        }
      },
      {
        "isEmpty": false,
        "getClassName": "Message",
        "receiver": {
          "isEmpty": false,
          "getClassName": "Message",
          "receiver": {
            "isEmpty": false,
            "getClassName": "Variable",
            "varName": "item",
            "ref": { }
          },
          "chainedMessage": {
            "isEmpty": false,
            "getClassName": "KeywordMessage",
            "arguments": [
              {
                "isEmpty": false,
                "getClassName": "Message",
                "receiver": {
                  "isEmpty": false,
                  "getClassName": "Variable",
                  "varName": "indexTwo",
                  "ref": { }
                }
              }
            ],
            "message": {
              "name": "at:"
            }
          }
        }
      }
    ],
    "message": {
      "name": "at:put:",
      "retType": { }
    }
  }
},
{
  "isEmpty": false,
  "getClassName": "Message",
  "receiver": {
    "isEmpty": false,
    "getClassName": "Variable",
    "varName": "item",
    "ref": { }
  },
  "chainedMessage": {
    "isEmpty": false,
    "getClassName": "KeywordMessage",
    "arguments": [
      {
```

```
                    "isEmpty": false,
                    "getClassName": "Message",
                    "receiver": {
                      "isEmpty": false,
                      "getClassName": "Variable",
                      "varName": "indexTwo",
                      "ref": { }
                    }
                  },
                  {
                    "isEmpty": false,
                    "getClassName": "Message",
                    "receiver": {
                      "isEmpty": false,
                      "getClassName": "Variable",
                      "varName": "temp",
                      "ref": { }
                    }
                  }
                ],
                "message": {
                  "name": "at:put:",
                  "retType": { }
                }
              }
            }
          ]
        }
      },
      "obj": { }
    },
    "varName": "swap"
  },
  {
    "isEmpty": false,
    "getClassName": "Message",
    "message": {
      "name": "size",
      "retType": { }
    },
    "receiver": {
      "isEmpty": false,
      "getClassName": "Variable",
      "varName": "item",
      "ref": { }
    },
    "varName": "itemCount"
  },
  {
    "isEmpty": false,
    "getClassName": "Message",
    "message": {
      "name": "whileTrue"
    },
    "receiver": {
      "isEmpty": false,
      "getClassName": "Block",
      "sequence": {
        "isEmpty": false,
        "getClassName": "Sequence",
        "statements": {
          "isEmpty": false,
          "getClassName": "Statements",
          "expressions": [
            {
              "isEmpty": false,
              "getClassName": "Message",
              "receiver": {
                "isEmpty": false,
                "getClassName": "Variable",
                "varName": "false",
                "ref": { },
                "obj": { }
              },
              "varName": "hasChanged"
            },
            {
              "isEmpty": false,
              "getClassName": "BinaryMessage",
              "operation": "-",
```

```
  "secondOperand": {
    "isEmpty": false,
    "getClassName": "Message",
    "receiver": {
      "isEmpty": false,
      "getClassName": "Integer",
      "obj": { },
      "value": 1
    }
  },
  "receiver": {
    "isEmpty": false,
    "getClassName": "Message",
    "receiver": {
      "isEmpty": false,
      "getClassName": "Variable",
      "varName": "itemCount",
      "ref": { }
    }
  },
  "varName": "itemCount"
},
{
  "isEmpty": false,
  "getClassName": "Message",
  "receiver": {
    "isEmpty": false,
    "getClassName": "Integer",
    "obj": { },
    "value": 1
  },
  "chainedMessage": {
    "isEmpty": false,
    "getClassName": "KeywordMessage",
    "arguments": [
      {
        "isEmpty": false,
        "getClassName": "Message",
        "receiver": {
          "isEmpty": false,
          "getClassName": "Variable",
          "varName": "itemCount",
          "ref": { }
        }
      },
      {
        "isEmpty": false,
        "getClassName": "Message",
        "receiver": {
          "isEmpty": false,
          "getClassName": "Block",
          "blockParamList": {
            "isEmpty": false,
            "getClassName": "BlockParams",
            "identifiers": [
              "index"
            ]
          },
          "sequence": {
            "isEmpty": false,
            "getClassName": "Sequence",
            "statements": {
              "isEmpty": false,
              "getClassName": "Statements",
              "expressions": [
                {
                  "isEmpty": false,
                  "getClassName": "BinaryMessage",
                  "operation": ">",
                  "secondOperand": {
                    "isEmpty": false,
                    "getClassName": "Message",
                    "receiver": {
                      "isEmpty": false,
                      "getClassName": "Message",
                      "receiver": {
                        "isEmpty": false,
                        "getClassName": "Variable",
                        "varName": "item",
                        "ref": { }
```

```
      },
      "chainedMessage": {
        "isEmpty": false,
        "getClassName": "KeywordMessage",
        "arguments": [
          {
            "isEmpty": false,
            "getClassName": "BinaryMessage",
            "operation": "+",
            "secondOperand": {
              "isEmpty": false,
              "getClassName": "Message",
              "receiver": {
                "isEmpty": false,
                "getClassName": "Integer",
                "obj": { },
                "value": 1
              }
            },
            "receiver": {
              "isEmpty": false,
              "getClassName": "Message",
              "receiver": {
                "isEmpty": false,
                "getClassName": "Variable",
                "varName": "index",
                "ref": { }
              }
            }
          }
        ],
        "message": {
          "name": "at:"
        }
      }
    }
  },
  "receiver": {
    "isEmpty": false,
    "getClassName": "Message",
    "receiver": {
      "isEmpty": false,
      "getClassName": "Message",
      "receiver": {
        "isEmpty": false,
        "getClassName": "Variable",
        "varName": "item",
        "ref": { }
      },
      "chainedMessage": {
        "isEmpty": false,
        "getClassName": "KeywordMessage",
        "arguments": [
          {
            "isEmpty": false,
            "getClassName": "Message",
            "receiver": {
              "isEmpty": false,
              "getClassName": "Variable",
              "varName": "index",
              "ref": { }
            }
          }
        ],
        "message": {
          "name": "at:"
        }
      }
    }
  },
  "chainedMessage": {
    "isEmpty": false,
    "getClassName": "KeywordMessage",
    "arguments": [
      {
        "isEmpty": false,
        "getClassName": "Message",
        "receiver": {
          "isEmpty": false,
          "getClassName": "Block",
```

```
    "sequence": {
      "isEmpty": false,
      "getClassName": "Sequence",
      "statements": {
        "isEmpty": false,
        "getClassName": "Statements",
        "expressions": [
          {
            "isEmpty": false,
            "getClassName": "Message",
            "receiver": {
              "isEmpty": false,
              "getClassName": "Variable",
              "varName": "swap",
              "ref": { }
            },
            "chainedMessage": {
              "isEmpty": false,
              "getClassName": "KeywordMessage",
              "arguments": [
                {
                  "isEmpty": false,
                  "getClassName": "Message",
                  "receiver": {
                    "isEmpty": false,
                    "getClassName": "Variable",
                    "varName": "index",
                    "ref": { }
                  }
                },
                {
                  "isEmpty": false,
                  "getClassName": "BinaryMessage",
                  "operation": "+",
                  "secondOperand": {
                    "isEmpty": false,
                    "getClassName": "Message",
                    "receiver": {
                      "isEmpty": false,
                      "getClassName": "Integer",
                      "obj": { },
                      "value": 1
                    }
                  },
                  "receiver": {
                    "isEmpty": false,
                    "getClassName": "Message",
                    "receiver": {
                      "isEmpty": false,
                      "getClassName": "Variable",
                      "varName": "index",
                      "ref": { }
                    }
                  }
                }
              ],
              "message": {
                "name": "value:value:",
                "retType": { }
              }
            }
          },
          {
            "isEmpty": false,
            "getClassName": "Message",
            "receiver": {
              "isEmpty": false,
              "getClassName": "Variable",
              "varName": "true",
              "ref": { },
              "obj": { }
            },
            "varName": "hasChanged"
          }
        ]
      }
    },
    "obj": { }
  }
}
```

```
                                ],
                                "message": {
                                  "name": "ifTrue:"
                                }
                              }
                            }
                          ]
                        }
                      },
                      "obj": { }
                    }
                  }
                ],
                "message": {
                  "name": "to:do:"
                }
              }
            },
            {
              "isEmpty": false,
              "getClassName": "Message",
              "receiver": {
                "isEmpty": false,
                "getClassName": "Variable",
                "varName": "hasChanged",
                "ref": { }
              }
            }
          ]
        }
      },
      "obj": { }
    }
  },
  {
    "isEmpty": false,
    "getClassName": "Message",
    "receiver": {
      "isEmpty": false,
      "getClassName": "Integer",
      "obj": { },
      "value": 1
    },
    "chainedMessage": {
      "isEmpty": false,
      "getClassName": "KeywordMessage",
      "arguments": [
        {
          "isEmpty": false,
          "getClassName": "Message",
          "message": {
            "name": "size",
            "retType": { }
          },
          "receiver": {
            "isEmpty": false,
            "getClassName": "Variable",
            "varName": "item",
            "ref": { }
          }
        },
        {
          "isEmpty": false,
          "getClassName": "Message",
          "receiver": {
            "isEmpty": false,
            "getClassName": "Block",
            "blockParamList": {
              "isEmpty": false,
              "getClassName": "BlockParams",
              "identifiers": [
                "index"
              ]
            },
            "sequence": {
              "isEmpty": false,
              "getClassName": "Sequence",
              "statements": {
                "isEmpty": false,
                "getClassName": "Statements",
```

```
                        "expressions": [
                          {
                            "isEmpty": false,
                            "getClassName": "Message",
                            "message": {
                              "name": "printfInt",
                              "retType": { }
                            },
                            "receiver": {
                              "isEmpty": false,
                              "getClassName": "Message",
                              "receiver": {
                                "isEmpty": false,
                                "getClassName": "Variable",
                                "varName": "item",
                                "ref": { }
                              },
                              "chainedMessage": {
                                "isEmpty": false,
                                "getClassName": "KeywordMessage",
                                "arguments": [
                                  {
                                    "isEmpty": false,
                                    "getClassName": "Message",
                                    "receiver": {
                                      "isEmpty": false,
                                      "getClassName": "Variable",
                                      "varName": "index",
                                      "ref": { }
                                    }
                                  }
                                ],
                                "message": {
                                  "name": "at:"
                                }
                              }
                            }
                          }
                        ]
                      }
                    },
                    "obj": { }
                  }
                }
              ],
              "message": {
                "name": "to:do:"
              }
            }
          }
        }
      ]
    }
  }
}
```


Приложение Г

```
public codegen(generator: CodeGenerator, varName?: string): llvm.Value {

    const varType = getLiteralArrayType(this);
    const typeParts = varType.split(' ');
    const arrType = llvm.ArrayType.get(generator.builder.getInt32Ty(), Number(typeParts[1]));

    const arr = generator.builder.CreateAlloca(arrType, null, 'lit_arr');

    if (!this.elements) {
        return arr;
    }

    for (let i = 0; i < this.elements.length; i++) {
        const ptrCast = generator.builder.CreatePointerCast(arr,
llvm.PointerType.getUnqual(arrType.getElementType()));
        const gep = generator.builder.CreateGEP(
            generator.builder.getInt32Ty(),
            ptrCast,
            generator.builder.getInt32(i),
            `lit_arr_gep` + i,
```

```
        );

        generator.builder.CreateStore(
            generator.builder.getInt32((this.elements[i] as STInteger).value ?? 0),
            gep
        );
    }

    if (varName) {
        const arrVar = generator.variables.get(varName)!;
        generator.builder.CreateStore(arr, generator.variables.get(varName)!);
        return arrVar;
    }

    return arr;
}



public codegen(generator: CodeGenerator, fromVal: llvm.Value, toVal: llvm.Value, blockFn:
llvm.Function): llvm.Value {
    const toDoBB = llvm.BasicBlock.Create(generator.rootContext, 'to_do', generator.currentFn);
    const exitBB = llvm.BasicBlock.Create(generator.rootContext, 'to_do_exit',
generator.currentFn);

    const indexAlloca = generator.builder.CreateAlloca(fromVal.getType(), null, 'i');
    generator.builder.CreateStore(fromVal, indexAlloca);
    let index = generator.builder.CreateLoad(fromVal.getType(), indexAlloca);

    const condOuter = generator.builder.CreateICmpSGT(index, toVal, 'to_do_cond_prev');
    generator.builder.CreateCondBr(condOuter, exitBB, toDoBB);

    generator.BBStack.pop();
    generator.BBStack.push(exitBB);

    generator.BBStack.push(toDoBB);
    generator.builder.SetInsertPoint(toDoBB);

    index = generator.builder.CreateLoad(fromVal.getType(), indexAlloca);
    const fnVal = generator.builder.CreateCall(blockFn, [index]);

    const indexAdd = generator.builder.CreateAdd(index, generator.builder.getInt32(1));
    const addStore = generator.builder.CreateStore(indexAdd, indexAlloca);
    index = generator.builder.CreateLoad(fromVal.getType(), indexAlloca);

    const cond = generator.builder.CreateICmpSGT(index, toVal, 'to_do_cond');
    generator.builder.CreateCondBr(cond, exitBB, toDoBB);

    generator.BBStack.pop();

    generator.builder.SetInsertPoint(exitBB);

    return indexAlloca;
}
```

Приложение Д

```
; ModuleID = 'cw-root'
source_filename = "cw-root"

@false = global i1 false
@true = global i1 true
@print_format = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@item = global [13 x i32]* null
@itemCount = global i32 0
@hasChanged = global i1 false
@fact = global i32 0
@indexOne = global i32 0
@indexTwo = global i32 0
@temp = global i32 0
@index = global i32 0
@index.3 = global i32 0

define i32 @main() {
main_bb:
  %lit_arr = alloca [13 x i32], align 4
  %0 = bitcast [13 x i32]* %lit_arr to i32*
```

```
  %lit_arr_gep0 = getelementptr i32, i32* %0, i32 0
  store i32 1, i32* %lit_arr_gep0, align 4
  %1 = bitcast [13 x i32]* %lit_arr to i32*
  %lit_arr_gep1 = getelementptr i32, i32* %1, i32 1
  store i32 5, i32* %lit_arr_gep1, align 4
  %2 = bitcast [13 x i32]* %lit_arr to i32*
  %lit_arr_gep2 = getelementptr i32, i32* %2, i32 2
  store i32 -3, i32* %lit_arr_gep2, align 4
  %3 = bitcast [13 x i32]* %lit_arr to i32*
  %lit_arr_gep3 = getelementptr i32, i32* %3, i32 3
  store i32 -7, i32* %lit_arr_gep3, align 4
  %4 = bitcast [13 x i32]* %lit_arr to i32*
  %lit_arr_gep4 = getelementptr i32, i32* %4, i32 4
  store i32 -32, i32* %lit_arr_gep4, align 4
  %5 = bitcast [13 x i32]* %lit_arr to i32*
  %lit_arr_gep5 = getelementptr i32, i32* %5, i32 5
  store i32 8, i32* %lit_arr_gep5, align 4
  %6 = bitcast [13 x i32]* %lit_arr to i32*
  %lit_arr_gep6 = getelementptr i32, i32* %6, i32 6
  store i32 -4, i32* %lit_arr_gep6, align 4
  %7 = bitcast [13 x i32]* %lit_arr to i32*
  %lit_arr_gep7 = getelementptr i32, i32* %7, i32 7
  store i32 90, i32* %lit_arr_gep7, align 4
  %8 = bitcast [13 x i32]* %lit_arr to i32*
  %lit_arr_gep8 = getelementptr i32, i32* %8, i32 8
  store i32 2, i32* %lit_arr_gep8, align 4
  %9 = bitcast [13 x i32]* %lit_arr to i32*
  %lit_arr_gep9 = getelementptr i32, i32* %9, i32 9
  store i32 4, i32* %lit_arr_gep9, align 4
  %10 = bitcast [13 x i32]* %lit_arr to i32*
  %lit_arr_gep10 = getelementptr i32, i32* %10, i32 10
  store i32 2, i32* %lit_arr_gep10, align 4
  %11 = bitcast [13 x i32]* %lit_arr to i32*
  %lit_arr_gep11 = getelementptr i32, i32* %11, i32 11
  store i32 0, i32* %lit_arr_gep11, align 4
  %12 = bitcast [13 x i32]* %lit_arr to i32*
  %lit_arr_gep12 = getelementptr i32, i32* %12, i32 12
  store i32 7, i32* %lit_arr_gep12, align 4
  store [13 x i32]* %lit_arr, [13 x i32]** @item, align 8
  %13 = load [13 x i32]*, [13 x i32]** @item, align 8
  %14 = call i32 @size_13_13([13 x i32]* %13)
  store i32 %14, i32* @itemCount, align 4
  br label %do_while

do_while:                                         ; preds = %do_while, %main_bb
  %do_while_body = call i1 @anonFn()
  br i1 %do_while_body, label %do_while, label %do_while_exit

do_while_exit:                                    ; preds = %do_while
  %15 = load [13 x i32]*, [13 x i32]** @item, align 8
  %16 = call i32 @size_13_13([13 x i32]* %15)
  %i = alloca i32, align 4
  store i32 1, i32* %i, align 4
  %17 = load i32, i32* %i, align 4
  %to_do_cond_prev = icmp sgt i32 %17, %16
  br i1 %to_do_cond_prev, label %to_do_exit, label %to_do

to_do:                                            ; preds = %to_do, %do_while_exit
  %18 = load i32, i32* %i, align 4
  %19 = call i32 @anonFn.4(i32 %18)
  %20 = add i32 %18, 1
  store i32 %20, i32* %i, align 4
  %21 = load i32, i32* %i, align 4
  %to_do_cond = icmp sgt i32 %21, %16
  br i1 %to_do_cond, label %to_do_exit, label %to_do

to_do_exit:                                       ; preds = %to_do, %do_while_exit
  ret i32 0
}

define i32 @swap(i32 %indexOne, i32 %indexTwo) {
swap:
  store i32 %indexOne, i32* @indexOne, align 4
  store i32 %indexTwo, i32* @indexTwo, align 4
  %0 = load [13 x i32]*, [13 x i32]** @item, align 8
  %1 = load i32, i32* @indexOne, align 4
  %2 = call i32 @at_13_13([13 x i32]* %0, i32 %1)
  store i32 %2, i32* @temp, align 4
  %3 = load [13 x i32]*, [13 x i32]** @item, align 8
  %4 = load i32, i32* @indexOne, align 4
```

```
  %5 = load [13 x i32]*, [13 x i32]** @item, align 8
  %6 = load i32, i32* @indexTwo, align 4
  %7 = call i32 @at_13_13([13 x i32]* %5, i32 %6)
  %8 = call i32 @at_put_13_13([13 x i32]* %3, i32 %4, i32 %7)
  %9 = load [13 x i32]*, [13 x i32]** @item, align 8
  %10 = load i32, i32* @indexTwo, align 4
  %11 = load i32, i32* @temp, align 4
  %12 = call i32 @at_put_13_13([13 x i32]* %9, i32 %10, i32 %11)
  ret i32 %12
}

define i32 @at_13_13([13 x i32]* %arr, i32 %index) {
at_13_13_fn:
  %index_corr = sub i32 %index, 1
  %0 = bitcast [13 x i32]* %arr to i32*
  %arr_at_gep = getelementptr i32, i32* %0, i32 %index_corr
  %1 = load i32, i32* %arr_at_gep, align 4
  ret i32 %1
}

define i32 @at_put_13_13([13 x i32]* %arr, i32 %index, i32 %elem) {
at_put_13_13_fn:
  %index_corr = sub i32 %index, 1
  %0 = bitcast [13 x i32]* %arr to i32*
  %arr_at_put_gep = getelementptr i32, i32* %0, i32 %index_corr
  store i32 %elem, i32* %arr_at_put_gep, align 4
  ret i32 %elem
}

define i32 @size_13_13([13 x i32]* %arr) {
size_13_13_fn:
  ret i32 13
}

define i1 @anonFn() {
anon_fn:
  %0 = load i1, i1* @false, align 1
  store i1 %0, i1* @hasChanged, align 1
  %1 = load i32, i32* @itemCount, align 4
  %2 = sub i32 %1, 1
  store i32 %2, i32* @itemCount, align 4
  %3 = load i32, i32* @itemCount, align 4
  %i = alloca i32, align 4
  store i32 1, i32* %i, align 4
  %4 = load i32, i32* %i, align 4
  %to_do_cond_prev = icmp sgt i32 %4, %3
  br i1 %to_do_cond_prev, label %to_do_exit, label %to_do

to_do:                                            ; preds = %to_do, %anon_fn
  %5 = load i32, i32* %i, align 4
  %6 = call i1 @anonFn.1(i32 %5)
  %7 = add i32 %5, 1
  store i32 %7, i32* %i, align 4
  %8 = load i32, i32* %i, align 4
  %to_do_cond = icmp sgt i32 %8, %3
  br i1 %to_do_cond, label %to_do_exit, label %to_do

to_do_exit:                                       ; preds = %to_do, %anon_fn
  %9 = load i1, i1* @hasChanged, align 1
  ret i1 %9
}

define i1 @anonFn.1(i32 %index) {
anon_fn:
  store i32 %index, i32* @index, align 4
  %0 = load [13 x i32]*, [13 x i32]** @item, align 8
  %1 = load i32, i32* @index, align 4
  %2 = call i32 @at_13_13([13 x i32]* %0, i32 %1)
  %3 = load [13 x i32]*, [13 x i32]** @item, align 8
  %4 = load i32, i32* @index, align 4
  %5 = add i32 %4, 1
  %6 = call i32 @at_13_13([13 x i32]* %3, i32 %5)
  %7 = icmp sgt i32 %2, %6
  br i1 %7, label %if_true, label %if_true_exit

if_true:                                          ; preds = %anon_fn
  %8 = call i1 @anonFn.2()
  br label %if_true_exit

if_true_exit:                                     ; preds = %if_true, %anon_fn
```

```
  ret i1 %7
}

define i1 @anonFn.2() {
anon_fn:
  %0 = load i32, i32* @index, align 4
  %1 = load i32, i32* @index, align 4
  %2 = add i32 %1, 1
  %3 = call i32 @swap(i32 %0, i32 %2)
  %4 = load i1, i1* @true, align 1
  store i1 %4, i1* @hasChanged, align 1
  ret i1 %4
}

define i32 @anonFn.4(i32 %index) {
anon_fn:
  store i32 %index, i32* @index.3, align 4
  %0 = load [13 x i32]*, [13 x i32]** @item, align 8
  %1 = load i32, i32* @index.3, align 4
  %2 = call i32 @at_13_13([13 x i32]* %0, i32 %1)
  %3 = call i32 @printf_int(i32 %2)
  ret i32 %3
}

define i32 @printf_int(i32 %num) {
printf_int_fn:
  %printf_call = call i32 ([4 x i8]*, ...) @printf([4 x i8]* @print_format, i32%num)
  ret i32 %printf_call
}

declare i32 @printf([4 x i8]*, ...)
```