

今回の内容

6.1 Minimum C コンパイラの字句解析部 . . . . .	6-1
6.2 演習問題 . . . . .	6-7

## 6.1 Minimum C コンパイラの字句解析部

Minimum C のトークンの文法を BNF 記法で表すと次のようなものとなります。〈*TokenSeq*〉がソースプログラム全体を表す非終端記号です。ただし〈*White*〉は、スペース、タブ、改行などの制御文字を表すものとし、〈*Any*〉は任意の文字を表すものとします。

$$\begin{aligned} \langle Digit \rangle &::= "0" \mid "1" \mid "2" \mid \dots \mid "9". \\ \langle Number \rangle &::= \langle Digit \rangle \{ \langle Digit \rangle \}. \\ \langle Alphabet \rangle &::= "a" \mid "b" \mid "c" \mid \dots \mid "z" \mid "A" \mid "B" \mid "C" \mid \dots \mid "Z". \\ \langle Ident \rangle &::= \langle Alphabet \rangle \{ \langle Alphabet \rangle \mid \langle Digit \rangle \mid "_" \}. \\ \langle Token \rangle &::= "(" \mid ")" \mid "{" \mid "}" \mid "*" \mid "%" \mid "+" \mid "-" \mid "," \mid ";" \mid "!" "=" \\ &\quad \mid "=" [ "=" ] \mid "<" [ "=" ] \mid ">" [ "=" ] \mid "/" [ "*" \{ \langle Any \rangle \} "*" "/" ] \\ &\quad \mid \langle Number \rangle \\ &\quad \mid "i" "n" "t" \mid "r" "e" "t" "u" "r" "n" \mid "i" "n" "p" "u" "t" \mid "p" "r" "i" "n" "t" \\ &\quad \mid "w" "h" "i" "l" "e" \mid "i" "f" \mid "e" "l" "s" "e" \\ &\quad \mid \langle Ident \rangle. \\ \langle TokenSeq \rangle &::= \{ \langle White \rangle \} \{ \langle Token \rangle \{ \langle White \rangle \} \}. \end{aligned}$$

### 字句解析部の仕事

この文法に基づいて Minimum C コンパイラの字句解析部を構築しますが、その仕事とは次のようなものです。

1. ソースファイル中の空白類の文字やコメント `/* ... */` は読み飛ばす。これらは、トークンとして構文解析部には渡されない。
2. 改行文字を数えながら、現在処理中の行がソースファイルの第何行目であるかを大域変数 `line` に保持する。この情報は、ソースプログラム中の文法間違いなどをコンパイラがユーザーに報告する際に用いられる。
3. ソースファイルの文字を1文字分先読みしながら、第4回で解説した構文解析の手法と同じ要領でトークンを切り出していく。
4. 切り出したトークンの種別を大域変数 `token` に格納する。
5. 識別子 〈*Ident*〉 に対しては、そのトークンが予約語でないことを確認した上で、変数 `token` に `mc_ident` (識別子であることを表す定数) を格納するとともに、識別子の名前 (文字列) を大域変数 `ident` に格納する。

6. 定数  $\langle \text{Number} \rangle$  に対しては、`token` の値を `mc_number` (定数であることを表す定数) とし、それを十進数と解釈したときの数値を大域変数 `number` に格納する。

メモ

### 字句解析部のプログラム (`scan.h` と `scan.c`)

以下は Minimum C コンパイラの字句解析部のプログラムです。このプログラムの中心となる関数は `GetToken()` で、この関数が呼ばれるごとに、大域変数 `token` の値に読み込んだトークンの種別が格納されます。このプログラムで使われている `strcmp()` は、文字列を比較する標準ライブラリ関数で、2つの文字列が同じであれば0を返します。

scan.h

```
1 /*
2 *      scan.h
3 */
4
5 #define MC_MAXNUM      (((unsigned long)1<<31)-1) /* 定数の最大値 */
6 #define MC_MAXIDLEN   (32)                       /* 識別子の最大長 */
7
8 typedef enum {
9     mc_null,
10    mc_ident,      /* <Ident> (識別子) */
11    mc_number,     /* <Number> (定数) */
12    mc_mult,       /* * */
13    mc_div,        /* / */
14    mc_mod,        /* % */
15    mc_plus,       /* + */
16    mc_minus,      /* - */
17    mc_assign,     /* = */
18    mc_eq,         /* == */
19    mc_neq,        /* != */
20    mc_lt,         /* < */
21    mc_le,         /* <= */
22    mc_gt,         /* > */
23    mc_ge,         /* >= */
24    mc_comma,      /* , */
25    mc_semicolon,  /* ; */
26    mc_lparen,     /* ( */
27    mc_rparen,     /* ) */
28    mc_lcb,        /* { */
29    mc_rcb,        /* } */
30    mc_int,        /* int */
31    mc_return,     /* return */
32    mc_input,      /* input */
33    mc_print,      /* print */
34    mc_while,      /* while */
35    mc_if,         /* if */
```

```

36     mc_else,          /* else */
37     mc_eof           /* ファイルの終り */
38 } Token;
39
40 extern void          Error(char *);
41 extern void          GetToken(void);
42 extern int           OpenFile(char *);
43 extern void          CloseFile(void);
44
45 extern Token         token;
46 extern char *        ident;
47 extern unsigned long number;
48 extern unsigned int  line;
49

```

メモ

```

1 /*
2 *      scan.c
3 */
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <string.h>
8 #include "scan.h"
9
10 Token          token;
11 char *         ident;
12 unsigned long  number;
13 unsigned int   line;
14
15 static FILE *  source;
16 static int     ch;
17
18 typedef struct {
19     char *      string;
20     Token       token;
21 } Keyword;
22
23 static Keyword keywords[] = {
24     { "int",          mc_int          },
25     { "return",      mc_return       },
26     { "input",       mc_input        },
27     { "print",       mc_print        },
28     { "while",       mc_while        },
29     { "if",          mc_if           },
30     { "else",        mc_else         },
31     { 0,             mc_null         }
32 };
33

```

```

34 void Error(char *message)
35 {
36     fprintf (stderr, "第%d行: %s\n", line, message);
37     exit(1);
38 }
39
40 static void GetChar(void)
41 {
42     if (ch == EOF)
43         return;
44     if (ch == '\n')
45         line ++;
46     ch = getc(source);
47 }
48
49 static void Identifier(void)
50 {
51     static char buf[MC_MAXIDLEN+1];
52     int i = 0;
53
54     /* 識別子あるいは予約語を buf に読み取る */
55     do {
56         if (MC_MAXIDLEN <= i)
57             Error("識別子が長すぎます");
58         buf[i++] = ch;
59         GetChar();
60     } while ('0' <= ch && ch <= '9' || ch == '_'
61             || 'A' <= ch && ch <= 'Z' || 'a' <= ch && ch <= 'z');
62     buf[i] = '\0';
63
64     /* 予約語の表の中で検索 */
65     for (i = 0; keywords[i].string != (char *)0; i ++ ) {
66         if (strcmp(keywords[i].string, buf) == 0) {
67             token = keywords[i].token;
68             return;
69         }
70     }
71
72     /* 予約語でないので識別子 */
73     token = mc_ident;
74     ident = buf;
75 }
76
77 static void Number(void)
78 {
79     unsigned long n = 0;
80     int d;
81
82     do {
83         d = ch - '0';
84         if ((MC_MAXNUM - n) / 10 < n)
85             Error("定数値が大きすぎます");
86         n = 10*n + d;
87         GetChar();
88     } while ('0' <= ch && ch <= '9');
89
90     token = mc_number;
91     number = n;
92 }
93

```

```

94 static void Comment(void)
95 {
96     /* コメントの読みとばし */
97     GetChar();
98     do {
99         while (ch != '*' && ch != EOF)
100             GetChar();
101         GetChar();
102     } while (ch != '/' && ch != EOF);
103     GetChar();
104 }
105
106 void GetToken(void)
107 {
108     again:
109     /* 空白の読みとばし */
110     while (ch <= ' ') {
111         if (ch == EOF) {
112             token = mc_eof;
113             return;
114         }
115         GetChar();
116     }
117
118     /* トークンの識別 */
119     switch (ch) {
120         case '(' : GetChar(); token = mc_lparen; break;
121         case ')' : GetChar(); token = mc_rparen; break;
122         case '{' : GetChar(); token = mc_lcb; break;
123         case '}' : GetChar(); token = mc_rcb; break;
124         case '*' : GetChar(); token = mc_mult; break;
125         case '%' : GetChar(); token = mc_mod; break;
126         case '+' : GetChar(); token = mc_plus; break;
127         case '-' : GetChar(); token = mc_minus; break;
128         case ',' : GetChar(); token = mc_comma; break;
129         case ';' : GetChar(); token = mc_semicolon; break;
130         case '!' : GetChar();
131             if (ch != '=')
132                 Error("! の次は = でなければなりません");
133             GetChar();
134             token = mc_neq;
135             break;
136         case '=' : GetChar();
137             if (ch == '=') {
138                 GetChar();
139                 token = mc_eq;
140             }
141             else
142                 token = mc_assign;
143             break;
144         case '<' : GetChar();
145             if (ch == '=') {
146                 GetChar();
147                 token = mc_le;
148             }
149             else
150                 token = mc_lt;
151             break;
152         case '>' : GetChar();
153             if (ch == '=') {

```

```

154         GetChar();
155         token = mc_ge;
156     }
157     else
158         token = mc_gt;
159     break;
160     case '/': GetChar();
161             if (ch == '*') {
162                 Comment();
163                 goto again;
164             }
165             else
166                 token = mc_div;
167     break;
168     case '0': case '1': case '2': case '3': case '4':
169     case '5': case '6': case '7': case '8': case '9':
170         Number();
171     break;
172     case 'A': case 'B': case 'C': case 'D': case 'E':
173     case 'F': case 'G': case 'H': case 'I': case 'J':
174     case 'K': case 'L': case 'M': case 'N': case 'O':
175     case 'P': case 'Q': case 'R': case 'S': case 'T':
176     case 'U': case 'V': case 'W': case 'X': case 'Y': case 'Z':
177     case 'a': case 'b': case 'c': case 'd': case 'e':
178     case 'f': case 'g': case 'h': case 'i': case 'j':
179     case 'k': case 'l': case 'm': case 'n': case 'o':
180     case 'p': case 'q': case 'r': case 's': case 't':
181     case 'u': case 'v': case 'w': case 'x': case 'y': case 'z':
182         Identifier();
183     break;
184     default: Error("無意味な文字が現れました");
185     break;
186 }
187 }
188
189 int OpenFile(char * filename)
190 {
191     source = fopen(filename, "r");
192     if (source == NULL)
193         return -1;
194     line = 1;
195     ch = ' ';
196     return 0;
197 }
198
199 void CloseFile(void)
200 {
201     fclose(source);
202 }

```

メモ

## 6.2 演習問題

1. *TokenSeq* の文法は、コメント `/*...*/` に関する曖昧性を持っていることを示しなさい。また、例えば次の文字列が、実際にはどのような *Token* の列として字句解析部 (`scan.c`) で認識されるか考察しなさい。

```
x/*y*/123*/z
```

2. *TokenSeq* の文法は、識別子 *Ident* となり得るような文字列に関する曖昧性を持っていることを示しなさい。また、例えば次の文字列が、実際にはどのような *Token* の列として字句解析部 (`scan.c`) で認識されるか考察しなさい。

```
whilexyz
```

3. 今回の字句解析部 (`scan.c` と `scan.h`) を利用して、文法チェックのみを行う Minimum C のパーザを書きなさい。構文図が決定的でなくても、第4回に解説した「構文図からのパーザの生成」法に従ってとりあえずパーザを書いてみることに。

## 付録：第4回演習問題の解答例

### 2. 機械的に構文図を変換した場合

```
#include <stdio.h>

void HizeroSuji(void)
{
    switch (token) {
        case '1':
            if (token == '1')
                gettoken();
            else
                error();
            break;
            :
        case '9':
            if (token == '9')
                gettoken();
            else
                error();
            break;
        default:
            error();
    }
}

void Suji(void)
{
    switch (token) {
        case '0':
            if (token == '0')
                gettoken();
            else
                error();
            break;
        case '1': case '2': case '3':
        case '4': case '5': case '6':
        case '7': case '8': case '9':
            HizeroSuji();
            break;
        default:
            error();
    }
}

void Shosubu(void)
{
    if (token == '.')
        gettoken();
    else
        error();
    while (token == '0') {
        if (token == '0')
            gettoken();
        else
            error();
    }
    HizeroSuji();
    while (token == '0' || token == '1' ||
           token == '2' || token == '3' ||
           token == '4' || token == '5' ||
           token == '6' || token == '7' ||
           token == '8' || token == '9') {
        while (token == '0') {
            if (token == '0')
                gettoken();
            else
                error();
        }
    }
}
```

### ある程度効率化した場合

```
#include <stdio.h>

void HizeroSuji(void)
{
    switch (token) {
        case '1':
        case '2':
            :
        case '9':
            gettoken();
            break;
        default:
            error();
    }
}

void Suji(void)
{
    switch (token) {
        case '0':
            gettoken();
            break;

        case '1': case '2': case '3':
        case '4': case '5': case '6':
        case '7': case '8': case '9':
            HizeroSuji();
            break;
        default:
            error();
    }
}

void Shosubu(void)
{
    if (token == '.')
        gettoken();
    else
        error();
    do {
        while (token == '0')
            gettoken();

        HizeroSuji();
    } while ('0' <= token
           && token == '9');
}
```



```

        }
        HizeroSuji();
    }
}

void Ichiljo(void)
{
    HizeroSuji();
    while (token == '0' || token == '1' ||
           token == '2' || token == '3' ||
           token == '4' || token == '5' ||
           token == '6' || token == '7' ||
           token == '8' || token == '9') {
        Suji();
    }
    if (token == '.')
        Shosubu();
}

void Jissu(void)
{
    switch (token) {
    case '0':
        if (token == '0')
            gettoken();
        else
            error();
        if (token == '.')
            Shosubu();
        break;
    case '1': case '2': case '3':
    case '4': case '5': case '6':
    case '7': case '8': case '9':
        Ichiljo();
        break;
    case '-':
        if (token == '-')
            gettoken();
        else
            error();
        switch (token) {
        case '0':
            if (token == '0')
                gettoken();
            else
                error();
            Shosubu();
            break;
        case '1': case '2': case '3':
        case '4': case '5': case '6':
        case '7': case '8': case '9':
            Ichiljo();
            break;
        default:
            error();
        }
        break;
    default:
        error();
    }
}

void parse(void)
{
    gettoken();
    Jissu();
    if (token != EOF)
        error();
}

```

```

void Ichiljo(void)
{
    HizeroSuji();
    while ('0' <= token
           && token == '9') {
        Suji();
    }

    if (token == '.')
        Shosubu();
}

void Jissu(void)
{
    switch (token) {
    case '0':
        gettoken();

        if (token == '.')
            Shosubu();
        break;
    case '1': case '2': case '3':
    case '4': case '5': case '6':
    case '7': case '8': case '9':
        Ichiljo();
        break;
    case '-':
        gettoken();

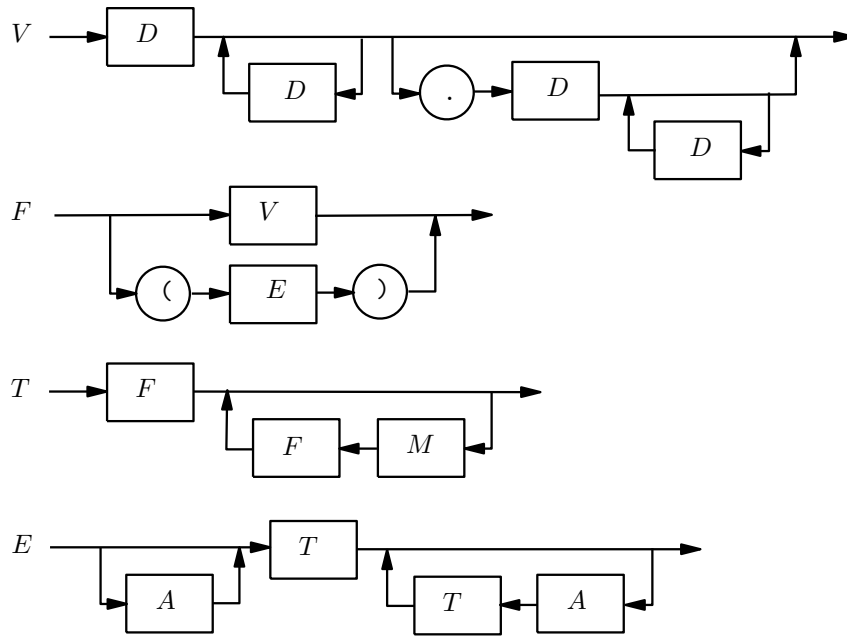
        switch (token) {
        case '0':
            gettoken();

            Shosubu();
            break;
        case '1': case '2': case '3':
        case '4': case '5': case '6':
        case '7': case '8': case '9':
            Ichiljo();
            break;
        default:
            error();
        }
        break;
    default:
        error();
    }
}

void parse(void)
{
    gettoken();
    Jissu();
    if (token != EOF)
        error();
}

```

3. (1)



(2)

```

1 double F(void)
2 {
3     double v;
4
5     if ('0' <= t && t <= '9') {
6         v = V();
7     }
8     else if (t == '(') {
9         gettoken();
10        v = E();
11        if (t != ')')
12            error();
13        gettoken();
14    }
15    else {
16        error();
17    }
18    return v;
19 }
20
21 double T(void)
22 {
23     double v, w;
24
25     v = F();
26     while (t == '*' || t == '/') {
27         if (t == '*') {
28             gettoken();
29             v *= F();
30         }
31         else {
32             gettoken();
33             w = F();
34             if (w == 0.0) {
35                 printf("Division by 0\n");

```

```

36             exit(1);
37         }
38         v /= w;
39     }
40 }
41 return v;
42 }
43
44 double E(void)
45 {
46     double v = 1.0;
47
48     if (t == '+' || t == '-') {
49         if (t == '+') {
50             gettoken();
51         }
52         else {
53             gettoken();
54             v = -1.0;
55         }
56     }
57     v *= T();
58     while (t == '+' || t == '-') {
59         if (t == '+') {
60             gettoken();
61             v += T();
62         }
63         else {
64             gettoken();
65             v -= T();
66         }
67     }
68     return v;
69 }
70
71 void L(void)
72 {
73     double v;
74
75     while (('0' <= t && t <= '9') || t == '+'
76           || t == '-' || t == '(') {
77         v = E();
78         if (t != '=')
79             error();
80         printf ("%f\n", v);
81         gettoken();
82     }
83 }

```