

計算機科学実験及び演習 3

ソフトウェア実験中間レポート 1

工学部情報学科 3 回生 1029255242

勝見久央

作成日: 平成 27 年 6 月 13 日

1 課題 1

この課題では Small C の文法に従ってソーティングプログラムを作成した。なお、Small C では BNF の記述から C 言語と異なって、関数内部で変数宣言を全て行ってから変数への値の代入を行わないとエラーが発生することに留意する。

1.1 課題 1 の回答

コードは次のようになった。

リスト 1: Small C で記述したバブルソートプログラム

```
1 int comp_num(int a, int b);
2 int sort_array[8];
3 int main(){
4     int i;
5     int j;
6     int h;
7
8     sort_array[0] = 6;
9     sort_array[1] = 4;
10    sort_array[2] = 2;
11    sort_array[3] = 5;
12    sort_array[4] = 7;
13    sort_array[5] = 8;
14    sort_array[6] = 1;
15    sort_array[7] = 3;
16
17
18    for(j = 8; j > 1; j = (j-1)){
19        for(i = 0; i < (j-1); i = (i+1)){
20            if(comp_num(sort_array[i+1], sort_array[i])){
21                h = sort_array[i+1];
22                sort_array[i+1] = sort_array[i];
23                sort_array[i] = h;
24            }
25        }
26    }
27    i = 0;
28    while(8 > i){
29        if(i != 7){
30            print(sort_array[i]);
31        }
32        else(print(sort_array[7]));
33        i = (i+1);
34    }
35
36
```

```

37 }
38
39 int comp_num(int a, int b){
40     if(a > b) return 0;
41     if(a < b) return 1;
42     if(a == b) return 2;
43 }

```

2 課題5

この課題では Small C の構文解析器を作成した. この構文解析器はプログラムを受け取って抽象構文木を主な終端記号の位置情報を含んだ形で返す.

2.1 課題5の回答

コードは次のようになった.

リスト 2: 構文解析器のプログラム

```

1 #lang racket
2 (require parser-tools/lex
3     (prefix-in : parser-tools/lex-sre)
4     parser-tools/yacc)
5 (provide (all-defined-out))
6
7 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
8 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;データ型定義;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
9 (struct declaration_st (type-spec declarator-list)#:transparent)
10 ;例) (int (id1 id2 *id3))
11
12 (struct func_declarator_st (name para-list)#:transparent)
13 (struct func_declarator_null_st (name)#:transparent)
14 (struct func_declarator_ast_st (name para-list)#:transparent)
15 (struct func_declarator_ast_null_st (name)#:transparent)
16 ;例) nameは関数名、para-listは引数のリスト.
17
18 (struct func_proto_st (type-spec func-declarator-st)#:transparent)
19 (struct func_def_st (type-spec func-declarator-st compound-state-list)#:transparent)
20 ;例) (int funcname1 (id1 id2 *id3) compound_list)
21 ;compound_listの中身は構造体declaration_listとstatement_listからなる.
22
23 (struct declarator_st (var)#:transparent)
24 (struct declarator_ast_st (var)#:transparent)
25 ;declaratorの定義
26
27 ;declaration_listは構造体declarationがリストになったもの.以下で定義.
28 (struct declaration_st2 (type-spec para-list)#:transparent);;;;
29 ;例) (int (id1 id2 *id3 id4[5]))
30
31 (struct para_declaration_st (type-spec para)#:transparent)
32 ;例) (int id1)
33
34 (struct exp_st (exp)#:transparent)
35 ;式を格納する構造体.
36
37 (struct assign_exp_st (dest src pos)#:transparent)
38 ;例) 代入を表す構造体. x = 3なら(x 3)
39
40 (struct logic_exp_st (log-ope op1 op2 pos)#:transparent)
41 ;例) (or a b)もしくは(and a b)
42
43 (struct rel_exp_st (rel-ope op1 op2 pos)#:transparent)
44 ;例) rel_opeは'equal 'not 'less 'and_less 'more 'and_moreで (less a b)など
45
46 (struct alge_exp_st (alge-ope op1 op2 pos)#:transparent)
47 ;例) alge_opeは'add 'sub 'mul 'divで('add a b)など
48
49 (struct id_st (name pos)#:transparent)
50 (struct id_ast_st (name pos)#:transparent)

```

```

51 ; 終端記号 identifierを表す構造体.
52
53 (struct array_st (name num pos)#:transparent); 宣言時.posはnameの位置.
54 (struct array_var_st (name num pos)#:transparent); 式の中で用いる.posはnameの位置.
55 ; 終端記号となりうる配列を表す構造体.
56
57 (struct spec_st (type pos)#:transparent)
58 ; データ型を表す構造体
59
60 (struct unary_exp_st (mark op pos)#:transparent)
61 ; postfix_expを表す構造体.markは'minus','ast','amp
62
63 (struct constant_st (cons pos)#:transparent)
64 ; 定数を表す構造体.
65
66 (struct null_statement_st (null)#:transparent)
67 ; セミコロンのみからなるstatementを表す構造体
68
69 (struct exp_with_semi_st (exp)#:transparent)
70 ; expressionとセミコロンからなる式を表す構造体.
71
72 (struct exp_in_paren_st (exp)#:transparent)
73 ; ()で囲まれたexpression
74
75 (struct if_st (cond-exp state pos)#:transparent); else無し.posはifの位置.
76 (struct if_else_st (cond-exp state else-state if-pos else-pos)#:transparent); elseあり
77 ; if文を表す構造体.
78
79 (struct while_st (cond-exp statement pos)#:transparent); posはwhileの位置.
80 ; while文を表す構造体.
81
82 (struct for_0_st (cond-exp1 cond-exp2 cond-exp3 statement pos)#:transparent)
83 (struct for_1_st (cond-exp1 cond-exp2 statement pos)#:transparent)
84 (struct for_2_st (cond-exp1 cond-exp2 statement pos)#:transparent)
85 (struct for_3_st (cond-exp1 cond-exp2 statement pos)#:transparent)
86 (struct for_4_st (cond-exp1 statement pos)#:transparent)
87 (struct for_5_st (cond-exp1 statement pos)#:transparent)
88 (struct for_6_st (cond-exp1 statement pos)#:transparent)
89 (struct for_7_st (statement pos)#:transparent)
90 ; for文を表す構造体.0~7の数字によってnullの位置が構文木作成の時点でわかる.
91
92 (struct return_st (exp pos)#:transparent); posはreturnの位置.
93 (struct return_null_st (exp pos)#:transparent)
94 ; return文を表す構造体.
95
96 (struct compound_st (declaration-list statement-list)#:transparent)
97 (struct compound_dec_st (declaration-list)#:transparent)
98 (struct compound_sta_st (statement-list)#:transparent)
99 (struct compound_null_st (null)#:transparent)
100 ; compound_statementを表す構造体.
101
102 (struct func_st (name para)#:transparent)
103 (struct func_nopara_st (name)#:transparent)
104 ; 関数呼び出しを表す構造体
105 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
106 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
107 (define-empty-tokens tokens-without-value
108   (+ * & - / =
109     l_small_paren r_small_paren ;()
110     l_mid_paren r_mid_paren ;[]
111     l_big_paren r_big_paren ;{}
112     int void
113     if while for else
114     or and equal not;||、&&、==、!=
115     less and_less;<、<=
116     more and_more;>、>=
117     return
118     semicolon comma
119     EOF))
120
121 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
122 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
123 (define-tokens tokens-with-value
124   (NUM VAR))
125
126 (define-lex-trans uinteger
127   (syntax-rules () ((_ d) (:+ d))))
128
129 (define-lex-abbrevs
130   (digit (char-range "0" "9")))

```

```

131 (number (uinteger digit))
132 (identifier-char (:or (char-range "a" "z")
133                       (char-range "A" "Z")
134                       "-"))
135 (identifier (:: identifier-char
136              (:* (:or identifier-char
137                    digit)))))
138
139 (define sub-program-lexer
140   (lexer-src-pos
141    ("(" (token-l_small_paren))
142    (")" (token-r_small_paren))
143    ("[" (token-l_mid_paren))
144    ("]" (token-r_mid_paren))
145    ("{" (token-l_big_paren))
146    ("}" (token-r_big_paren))
147    ("int" (token-int))
148    ("void" (token-void))
149    ("if" (token-if))
150    ("while" (token-while))
151    ("for" (token-for))
152    ("else" (token-else))
153    ("||" (token-or))
154    ("&&" (token-and))
155    ("==" (token-equal))
156    ("!=" (token-not))
157   ("<" (token-less))
158   ("<=" (token-and_less))
159   (">" (token-more))
160   (">=" (token-and_more))
161    ("return" (token-return))
162    (";" (token-semicolon))
163    ("," (token-comma))
164    ("+" (token-+))
165    ("*" (token-*))
166    ("&" (token-&))
167    ("_" (token--))
168    ("/" (token-/))
169    ("=" (token-=))
170    (number (token-NUM (string->number lexeme)))
171    (identifier (token-VAR (string->symbol lexeme)))
172    (whitespace (return-without-pos (sub-program-lexer input-port)))
173    ((eof) (token-EOF))))
174 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
175 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
176
177 (define program-parser
178   (parser
179    (start program); 開始記号に当たる非終端記号
180    (end EOF); 入力 of 終端に達した時に字句解析器が返すトークン
181    (src-pos); 位置情報を含むオブジェクトを返す
182    (debug "siple-parser.tbl")
183    (error (lambda (tok-ok? tok-name tok-value start-pos end-pos)
184             (error "parse error:" tok-name tok-value)))
185    (tokens tokens-with-value tokens-without-value)
186
187    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
188    (grammar
189     (program ((external_declaration) $1)
190              ((program external_declaration) (cons $1 $2)))
191     (external_declaration ((declaration) $1)
192                           ((function_prototype) $1)
193                           ((function_definition) $1))
194     (declaration ((type_specifier declarator_list semicolon) (declaration_st $1 $2)))
195     (declarator_list ((declarator) $1)
196                     ((declarator_list comma declarator) (cons $1 $3)))
197     (declarator ((direct_declarator) (declarator_st $1))
198                ((* direct_declarator) (declarator_ast_st $2)))
199
200     (direct_declarator ((VAR) (id_st $1 $1-start-pos)); 構造体 id_st を作成.
201                       ((VAR l_small_paren NUM r_small_paren) (array_st $1 $3 $1-start-pos)))
202
203     (function_prototype ((type_specifier function_declarator semicolon)
204                          (func_proto_st $1 $2)); 構造体 func_proto_st を作成.
205     (function_declarator ((VAR l_small_paren parameter_type_list r_small_paren)
206                           (func_declarator_st $1 $3)); 構造体 func_declarator_st を作成.
207                          ((VAR l_small_paren r_small_paren) (func_declarator_null_st $1))
208                          ((* VAR l_small_paren parameter_type_list r_small_paren)
209                           (func_declarator_ast_st (id_ast_st $2 $2-start-pos) $4))
210                          ((* VAR l_small_paren r_small_paren)

```

```

211         (func_declarator_ast_null_st (id_ast_st $2 $2-start-pos))))
212
213 (function_definition ((type_specifier function_declarator compound_statement)
214                     (func_def_st $1 $2 $3))); 構造体 func_def_st を作成.
215 (parameter_type_list ((parameter_declaration) $1)
216                     ((parameter_type_list comma parameter_declaration)(cons $1 $3)))
217 (parameter_declaration ((type_specifier parameter_declarator)(para_declaration_st $1 $2)))
218 (parameter_declarator ((VAR) (id_st $1 $1-start-pos)); 構造体 id_st を作成.
219                      ((* VAR)(id_ast_st $2 $2-start-pos)); 構造体 id_ast_st を作成.
220 (type_specifier ((int) (spec_st 'int $1-start-pos)); 構造体 spec_st を作成.
221                ((void) (spec_st 'void $1-start-pos)); 構造体 spec_st を作成.
222 (statement ((semicolon) (null_statement_st 'null))
223            ((expression semicolon)(exp_with_semi_st $1)); 構造体 exp_st を作成.
224            ((compound_statement) $1)
225            ((if l_small_paren expression r_small_paren statement)
226             (if_st $3 $5 $1-start-pos)); 構造体 if_st を作成.
227            ((if l_small_paren expression r_small_paren statement else statement)
228             (if_else_st $3 $5 $7 $1-start-pos $6-start-pos)); 構造体 if_else_st を作成.
229            ((while l_small_paren expression r_small_paren statement)
230             (while_st $3 $5 $1-start-pos)); 構造体 while_st を作成.
231            ((for l_small_paren
232                 expression semicolon expression semicolon expression
233                 r_small_paren statement)
234             (for_0_st $3 $5 $7 $9 $1-start-pos)); 構造体 for_0_st を作成.
235            ((for l_small_paren
236                 semicolon expression semicolon expression
237                 r_small_paren statement)
238             (for_1_st $4 $6 $8 $1-start-pos)); 構造体 for_1_st を作成.
239            ((for l_small_paren
240                 expression semicolon semicolon expression
241                 r_small_paren statement)
242             (for_2_st $3 $6 $8 $1-start-pos)); 構造体 for_2_st を作成.
243            ((for l_small_paren
244                 expression semicolon expression semicolon
245                 r_small_paren statement)
246             (for_3_st $3 $5 $8 $1-start-pos)); 構造体 for_3_st を作成.
247            ((for l_small_paren expression
248                 semicolon semicolon
249                 r_small_paren statement)
250             (for_4_st $3 $7 $1-start-pos)); 構造体 for_4_st を作成.
251            ((for l_small_paren
252                 semicolon expression semicolon
253                 r_small_paren statement)
254             (for_5_st $4 $7 $1-start-pos)); 構造体 for_5_st を作成.
255            ((for l_small_paren
256                 semicolon semicolon expression
257                 r_small_paren statement)
258             (for_6_st $5 $7 $1-start-pos)); 構造体 for_6_st を作成.
259            ((for l_small_paren
260                 semicolon semicolon
261                 r_small_paren statement)
262             (for_7_st $6 $1-start-pos)); 構造体 for_7_st を作成.
263            ((return expression semicolon)(return_st $2 $1-start-pos)); 構造体 return_st を作成.
264            ((return semicolon)(return_null_st 'null $1-start-pos)); 構造体 return-null_st を作成.
265 (compound_statement ((l_big_paren declaration_list statement_list r_big_paren)
266                    (compound_st $2 $3)); 構造体 compound_st を作成.
267                    ((l_big_paren declaration_list r_big_paren)
268                     (compound_dec_st $2)); 構造体 compound_dec_st を作成.
269                    ((l_big_paren statement_list r_big_paren)
270                     (compound_sta_st $2)); 構造体 compound_sta_st を作成.
271                    ((l_big_paren r_big_paren)
272                     (compound_null_st 'null)); 構造体 compound_null_st を作成.
273 (declaration_list ((declaration) $1)
274                  ((declaration_list declaration)(cons $1 $2)))
275 (statement_list ((statement) $1)
276                ((statement_list statement)(cons $1 $2)))
277 (expression ((assign_expr) $1)
278            ((expression comma assign_expr)(cons $1 $3)))
279 (assign_expr ((logical_OR_expr) $1)
280             ((logical_OR_expr = assign_expr)
281              (assign_exp_st $1 $3 $2-start-pos)); 構造体 assign_exp_st を作成.
282 (logical_OR_expr ((logical_AND_expr) $1)
283                 ((logical_OR_expr or logical_AND_expr)
284                  (logic_exp_st 'or $1 $3 $2-start-pos)); 構造体 logic_exp_st を作成.
285 (logical_AND_expr ((equality_expr) $1)
286                  ((logical_AND_expr and equality_expr)
287                   (logic_exp_st 'and $1 $3 $2-start-pos)); 構造体 logic_exp_st を作成.
288 (equality_expr ((relational_expr) $1)
289               ((equality_expr equal relational_expr)
290                (rel_exp_st 'equal $1 $3 $2-start-pos)); 構造体 rel_exp_st を作成.

```

```

291      ((equality_expr not relational_expr)
292       (rel_exp_st 'not $1 $3 $2-start-pos))); 構造体rel_exp_stを作成.
293
294      (relational_expr ((add_expr) $1)
295                       ((relational_expr less add_expr)
296                        (rel_exp_st 'less $1 $3 $2-start-pos))); 構造体rel_exp_stを作成.
297                       ((relational_expr more add_expr)
298                        (rel_exp_st 'more $1 $3 $2-start-pos))); 構造体rel_exp_stを作成.
299                       ((relational_expr and_less add_expr)
300                        (rel_exp_st 'and_less $1 $3 $2-start-pos))); 構造体rel_exp_stを作成.
301                       ((relational_expr and_more add_expr)
302                        (rel_exp_st 'adn_more $1 $3 $2-start-pos))); 構造体rel_exp_stを作成.
303
304      (add_expr ((mult_expr) $1)
305               ((add_expr + mult_expr)
306                (alge_exp_st 'add $1 $3 $2-start-pos))); 構造体alge_exp_stを作成.
307               ((add_expr - mult_expr)
308                (alge_exp_st 'sub $1 $3 $2-start-pos))); 構造体alge_exp_stを作成.
309      (mult_expr ((unary_expr) $1)
310               ((mult_expr * unary_expr)
311                (alge_exp_st 'mul $1 $3 $2-start-pos))); 構造体alge_exp_stを作成.
312               ((mult_expr / unary_expr)
313                (alge_exp_st 'div $1 $3 $2-start-pos))); 構造体alge_exp_stを作成.
314      (unary_expr ((postfix_expr) $1)
315                ((~ unary_expr)
316                 (unary_exp_st 'minus $2 $1-start-pos))); 構造体unary-exp-stを作成.
317                ((& unary_expr)
318                 (unary_exp_st 'amp $2 $1-start-pos))); 構造体unary-exp-stを作成.
319                ((* unary_expr)
320                 (unary_exp_st 'ast $2 $1-start-pos))); 構造体unary-exp-stを作成.
321
322      (postfix_expr ((primary_expr) $1)
323                  ((postfix_expr l_mid_paren expression r_mid_paren)
324                   (array_var_st $1 $3 $1-start-pos))); 構造体array_var_stを作成.
325                  ((VAR l_small_paren argument_expression_list r_small_paren)
326                   (func_st $1 $3))); 構造体func_stを作成.
327                  ((VAR l_small_paren r_small_paren)
328                   (func_nopara_st $1))); 構造体func_nopara_stを作成.
329      (primary_expr ((VAR)(id_st $1 $1-start-pos))); 構造体id_stを作成.
330                  ((NUM) (constant_st $1 $1-start-pos))
331                  ((l_small_paren expression r_small_paren)
332                   (exp_in_paren_st $2))); 構造体exp_stを作成.
333      (argument_expression_list ((assign_expr) $1)
334                               ((argument_expression_list comma assign_expr)(cons $1 $3)))
335    )
336  )
337 )
338 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
339 (define (parse-string s)
340   (let ((p (open-input-string s)))
341     (program-parser (lambda () (sub-program-lexer p)))))
342
343 (define (parse-port p)
344   (program-parser (lambda () (sub-program-lexer p))))
345
346 ; 実行例
347 ;(define p (open-input-file "kadai01.sc"))
348 ;(port-count-lines! p)
349 ;(parse-port p)

```

2.2 設計方針

この構文解析器は構文解析の際にプログラム（文字列）-> トークン列-> 抽象構文木という2段階の流れをとっている。すなわち、構文解析器はプログラムをトークンに置き換える部分とトークンの列から抽象構文木を作成する部分からなる。

2.3 各部の説明

前節で述べた「プログラム (文字列) -> トークン列」の変換を行うのが関数 `sub-program-lexer` である。この関数内部では関数 `lexer-src-pos` が使われており、これは指定した文字列もしくは指定した文字列に対応するトークンを文字列中に発見したときに返すトークンを個別に指定できる関数である。また、`lexer-src-pos` の戻り値には位置情報が含まれる。なお、特定の文字列、正規表現とトークンの対応付けは `define-lex-abbrevs` で定義し、その他のトークンは `define-empty-tokens` で定義される。次に、「トークン列 -> 抽象構文木」の変換においては、Small C の BNF の非終端記号の終端記号への置換えを `parser` 関数の `grammar` で指定している。ここでは特定の非終端記号もしくは終端記号を解析したときに起こすアクションをそれぞれ個別に指定できる。このプログラムの場合は抽象構文木のノードとなる構造体を返す。また、それらの構造体はプログラムの冒頭部分で `#:transparent` をつけて定義している。なお、プログラムの実行は、関数 `parse-string` に文字列としてプログラムを直接渡すか、ファイルからポートオブジェクトを生成して関数 `parse-port` に渡すことで行う。

3 課題 6

この課題では、課題 5 のプログラムを使って実際に構文解析を行う。

3.1 課題 6 の実行結果

`test.sc` に次のような内容を記述する。

リスト 3: 解析対象の例

```
1 int func1(int a, int b);
2 int func2(int a, int b);
3 int array[2];
4
5 void main(){
6     array[0] = 100;
7     array[1] = 9999;
8     func2(999, func1(1, 9));
9 }
10
11 int func1(int a, int b){
12     b = (a + b);
13     if(b > 8){
14         b = (b * 100);
15     }
16     else{
17         b = 0;
18     }
19     return b;
20 }
21
22 int func2(int a, int b){
23     int d;
24     while(a > b){
25         b = ((b + 1) * 2);
26     }
27     array[0] = b;
28     array[1] = a;
29     d = (array[0]+array[1]);
30     return d;
31 }
```

これをポートオブジェクトに変換して構文解析を行った。

リスト 4: 構文解析器プログラムの実行例

```

1 >(define p (open-input-file "test.sc"))
2 >(port-count-lines! p)
3 >(parse-port p)
4
5 (cons
6   (cons
7     (cons
8       (cons
9         (func_proto_st
10          (spec_st 'int (position 1 1 0))
11          (func_declarator_st
12            'func1
13            (cons
14              (para_declaration_st (spec_st 'int (position 11 1 10)) (id_st 'a (position 15 1 14)))
15              (para_declaration_st (spec_st 'int (position 18 1 17)) (id_st 'b (position 22 1 21))))))
16          (func_proto_st
17            (spec_st 'int (position 26 2 0))
18            (func_declarator_st
19              'func2
20              (cons
21                (para_declaration_st (spec_st 'int (position 36 2 10)) (id_st 'a (position 40 2 14)))
22                (para_declaration_st (spec_st 'int (position 43 2 17)) (id_st 'b (position 47 2 21))))))
23          (declaration_st (spec_st 'int (position 51 3 0))
24            (declarator_st (array_st 'array 2 (position 55 3 4))))))
25   (func_def_st
26     (spec_st 'void (position 66 5 0))
27     (func_declarator_null_st 'main)
28     (compound_sta_st
29       (cons
30         (cons
31           (exp_with_semi_st
32             (assign_exp_st
33               (array_var_st (id_st 'array (position 83 6 4))
34                 (constant_st 0 (position 89 6 10))
35                 (position 83 6 4))
36               (constant_st 100 (position 94 6 15))
37               (position 92 6 13)))
38           (exp_with_semi_st
39             (assign_exp_st
40               (array_var_st (id_st 'array (position 103 7 4))
41                 (constant_st 1 (position 109 7 10))
42                 (position 103 7 4))
43               (constant_st 9999 (position 114 7 15))
44               (position 112 7 13)))
45           (exp_with_semi_st
46             (func_st
47               'func2
48               (cons
49                 (constant_st 999 (position 131 8 10))
50                 (func_st 'func1 (cons (constant_st 1 (position 142 8 21))
51                   (constant_st 9 (position 145 8 24))))))))))
52   (func_def_st
53     (spec_st 'int (position 153 11 0))
54     (func_declarator_st
55       'func1
56       (cons
57         (para_declaration_st (spec_st 'int (position 163 11 10))
58           (id_st 'a (position 167 11 14)))
59         (para_declaration_st (spec_st 'int (position 170 11 17))
60           (id_st 'b (position 174 11 21))))))
61   (compound_sta_st
62     (cons
63       (cons
64         (exp_with_semi_st
65           (assign_exp_st
66             (id_st 'b (position 182 12 4))
67             (exp_in_paren_st (alge_exp_st 'add (id_st 'a (position 187 12 9))
68               (id_st 'b (position 191 12 13))
69               (position 189 12 11)))
70             (position 184 12 6)))
71         (if_else_st
72           (rel_exp_st 'more (id_st 'b (position 202 13 7))
73             (constant_st 8 (position 206 13 11)) (position 204 13 9))
74           (compound_sta_st
75             (exp_with_semi_st
76               (assign_exp_st
77                 (id_st 'b (position 218 14 8))

```



```

79         (exp_in_paren_st (alge_exp_st 'mul (id_st 'b (position 223 14 13))
80                               (constant_st 100 (position 227 14 17))
81                               (position 225 14 15))))
82     (position 220 14 10))))
83 (compound_sta_st
84   (exp_with_semi_st (assign_exp_st (id_st 'b (position 257 17 8))
85                                     (constant_st 0 (position 261 17 12))
86                                     (position 259 17 10))))
87   (position 199 13 4)
88   (position 243 16 4)))
89 (return_st (id_st 'b (position 281 19 11)) (position 274 19 4))))))
90 (func_def_st
91   (spec_st 'int (position 287 22 0))
92   (func_declarator_st
93     'func2
94     (cons
95       (para_declaration_st (spec_st 'int (position 297 22 10))
96                             (id_st 'a (position 301 22 14)))
97       (para_declaration_st (spec_st 'int (position 304 22 17))
98                             (id_st 'b (position 308 22 21))))))
99 (compound_st
100   (declaration_st (spec_st 'int (position 316 23 4))
101                   (declarator_st (id_st 'd (position 320 23 8))))
102   (cons
103     (cons
104       (cons
105         (while_st
106           (rel_exp_st 'more (id_st 'a (position 333 24 10))
107                           (id_st 'b (position 337 24 14)) (position 335 24 12))
108           (compound_sta_st
109             (exp_with_semi_st
110               (assign_exp_st
111                 (id_st 'b (position 349 25 8))
112                 (exp_in_paren_st
113                   (alge_exp_st
114                     'mul
115                     (exp_in_paren_st (alge_exp_st 'add (id_st 'b (position 355 25 14))
116                                                         (constant_st 1 (position 359 25 18))
117                                                         (position 357 25 16)))
118                     (constant_st 2000000 (position 364 25 23))
119                     (position 362 25 21)))
120                 (position 351 25 10))))
121             (position 327 24 4))
122             (exp_with_semi_st
123               (assign_exp_st
124                 (array_var_st (id_st 'array (position 383 27 4))
125                               (constant_st 0 (position 389 27 10))
126                               (position 383 27 4))
127                 (id_st 'b (position 394 27 15))
128                 (position 392 27 13))))
129             (exp_with_semi_st
130               (assign_exp_st
131                 (array_var_st (id_st 'array (position 401 28 4))
132                               (constant_st 1 (position 407 28 10))
133                               (position 401 28 4))
134                 (id_st 'a (position 412 28 15))
135                 (position 410 28 13))))
136             (exp_with_semi_st
137               (assign_exp_st
138                 (id_st 'd (position 419 29 4))
139                 (exp_in_paren_st
140                   (alge_exp_st
141                     'add
142                     (array_var_st (id_st 'array (position 424 29 9))
143                                   (constant_st 0 (position 430 29 15))
144                                   (position 424 29 9))
145                     (array_var_st (id_st 'array (position 433 29 18))
146                                   (constant_st 1 (position 439 29 24))
147                                   (position 433 29 18))
148                     (position 432 29 17)))
149                 (position 421 29 6))))
150             (return_st (id_st 'd (position 455 30 11)) (position 448 30 4))))))
151

```

さらに、例えば test.sc の内容を

```

1 int func1(int a, int b);
2 int func2(int a, int b);
3 int array[2];

```

```

4
5 void main(){
6     array[0] = 100;
7     array[1] = 9999;
8     func2(999, func1(1, 9));
9     unvalid = + unvalid
10 }

```

と書き換えて同様に構文解析を行うと、構文解析器は停止し、何も出力しない。また処理系の Dr.Racket 側ではエラーが

```
parse error: + #f
```

と出力され、文字列を右側から構文解析し、

```
= +
```

という文字列を読み取った途端このプログラムが定義していた BNF から外れたことを発見し、エラーを出力している事がわかる。

4 課題7

この課題では、課題5の内容とは逆に、抽象構文木を受け取って、プログラムとして文字列を返すような関数を作成する。

4.1 課題7の回答

コードは次のようになった。

リスト 5: 抽象構文器をプログラムに変換する関数

```

1 #lang racket
2 (require parser-tools/lex
3     (prefix-in : parser-tools/lex-sre))
4 (require parser-tools/yacc)
5 (provide (all-defined-out))
6
7
8 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
9 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
10 ;programはexternal-declarationのリスト.
11 ;external-declarationはdeclaartion、func-Proto、func-defからなるリスト
12 (struct declaration_st (type-spec declarator-list)#:transparent)
13 ;例) (int (id1 id2 *id3))
14
15 (struct func_declarator_st (name para-list)#:transparent)
16 (struct func_declarator_null_st (name)#:transparent)
17 (struct func_declarator_ast_st (name para-list)#:transparent)
18 (struct func_declarator_ast_null_st (name)#:transparent)
19 ;例) nameは関数名、para-listは引数のリスト.
20
21 (struct func_proto_st (type-spec func-declarator-st)#:transparent)
22 (struct func_def_st (type-spec func-declarator-st compound-state-list)#:transparent)
23 ;例) (int funcname1 (id1 id2 *id3) compound_list)
24 ;compound_listの中身は構造体declaration_listとstatement_listからなる.
25
26 (struct declarator_st (var)#:transparent)
27 (struct declarator_ast_st (var)#:transparent)
28 ;declaratorの定義
29
30 ;declaration_listは構造体declarationがリストになったもの.以下で定義.
31 (struct declaration_st2 (type-spec para-list)#:transparent);;;;
32 ;例) (int (id1 id2 *id3 id4[5]))
33

```

```

34 (struct para_declaration_st (type-spec para)#:transparent)
35 ;例) (int id1)
36
37 (struct exp_st (exp)#:transparent)
38 ;式を格納する構造体.
39
40 (struct assign_exp_st (dest src pos)#:transparent)
41 ;例) 代入を表す構造体. x = 3なら(x 3)
42
43 (struct logic_exp_st (log-ope op1 op2 pos)#:transparent)
44 ;例) (or a b)もしくは(and a b)
45
46 (struct rel_exp_st (rel-ope op1 op2 pos)#:transparent)
47 ;例) rel_opeは'equal 'not 'less 'and_less 'more 'and_moreで (less a b)など
48
49 (struct alge_exp_st (alge-ope op1 op2 pos)#:transparent)
50 ;例) alge_opeは'add 'sub 'mul 'divで('add a b)など
51
52 (struct id_st (name pos)#:transparent)
53 (struct id_ast_st (name pos)#:transparent)
54 ;終端記号 identifierを表す構造体.
55
56 (struct array_st (name num pos)#:transparent);宣言時.posはnameの位置.
57 (struct array_var_st (name num pos)#:transparent);式の中で用いる.posはnameの位置.
58 ;終端記号となりうる配列を表す構造体.
59
60 (struct spec_st (type pos)#:transparent)
61 ;データ型を表す構造体
62
63 (struct unary_exp_st (mark op pos)#:transparent)
64 ;postfix_expを表す構造体.markは'minus、'ast、'amp
65
66 (struct constant_st (cons pos)#:transparent)
67 ;定数を表す構造体.
68
69 (struct null_statement_st (null)#:transparent)
70 ;セミコロンのみからなるstatementを表す構造体
71
72 (struct exp_with_semi_st (exp)#:transparent)
73 ;expressionとセミコロンのみからなる式を表す構造体.
74
75 (struct exp_in_paren_st (exp)#:transparent)
76 ;()で囲まれたexpression
77
78 (struct if_st (cond-exp state pos)#:transparent);else無し.posはifの位置.
79 (struct if_else_st (cond-exp state else-state if-pos else-pos)#:transparent);elseあり
80 ;if文を表す構造体.
81
82 (struct while_st (cond-exp statement pos)#:transparent);posはwhileの位置.
83 ;while文を表す構造体.
84
85 (struct for_0_st (cond-exp1 cond-exp2 cond-exp3 statement pos)#:transparent)
86 (struct for_1_st (cond-exp1 cond-exp2 statement pos)#:transparent)
87 (struct for_2_st (cond-exp1 cond-exp2 statement pos)#:transparent)
88 (struct for_3_st (cond-exp1 cond-exp2 statement pos)#:transparent)
89 (struct for_4_st (cond-exp1 statement pos)#:transparent)
90 (struct for_5_st (cond-exp1 statement pos)#:transparent)
91 (struct for_6_st (cond-exp1 statement pos)#:transparent)
92 (struct for_7_st (statement pos)#:transparent)
93 ;for文を表す構造体.0~7の数字によってnullの位置が構文木作成の時点でわかる.
94
95 (struct return_st (exp pos)#:transparent);posはreturnの位置.
96 (struct return_null_st (exp pos)#:transparent)
97 ;return文を表す構造体.
98
99 (struct compound_st (declaration-list statement-list)#:transparent)
100 (struct compound_dec_st (declaration-list)#:transparent)
101 (struct compound_sta_st (statement-list)#:transparent)
102 (struct compound_null_st (null)#:transparent)
103 ;compound_statementを表す構造体.
104
105 (struct func_st (name para)#:transparent)
106 (struct func_nopara_st (name)#:transparent)
107 ;関数呼び出しを表す構造体
108 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
109 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
110
111
112
113 ;syn-to-codeは構文木として構造体を文字列に変換する.

```

```

114
115 ;dec-list-to-codeは構造体(declaration_st-declarator-list x)を文字列に変換する際の補助関数.
116 ;para-list-to-codeは構造体(func_declarator_st-para-list x)を文字列に変換する際の補助関数.
117 ;arg-list-to-codeは構造体(func_st-para x)を文字列に変換する際の補助関数.
118 ;上記の関数はカンマの数などがわかっていないときの処理に使用する.
119
120 (define (syn-to-code x)
121   (cond ((struct? x) (cond ((declaration_st? x)
122                             (string-append (syn-to-code (declaration_st-type-spec x))
123                                               " "
124                                               (dec-list-to-code (declaration_st-declarator-list x))
125                                               ";"))
126
127         ((func_declarator_st? x)
128          (string-append (symbol->string (func_declarator_st-name x))
129                          "("
130                          (para-list-to-code (func_declarator_st-para-list x))
131                          ")"))
132         ;(struct func_declarator_st (name para-list)#:transparent)
133
134         ((func_declarator_null_st? x)
135          (string-append (symbol->string (func_declarator_null_st-name x))
136                          "()"))
137         ;(struct func_declarator_null_st (name)#:transparent)
138
139         ((func_declarator_ast_st? x)
140          (string-append "*"
141                          (syn-to-code (func_declarator_ast_st-name x))
142                          (syn-to-code (func_declarator_ast_st-para-list x))))
143         ;(struct func_declarator_ast_st (name para-list)#:transparent)
144
145         ((func_declarator_ast_null_st? x)
146          (string-append "*"
147                          (syn-to-code (func_declarator_ast_null_st-name x))))
148         ;(struct func_declarator_ast_null_st (name)#:transparent)
149
150         ((func_proto_st? x)
151          (string-append (syn-to-code (func_proto_st-type-spec x))
152                          " "
153                          (syn-to-code (func_proto_st-func-declarator-st x))
154                          ";"))
155         ;(struct func_proto_st (type-spec func-declarator-st)#:transparent)
156
157         ((func_def_st? x)
158          (string-append (syn-to-code (func_def_st-type-spec x))
159                          " "
160                          (syn-to-code (func_def_st-func-declarator-st x))
161                          (syn-to-code (func_def_st-compound-state-list x))))
162         ;(struct func_def_st
163          ;(type-spec func-declarator-st
164          ;compound-state-list)#:transparent)
165
166         ((declarator_st? x) (syn-to-code (declarator_st-var x)))
167         ;(struct declarator_st (var)#:transparent)
168
169         ((declarator_ast_st? x)
170          (string-append "*"
171                          (syn-to-code (declarator_ast_st-var x))))
172         ;(struct declarator_ast_st (var)#:transparent)
173
174         ((para_declaration_st? x)
175          (string-append (syn-to-code (para_declaration_st-type-spec x))
176                          " "
177                          (syn-to-code (para_declaration_st-para x))))
178         ;(struct para_declaration_st (type-spec para)#:transparent)
179
180         ((exp_st? x) (syn-to-code (exp_st-exp x)))
181         ;(struct exp_st (exp)#:transparent)
182
183         ((assign_exp_st? x) (string-append (syn-to-code (assign_exp_st-dest x))
184                                              " = "
185                                              (syn-to-code (assign_exp_st-src x))))
186         ;(struct assign_exp_st (dest src pos)#:transparent)
187
188         ((logic_exp_st? x)
189          (string-append (syn-to-code (logic_exp_st-op1 x))
190                          (cond ((eq? (logic_exp_st-log-ope x) 'or)
191                                " || ")
192                                ((eq? (logic_exp_st-log-ope x) 'and)
193                                " && ")
194                                (t "")))
195

```

```

194         (syn-to-code (logic_exp_st-op2 x))))
195 ;(struct logic_exp_st (log-ope op1 op2 pos)#:transparent)
196
197 ((rel_exp_st? x) (string-append
198   (syn-to-code (rel_exp_st-op1 x))
199   (cond ((eq? (rel_exp_st-rel-ope x) 'equal)
200         " == ")
201         ((eq? (rel_exp_st-rel-ope x) 'not)
202         " != ")
203         ((eq? (rel_exp_st-rel-ope x) 'less)
204         " < ")
205         ((eq? (rel_exp_st-rel-ope x) 'and_less)
206         " <= ")
207         ((eq? (rel_exp_st-rel-ope x) 'more)
208         " > ")
209         ((eq? (rel_exp_st-rel-ope x) 'and_more)
210         " >= ")))
211   (syn-to-code (rel_exp_st-op2 x))))
212 ;(struct rel_exp_st (rel-ope op1 op2 pos)#:transparent)
213
214 ((alge_exp_st? x) (string-append
215   (syn-to-code (alge_exp_st-op1 x))
216   (cond ((eq? (alge_exp_st-alge-ope x) 'add)
217         " + ")
218         ((eq? (alge_exp_st-alge-ope x) 'sub)
219         " - ")
220         ((eq? (alge_exp_st-alge-ope x) 'mul)
221         " * ")
222         ((eq? (alge_exp_st-alge-ope x) 'div)
223         " / ")))
224   (syn-to-code (alge_exp_st-op2 x))))
225 ;(struct alge_exp_st (alge-ope op1 op2 pos)#:transparent)
226
227 ((id_st? x) (symbol->string (id_st-name x)))
228 ;(struct id_st (name pos)#:transparent)
229 ((id_ast_st? x)
230  (string-append "*"
231    (symbol->string (id_ast_st-name x))))
232 ;(struct id_ast_st (name pos)#:transparent)
233
234 ((array_st? x) (string-append (symbol->string (array_st-name x))
235   "["
236   (number->string (array_st-num x)) "]""))
237 ;(struct array_st (name num pos)#:transparent);宣言時.posはnameの位置.
238 ((array_var_st? x)
239  (string-append (syn-to-code (array_var_st-name x))
240    "["
241    (syn-to-code (array_var_st-num x)) "]""))
242 ;(struct array_var_st (name num pos)#:transparent);式の中で用いる.
243 ;posはnameの位置.
244
245 ((spec_st? x) (symbol->string (spec_st-type x)))
246 ;(struct spec_st (type pos)#:transparent)
247
248 ((unary_exp_st? x)
249  (string-append (cond ((eq? (unary_exp_st-mark x) 'minus) "-")
250    ((eq? (unary_exp_st-mark x) 'ast) "*")
251    ((eq? (unary_exp_st-mark x) 'amp) "&"))
252    (unary_exp_st-op x)))
253 ;(struct unary_exp_st (mark op pos)#:transparent)
254
255 ((constant_st? x) (number->string (constant_st-cons x)))
256 ;(struct constant_st (cons pos)#:transparent)
257
258 ((null_statement_st? x) ";")
259 ;(struct null_statement_st (null))
260
261 ((exp_with_semi_st? x)
262  (string-append (syn-to-code (exp_with_semi_st-exp x)) ";"))
263 ;(struct exp_with_semi_st (exp)#:transparent)
264
265 ((exp_in_paren_st? x)
266  (string-append "(" (syn-to-code (exp_in_paren_st-exp x)) ")"))
267 ;(struct exp_in_paren_st (exp)#:transparent)
268
269 ((if_st? x) (string-append "if ("
270   (syn-to-code (if_st-cond-exp x))
271   ")"
272   (syn-to-code (if_st-state x))))
273 ;(struct if_st (cond-exp state pos)#:transparent);else無し.

```

```

274 ;posはifの位置.
275 ((if_else_st? x) (string-append "if("
276                               (syn-to-code (if_else_st-cond-exp x))
277                               ")")
278                               (syn-to-code (if_else_st-state x))
279                               "else"
280                               (syn-to-code (if_else_st-else-state x))))
281
282 ;(struct if_else_st
283 ;(cond-exp state else-state if-pos else-pos)#:transparent)
284 ;elseあり
285
286 ((while_st? x) (string-append "while("
287                               (syn-to-code (while_st-cond-exp x))
288                               ")")
289                               (syn-to-code (while_st-statement x))))
290 ;(struct while_st (cond-exp statement pos)#:transparent)
291 ;posはwhileの位置.
292
293 ((for_0_st? x) (string-append "for("
294                               (syn-to-code (for_0_st-cond-exp1 x))
295                               ","
296                               (syn-to-code (for_0_st-cond-exp2 x))
297                               ","
298                               (syn-to-code (for_0_st-cond-exp3 x))
299                               ")")
300                               (syn-to-code (for_0_st-statement x))))
301 ;(struct for_0_st
302 ;      (cond-exp1 cond-exp2 cond-exp3 statement pos)#:transparent)
303
304 ((for_1_st? x) (string-append "for("
305                               (syn-to-code (for_1_st-cond-exp1 x))
306                               ","
307                               (syn-to-code (for_1_st-cond-exp2 x))
308                               ")")
309                               (syn-to-code (for_1_st-statement x))))
310 ;(struct for_1_st (cond-exp1 cond-exp2 statement pos)#:transparent)
311
312 ((for_2_st? x) (string-append "for("
313                               (syn-to-code (for_2_st-cond-exp1 x))
314                               ";;"
315                               (syn-to-code (for_2_st-cond-exp2 x))
316                               ")")
317                               (syn-to-code (for_2_st-statement x))))
318 ;(struct for_2_st (cond-exp1 cond-exp2 statement pos)#:transparent)
319
320 ((for_3_st? x) (string-append "for("
321                               (syn-to-code (for_3_st-cond-exp1 x))
322                               ","
323                               (syn-to-code (for_3_st-cond-exp2 x))
324                               ");")
325                               (syn-to-code (for_3_st-statement x))))
326 ;(struct for_3_st (cond-exp1 cond-exp2 statement pos)#:transparent)
327
328 ((for_4_st? x) (string-append "for("
329                               (syn-to-code (for_4_st-cond-exp1 x))
330                               ";;")
331                               (syn-to-code (for_4_st-statement x))))
332 ;(struct for_4_st (cond-exp1 statement pos)#:transparent)
333
334 ((for_5_st? x) (string-append "for("
335                               (syn-to-code (for_5_st-cond-exp1 x))
336                               ");")
337                               (syn-to-code (for_5_st-statement x))))
338 ;(struct for_5_st (cond-exp1 statement pos)#:transparent)
339
340 ((for_6_st? x) (string-append "for("
341                               (syn-to-code (for_6_st-cond-exp1 x))
342                               ");")
343                               (syn-to-code (for_6_st-statement x))))
344 ;(struct for_6_st (cond-exp1 statement pos)#:transparent)
345
346 ((for_7_st? x) (string-append "for("
347                               (syn-to-code (for_7_st-cond-exp1 x))
348                               ");")
349                               (syn-to-code (for_7_st-statement x))))
350 ;(struct for_7_st (statement pos)#:transparent)
351
352 ((return_st? x) (string-append "return"
353                               (syn-to-code (return_st-exp x))
354                               ";"))
355 ;(struct return_st (exp pos)#:transparent);posはreturnの位置.

```

```

354      ((return_null_st? x)
355       (string-append "return"
356                     (syn-to-code (return_null_st-exp x))
357                     ";"))
358      ;(struct return_null_st (exp pos)#:transparent)
359
360      ((compound_st? x)
361       (string-append "{"
362                     (syn-to-code (compound_st-declaration-list x))
363                     (syn-to-code (compound_st-statement-list x))
364                     "}"))
365      ;(struct compound_st (declaration-list statement-list)#:transparent)
366
367      ((compound_dec_st? x)
368       (string-append "{"
369                     (syn-to-code (compound_dec_st-declaration-list x))
370                     "}"))
371      ;(struct compound_dec_st (declaration-list)#:transparent)
372
373      ((compound_sta_st? x)
374       (string-append
375        "{"
376        (syn-to-code (compound_sta_st-statement-list x)) ;;;;
377        "}"))
378      ;(struct compound_sta_st (statement-list)#:transparent)
379
380      ((compound_null_st? x) "{}")
381      ;(struct compound_null_st (null)#:transparent)
382
383      ((func_st? x) (string-append (symbol->string (func_st-name x))
384                                  "("
385                                  (arg-list-to-code (func_st-para x))
386                                  ")"))
387      ;(struct func_st (name para)#:transparent)
388
389      ((func_nopara_st? x)
390       (string-append (symbol->string (func_nopara_st-name x))
391                     "("
392                     ")"))
393      ;(struct func_nopara_st (name)#:transparent)
394
395
396
397      ;(#t "error: unknown syntax")
398      ))
399
400
401      (else (string-append (syn-to-code (car x))
402                          " "
403                          (syn-to-code (cdr x)))))
404
405  (define (dec-list-to-code x)
406    (cond ((struct? x) (syn-to-code x))
407          (else (string-append (dec-list-to-code (car x))
408                              " "
409                              (syn-to-code (cdr x)))))
410
411  (define (para-list-to-code x)
412    (cond ((struct? x) (syn-to-code x))
413          (else (string-append (para-list-to-code (car x))
414                              " "
415                              (syn-to-code (cdr x)))))
416
417  (define (arg-list-to-code x)
418    (cond ((struct? x) (syn-to-code x))
419          (else (string-append (arg-list-to-code (car x))
420                              " "
421                              (syn-to-code (cdr x)))))
422

```

実行結果は以下ようになった.

リスト 6: プログラムへの変換例

```

1 #lang racket
2 > (syn-to-code
3   (cons
4     (cons

```

```

5 (cons
6 (cons
7 (cons
8 (func_proto_st
9 (spec_st 'int (position 1 1 0))
10 (func_declarator_st
11 'func1
12 (cons
13 (para_declaration_st (spec_st 'int (position 11 1 10))
14 (id_st 'a (position 15 1 14)))
15 (para_declaration_st (spec_st 'int (position 18 1 17))
16 (id_st 'b (position 22 1 21))))))
17 (func_proto_st
18 (spec_st 'int (position 26 2 0))
19 (func_declarator_st
20 'func2
21 (cons
22 (para_declaration_st (spec_st 'int (position 36 2 10))
23 (id_st 'a (position 40 2 14)))
24 (para_declaration_st (spec_st 'int (position 43 2 17))
25 (id_st 'b (position 47 2 21))))))
26 (declaration_st (spec_st 'int (position 51 3 0))
27 (declarator_st (array_st 'array 2 (position 55 3 4))))))
28 (func_def_st
29 (spec_st 'void (position 66 5 0))
30 (func_declarator_null_st 'main)
31 (compound_sta_st
32 (cons
33 (cons
34 (exp_with_semi_st
35 (assign_exp_st
36 (array_var_st (id_st 'array (position 83 6 4))
37 (constant_st 0 (position 89 6 10)) (position 83 6 4))
38 (constant_st 100 (position 94 6 15))
39 (position 92 6 13)))
40 (exp_with_semi_st
41 (assign_exp_st
42 (array_var_st (id_st 'array (position 103 7 4))
43 (constant_st 1 (position 109 7 10)) (position 103 7 4))
44 (constant_st 9999 (position 114 7 15))
45 (position 112 7 13)))
46 (exp_with_semi_st
47 (func_st
48 'func2
49 (cons
50 (constant_st 999 (position 131 8 10))
51 (func_st 'func1 (cons (constant_st 1 (position 142 8 21))
52 (constant_st 9 (position 145 8 24))))))))))
53 (func_def_st
54 (spec_st 'int (position 153 11 0))
55 (func_declarator_st
56 'func1
57 (cons
58 (para_declaration_st (spec_st 'int (position 163 11 10))
59 (id_st 'a (position 167 11 14)))
60 (para_declaration_st (spec_st 'int (position 170 11 17))
61 (id_st 'b (position 174 11 21))))))
62 (compound_sta_st
63 (cons
64 (cons
65 (exp_with_semi_st
66 (assign_exp_st
67 (id_st 'b (position 182 12 4))
68 (exp_in_paren_st (alge_exp_st 'add (id_st 'a (position 187 12 9))
69 (id_st 'b (position 191 12 13))
70 (position 189 12 11)))
71 (position 184 12 6)))
72 (if_else_st
73 (rel_exp_st 'more (id_st 'b (position 202 13 7))
74 (constant_st 8 (position 206 13 11))
75 (position 204 13 9))
76 (compound_sta_st
77 (exp_with_semi_st
78 (assign_exp_st
79 (id_st 'b (position 218 14 8))
80 (exp_in_paren_st
81 (alge_exp_st 'mul (id_st 'b (position 223 14 13))
82 (constant_st 100 (position 227 14 17))
83 (position 225 14 15)))
84 (position 220 14 10))))))

```



```

85         (compound_sta_st
86         (exp_with_semi_st
87         (assign_exp_st (id_st 'b (position 257 17 8))
88         (constant_st 0 (position 261 17 12))
89         (position 259 17 10))))
90         (position 199 13 4)
91         (position 243 16 4))
92         (return_st (id_st 'b (position 281 19 11)) (position 274 19 4))))))
93 (func_def_st
94 (spec_st 'int (position 287 22 0))
95 (func_declarator_st
96 'func2
97 (cons
98 (para_declaration_st (spec_st 'int (position 297 22 10))
99 (id_st 'a (position 301 22 14)))
100 (para_declaration_st (spec_st 'int (position 304 22 17))
101 (id_st 'b (position 308 22 21))))))
102 (compound_st
103 (declaration_st (spec_st 'int (position 316 23 4))
104 (declarator_st (id_st 'd (position 320 23 8))))
105 (cons
106 (cons
107 (cons
108 (while_st
109 (rel_exp_st 'more (id_st 'a (position 333 24 10))
110 (id_st 'b (position 337 24 14))
111 (position 335 24 12))
112 (compound_sta_st
113 (exp_with_semi_st
114 (assign_exp_st
115 (id_st 'b (position 349 25 8))
116 (exp_in_paren_st
117 (alge_exp_st
118 'mul
119 (exp_in_paren_st
120 (alge_exp_st 'add (id_st 'b (position 355 25 14))
121 (constant_st 1 (position 359 25 18))
122 (position 357 25 16)))
123 (constant_st 200000 (position 364 25 23))
124 (position 362 25 21)))
125 (position 351 25 10))))
126 (position 327 24 4))
127 (exp_with_semi_st
128 (assign_exp_st
129 (array_var_st (id_st 'array (position 383 27 4))
130 (constant_st 0 (position 389 27 10))
131 (position 383 27 4))
132 (id_st 'b (position 394 27 15))
133 (position 392 27 13))))
134 (exp_with_semi_st
135 (assign_exp_st
136 (array_var_st (id_st 'array (position 401 28 4))
137 (constant_st 1 (position 407 28 10))
138 (position 401 28 4))
139 (id_st 'a (position 412 28 15))
140 (position 410 28 13))))
141 (exp_with_semi_st
142 (assign_exp_st
143 (id_st 'd (position 419 29 4))
144 (exp_in_paren_st
145 (alge_exp_st
146 'add
147 (array_var_st (id_st 'array (position 424 29 9))
148 (constant_st 0 (position 430 29 15))
149 (position 424 29 9))
150 (array_var_st (id_st 'array (position 433 29 18))
151 (constant_st 1 (position 439 29 24))
152 (position 433 29 18))
153 (position 432 29 17)))
154 (position 421 29 6))))
155 (return_st (id_st 'd (position 455 30 11)) (position 448 30 4))))))
156 )
157 "int func1(int a, int b); int func2(int a, int b); int array[2]; void main(){array[0] = 100; array[1] = 9999; func2(999,
158

```

4.2 設計方針

このプログラムは抽象構文木として受け取ったプログラムを文字列に変換する。具体的には抽象構文木の構造体の種類によって行う処理を 1 つずつ `cond` 文で定める。また、受け取る抽象構文木は構造体を属性値として持つ構造体であるから、この動作を再帰的に繰り返すプログラムを作成する。

4.3 各部の説明

このプログラムは、構造体を受け取って受け取った構造体の種類に応じて様々な処理を行う関数 `syn-to-code` を再帰的に定義することで作成されている。基本的にはある構造体を `syn-to-code` が受け取ったら、その中の属性である構造体を再び `syn-to-code` に渡し、その結果を `string-append` でつなぐという設計をとっている。ただし、属性の数が任意の個数の構造体もしくはシンボルになり得るような構造体、すなわち、関数定義の際の引数のリスト等进行处理する際は、他のものと同じような再帰的处理の方法ではカンマの数が不適当になるため、別の関数を用意する必要がある。それらは関数 `dec-list-to-code`、`para-list-to-code`、`arg-list-to-code` として定義されている。

5 感想

今回の実験では前年度のコンパイラの授業を受けてコンパイラの作成をするということで、その初期段階として構文解析器を作成しましたが、コンパイラの授業がなかなか理解できず苦戦した僕にとっては非常に苦労した課題でした。しかし実際に作ってみると去年なかなか理解できなかったものが意外とすんなりと理解でき、作りながら理解するという実験の恩恵をストレートに感じる事が出来ました。