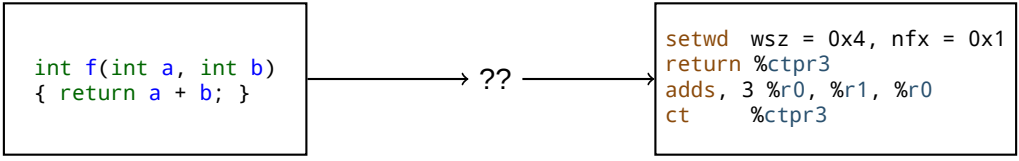
Проектирование компиляторов

Лекция 2.
Лексический анализ.

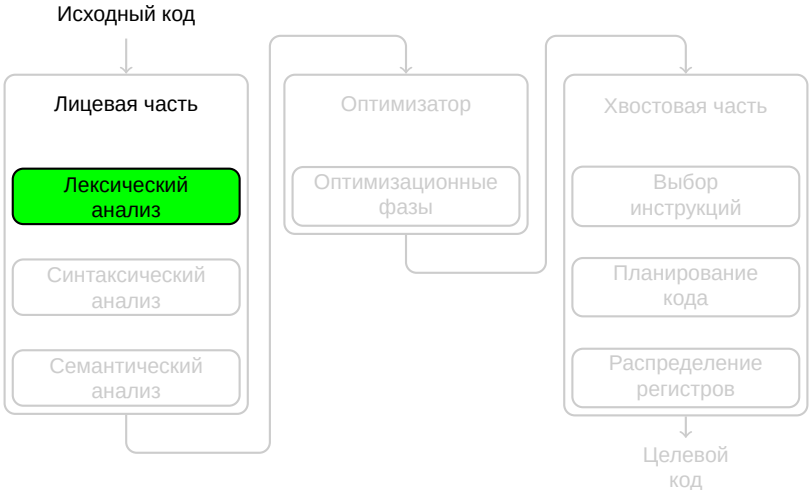
Маркин А. Л.
alexanius@gmail.com

2021

Компиляция программы



Компиляция программы



Компиляция программы

Лицевая часть (Frontend) — программа для преобразования текста исходной программы в промежуточное представление.

Промежуточное представление (Intermediate Representation, IR) — структура данных, содержащая в себе исходную программу в пригодном для обработки компилятором виде.

Лексический анализ

Исходная программа в окне текстового редактора:

```
int f(int a, int b)
{ return a + b; }
```

Исходная программа с точки зрения лексического анализатора:

```
int f(int a, int b)\n{ return a + b; }
```

Задача лексического анализатора — выделить из входного потока слова и классифицировать.

Токены

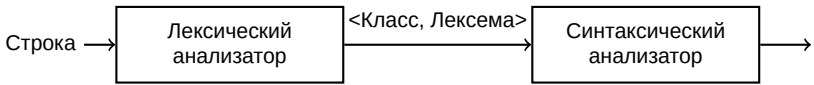
Классифицированные слова входного потока называются **токенами**. Токен состоит из пары <класс, лексема>.

Пример класса токена:

В русском языке: существительное, глагол, прилагательное

В языке Си: идентификатор, ключевое слово, число

Лексема — последовательность символов, составляющих отдельный токен.



Токены

Поток входных символов в лексическом анализаторе:

```
int f(int a, int b)\n{ return a + b; }
```

Поток токенов, получаемый на основе данного исходного кода:

```
<Type, "int"> <Id, "f"> <LParent, "(">, <Type, "int">, <Id, "a">, <Type, "int">,
<Id, "b">, <RParent, ")">, <LBrace, "{">, <Ret, "return">, <Id, "a">,
<Plus, "+"> <Id, "b">, <Semi, ";"> <RBrace, "}">
```

Лексемы

Является ли `PI` лексемой:

```
#define PI 3.14159265359

float circleSquare(float radius)
{
   return radius * PI * PI;
}
```

Лексемы

Является ли `PI` лексемой:

```
#define PI 3.14159265359

float circleSquare(float radius)
{
   return radius * PI * PI;
}
```

В простых случаях лексемы `PI` в коде не будет, т.к. она исчезнет после препроцессирования:

```
<Type, "float">, <Id, "circleSquare">, <LParent, "(">, <Type, "float">,
<Id, "radius">, <RParent, ")", <LBrace, "{">, <Ret, "return">, <Id, "radius">,
<Mul, "*">, <FpLit, "3.14159265359">, <Mul, "*">, <FpLit, "3.14159265359">,
<Semi, ";">, <RBrace, "}">
```

Лексемы

Стадии трансляции в соответствии со стандартом языка Си (5.1.1.2 Translation phases):

1. Отображение многобайтовых символов в допустимое множество символов, замена триграфов соответствующими символами.
2. Удаление переносов строк, стоящих после символа `'\'`.
3. Выделение токенов с командами препроцессора, замена комментариев на символ пробела, все последовательности пробелов заменяются одним пробелом.
4. Раскрытие команд препроцессора.
5. Преобразование управляющих символов в соответствующие члены множества исполняющих символов.
6. Соединение разделённых строковых литералов

Далее запускается синтаксический анализ и последующие стадии компиляции.

Токены

Чтобы выделить лексему и определить её класс, необходимо как-то её описать.

Неформальное описание идентификатора: *набор букв или цифр или подчёркиваний _, начинающихся с не цифры*.

Более формальное описание идентификатора в языке Си: *An identifier is a sequence of non digit characters (including the underscore _, the lowercase and uppercase Latin letters, and other characters) and digits, which designates one or more entities as described in 6.2.1. Lowercase and uppercase letters are distinct.*

Токены

An identifier is a sequence of non digit characters (including the underscore _, the lowercase and uppercase Latin letters, and other characters) and digits, which designates one or more entities as described in 6.2.1. Lowercase and uppercase letters are distinct.

Проблема: для описания лексем необходима более формальная нотация.

Наиболее распространённой нотацией являются **регулярные выражения**:

`[a-zA-Z_][a-zA-Z0-9_]*` — регулярное выражение, описывающее идентификатор.

Регулярные выражения

Алфавит — конечное непустое множество символов.

Пример

$V_1 = \{0, 1\}$

$V_2 = \{a, b, \ldots, z\}$

Цепочка (слово) — конечная последовательность символов алфавита.

Пример

0100101 — слово алфавита V_1

elbrus — слово алфавита V_2

Регулярные выражения

Язык — множество цепочек, каждая из которых принадлежит V^*:
$V^* = \{\varepsilon\} \cup V^+ = \{\varepsilon\} \cup V^1 \cup V^2 \cup V^3 \cup \ldots$

$\{\varepsilon\}$ — пустая цепочка

V^n — множество всех цепочек длины n

$L \subseteq V^*$ — язык над V. Алфавит языка обязан быть конечным.

Проблема — вопрос о том, является ли данная цепочка элементом определённого языка

Регулярные выражения

Регулярное выражение — способ алгебраического описания языка.

Операции над языками, задаваемыми регулярными выражениями:

Объединение языков L и M: $L \cup M$ — множество цепочек, содержащихся либо в L, либо в M, либо в обоих языках.

Конкатенация языков L и M: $L.M$ — множество цепочек, образующихся дописыванием любой цепочке из L любой цепочки из M.

Замыкание «Клини» языка L: L^* — множество всех цепочек, образующихся конкатенацией любого количества цепочек из L.

Регулярные выражения

Регулярные выражения были разработаны Клини в 1950-х годах как способ описания конечно-автоматной модели нервной деятельности.

Позднее Томпсон встроил их в редактор QED, а затем а ed, написанных для ОС UNIX, после чего они и получили свою популярность.

Сейчас регулярные выражения используются во всех программах обработки текста.

Регулярные выражения

Базовые регулярные выражения POSIX (BRE — Basic Regular Expressions) — представляют из себя традиционный синтаксис регулярных выражений UNIX, определённый в POSIX.

Расширенные регулярные выражения POSIX (ERE — extended regular expressions) — синтаксис аналогичен BRE, добавлены некоторые метасимволы, изменено поведение некоторых выражений.

Регулярные выражения, совместимые с Perl (PCRE — Perl Compatible Regular Expressions) — данные выражения обладают значительно более мощным и богатым синтаксисом чем POSIX–совместимые.

Регулярные выражения

Элементы синтаксиса PCRE:

`c`	Просто символ `'c'`
`.`	Произвольный символ
`c*`	Символ `'c'` повторяется 0 или более раз
`c+`	Символ `'c'` повторяется 1 или более раз
`c?`	Символ `'c'` повторяется 0 или 1 раз
`|`	альтернатива (или)
`()`	Группировка символов
`[]`	Класс символов

Примеры:

`[a-zA-Z_][a-zA-Z0-9_]*` — идентификатор

`(if|then|else)` — некоторые ключевые слова

`[0-9]+` — целое число

Регулярные выражения

Пример ключевых слов в языке Си:

```
auto            * if            unsigned
break             inline        void
case              int           volatile
char              long          while
const             register      _Alignas
continue          restrict      _Alignof
default           return        _Atomic
do                short         _Bool
double            signed        _Complex
else              sizeof        _Generic
enum              static        _Imaginary
extern            struct        _Noreturn
float             switch        _Static_assert
for               typedef       _Thread_local
goto              union
```

Регулярные выражения

Часть регулярного выражения, контролирующего корректность адреса электронной почты в соответствии с RFC822 (оно не поддерживает все возможности, представленные в стандарте):

```
(?:(?:\r\n)?[ \t])*(?:(?:(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\
    Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t]))*"(?:(?:\r\n
    )?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\] \000-\031]+(?:(?:(?:\r
    \n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|"(?:[^\"\r\\]|\\.|(?:(?:\r\n)?[ \t])
    )*"(?:(?:\r\n)?[ \t])*))*@(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\]
    \000-\031]+(?:(?:(?:\r\n)?[ \t])+|\Z|(?=[\["()<>@,;:\\".\[\]]))|\[([^\[\]\r
    \\]|\\.)*\](?:(?:\r\n)?[ \t])*)(?:\.(?:(?:\r\n)?[ \t])*(?:[^()<>@,;:\\".\[\]
    \000-\031]+(?:(?:(?:\r\n)?[ \t])+
```

Полная версия содержит 6343 символа.

Ошибки проектирования языков

Для упрощения реализации лексического анализатора в некоторых языках делаются допущения, которые в последствии могут приводить к ошибкам на стороне программиста.

Например в языках Fortran и PL/1 синтаксический анализатор игнорировал пробелы. Из-за этого появлялись схожие на вид, но совершенно разные конструкции:

`DO 5 I = 1.25`	Присвоение `DO5I := 1.25`
`DO 5 I = 1,25`	Цикл `for i := 1 to 25 do ...`

Ошибки проектирования языков

В лексическом анализаторе языке PL/1 не было предусмотрено ключевых слов, поэтому следующая конструкция являлась корректной:

```
IF ELSE THEN THEN = ELSE; ELSE ELSE = THEN
```

Ошибки проектирования языков

В лексическом анализаторе языке PL/1 не было предусмотрено ключевых слов, поэтому следующая конструкция являлась корректной:

```
IF ELSE THEN THEN = ELSE; ELSE ELSE = THEN
```

А две конструкции с разницей в один символ несли разную семантику:

`DECLARE (A, B, C, D);`	Вызов процедуры DECLARE
`DECLARE (A, B, C, D)=`	Начало объявления процедуры

Лексические анализаторы

В исходном тексте возможны неоднозначности. Например:

- `if8` — является ли идентификатором или двумя лексемами `if` и 8?
- `(if` — является ли идентификатором или ключевым словом?

Лексические анализаторы

В исходном тексте возможны неоднозначности. Например:

- `if8` — является ли идентификатором или двумя лексемами `if` и 8?
- `(if` — является ли идентификатором или ключевым словом?

Обычно лексический анализатор следует следующим правилам:

- **Наибольшее совпадение** — совпадением выбирается подстрока, являющаяся наиболее длинным вхождением.
- **Правило приоритета** — если лексема соответствует нескольким выражениям, то выбирается то, которое было записано первым.

Лексические анализаторы

Проблема: регулярное выражение способно описывать лексемы, но оно не даёт машине понять как проверять строки на соответствие.

Для разбора регулярного выражения необходим механизм, по заданному выражению выдающий алгоритм действий, позволяющий проверить строку на соответствие данному выражению.

Конечные автоматы

Конечный автомат — набор состояний и правил переходов между ними.

Детерминированный конечный автомат (ДКА) $A = (Q, V, \delta, q_0, F)$

Q	—	конечное множество состояний
V	—	множество входных символов
δ	—	функция перехода
$q_0 \in Q$	—	начальное состояние
$F \in Q$	—	множество допускающих состояний

Конечные автоматы

Как работает конечный автомат:

1. Попадаем в стартовое состояние q_0.
2. Считываем входной символ v_i и переходим в состояние в соответствии с правилом из δ: $q_i \xrightarrow{v_i} q_j$. Повторяем пока не закончатся входные символы.
3. Если входные символы кончились и мы в допускающем состоянии $q_i \in F$, то допускаем цепочку.
4. В противном случае - отвергаем.

Конечные автоматы

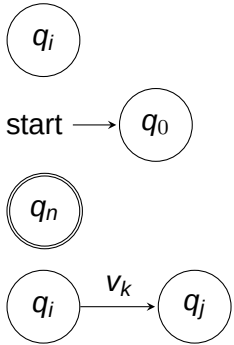
Пример простого конечного автомата:

$$
\begin{array}{ll}
Q & \{q_0, q_1, q_2\} \\
V & \{0, 1\} \\
\delta & \begin{cases}
\delta(q_0, 0) = q_0 \\
\delta(q_0, 1) = q_1 \\
\delta(q_1, 0) = q_1 \\
\delta(q_1, 1) = q_2 \\
\delta(q_2, 0) = q_2 \\
\delta(q_2, 1) = q_2
\end{cases} \\
q_0 \in Q & \\
F \in Q & \{q_2\}
\end{array}
$$

Конечные автоматы

Для визуализации конечных автоматов используют **диаграммы переходов**:

- Состояние
- Начальное состояние
- Допускающее состояние
- Переход



Конечные автоматы

Q $\{q_0, q_1, q_2\}$

V $\{0, 1\}$

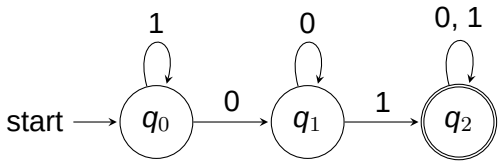
δ

$$\begin{cases} \delta(q_0, 0) = q_1 \\ \delta(q_0, 1) = q_0 \\ \delta(q_1, 0) = q_1 \\ \delta(q_1, 1) = q_2 \\ \delta(q_2, 0) = q_2 \\ \delta(q_2, 1) = q_2 \end{cases}$$

$q_0 \in Q$

$F \in Q$ $\{q_2\}$

Диаграмма для данного ДКА:



Конечные автоматы

Недетерминированный конечный автомат (НКА) $A = (Q, V, \delta, q_0, F)$

Q	—	конечное множество состояний
V	—	множество входных символов
δ	—	функция перехода, возвращающая **множество** состояний
$q_0 \in Q$	—	начальное состояние
$F \in Q$	—	множество допускающих состояний

НКА может одновременно находиться в нескольких состояниях одновременно.

Конечные автоматы

Q $\{q_0, q_1, q_2\}$

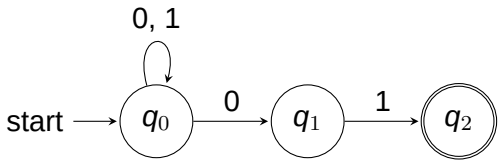
V $\{0, 1\}$

δ
$$\begin{cases} \delta(q_0, 0) = \{q_0, q_1\} \\ \delta(q_0, 1) = q_1 \\ \delta(q_1, 1) = q_2 \\ \delta(q_2, 0) = q_2 \\ \delta(q_2, 1) = q_2 \end{cases}$$

$q_0 \in Q$

$F \in Q$ $\{q_2\}$

Диаграмма для данного НКА:



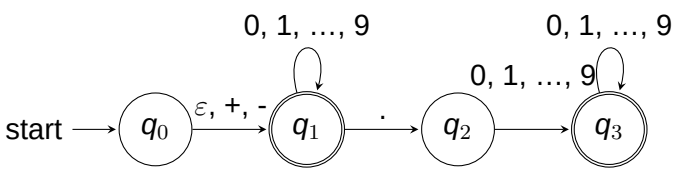
Конечные автоматы

Для удобства в конечные автоматы можно добавить переход по пустому символу — ε-переход.

Конечные автоматы

Для удобства в конечные автоматы можно добавить переход по пустому символу — ε-переход.

ε–НКА допускающий подмножество плавающих чисел:



Конечные автоматы

Для автомата $A = (Q, V, \delta, q_0, F)$ можно определить язык $L(A)$:
$L(A) = \{w | \delta(q_0, w) \in F\}$.

Если язык L есть $L(A)$ для некоторого НКА A, то говорят что L является **регулярным языком**.

ДКА, НКА и ε–НКА имеют одинаковую мощность.

Конечные автоматы

Если $L = L(A)$ для некоторого ДКА A, то существует регулярное выражение R, причём $L = L(R)$.

Любой язык, определённый некоторым регулярным выражением можно задать конечным автоматом.

Конечные автоматы

В привычном виде модель конечного автомата появилась благодаря работам Хаффмана и Мура в 1955-1956 годах. Недетерминированные конечные автоматы были предложены Рабином и Скоттом в 1959 году. Они же показали эквивалентность ДКА и НКА.

Литература

- *А. В. Ахо, М. С. Лам, Р. Сети, Д. Д. Ульман* Компиляторы: принципы, технологии и инструменты.
- *K. D. Cooper, L. Torczon* Engineering a compiler.
- *Д. Хопкрофт, Р. Мотвани, Д. Ульман* Введение в теорию автоматов, языков и вычислений
- *Р. Хантер* Основные концепции компиляторов
- CS 143 — Compilers, Stanford University