1 19. Трансляторы, интерпретаторы и компиляторы. Стадии работы компилятора. Лексический и синтаксический анализ. Синтаксически управляемая трансляция.

Компилятор – это программа, которая считывает текст программы, написанной на одном языке – *исходном*, и транслирует (переводит) его в эквивалентный текст на другом языке – *целевом*. Одна из важных ролей компилятора состоит в сообщении об ошибках в исходной программе, обнаруженных в процессе трансляции. (NB. Wiki говорит, что на самом деле это транслятор. Компилятор является частным случаем транслятора, к. транслирует программу, составленную на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком к машинному коду).

Если целевая программа представляет собой программу на машинном языке, она затем может быть вызвана пользователем для обработки некоторых входных данных и получения некоторых выходных данных.

Интерпретатор представляет собой еще один распространенный вид языкового процессора. Вместо получения целевой программы, как в случае транслятора, интерпретатор непосредственно выполняет операции, указанные в исходной программе, над входными данными, предоставляемыми пользвателем.

Пример. Языковой процессор Java объединяет в себе и компиляцию, и интерпретацию. Исходная программа на Java может сначала компилироваться в промежуточный вид, именуемый байт-кодом. Затем байт-код интерпретируется виртуальной машиной.

Кроме компиляторов, потребовать создания выполнимой целевой программы могут и другие программы. Исходная программа может быть разделена на модули, находящиеся в различных файлах. Сборка исходной программы иногда поручается отдельной программе, именуемой препроцессором. Препроцессор может также раскрывать сокращения, именуемые макросами, в инструкции исходного языка. Модифицированння исходная программа затем передается компилятору. Компилятор может выдать в качестве входных данных программу на языке ассемблера. Язык ассемблера затем обрабатывается программой, которая называется *ассемблер*, и дает в качестве выходных данных перемещаемый машинный код. Большие программы компилируются по частям, так что машинный код должен быть скомпонован совместно с другими перемещаемыми объектными файлами и библиотечными файлами в код, который можно бу-

дет выполнять на данной машине. *Компоновщик* выполняет разрешение внешних адресов памяти, по которым код из одного файла может обращаться к информации из другого файла. *Загрузчик* затем помещает все выполнимые объектные файлы в память для выполнения.

1.1 Структура компилятора

Компиляция разделяется на две части: анализ и синтез.

Анализ разбивает исходную программу на составные части и накладывает на них грамматическую структуру. Затем он использует эту структуру для создания промежуточного представления исходной программы. Если анализ обнаруживает, что исходная программа неверно составлена синтаксически либо дефектна семантически, он должен выдать информативные сообщения об этом. Анализ также собирает информацию об исходной программе и сохраняет её в структуре данных, именуемой *таблицей символов*, которая передается вместе с промежуточный представлением синтезу.

Синтез строит требуемую целевую программу на основе промежуточного представлени и информации из таблицы символов.

1.2 Фазы компиляции

Перавя фаза компиляции называется *лексическим анализом* или *сканированием*. Лексический анализатор читает поток символов, составляющих исходную программу, и группирует эти символы в значащие последовательности, называющиеся *лексемами*. Для каждой лексемы анализатор строит выходной *токен* вида <имя, значение>. Он передается последующей фазе, синтаксическому анализу.

Вторая фаза компилятора – *синтаксический анализ* или *разбор*. Анализатор использует первые компоненты токенов, полученных при лексическом анализе, для создания древовидного промежуточного представления, которое описывает грамматическую структуру потока токенов. Типичным представленим является *синтаксическое дерево*, в котором каждый внутренний узел представляет операцию, а дочерние узлы – аргументы этой операции. Последующие фазы компилятора используют грамматическую структуру, которая помогает проанализировать исходную и сгенерировать целевую программу.

Третья фаза компилятора – *семантический анализ*. Семантический анализатор использует синтаксическое дерево и информацию из таблицы символов для проверки исходной программы на семантическую согласованность с определением языка. Он также собирает информацию о

типах и сохраняет её в синтаксическом дереве или в таблице символов для последующего использования в процессе генерации промежуточного кода. Важной частью семантического анализа является *проверка типов*, когда компилятор проверяет, имеет ли каждый оператор операнды соответствующего типа.

Четвертая фаза – *генерация промежуточного кода*. После синтаксического и семантического анализа многие компиляторы генерируют явное низкоуровневое или машинное промежуточное представление исходной программы, которое можно рассматривать как программу для абстрактной вычислительной машины. Такое промеждуточное представление должно обладать двумя важными свойствами: оно должно легко генерироваться и легко транслироваться в целевой машинный язык. Одним из примеров является *трехадресный код* – последовательность команд, напоминающих ассемблерные, причем в каждой команде имеется три операнда.

Пятая фаза – *оптимизация кода*. Фаза машинно-независимой оптимизации кода пытается улучшить промежуточный код, чтобы затем получить более качественный целевой код. Обычно это означает "более быстрый но может быть и "более короткий"или "использующий меньше ресурсов".

Шестая фаза – *генерация кода*. Генератор кода получает в качестве входных данных промежуточное представление исходной программы и отображает его в целевой язык. Если целевой язык представляет собой машинный код, для каждой переменной, используемой программой, выбираются соответствующие регистры или ячейки памяти. Затем промежуточные команды транслируются в последовательность машинных команд, выполняющих те же действия. Ключевым моментом генерации кода является аккуратное распределение регистров для хранения переменных.

При реализации работа разных фаз может быть сгруппирована в *проходы*, которые считывают входной файл и записывают выходной. Например, фазы анализа – лексический, синтаксический, семантический анализы и генерация промежуточного кода – могут быть объединены в один проход. Оптимизация кода может представлять собой необязательный проход. Затем может быть еще один проход, заключающийся в генерации кода для конкретной целевой машины.

1.3 Синтаксически управляемая трансляция

Синтаксически управляемая трансляция выполняется путем присоединения правил или программных фрагментов к продукциям граммати-

ки. Синтаксически управляемая трансляция работает за счет добавления действий в контекстно-свободную грамматику. Эти действия будут осуществляться, когда соответствующее правило используется в выводе. Описание грамматики с такими действиями называется схемой синтаксически управляемой трансляции (или просто схемой трансляции). Каждый символ в грамматике может иметь атрибуты, которые содержат данные. Обычно такие атрибуты могут включать в себя тип переменной, значение выражения, и т.п. Для символа X с атрибутом t обращение к атрибуту может выглядеть как X.t. Таким образом, используя действия и атрибуты, грамматика может быть применена для перевода текста с языка, порождаемого ею, выполнением действий и переносом информации через атрибуты символов.