Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Архитектура вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

«Компилятор»

БГУИР КП 1-40 04 01 009 ПЗ

Студент гр. 753505
Лесун А.И.
Руководитель
Ассистент кафедры информатики
Леченко А.В.

Минск 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Программист должен понимать, что язык программирования определяет характер программы, а не способ ее выполнения компьютером. Есть два способа выполнения программы компьютером: она может быть подвергнута компиляции или интерпретации. Программа, написанная на любом языке программирования, может как компилироваться, так и интерпретироваться, однако многие языки изначально созданы для выполнения преимущественно одним из этих способов. Например, Java рассчитан в основном на интерпретацию программы, а язык C — на компиляцию. Необходимо помнить, что при разработке языка C его конструкции оптимизировались специально для компиляции. И хотя интерпретаторы C существуют и доступны для программистов, C разрабатывался преимущественно для компиляции. Поэтому при разработке программ на C большинство программистов используют именно компилятор, а не интерпретатор.

В простейшем случае интерпретатор читает исходный текст программы по одной строке за раз, выполняет эту строку и только после этого переходит к следующей. Так работали ранние версии языка Basic. В языках типа Java исходный текст программы сначала конвертируется в промежуточную форму, а затем интерпретируется. В этом случае программа также интерпретируется в процессе выполнения.

Компилятор читает сразу всю программу и конвертирует ее в объектный код, то есть транслирует исходный текст программы в форму, более пригодную для непосредственного выполнения компьютером. Объектный код также называют двоичным или машинным кодом. Когда программа скомпилирована, в ее коде уже нет отдельных строк исходного кода.

В общем случае интерпретируемая программа выполняется медленнее, чем скомпилированная. Необходимо помнить, что компилятор преобразует исходный текст программы в объектный код, который выполняется компьютером непосредственно. Значит, потеря времени на компиляцию происходит лишь единожды, а в случае интерпретации — каждый раз при очередной компиляции фрагмента программы в процессе ее выполнения.

Глава 1. Теория

Определим пару важных понятий для данного проекта:

Компиляция — сборка программы, включающая трансляцию всех модулей программы, написанных на одном или нескольких исходных языках программирования высокого уровня и/или языке ассемблера, в эквивалентные программные модули на низкоуровневом языке, близком машинному коду (абсолютный код, объектный модуль, иногда на язык ассемблера) или непосредственно на машинном языке или ином двоично кодовом низкоуровневом командном языке и последующую сборку исполняемой машинной программы.

Компилятор — программа или техническое средство, выполняющее компиляцию программы.

Трансляция программы как неотъемлемая составляющая компиляции включает в себя:

1. Лексический анализ. На этом этапе последовательность символов исходного файла преобразуется в последовательность лексем.

2. Синтаксический (грамматический) анализ. Последовательность лексем преобразуется в дерево разбора.

3. Семантический анализ. Дерево разбора обрабатывается с целью установления его семантики (смысла) — например, привязка идентификаторов к их декларациям, типам, проверка совместимости, определение типов выражений и т. д. Результат обычно называется «промежуточным представлением/кодом», и может быть дополненным деревом разбора, новым деревом, абстрактным набором команд или чем-то ещё, удобным для дальнейшей обработки.

4. Оптимизация. Выполняется удаление излишних конструкций и упрощение кода с сохранением его смысла. Оптимизация может быть на разных уровнях и этапах — например, над промежуточным кодом или над конечным машинным кодом.

5. Генерация кода. Из промежуточного представления порождается код на целевом машинно-ориентированном языке.

Абстрактное синтаксическое дерево (АСД) — в информатике конечное помеченное ориентированное дерево, в котором внутренние вершины сопоставлены (помечены) с операторами языка программирования, а листья — с соответствующими операндами. Таким образом, листья являются пустыми операторами и представляют только переменные и константы.

Синтаксические деревья используются в парсерах для промежуточного представления программы между деревом разбора (конкретным

синтаксическим деревом) и структурой данных, которая затем используется в качестве внутреннего представления в компиляторе или интерпретаторе компьютерной программы для оптимизации и генерации кода. Возможные варианты подобных структур описываются абстрактным синтаксисом.

Эзотерический язык программирования — язык программирования, разработанный для исследования границ возможностей разработки языков программирования, для доказательства потенциально возможной реализации некой идеи (так называемое «доказательство концепции»), в качестве произведения программного искусства или в качестве шутки (компьютерного юмора).

Данный компилятор предназначен для компиляции языка программирования Brainfuck. Brainfuck — один из известнейших эзотерических языков программирования, придуман Урбаном Мюллером в 1993 году, известен своим минимализмом. Название языка можно перевести на русский как вынос мозга. Язык имеет восемь команд, каждая из которых записывается одним символом. Исходный код программы на Brainfuck представляет собой последовательность этих символов без какого-либо дополнительного синтаксиса. Машина, которой управляют команды Brainfuck, состоит из упорядоченного набора ячеек и указателя текущей ячейки, напоминая ленту и головку машины Тьюринга. Кроме того, подразумевается устройство общения с внешним миром через поток ввода и поток вывода.

Вот сами команды:

Команда Brainfuck	Эквивалент на Си	Описание команды
`>`	`i++;`	перейти к следующей ячейке
`<`	`i--;`	перейти к предыдущей ячейке
`+`	`arr[i]++;`	увеличить значение в текущей ячейке на 1
`-`	`arr[i]--;`	уменьшить значение в текущей ячейке на 1
`.`	`putchar(arr[i]);`	напечатать значение из текущей ячейки
`,`	`arr[i] = getchar();`	ввести извне значение и сохранить в текущей ячейке
`[`	`while(arr[i]){`	если значение текущей ячейки ноль, перейти вперёд по тексту программы на ячейку, следующую за соответствующей `]` (с учётом вложенности)
`]`	`}`	если значение текущей ячейки не нуль, перейти назад по тексту программы на символ `[` (с учётом вложенности)

Глава 2. Программа

Программа состоит из следующих сущностей:

- **AstPrinter** — предназначен для вывода абстрактного синтаксического дерева.
- **CodeGenerator** — предназначен для генерации кода в выходной файл из переданного абстрактного синтаксического дерева.
- **FileHandler** — предназначен для удобного использования файлов.
- **LexicalAnalyzer** — предназначен для лексического анализа входного файла.
- **Node** — узел в абстрактном синтаксическом дереве.
- **Optimizer** — предназначен для оптимизации кода, на основе переданного абстрактного синтаксического дерева.
- **Parser** — предназначен для построения абстрактного синтаксического дерева из кода входного файла.
- **Settings** — предназначен для хранения заданных настроек компиляции, переданные через аргументы командной строки.

Использование компилятора:

BrainfuckCompiler.exe [options] file

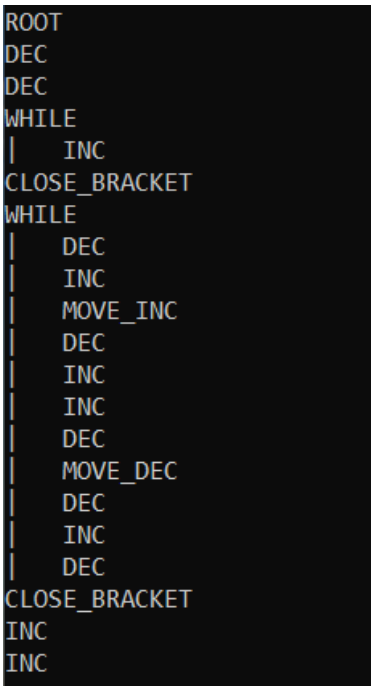
file — имя входного файла.

Options:

- --help — отображение помощи по использования компилятора.
- --la — произвести только лексический анализ входного файла.
- --no-opt — скомпилировать файл без оптимизаций.
- --pr-ast — показать абстрактное синтаксическое дерево перед оптимизациями.
- --pr-opt-ast — показать абстрактное синтаксическое дерево после оптимизаций.
- -o <result_file> — задать имя выходного файла (по умолчанию — file.c).

Также в компиляторе присутствует оптимизатор, который можно отключать при компиляции. Посмотрим разницу на основе абстрактного синтаксического дерева:

Компилируемый код: --[+][-+>-++-<-+-]++



До оптимизации

После оптимизации

ЗАКЛЮЧЕНИЕ

В итоге был написан компилятор на языке C++ для компиляции программы, написанной на эзотерическом языке Brainfuck. Он разделен на такие отдельные части как лексический анализатор, парсер, оптимизатор и генератор кода. Также с помощью параметров командной строки можно настраивать компиляцию, например, только проанализировать код на Brainfuck, вывести абстрактное синтаксическое дерево перед оптимизациями, вывести абстрактное синтаксическое дерево после оптимизации, задать имя выходного файла. Языком выходного файла компиляции был выбран C, так как он намного проще чем Assembler и намного быстрее чем Java, Python и т.д.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://en.cppreference.com/w/.
2. https://ru.wikipedia.org/wiki/Brainfuck.
3. Steven Muchnick , Advanced Compiler Design and Implementation, 1997.