

ПОСТРОЕНИЕ КОМПИЛЯТОРОВ

ИЗДАТЕЛЬСТВО
ТЕХНИКО-ТЕОРЕТИЧЕСКОЙ ЛИТЕРАТУРЫ
КОМАНДЫ РАЗРАБОТЧИКОВ
КОМПИЛЯТОРА ОБЕРОНА

Р И Г А 2 0 2 0

Часть I

Введение в устройство компилятора

Глава 1

Общее подразделение компилятора

Компилятор преобразует исходный код программы в машинный код электронно-вычислительной машины и состоит из следующих модулей:

- драйвер текста,
- лексический анализатор,
- синтаксический анализатор,
- символьная таблица,
- кодогенератор.

Драйвер текста разбивает исходный код программы на отдельные литеры и передаёт их по одной лексическому анализатору¹.

Лексический анализатор принимает литеры одну за другой и собирает их в лексемы языка программирования. Примеры лексем: `module`, `BEGIN`, `+`, `:=`, `"Михи-джаки!"`, `47.813`, `>=`, `myVariable`. Лексемы передаются синтаксическому анализатору по одной, при этом сам лексический анализатор ровным счётом ничего не понимает (если здесь может идти речь о понимании) в том, что́ из этих лексем может получиться и как они должны взаимодействовать друг с другом. Другими словами,

¹В указанном взаимоотношении не драйвер текста, а лексический анализатор играет ведущую роль — он вызывает процедуру драйвера текста (`ReadCh`), которая только тогда считывает очередную литеру.

на данном этапе ничего не известно о *синтаксисе* языка, известно только то, какие *слова* существуют в этом языке. Лексический анализатор представлен внешнему миру (т. е. синтаксическому анализатору) процедурой `Get`.

Синтаксический анализатор собирает лексемы в синтаксические конструкции в соответствии с описанием языка программирования, такие как `IF` условие `THEN` действие `END` и обозначение `:=` выражение².

Синтаксический анализатор представляет собой главный модуль компилятора. Именно он руководит всей его работой.

Символьная таблица — это вспомогательный модуль, который использует синтаксический анализатор для хранения данных об объявленных в программе переменных, процедурах и т. д.³ Эти данные используются в процессе компиляции. Символьная таблица также отвечает за импорт модулей в компилируемую программу (таких, например, как `In` и `Out`).

Кодогенератор содержит набор процедур, которые вызываются синтаксическим анализатором в процессе разбора исходного кода и помещают в выходной файл генерируемый машинный код.

Итак, общая схема такова:⁴

Файл с исходным кодом «.mod» → (Исходный код программы) → Драйвер текста → (Литеры) → Лексический анализатор → (Лексемы) → Синтаксический анализатор и символьная таблица → Кодогенератор → (Машинный код) → Файл с машинным кодом

Как видно, данный процесс порождает файл с машинным кодом. В простейшем случае (когда вся программа состоит из одного модуля), на выходе получается исполнимый файл. В ДОС и Windows такой файл

²В данных примерах слова «условие», «действие» и т. д. подразумевают под собой некоторые другие синтаксические конструкции.

³Типы, константы, переменные и процедуры вместе называются объектами.

⁴Слова, указанные в скобках означают передаваемые данные, а слова без скобок — хранители или обработчики данных.

имеет расширение «.exe»⁵, тогда как в других операционных системах он может вовсе не иметь никакого расширения.

В более сложном случае, программа состоит из нескольких модулей, каждый из которых компилируется отдельно от остальных, в результате чего из каждого mod-файла получается два файла: «.sym» и «.o».

Файл с расширением «.o» называется *о б ъ е к т н ы м ф а й л о м*. Он содержит машинный код какого-то одного скомпилированного модуля. «.sym»-файл содержит данные об этом модуле, необходимые для того, чтобы из всех «.o»-файлов собрать исполнимый файл. Этот процесс называется *к о м п о н о в к о й*.

⁵Исполнимые файлы ДОС могут также иметь расширение «.com» — такой файл содержит чистый машинный код без каких либо дополнительных данных. Он загружается в ОЗУ (всегда в одно и то же место) и просто запускается.

Глава 2

Начинаем писать компилятор

2.1 Простейший драйвер текста

Драйвер текста представляет собой процедуру `readCh`. Её задача — считывать литеры из потока входных данных (исходного кода компилируемой программы) одну за другой. Результат работы процедуры `readCh` помещается в глобальную переменную `ch` типа `CHAR`.

В простейшем случае драйвер текста может брать код программы из переменной строкового типа. Тогда ещё необходима переменная-бегунок по строке — `i`.

```
1 VAR s: ARRAY 64 OF CHAR;  
2   i: INTEGER;
```

2.2 Простейший лексический анализатор

```
1 MODULE Lexer1;  
2 IMPORT In, Out, Files;  
3 CONST (* Допустимые лексемы *)  
4   null = 0; module = 1; import = 2; const = 3; var = 4; begin =  
5     5;  
6   end = 6; if = 7; then = 8; else = 9; elsif = 10; while = 11;  
7   ident = 12; semicol = 13; colon = 14; comma = 15; period = 16;  
8   becomes = 17; equ = 18; neq = 19; les = 20; leq = 21; gtr = 22;  
9   geq = 23; lparen = 24; rparen = 25; lbrak = 26; rbrak = 27;  
10  lbrace = 28; rbrace = 29; int = 30; string = 31; not = 32;  
11  and = 33; or = 34; plus = 35; minus = 36; rdiv = 37; div = 38;  
12  mod = 39; times = 40; eot = 64;  
13 VAR f: Files.File;  
14     r: Files.Rider;  
15     ch: CHAR;  
16     sym: INTEGER;  
17     id: ARRAY 32 OF CHAR;  
18     idlen: INTEGER;  
19     ival: INTEGER;
```

6

```
19
20 PROCEDURE ReadCh;
21 BEGIN
22     IF r.eof THEN ch := 0X
23     ELSE Files.Read(r, ch)
24     END
25 END ReadCh;
26
27 PROCEDURE Get;
28 BEGIN
29     WHILE (ch # 0X) & (ch <= " ") DO
30         ReadCh
31     END;
32
33     IF r.eof THEN sym := eot
34     ELSIF ch = "," THEN sym := comma; ReadCh
35     ELSIF ch = "." THEN sym := period; ReadCh
36     ELSIF ch = ";" THEN sym := semicol; ReadCh
37     ELSIF ch = "+" THEN sym := plus; ReadCh
38     ELSIF ch = "-" THEN sym := minus; ReadCh
39     ELSIF ch = "&" THEN sym := and; ReadCh
40     ELSIF ch = "~" THEN sym := not; ReadCh
41     ELSIF ch = "(" THEN sym := lparen; ReadCh
42     ELSIF ch = ")" THEN sym := rparen; ReadCh
43     ELSIF ch = "[" THEN sym := lbrak; ReadCh
44     ELSIF ch = "]" THEN sym := rbrak; ReadCh
45     ELSIF ch = "{" THEN sym := lbrace; ReadCh
46     ELSIF ch = "}" THEN sym := rbrace; ReadCh
47     ELSIF ch = ":" THEN
48         ReadCh;
49         IF ch = "=" THEN sym := becomes; ReadCh
50         ELSE sym := colon
51         END
52     ELSIF ("A" <= ch) & (ch <= "Z") OR ("a" <= ch) & (ch <= "z")
53     THEN
54         id[0] := ch; idlen := 1; ReadCh;
55         WHILE ("A" <= ch) & (ch <= "Z") OR
56             ("a" <= ch) & (ch <= "z") OR
57             ("0" <= ch) & (ch <= "9") DO
58             IF idlen < LEN(id) - 1 THEN
59                 id[idlen] := ch;
60                 INC(idlen)
61             END;
62             ReadCh
63         END;
64         id[idlen] := 0X;
```

```

64     IF id = "MODULE" THEN sym := module
65     ELSIF id = "IMPORT" THEN sym := import
66     ELSIF id = "CONST" THEN sym := const
67     ELSIF id = "VAR" THEN sym := var
68     ELSIF id = "BEGIN" THEN sym := begin
69     ELSIF id = "END" THEN sym := end
70     ELSIF id = "IF" THEN sym := if
71     ELSIF id = "THEN" THEN sym := then
72     ELSIF id = "ELSE" THEN sym := else
73     ELSIF id = "ELSIF" THEN sym := elsif
74     ELSIF id = "WHILE" THEN sym := while
75     ELSIF id = "DIV" THEN sym := div
76     ELSIF id = "MOD" THEN sym := mod
77     ELSIF id = "OR" THEN sym := or
78     ELSE sym := ident
79     END
80   ELSIF ("0" <= ch) & (ch <= "9") THEN
81     ival := ORD(ch) - ORD("0"); ReadCh;
82     WHILE ("0" <= ch) & (ch <= "9") DO
83       ival := ival * 10 + ORD(ch) - ORD("0");
84       ReadCh
85     END;
86     sym := int;
87   ELSE
88     sym := null;
89     ReadCh
90   END
91 END Get;
92
93 BEGIN
94   f := Files.Old("Test.Mod");
95   IF f = NIL THEN
96     Out.String("No file");
97     Out.Ln
98   ELSE
99     Files.Set(r, f, 0);
100    ReadCh;
101    Get;
102    WHILE sym # eot DO
103      Out.Int(sym, 4);
104      IF sym = int THEN
105        Out.Char("("); Out.Int(ival, 0); Out.Char(")")
106      END;
107      Get
108    END
109  END;

```

8

```
110   Out.Ln
111 END Lexer1.
```

Оглавление

Введение в устройство компилятора	2
Общее подразделение компилятора	2
Начинаем писать компилятор	5
Простейший драйвер текста	5
Простейший лексический анализатор	5