情報工学実験 C コンパイラ実験最終レポート

氏名: 寺岡 久騎 (TERAOKA, Hisaki)
学生番号: 09B22433

出題日: 2025 年 1 月 26 日
提出日: 2025 年 2 月 4 日
締切日: 2025 年 2 月 4 日

1 本実験の目的

本実験の目的は, これまで学んできた C 言語についての知識を再確認するとともに, それを使って,

- ソフトウェア全体の仕様の決定
- プログラムで利用するデータ構造，アルゴリズムの考案と実装
- 動作のテスト，デバッグの作業

これら一連の作業を通して大規模なプログラムの作成の経験をすること，そして lex，yacc といったプログラムジェネレータを使用したコンパイラプログラムの作成を通して，コード解析やファイルシステムに利用される木構造の取り扱い，ソースコードとアセンブリ言語との対応についての理解と習熟を深めることである.

2 作成した言語の定義

以下に作成した言語の定義として，文法規則を記述した yacc のルールを示す．言語は，最終課題 1–5 が実行できる実装となっている.

```
%union {
  struct node *np;
  char* sp;
  int   ival;
}

%type <np> program declarations func_define decl_statement decl_part
           statements statement func_var_decl func_arg_part
           assignment_stmt assignment loop_stmt cond_stmt func_call
           while_stmt if_stmt else_stmt elif_stmt for_stmt
           expressions expression
           condition array array_index term factor unary_factor
           comp_op unary_op bit_op var idents

%type <ival> add_op mul_op

%token DEFINE ASSIGN ARRAY_DEF
       L_BRACKET R_BRACKET L_PARAN R_PARAN L_BRACE R_BRACE
```

```
19:         SEMIC COMMA ADD SUB MUL DIV REM INCREM DECREM EQ NE LT GT LTE GTE
20:         AND OR XOR NOT L_SHIFT R_SHIFT
21:         FUNCDECL FUNCCALL
22:         FOR WHILE IF ELSE
23:
24: %token <sp> IDENT
25: %token <ival> NUMBER
26:
27: %define parse.error verbose
28:
29: %%
30:
31: program
32:   : declarations statements
33:   | declarations
34: ;
35:
36: declarations
37:   : decl_statement declarations
38:   | decl_statement
39: ;
40:
41: array_index
42:   : L_BRACKET expression R_BRACKET
43:   | L_BRACKET expression R_BRACKET L_BRACKET expression R_BRACKET
44:   | L_BRACKET R_BRACKET
45: ;
46:
47: array
48:   : IDENT array_index
49: ;
50:
51: decl_part
52:   : DEFINE idents
53:   | ARRAY_DEF array
54: ;
55:
56: func_arg_part
57:   : DEFINE IDENT
58:   | ARRAY_DEF array
59: ;
60:
61: decl_statement
62:   : decl_part SEMIC
63:   | func_define
64: ;
65:
66: func_var_decl
67:   : func_arg_part
68:   | func_arg_part COMMA func_var_decl
69: ;
70:
71: func_define
72:   : FUNCDECL IDENT L_PARAN func_var_decl R_PARAN L_BRACE declarations statements R_BRACE
73:   | FUNCDECL IDENT L_PARAN R_PARAN L_BRACE declarations statements R_BRACE
74: ;
75:
76: func_call
77:   : FUNCCALL IDENT L_PARAN expressions R_PARAN SEMIC
78:   | FUNCCALL IDENT L_PARAN  R_PARAN SEMIC
79: ;
80:
81: statements
```

```
82:   : statement statements
83:   | statement
84: ;
85:
86: statement
87:   : assignment_stmt
88:   | loop_stmt
89:   | cond_stmt
90:   | func_call
91: ;
92:
93: assignment
94:   : IDENT ASSIGN expression
95:   | array ASSIGN expression
96:   | unary_factor
97: ;
98:
99: assignment_stmt
100:   : assignment SEMIC
101: ;
102:
103: expressions
104:   : expression COMMA expressions
105:   | expression
106: ;
107:
108: expression
109:   : expression add_op term
110:   | term
111: ;
112:
113: term
114:   : term mul_op factor
115:   | term bit_op factor
116:   | factor
117: ;
118:
119: unary_factor
120:   : IDENT unary_op
121:   | unary_op IDENT
122: ;
123:
124: factor
125:   : var
126:   | L_PARAN expression R_PARAN
127:   | unary_factor
128:   | NOT IDENT
129: ;
130:
131: add_op
132:   : ADD
133:   | SUB
134: ;
135:
136: mul_op
137:   : MUL
138:   | DIV
139:   | REM
140:
141: unary_op
142:   : INCREM
143:   | DECREM
144: ;
```

```
145:
146: bit_op
147:   : AND
148:   | OR
149:   | XOR
150:   | L_SHIFT
151:   | R_SHIFT
152: ;
153:
154: var
155:   : IDENT
156:   | NUMBER
157:   | IDENT array_index
158: ;
159:
160: loop_stmt
161:   : while_stmt
162:   | for_stmt
163: ;
164:
165: while_stmt
166:   : WHILE L_PARAN condition R_PARAN L_BRACE statements R_BRACE
167: ;
168:
169: for_stmt
170:   : FOR L_PARAN assignment SEMIC condition SEMIC assignment R_PARAN
171:   L_BRACE statements R_BRACE
172: ;
173:
174: cond_stmt
175:   : if_stmt
176:   | elif_stmt
177: ;
178:
179: if_stmt
180:   : IF L_PARAN condition R_PARAN L_BRACE statements
181: ;
182:
183: else_stmt
184:   : ELSE L_BRACE statements R_BRACE
185: ;
186:
187: elif_stmt
188:   : if_stmt else_stmt
189: ;
190:
191: condition
192:   : expression comp_op expression
193: ;
194:
195: comp_op
196:   : EQ
197:   | NE
198:   | LT
199:   | GT
200:   | LTE
201:   | GTE
202: ;
203:
204: idents
205:   : IDENT COMMA idents
206:   | IDENT
207: ;
```

```
208:
209: %%
```

2.1 定義した言語で受理されるプログラム

2.1.1 全体の構造

作成した言語では，プログラム全体はまず

- 変数宣言部
- 処理文集合

からなる仕様となっており，使用される変数はプログラムの冒頭で全て宣言され，その後に変数が使用される様々な処理文の集合が記述される．従って，文集合中に変数宣言は行われない．

2.1.2 変数の宣言

プログラムの変数宣言部では，4 バイトの整数値を格納できる通常の変数と，この整数値を複数格納できる配列 (1 次元，2 次元) が記述される．yacc の規則から，プログラムにおいて変数の宣言として以下の記述ができる．ここで，**<識別子名>**は変数名であり，先頭がアルファベットの英数字列である．

整数変数: `define` **<識別子名>**`;`
整数変数 (複数): `define` **<識別子名>**`, ...;`
配列 (1 次元): `array` **<識別子名>** `[`**自然数**`];`
配列 (2 次元): `array` **<識別子名>** `[`**自然数**`][`**自然数**`];`

3 コード生成の概要

3.1 メモリの使用方法

3.2 汎用レジスタの使用方法

3.3 算術式のコード生成の方法

4 工夫した点について

5 最終課題のプログラム及び実行結果

6 考察