抽象構文木（AST': Abstract Syntax Tree）

grammar rule

2.10.2 Shell Grammar Rules
[Command Name]
When the TOKEN is exactly a reserved word, the token identifier for that reserved word shall result. Otherwise, the token WORD shall be returned. Also, if the parser is in any state where only a reserved word could be the next correct token, proceed as above.
Note:
Because at this point characters are retained in the token, quoted strings cannot be recognized as reserved words. This rule also implies that reserved words are not recognized except in certain positions in the input, such as after a or ; the grammar presumes that if the reserved word is intended, it is properly delimited by the user, and does not attempt to reflect that requirement directly. Also note that line joining is done before tokenization, as described in Escape Character (Backslash), so escaped characters are already removed at this point.
Rule 1 is not directly referenced in the grammar, but is referred to by other rules, or applies globally.
[Redirection to or from filename]
The expansions specified in Redirection shall occur. As specified there, exactly one field can result (or the result is unspecified), and there are additional requirements on pathname expansion.
[Redirection from here-document]
Quote removal shall be applied to the word to determine the delimiter that is used to find the end of the here-document that begins after the next .
[Case statement termination]
When the TOKEN is exactly the reserved word esac, the token identifier for esac shall result. Otherwise, the token WORD shall be returned.
[NAME in for]
When the TOKEN meets the requirements for a name (see XBD Name), the token identifier NAME shall result. Otherwise, the token WORD shall be returned.
[Third word of for and case]
[case only]
When the TOKEN is exactly the reserved word in, the token identifier for in shall result. Otherwise, the token WORD shall be returned.
[for only]
When the TOKEN is exactly the reserved word in or do, the token identifier for in or do shall result, respectively. Otherwise, the token WORD shall be returned.
(For a. and b.: As indicated in the grammar, a linebreak precedes the tokens in and do. If characters are present at the indicated location, it is the token after them that is treated in this fashion.)
[Assignment preceding command name]
[When the first word]
If the TOKEN does not contain the character '=', rule 1 is applied. Otherwise, 7b shall be applied.
[Not the first word]
If the TOKEN contains an unquoted (as determined while applying rule 4 from Token Recognition) character that is not part of an embedded parameter expansion, command substitution, or arithmetic expansion construct (as determined while applying rule 5 from Token Recognition):
If the TOKEN begins with '=', then rule 1 shall be applied.
If all the characters in the TOKEN preceding the first such form a valid name (see XBD Name), the token

ASSIGNMENT_WORD shall be returned.
Otherwise, it is unspecified whether rule 1 is applied or ASSIGNMENT_WORD is returned.
Otherwise, rule 1 shall be applied.
Assignment to the name within a returned ASSIGNMENT_WORD token shall occur as specified in Simple Commands.
[NAME in function]
When the TOKEN is exactly a reserved word, the token identifier for that reserved word shall result. Otherwise, when the TOKEN meets the requirements for a name, the token identifier NAME shall result. Otherwise, rule 7 applies.
[Body of function]
Word expansion and assignment shall never occur, even when required by the rules above, when this rule is being parsed. Each TOKEN that might either be expanded or have assignment applied to it shall instead be returned as a single WORD consisting only of characters that are exactly the token described in Token Recognition.

バッカス・ナウア記法によるアメリカの住所表記

```
<住所> ::= <名前> <住所> <郵便番号>
<名前> ::= <個人名> <姓> <敬称> <EOL>
           ¦ <個人名> <名前> <EOL>

<個人名> ::= <名> ¦ <頭文字> "."
<住所> ::= <部屋番号> <番号> <丁目> <EOL>
 <郵便番号> ::= <町名> "," <州名> <郵便コード> <EOL>
```

バッカス・ナウア記法によるshellコマンドの表記（課題の範囲のみ）

```
command_line ::=
    "\n"
  ¦ sequencial_commands delimiter "\n"
  ¦ sequencial_commands "\n"
delimiter ::=
    ";"
sequencial_commands ::=
    piped_commands delimiter sequencial_commands
  ¦ piped_commands
piped_commands ::=
    command "¦" piped_commands
  ¦ command
command ::=
    arguments
arguments ::=
    redirection
  ¦ redirection arguments
  ¦ string
  ¦ string arguments
string ::=
    expandable <no_space> string
  ¦ expandable
  ¦ not_expandable <no_space> string
  ¦ not_expandable
  ¦ expandable_quoted <no_space> string
  ¦ expandable_quoted
redirection ::=
    "<" string
  ¦ ">" string
  ¦ ">>" string
  ¦ "<<" string
```

【参考】バッカス・ナウア記法によるshellコマンドの表記（すべて）

```
/* -------------------------------------------------------
   The Grammar
   ------------------------------------------------------- */
%start program
%%
program          : linebreak complete_commands linebreak
                 ¦ linebreak
                 ;
complete_commands: complete_commands newline_list complete_command
                 ¦                                complete_command
                 ;
complete_command : list separator_op
                 ¦ list
                 ;
list             : list separator_op and_or
                 ¦                   and_or
                 ;
and_or           :                         pipeline
                 ¦ and_or AND_IF linebreak pipeline
                 ¦ and_or OR_IF  linebreak pipeline
                 ;
pipeline         :      pipe_sequence
                 ¦ Bang pipe_sequence
                 ;
pipe_sequence    :                             command
                 ¦ pipe_sequence '¦' linebreak command
                 ;
command          : simple_command
                 ¦ compound_command
                 ¦ compound_command redirect_list
                 ¦ function_definition
                 ;
compound_command : brace_group
                 ¦ subshell
                 ¦ for_clause
                 ¦ case_clause
                 ¦ if_clause
                 ¦ while_clause
                 ¦ until_clause
                 ;
subshell         : '(' compound_list ')'
                 ;
compound_list    : linebreak term
                 ¦ linebreak term separator
                 ;
term             : term separator and_or
                 ¦                and_or
                 ;
```

```
for_clause      : For name                              do_group
               ¦ For name                   sequential_sep do_group
               ¦ For name linebreak in         sequential_sep do_group
               ¦ For name linebreak in wordlist sequential_sep do_group
               ;
name            : NAME                   /* Apply rule 5 */
               ;
in              : In                     /* Apply rule 6 */
               ;
wordlist        : wordlist WORD
               ¦        WORD
               ;
case_clause     : Case WORD linebreak in linebreak case_list    Esac
               ¦ Case WORD linebreak in linebreak case_list_ns Esac
               ¦ Case WORD linebreak in linebreak              Esac
               ;
case_list_ns    : case_list case_item_ns
               ¦         case_item_ns
               ;
case_list       : case_list case_item
               ¦         case_item
               ;
case_item_ns    :     pattern ')' linebreak
               ¦    pattern ')' compound_list
               ¦ '(' pattern ')' linebreak
               ¦ '(' pattern ')' compound_list
               ;
case_item       :     pattern ')' linebreak     DSEMI linebreak
               ¦    pattern ')' compound_list DSEMI linebreak
               ¦ '(' pattern ')' linebreak     DSEMI linebreak
               ¦ '(' pattern ')' compound_list DSEMI linebreak
               ;
pattern         :            WORD        /* Apply rule 4 */
               ¦ pattern '¦' WORD        /* Do not apply rule 4 */
               ;
if_clause       : If compound_list Then compound_list else_part Fi
               ¦ If compound_list Then compound_list           Fi
               ;
else_part       : Elif compound_list Then compound_list
               ¦ Elif compound_list Then compound_list else_part
               ¦ Else compound_list
               ;
while_clause    : While compound_list do_group
               ;
until_clause    : Until compound_list do_group
               ;
function_definition : fname '(' ')' linebreak function_body
               ;
```

```
function_body    : compound_command               /* Apply rule 9 */
                 ¦ compound_command redirect_list  /* Apply rule 9 */
                 ;
fname            : NAME                            /* Apply rule 8 */
                 ;
brace_group      : Lbrace compound_list Rbrace
                 ;
do_group         : Do compound_list Done           /* Apply rule 6 */
                 ;
simple_command   : cmd_prefix cmd_word cmd_suffix
                 ¦ cmd_prefix cmd_word
                 ¦ cmd_prefix
                 ¦ cmd_name cmd_suffix
                 ¦ cmd_name
                 ;
cmd_name         : WORD                   /* Apply rule 7a */
                 ;
cmd_word         : WORD                   /* Apply rule 7b */
                 ;
cmd_prefix       :            io_redirect
                 ¦ cmd_prefix io_redirect
                 ¦            ASSIGNMENT_WORD
                 ¦ cmd_prefix ASSIGNMENT_WORD
                 ;
cmd_suffix       :            io_redirect
                 ¦ cmd_suffix io_redirect
                 ¦            WORD
                 ¦ cmd_suffix WORD
                 ;
redirect_list    :               io_redirect
                 ¦ redirect_list io_redirect
                 ;
io_redirect      :           io_file
                 ¦ IO_NUMBER io_file
                 ¦           io_here
                 ¦ IO_NUMBER io_here
                 ;
io_file          : '<'       filename
                 ¦ LESSAND   filename
                 ¦ '>'       filename
                 ¦ GREATAND  filename
                 ¦ DGREAT    filename
                 ¦ LESSGREAT filename
                 ¦ CLOBBER   filename
                 ;
filename         : WORD                      /* Apply rule 2 */
                 ;
io_here          : DLESS     here_end
```

```
                 ¦ DLESSDASH here_end
                 ;
here_end         : WORD                    /* Apply rule 3 */
                 ;
newline_list     :              NEWLINE
                 ¦ newline_list NEWLINE
                 ;
linebreak        : newline_list
                 ¦ /* empty */
                 ;
separator_op     : '&'
                 ¦ ';'
                 ;
separator        : separator_op linebreak
                 ¦ newline_list
                 ;
sequential_sep   : ';' linebreak
                 ¦ newline_list
                 ;
```