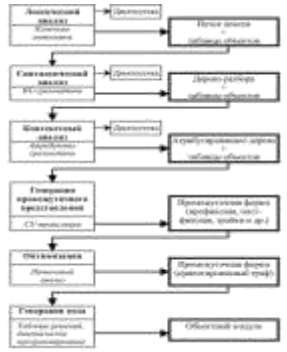
Обобщенная структура компилятора и основные фазы компиляции



Лексический анализ

На фазе лексического анализа входная программа, представляющая собой поток литер, разбивается на лексемы - слова в соответствии с определениями языка.

Основными формализмами, лежащим в основе реализации лексических анализаторов, являются конечные автоматы и регулярные выражения.

Лексический анализатор может работать в двух основных режимах: либо как подпрограмма, вызываемая синтаксическим анализатором для получения очередной лексемы, либо как полный проход, результатом которого является файл лексем.

Лексический анализ

В процессе выделения лексем лексический анализатор может как самостоятельно строить таблицы объектов (идентификаторов, строк, чисел и т.д.), так и выдавать значения для каждой лексемы при очередном к нему обращении. В этом случае таблицы объектов строятся в последующих фазах (например, в процессе синтаксического анализа).

На этапе лексического анализа обнаруживаются некоторые (простейшие) ошибки (недопустимые символы, неправильная запись чисел, идентификаторов и др.).

Синтаксический анализ

Основная задача синтаксического анализа - разбор структуры программы.

Как правило, под структурой понимается дерево, соответствующее разбору в контекстно-свободной грамматике языка.

Чаще всего используется

LL(1)-анализ (и его вариант - рекурсивный спуск),

LR(1)-анализ и его варианты (LR(0), SLR(1), LALR(1) и другие).

Результатом синтаксического анализа является синтаксическое дерево со ссылками на таблицы объектов. В процессе синтаксического анализа также обнаруживаются ошибки, связанные со структурой программы.

Контекстный анализ

На этапе контекстного анализа выявляются зависимости между частями программы, которые не могут быть описаны контекстно-свободным синтаксисом.

Это в основном связи «описание-использование», в частности, анализ типов объектов, анализ областей видимости, соответствие параметров, метки и другие.

В процессе контекстного анализа таблицы объектов пополняются информацией об описаниях (свойствах) объектов.

Основным формализмом, использующимся при контекстном анализе, является аппарат атрибутных грамматик.

Результатом контекстного анализа является атрибутированное дерево программы.

Информация об объектах может быть как рассредоточена в самом дереве, так и сосредоточена в отдельных таблицах объектов. В процессе контекстного анализа также могут быть обнаружены ошибки, связанные с неправильным использованием объектов.

Генерация внутреннего представления

Затем программа может быть переведена во внутреннее представление. Это делается для целей оптимизации и/или удобства генерации кода.

Еще одной целью преобразования программы во внутреннее представление является желание иметь переносимый компилятор. Тогда только последняя фаза (генерация кода) является машинно-зависимой.

В качестве внутреннего представления может использоваться префиксная или постфиксная запись, ориентированный граф, тройки, четверки и другие.

Оптимизация

Фаз оптимизации может быть несколько. Оптимизации обычно делят на машинно-зависимые и машинно-независимые, локальные и глобальные. машинно-зависимые и машинно-независимые, локальные и глобальные.

Часть машинно-зависимой оптимизации выполняется на фазе генерации кода.

Глобальная оптимизация пытается принять во внимание структуру всей программы, локальная - только небольших ее фрагментов.

Глобальная оптимизация основывается на глобальном потоковом анализе, который выполняется на графе программы и представляет по существу преобразование этого графа.

При этом могут учитываться такие свойства программы, как межпроцедурный анализ, межмодульный анализ, анализ областей жизни переменных и т.д.

Генерация кода

Генерация кода - последняя фаза трансляции. Результатом ее является либо ассемблерный модуль, либо объектный (или загрузочный) модуль.

В процессе генерации кода могут выполняться некоторые локальные оптимизации, такие как распределение регистров, выбор длинных или коротких переходов, учет стоимости команд при выборе конкретной последовательности команд.

Для генерации кода разработаны различные методы, такие как таблицы решений, сопоставление образцов, включающее динамическое программирование, различные синтаксические методы.

Выводы

Конечно, те или иные фазы транслятора могут либо отсутствовать совсем, либо объединяться.

В простейшем случае однопроходного транслятора нет явной фазы генерации промежуточного представления и оптимизации, остальные фазы объединены в одну, причем нет и явно построенного синтаксического дерева.