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 2

工学部情報学科 3 回生 1029255242
勝見久央

作成日: 平成 27 年 7 月 7 日

1 課題 8

この課題では

- 単項演算 -x
- else 節のない if 文
- for 文 for(e1, e2, e3) s
- 配列参照式 e1[e2]

のシンタックスシュガーを含めたプログラムの抽象構文木への変換処理を実装した. シンタックスシュガーの実装においてはそれぞれを

- `0-x`
- `if(condition){compound-statement}else{}`
- `e1; while(e2){s e3;}`
- `*(e1+e2)`（結果として発生する式`&(*e)` の形は `e` に変換）

と変換した. さらに、組み込み関数 print のプロトタイプ宣言を抽象構文着生成の段階でプログラムに付加するようにした.

1.1 課題 8 の回答

コードは次のようになった.

リスト 1: シンタックスシュガーを含む抽象構文木への変換処理

```
#lang racket
(require parser-tools/lex
         (prefix-in : parser-tools/lex-sre)
         parser-tools/yacc
         (prefix-in stx: "mysyntax.rkt")
         (prefix-in k07u: "kadai07upgrade.rkt")
         )
(provide (all-defined-out))
```

```
9
10 (define-empty-tokens tokens-without-value
11   (+ * & - / =
12      l_small_paren r_small_paren ;()
13      l_mid_paren r_mid_paren ;[]
14      l_big_paren r_big_paren ;{}
15      int void
16      if while for else
17      or and equal not;||、&&、==、!=
18      less and_less;<、<=
19      more and_more;>、>=
20      return
21      semicolon comma
22      EOF))
23
24 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
25 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
26 (define-tokens tokens-with-value
27   (NUM VAR))
28
29 (define-lex-trans uinteger
30   (syntax-rules () ((_ d) (:+ d))))
31
32 (define-lex-abbrevs
33   (digit   (char-range "0" "9"))
34   (number  (uinteger digit))
35   (identifier-char (:or (char-range "a" "z")
36                         (char-range "A" "Z")
37                         "_"))
38   (identifier (:: identifier-char
39                   (:* (:or identifier-char
40                            digit)))))
41
42 (define sub-program-lexer
43   (lexer-src-pos
44    ("(" (token-l_small_paren))
45    (")" (token-r_small_paren))
46    ("[" (token-l_mid_paren))
47    ("]" (token-r_mid_paren))
48    ("{" (token-l_big_paren))
49    ("}" (token-r_big_paren))
50    ("int" (token-int))
51    ("void" (token-void))
52    ("if" (token-if))
53    ("while" (token-while))
54    ("for" (token-for))
55    ("else" (token-else))
56    ("||" (token-or))
57    ("&&" (token-and))
58    ("==" (token-equal))
59    ("!=" (token-not))
60    ("<" (token-less))
61    ("<=" (token-and_less))
62    (">" (token-more))
63    (">=" (token-and_more))
64    ("return" (token-return))
65    (";" (token-semicolon))
66    ("," (token-comma))
67    ("+" (token-+))
68    ("*" (token-*))
69    ("&" (token-&))
70    ("-" (token--))
71    ("/" (token-/))
72    ("=" (token-=))
73    (number (token-NUM (string->number lexeme)))
74    (identifier (token-VAR (string->symbol lexeme)))
75    (whitespace (return-without-pos (sub-program-lexer input-port)))
76    ((eof) (token-EOF))))
77 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
78 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
79
80 (define program-parser
81   (parser
82    (start program);開始記号に当たる非終端記号
83    (end EOF);入力の終端に達した時に字句解析器が返すトークン
84    (src-pos);位置情報を含むオブジェクトを返す
85    (debug "siple-parser.tbl")
86    (error (lambda (tok-ok? tok-name tok-value start-pos end-pos)
87             (error "parse error:" tok-name tok-value)))
88    (tokens tokens-with-value tokens-without-value)
```

```
89
90      ;;;;;;;;;;;;;;;;;;;;;;;;BNFの指定;;;;;;;;;;;;;;;;;;;;;;;;;;
91      (grammar
92       (program ((program_with_print)
93                 (cons
94                  (stx:func_proto_st
95                   (stx:spec_st 'void 'print-proto)
96                   (stx:func_declarator_st 'print
97                                           (stx:para_declaration_st
98                                            (stx:spec_st 'int 'print-proto)
99                                            (stx:id_st 'v 'print-proto))
100                                          'print-proto))
101                 $1)));プログラムの冒頭に組込み関数 printのプロトタイプ宣言をつける.
102
103      (program_with_print ((external_declaration) $1)
104                          ((program_with_print external_declaration) (cons $1 $2)))
105
106      #;(stx:func_proto_st
107         (stx:spec_st 'int 'print-proto)
108         (stx:func_declarator_st 'print
109                                 (stx:para_declaration_st (stx:spec_st 'int 'print-proto)
110                                                          (stx:id_st 'i 'print-proto))))
111
112      (external_declaration ((declaration) $1)
113                            ((function_prototype) $1)
114                            ((function_definition) $1))
115      (declaration ((type_specifier declarator_list semicolon)
116                    (stx:declaration_st  $1 $2)))
117      (declarator_list ((declarator) $1)
118                       ((declarator_list comma declarator)(cons $1 $3)))
119      (declarator ((direct_declarator)
120                   (stx:declarator_st $1))
121                  ((* direct_declarator)
122                   (stx:declarator_ast_st $2)
123                   ))
124
125      (direct_declarator ((VAR) (stx:id_st $1 $1-start-pos))
126                         ((VAR l_mid_paren NUM r_mid_paren)
127                          (stx:array_st $1 $3 $1-start-pos)))
128
129      (function_prototype ((type_specifier function_declarator semicolon)
130                           (stx:func_proto_st $1 $2)))
131      (function_declarator ((VAR l_small_paren parameter_type_list r_small_paren)
132                            (stx:func_declarator_st $1 $3 $1-start-pos))
133                           ((VAR l_small_paren r_small_paren)
134                            (stx:func_declarator_null_st $1 $1-start-pos))
135                           ((* VAR l_small_paren parameter_type_list r_small_paren)
136                            (stx:func_declarator_ast_st $2 $4 $2-start-pos))
137                           ((* VAR l_small_paren r_small_paren)
138                            (stx:func_declarator_ast_null_st $2 $2-start-pos)))
139
140      (function_definition ((type_specifier function_declarator compound_statement)
141                            (stx:func_def_st $1 $2 $3)))
142      (parameter_type_list ((parameter_declaration) $1)
143                           ((parameter_type_list comma parameter_declaration)(cons $1 $3)))
144      (parameter_declaration ((type_specifier parameter_declarator)
145                              (stx:para_declaration_st $1 $2)))
146      (parameter_declarator ((VAR) (stx:id_st $1 $1-start-pos))
147                            ((* VAR)(stx:id_ast_st $2 $2-start-pos)))
148      (type_specifier ((int) (stx:spec_st 'int $1-start-pos))
149                      ((void) (stx:spec_st 'void $1-start-pos)))
150      (statement ((semicolon) (stx:null_statement_st 'null))
151                 ((expression semicolon)$1)
152                 ((compound_statement) $1)
153                 ((if l_small_paren expression r_small_paren statement)
154                  ;シンタックスシュガー
155                  (stx:if_else_st $3
156                                  $5
157                                  (stx:null_statement_st 'null)
158                                  $1-start-pos 'syntax-sygar))
159                 ((if l_small_paren expression r_small_paren statement else statement)
160                  (stx:if_else_st $3 $5 $7 $1-start-pos $6-start-pos))
161                 ((while l_small_paren expression r_small_paren statement)
162                  (stx:while_st $3 $5 $1-start-pos))
163                 ((for l_small_paren expression
164                    semicolon expression
165                    semicolon expression
166                    r_small_paren statement)
167                  ;シンタックスシュガー
168                  (stx:compound_sta_st
```

```
                  (cons
                   $3
                   (stx:while_st $5
                                 (stx:compound_sta_st (cons $9 $7))
                                 'syntax-sugar))))
                ((for l_small_paren
                   semicolon expression
                   semicolon expression
                   r_small_paren statement)
                 ;シンタックスシュガー
                 (stx:while_st $4
                               (stx:compound_sta_st (cons $8 $6))
                               'syntax-sugar))
                ((for l_small_paren expression
                   semicolon
                   semicolon expression
                   r_small_paren statement)
                 ;シンタックスシュガー
                 (stx:compound_sta_st
                  (cons
                   $3
                   (stx:while_st (stx:null_statement_st 'null)
                                 (stx:compound_sta_st (cons $8 $6))
                                 'syntax-sugar))))
                ((for l_small_paren expression
                   semicolon expression
                   semicolon
                   r_small_paren statement)
                 ;シンタックスシュガー
                 (stx:compound_sta_st
                  (cons
                   $3
                   (stx:while_st $5
                                 $8
                                 'syntax-sugar))))
                ((for l_small_paren expression
                   semicolon
                   semicolon
                   r_small_paren statement)
                 ;シンタックスシュガー
                 (stx:compound_sta_st
                  (cons
                   $3
                   (stx:while_st (stx:null_statement_st 'null)
                                 $7
                                 'syntax-sugar))))
                ((for l_small_paren
                   semicolon expression
                   semicolon
                   r_small_paren statement)
                 ;シンタックスシュガー
                 (stx:while_st $4
                               $7
                               'syntax-sugat))
                ((for l_small_paren
                   semicolon
                   semicolon expression
                   r_small_paren statement)
                 ;シンタックスシュガー
                 (stx:while_st (stx:null_statement_st 'null)
                               (stx:compound_sta_st (cons $7 $5))))
                ((for l_small_paren
                   semicolon
                   semicolon
                   r_small_paren statement)
                 ;シンタックスシュガー
                 (stx:while_st (stx:null_statement_st 'null)
                               $6
                               'syntax-sugar))

                ((return expression semicolon)(stx:return_st $2 $1-start-pos))
                ((return semicolon)(stx:return_st 'noreturn $1-start-pos)))
      (compound_statement ((l_big_paren declaration_list statement_list r_big_paren)
                           (stx:compound_st $2 $3))
                          ((l_big_paren declaration_list r_big_paren)
                           (stx:compound_dec_st $2))
                          ((l_big_paren statement_list r_big_paren)
                           (stx:compound_sta_st $2))
                          ((l_big_paren r_big_paren)
                           (stx:compound_null_st 'null)))
```

```
(declaration_list ((declaration) $1)
                  ((declaration_list declaration)(cons $1 $2)))
(statement_list ((statement) $1)
                ((statement_list statement)(cons $1 $2)))
(expression ((assign_expr) $1)
            ((expression comma assign_expr)(cons $1 $3)))
(assign_expr ((logical_OR_expr) $1)
             ((logical_OR_expr = assign_expr)
              (stx:assign_exp_st $1 $3 $2-start-pos)))
(logical_OR_expr ((logical_AND_expr) $1)
                 ((logical_OR_expr or logical_AND_expr)
                  (stx:logic_exp_st 'or $1 $3 $2-start-pos)))
(logical_AND_expr ((equality_expr) $1)
                  ((logical_AND_expr and equality_expr)
                   (stx:logic_exp_st 'and $1 $3 $2-start-pos)))
(equality_expr ((relational_expr) $1)
               ((equality_expr equal relational_expr)
                (stx:rel_exp_st 'equal $1 $3 $2-start-pos))
               ((equality_expr not relational_expr)
                (stx:rel_exp_st 'not $1 $3 $2-start-pos)))

(relational_expr ((add_expr) $1)
                 ((relational_expr less add_expr)
                  (stx:rel_exp_st 'less $1 $3 $2-start-pos))
                 ((relational_expr more add_expr)
                  (stx:rel_exp_st 'more $1 $3 $2-start-pos))
                 ((relational_expr and_less add_expr)
                  (stx:rel_exp_st 'and_less $1 $3 $2-start-pos))
                 ((relational_expr and_more add_expr)
                  (stx:rel_exp_st 'adn_more $1 $3 $2-start-pos)))

(add_expr ((mult_expr) $1)
          ((add_expr + mult_expr)
           (stx:alge_exp_st 'add $1 $3 $2-start-pos))
          ((add_expr - mult_expr)
           (stx:alge_exp_st 'sub $1 $3 $2-start-pos)))
(mult_expr ((unary_expr) $1)
           ((mult_expr * unary_expr)
            (stx:alge_exp_st 'mul $1 $3 $2-start-pos))
           ((mult_expr / unary_expr)
            (stx:alge_exp_st 'div $1 $3 $2-start-pos)))
(unary_expr ((postfix_expr) $1)
            ((- unary_expr)
             ;シンタックスシュガー
             (stx:alge_exp_st 'sub
                              (stx:constant_st 0 'syntax-sugar)
                              $2
                              $1-start-pos))
            ((& unary_expr)
             (if (stx:exp_in_paren_st? $2)
                 (if (stx:unary_exp_st? (stx:exp_in_paren_st-exp $2))
                     (if (equal? 'ast
                                 (stx:unary_exp_st-mark (stx:exp_in_paren_st-exp $2)))
                         ;間接参照式のシンタックスシュガー
                         (stx:unary_exp_st-op (stx:exp_in_paren_st-exp $2))
                         (stx:unary_exp_st 'amp $2 $1-start-pos))
                     (stx:unary_exp_st 'amp $2 $1-start-pos))
                 (if (stx:unary_exp_st? $2)
                     (if (equal? 'ast (stx:unary_exp_st-mark $2))
                         (stx:unary_exp_st-op $2)
                         (stx:unary_exp_st 'amp $2 $1-start-pos))
                     (stx:unary_exp_st 'amp $2 $1-start-pos))))
            ((* unary_expr)
             (stx:unary_exp_st 'ast $2 $1-start-pos)))

(postfix_expr ((primary_expr) $1)
              ((postfix_expr l_mid_paren expression r_mid_paren)
               ;(stx:array_var_st $1 $3  $1-start-pos)
               ;配列参照式のシンタックスシュガー
               (stx:unary_exp_st 'ast
                                 (stx:exp_in_paren_st
                                  (stx:alge_exp_st 'add $1 $3 'syntax-sugar))
                                 'syntax-sugar))
              ((VAR l_small_paren argument_expression_list r_small_paren)
               (stx:func_st $1 $3))
              ((VAR l_small_paren r_small_paren)
               (stx:func_st $1 'nopara)))
(primary_expr ((VAR)(stx:id_st $1 $1-start-pos))
              ((NUM) (stx:constant_st $1 $1-start-pos))
              ((l_small_paren expression r_small_paren)
```

```
                    (stx:exp_in_paren_st $2)))
    (argument_expression_list ((assign_expr) $1)
                              ((argument_expression_list comma assign_expr)(cons $1 $3))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(define (parse-string s)
  (let ((p (open-input-string s)))
    (program-parser (lambda () (sub-program-lexer p)))))

(define (parse-port p)
  (program-parser (lambda () (sub-program-lexer p))))
#;(begin
  (define p9999 (open-input-file "kadai01.c"))
  (port-count-lines! p9999)
  (parse-port p9999))
```

1.2 設計方針

この課題は前節の課題 5 で作成した抽象構文木作成の手続きに上記のシンタックスシュガーの変換処理と print 関数のプロトタイプ宣言を付け加えるだけである. 具体的にはシンタックスシュガーに当てはまる文字列を受け取った際に、返す構文木をシンタックスシュガーに対応する構文木に変更し、プログラムを読み込むと同時にその先頭に組込み関数 print のプロトタイプ宣言を表す構文木を付加するようにしただけである.

1.3 各部の説明

単項演算のシンタックスシュガーは次のようになる.

リスト 2: 単項演算のシンタックスシュガー

```
((- unary_expr)
 ;シンタックスシュガー
 (stx:alge_exp_st 'sub
    (stx:constant_st 0 'syntax-sugar) $2 $1-start-pos))
```

else 節のない if 文のシンタックスシュガーは次のようなる.

リスト 3: if 文のシンタックスシュガー

```
((if l_small_paren expression r_small_paren statement)
 ;シンタックスシュガー
 (stx:if_else_st $3
                 $5
                 (stx:null_statement_st 'null)
                 $1-start-pos
                 'syntax-sygar))
\end{lstlising}
for文のシンタックスシュガーは次のようになる.
なお、for文を表す構造体は条件節の式の有無に応じて8パターン
存在するが、ここではその一つを紹介する.
\begin{lstlisting}[caption=for文のシンタックスシュガー]
((for l_small_paren expression
                 semicolon expression
                 semicolon expression
                 r_small_paren statement)
                ;シンタックスシュガー
                (stx:compound_sta_st
                 (cons
                 $3
                 (stx:while_st $5
                               (stx:compound_sta_st (cons $9 $7))
                               'syntax-sugar))))
```

配列参照式のシンタックスシュガーは次のようになる.

リスト 4: 配列参照式のシンタックスシュガー

```
((postfix_expr l_mid_paren expression r_mid_paren)
                   ;(stx:array_var_st $1 $3  $1-start-pos)
                   ;配列参照式のシンタックスシュガー
                   (stx:unary_exp_st 'ast
                                     (stx:alge_exp_st 'add $1 $3 'syntax-sugar)
                                     'syntax-sugar))
```

```
(unary_expr ((postfix_expr) $1)
                ((- unary_expr)
                 ;シンタックスシュガー
                 (stx:alge_exp_st 'sub
                                  (stx:constant_st 0 'syntax-sugar)
                                  $2
                                  $1-start-pos))
                ((& unary_expr)
                 (if (stx:exp_in_paren_st? $2)
                     (if (stx:unary_exp_st? (stx:exp_in_paren_st-exp $2))
                         (if (equal? 'ast
                                     (stx:unary_exp_st-mark (stx:exp_in_paren_st-exp $2)))
                             ;間接参照式のシンタックスシュガー
                             (stx:unary_exp_st-op (stx:exp_in_paren_st-exp $2))
                             (stx:unary_exp_st 'amp $2 $1-start-pos))
                         (stx:unary_exp_st 'amp $2 $1-start-pos))
                     (if (stx:unary_exp_st? $2)
                         (if (equal? 'ast (stx:unary_exp_st-mark $2))
                             (stx:unary_exp_st-op $2)
                             (stx:unary_exp_st 'amp $2 $1-start-pos))
                         (stx:unary_exp_st 'amp $2 $1-start-pos))))
                ((* unary_expr)
                 (stx:unary_exp_st 'ast $2 $1-start-pos)))
```

また、pirnt 関数のプロトタイプ宣言は次のように実装した.

リスト 5: print 関数のプロトタイプ

```
(program
    ((program_with_print)
        (cons
          (stx:func_proto_st
             (stx:spec_st 'void 'print-proto)
             (stx:func_declarator_st
                  'print
             (stx:para_declaration_st
                 (stx:spec_st 'int 'print-proto)
              (stx:id_st 'v 'print-proto))
             'print-proto))
          $1)))
```

2 課題 10

この課題では、課題 8 で生成された抽象構文木に対して

- 意味解析
- 式の形の検査
- 式の型の検査

行う手続きを作成する. それぞれのソースコードは次のようになった.

リスト 6: 意味解析について (README.md)

```
# compiler
##意味解析部
###analy-declaration_st(チェック部分開発途中)
;(stx:declaration_st...) と
;__分析に使う環境env__と
;__current-lev__
;を受け取って
;(stx;declaration_st type-spec (list (obj...) (obj...)...))
;を返す.
;同時にlistの形で環境に追加.
;同時に環境のチェックも行う.

###analy-func_proto_st
;(stx:func_proto_st...)
;を受け取って
;(stx:func_proto_st (stx:spec_st...)
;                   (stx:func_declarator/_ast/_st 関数名
;                                (list obj...)))
;を返し
;同時に関数プロトタイプのobject(obj name 0 'proto type)を
;環境に登録.
;パラメータのobject(obj name 1 'parm type)の(list obj...)を
;パラメータ専用の環境をまず初期化してから登録

###analy-func_def_st
;stx:func_def_stを
;引数に取り
;(stx:func_def_st stx:spec_st
;                 (func_declarator_st '関数宣言のオブジェクト'
;                                     'パラメータのオブジェクトのlist')
;                 compound-statement)
;(compound-statement部分については関数analy-compound_stに任せる.)
;を返す.
;同時にパラメータのオブジェクトをパラメータ専用の環境に追加、チェック
;同時に関数宣言のオブジェクトを環境に追加、チェック

###analy-compound_st
;stx:compound_stか
;stx:compound_dec_stか
;stx:compound_sta_stか
;stx:compound_null_stと
;lev
;を受け取って
;(stx:compound_st declaration-list statement-list)
;を返す.
;ただし
;declaration-listが無いときは'nodecl
;statement-listが無いときは'nostat
;を入れる
;同時に
;意味解析開始時にcurrent-levをひとつ上げる
;終了時に1つ下げる

###analy-compdecl
;__analy-declaration_stの派生__
;(list* (stx:declaration_st...)...)と
;lev(解析中のcompound-statementのブロックレベル)
;を引数に取り
;(list* obj)
;を返す関数
;_analy-declarationとは違って外部の大域の環境を更新しない._
;compound-statementの意味解析結果の環境としては(list* obj)を直接使用することとする.

###analy-compstate
;levと
;envと
;compound_stなどに入るstatementを
;引数に取り
;それぞれのobjを
;返す関数
;同時にenvをもとにstatement内の定義をチェックする.
```

リスト 7: 意味解析を行う処理 semantic-analy.rkt

```
#lang racket
(require "myenv.rkt")
```

```
(require (prefix-in stx: "mysyntax.rkt"))
(require (prefix-in k08: "kadai08.rkt"))
(require "mymap.rkt")
(provide (all-defined-out))

;(define current-lev 0)
;(define comp-lev 0)
(define env '())
(define para-env '())
(define comp-env '())


(define (analy-declaration_st st lev)
  ;;;;
  ;内部定義
  ;(stx:declarator_st...)と
  ;levと
  ;'intもしくは'void
  ;を引数にとり
  ;obj
  ;を返す関数.
  (define (make-obj-from-decl decl type lev)
    (let* ((id (cond ((stx:declarator_st? decl)
                      (stx:declarator_st-var decl))
                     ((stx:declarator_ast_st? decl)
                      (stx:declarator_ast_st-var decl))))
           (name (cond ((stx:id_st? id) (stx:id_st-name id))
                       ((stx:array_st? id) (stx:array_st-name id))))
           (flag (cond ((stx:declarator_st? decl) 'nomal)
                       ((stx:declarator_ast_st? decl) 'pointer)))
           (kind 'var)
           (type (cond ((stx:array_st? id) (type_array type (stx:array_st-num id)))
                       (else (cond ((equal? flag 'nomal) type)
                                   ((equal? flag 'pointer) (type_pointer 'pointer type))))))
           (pos (cond ((stx:id_st? id)
                       (stx:id_st-pos id))
                      ((stx:id_ast_st? id)
                       (stx:id_ast_st-pos id))
                      ((stx:array_st? id)
                       (stx:array_st-pos id)))))
      (obj name lev kind type pos)))
  ;;;;
  (let* (;typeに入っているのは (stx:spec_st 'intか'void ポジション)
         (type (stx:declaration_st-type-spec st))
         (declarator-list (stx:declaration_st-declarator-list st))
         ;objのlistを作成する.
         (obj-list (map*
                    (lambda (x) (make-obj-from-decl x (stx:spec_st-type type) lev))
                    declarator-list)))
    ;意味解析上のエラーがないか確認する.
    (map (lambda (x) (check-decl x env)) obj-list)
    (map (lambda (x) (check-decl x para-env)) obj-list)
    ;なければ環境に追加.
    (set! env (add-list obj-list env))
    ;構造体を返す.
    (stx:declaration_st type obj-list)))


(define (analy-func_proto_st st)
  ;;;;;内部定義
  ;make-obj-from-paralistは
  ;(list* (stx:para_declaration_st...)...)
  ;を引数に取り、
  ;(list obj...)
  ;を返す.
  ;'noparaの時は関数の外で処理する.
  (define (make-obj-from-paralist para-list)
    (map* (lambda (para-decl)
            (let* (;typeは'intとか.この時点では確定しない.最終的にはflagと合わせて決定.
                   (type (stx:spec_st-type (stx:para_declaration_st-type-spec para-decl)))
                   ;idはstx:id_stかstx:id_ast_st
                   (id (stx:para_declaration_st-para para-decl))
                   ;flagはポインタ型なら'pointer、そうでなければ'normal
                   (flag (cond ((stx:id_st? id) 'normal)
                               ((stx:id_ast_st? id) 'pointer)))
                   (name (cond ((equal? flag 'normal) (stx:id_st-name id))
                               ((equal? flag 'pointer)(stx:id_ast_st-name id))))
                   (pos (cond ((stx:id_st? id) (stx:id_st-pos id))
                              ((stx:id_ast_st? id) (stx:id_ast_st-pos id))))
                   (lev 1)
```

```
                    (kind 'parm)
                    (type (cond ((equal? flag 'normal) type)
                                ((equal? flag 'pointer)(type_pointer 'pointer type)))))
                (obj name lev kind type pos)))
           para-list))
  ;;;;;内部定義ここまで
  (let* (;このspecがintで返り値が*intの場合あり.
         ;返り値は最終的にはここの型とflagで決定される.
         ;specはstx:spec_st
         (spec (stx:func_proto_st-type-spec st))
         ;declはstx:func_declarator/_null/_ast/_stの4つの場合がある.
         (decl (stx:func_proto_st-func-declarator-st st))
         ;返り値がnormalかpointerか、パラメータの有無がnoremalかnoneか
         ;para_flagは(struct para_flag (out-type para))で定義される構造体.
         (flag (cond ((stx:func_declarator_st? decl)(para_flag 'normal 'normal))
                     ((stx:func_declarator_null_st? decl)(para_flag 'normal 'none))
                     ((stx:func_declarator_ast_st? decl) (para_flag 'pointer 'normal))
                     ((stx:func_declarator_ast_null_st? decl) (para_flag 'pointer 'none))))
         (proto-name (cond ((stx:func_declarator_st? decl)
                            (stx:func_declarator_st-name decl))
                           ((stx:func_declarator_null_st? decl)
                            (stx:func_declarator_null_st-name decl))
                           ((stx:func_declarator_ast_st? decl)
                            (stx:func_declarator_ast_st-name decl))
                           ((stx:func_declarator_ast_null_st? decl)
                            (stx:func_declarator_ast_null_st-name decl))))
         ;プロトタイプの位置情報
         (proto-pos (cond ((stx:func_declarator_st? decl)
                           (stx:func_declarator_st-pos decl))
                          ((stx:func_declarator_null_st? decl)
                           (stx:func_declarator_null_st-pos decl))
                          ((stx:func_declarator_ast_st? decl)
                           (stx:func_declarator_ast_st-pos decl))
                          ((stx:func_declarator_ast_null_st? decl)
                           (stx:func_declarator_ast_null_st-pos decl))))
         ;para-listは(list* (stx:para_declaration_st...)...)
         ;もしくはパラメータが無いときは'noparaが入っている.
         (para-list (cond ((stx:func_declarator_st? decl)
                           (stx:func_declarator_st-para-list decl))
                          ((stx:func_declarator_null_st? decl)
                           'nopara)
                          ((stx:func_declarator_ast_st? decl)
                           (stx:func_declarator_ast_st-para-list decl))
                          ((stx:func_declarator_ast_null_st? decl)
                           'nopara)))
         ;para-obj-listは(list obj...)もしくは'nopara
         (para-obj-list (cond ((equal? para-list 'nopara) 'nopara)
                              (else (make-obj-from-paralist para-list))))
         (proto-type (cond ((equal? 'normal (para_flag-out-type flag))
                            (type_fun 'fun
                                      (stx:spec_st-type spec)
                                      (cond ((equal? 'nopara para-obj-list)
                                             'nopara)
                                            (else (map (lambda (x) (obj-type x))
                                                       para-obj-list)))))
                           ;(struct type-pointer (pointer type) #:transparent)
                           ((equal? 'pointer (para_flag-out-type flag))
                            (type_fun 'fun
                                      (type_pointer 'pointer (stx:spec_st-type spec))
                                      (cond ((equal? 'nopara para-obj-list)
                                             'nopara)
                                            (else (map (lambda (x) (obj-type x))
                                                       para-obj-list)))))))
         (proto-obj (obj proto-name 0 'proto proto-type proto-pos)))
    ;プロトタイプのオブジェクトのチェック
    (check-proto proto-obj env)
    ;プロトタイプのオブジェクトを環境に追加.
    (set! env (extend-env proto-obj env))
    ;パラメータ内の二重宣言をチェック.
    (check-proto-para para-obj-list)
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
    ;返す構造体
    ;ここで返したいものはlet*で取り出しておく必要がある.
    (stx:func_proto_st spec (stx:func_declarator_st proto-obj para-obj-list proto-pos))))

(define (analy-func_def_st st)
  ;;;;;内部定義
  ;make-obj-from-paralistは
  ;(list* (stx:para_declaration_st...)...)
  ;を引数に取り、
```

```
;(list obj...)
;を返す.
;'noparaの時は関数の外で処理する.
(define (make-obj-from-paralist para-list)
  (map* (lambda (para-decl)
          (let* (;typeは'intとか.この時点では確定しない.最終的にはflagと合わせて決定.
                 (type (stx:spec_st-type (stx:para_declaration_st-type-spec para-decl)))
                 ;idはstx:id_stかstx:id_ast_st
                 (id (stx:para_declaration_st-para para-decl))
                 ;flagはポインタ型なら'pointer、そうでなければ'normal
                 (flag (cond ((stx:id_st? id) 'normal)
                             ((stx:id_ast_st? id) 'pointer)))
                 (name (cond ((equal? flag 'normal) (stx:id_st-name id))
                             ((equal? flag 'pointer)(stx:id_ast_st-name id))))
                 (pos (cond ((stx:id_st? id) (stx:id_st-pos id))
                            ((stx:id_ast_st? id) (stx:id_ast_st-pos id))))
                 (lev 1)
                 (kind 'parm)
                 (type (cond ((equal? flag 'normal) type)
                             ((equal? flag 'pointer)(type_pointer 'pointer type)))))
            (obj name lev kind type pos)))
        para-list))
;;;;;内部定義ここまで
(let* (;このspecがintで返り値が*intの場合あり.
       ;返り値は最終的にはここの型とflagで決定される.
       ;specはstx:spec_st
       (spec (stx:func_def_st-type-spec st))
       ;declはstx:func_declarator/_null/_ast/_stの4つの場合がある.
       (decl (stx:func_def_st-func-declarator-st st))
       (compo (stx:func_def_st-compound-state-list st))
       ;返り値がnormalかpointerか、パラメータの有無がnoremalかnoneか
       ;fundef_flagは(struct fundef_flag (out-type para))で定義される構造体.
       (flag (cond ((stx:func_declarator_st? decl)(fundef_flag 'normal 'normal))
                   ((stx:func_declarator_null_st? decl)(fundef_flag 'normal 'none))
                   ((stx:func_declarator_ast_st? decl) (fundef_flag 'pointer 'normal))
                   ((stx:func_declarator_ast_null_st? decl) (fundef_flag 'pointer 'none))))
       (fundef-name (cond ((stx:func_declarator_st? decl)
                           (stx:func_declarator_st-name decl))
                          ((stx:func_declarator_null_st? decl)
                           (stx:func_declarator_null_st-name decl))
                          ((stx:func_declarator_ast_st? decl)
                           (stx:func_declarator_ast_st-name decl))
                          ((stx:func_declarator_ast_null_st? decl)
                           (stx:func_declarator_ast_null_st-name decl))))
       ;関数定義の位置情報
       (fundef-pos (cond ((stx:func_declarator_st? decl)
                          (stx:func_declarator_st-pos decl))
                         ((stx:func_declarator_null_st? decl)
                          (stx:func_declarator_null_st-pos decl))
                         ((stx:func_declarator_ast_st? decl)
                          (stx:func_declarator_ast_st-pos decl))
                         ((stx:func_declarator_ast_null_st? decl)
                          (stx:func_declarator_ast_null_st-pos decl))))
       ;para-listは(list* (stx:para_declaration_st...)...)
       ;もしくはパラメータが無いときは'noparaが入っている.
       (para-list (cond ((stx:func_declarator_st? decl)
                         (stx:func_declarator_st-para-list decl))
                        ((stx:func_declarator_null_st? decl)
                         'nopara)
                        ((stx:func_declarator_ast_st? decl)
                         (stx:func_declarator_ast_st-para-list decl))
                        ((stx:func_declarator_ast_null_st? decl)
                         'nopara)))
       ;para-obj-listは(list obj...)もしくは'nopara
       (para-obj-list (cond ((equal? para-list 'nopara) 'nopara)
                            (else (make-obj-from-paralist para-list))))
       (fundef-type (cond ((equal? 'normal (fundef_flag-out-type flag))
                           (type_fun 'fun
                                     (stx:spec_st-type spec)
                                     (cond ((equal? 'nopara para-obj-list) 'nopara)
                                           (else(map (lambda (x) (obj-type x))
                                                     para-obj-list)))))
                          ((equal? 'pointer (fundef_flag-out-type flag))
                           (type_fun 'fun
                                     (type_pointer 'pointer (stx:spec_st-type spec))
                                     (cond ((equal? 'nopara para-obj-list) 'nopara)
                                           (else (map (lambda (x) (obj-type x))
                                                      para-obj-list)))))
                          (else (error "IN VALID FUNCTION"))))
       (fundef-obj (obj fundef-name 0 'fun fundef-type fundef-pos)))
```

```
243     ;関数定義のオブジェクトのチェック
244     (check-func fundef-obj env)
245     ;関数定義のオブジェクトを環境に追加.
246     (set! env (extend-env fundef-obj env))
247     ;パラメータ内の二重宣言をチェック
248     (check-def-para para-obj-list)
249     ;パラメータの環境を登録
250     (set! para-env para-obj-list)
251     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
252     ;返す構造体
253     ;ここで返したいものはlet*で取り出しておく必要がある.
254     (stx:func_def_st spec
255                      (stx:func_declarator_st fundef-obj para-obj-list fundef-pos)
256                      (analy-compound_st compo 1 env fundef-obj)
257                      )))
258
259 (define (analy-compound_st st lev outer-env func-tag)
260   (let* ((flag (cond ((stx:compound_st? st) (comp_flag 'normal 'normal 1))
261                      ((stx:compound_dec_st? st) (comp_flag 'normal 'nostat 2))
262                      ((stx:compound_sta_st? st) (comp_flag 'nodecl 'normal 3))
263                      ((or (stx:null_statement_st? st)
264                           (stx:compound_null_st? st)) (comp_flag 'nodecl 'nostat 4))
265                      (else (error "UNEXPECTED STRUCTURE! ERROR IN ANALY-COMPOUND." st))))
266          ;decl-listには(list* stx:declaration_st...)
267          (decl-list (cond ((equal? 1 (comp_flag-n flag))
268                            (stx:compound_st-declaration-list st))
269                           ((equal? 2 (comp_flag-n flag))
270                            (stx:compound_dec_st-declaration-list st))
271                           ((or (equal? 3 (comp_flag-n flag))
272                                (equal? 4 (comp_flag-n flag))) 'nodecl)))
273          ;statement-listにはstatementのlist*が入る.
274          ;処理する際はmap*で
275          (stat-list (cond ((equal? 1 (comp_flag-n flag))
276                            (stx:compound_st-statement-list st))
277                           ((equal? 3 (comp_flag-n flag))
278                            (stx:compound_sta_st-statement-list st))
279                           ((or (equal? 2 (comp_flag-n flag))(equal? 4 (comp_flag-n flag))) 'nostat)))
280          ;意味解析開始時にlevを一つ繰り上げる
281          (this-lev (+ lev 1))
282          ;decl-listに入っているのは(list stx:declaration_st...)か'nodecl
283          (decl-list (cond ((equal? 'normal (comp_flag-decl flag))
284                            ;このときオブジェクトはcomp-envに追加する必要がある.
285                            ;(map* analy-compdecl decl-list
286                            (map* (lambda (x) (analy-compdecl x this-lev)) decl-list))
287                           ((equal? 'nodecl (comp_flag-decl flag))
288                            'nodecl)))
289          ;decl-listからこのcompoun-statement内で新しく生成される環境を格納する.
290          ;comp-env内はこのcompound-statement内で新しく定義されたオブジェクトのlist
291          (comp-env
292           (cond
293             ((equal? 'nodecl decl-list) 'nodecl)
294             (else ;listの各要素がlistであることを保証させるためのmake-list-list
295              (flatten (map (lambda (x) (stx:declaration_st-declarator-list x)) decl-list)))))
296          ;comp-envのチェック
297          ;代入には意味は無い.let*の代入文の段階でチェックを実行しておく必要があるためこのようにした.
298          ;comp-env内のみで二重定義などが無いかどうかをチェックする.
299          (comp-env-check (check-comp-env comp-env))
300          ;大域環境と照らし合わせる
301          (comp-env-check (cond ((equal? 'nodecl comp-env) #t)
302                                (else (map (lambda (x) (check-decl x outer-env)) comp-env))))
303          (comp-env-check (cond ((equal? 'nodecl comp-env) #t)
304                                (else (map (lambda (x) (check-decl x para-env)) comp-env))))
305          ;outer-envは大域環境を含む.
306          (new-comp-env (cond ((equal? 'nodecl comp-env) outer-env)
307                              (else (append comp-env outer-env))))
308          (stat-list (cond ((equal? 'normal (comp_flag-stat flag))
309                            (map*
310                             (lambda (x) (analy-compstate x this-lev new-comp-env func-tag))
311                             stat-list))
312                           ((equal? 'nostat (comp_flag-stat flag))
313                            'nostat))))
314     (stx:compound_st decl-list stat-list)))
315
316 (define (analy-compdecl st lev)
317   ;;;;
318   ;内部定義
319   ;(stx:declarator_st...)と
320   ;levと
321   ;'intもしくは'void
322   ;を引数にとり
```

```
  ;obj
  ;を返す関数.
  (define (make-obj-from-decl decl type lev)
    (let* ((id (cond ((stx:declarator_st? decl)
                      (stx:declarator_st-var decl))
                     ((stx:declarator_ast_st? decl)
                      (stx:declarator_ast_st-var decl))))
           (name (cond ((stx:id_st? id) (stx:id_st-name id))
                       ((stx:id_ast_st? id) (stx:id_ast_st-name id))
                       ((stx:array_st? id) (stx:array_st-name id))))
           (pos (cond ((stx:id_st? id) (stx:id_st-pos id))
                      ((stx:id_ast_st? id) (stx:id_ast_st-pos id))
                      ((stx:array_st? id) (stx:array_st-pos id))))
           (flag (cond ((stx:declarator_st? decl) 'nomal)
                       ((stx:declarator_ast_st? decl) 'pointer)))
           (kind 'var)
           (type (cond ((stx:array_st? id) (type_array type (stx:array_st-num id)))
                       (else (cond ((equal? flag 'nomal) type)
                                   ((equal? flag 'pointer) (type_pointer 'pointer type)))))))
      (obj name lev kind type pos)))
  ;;;;内部定義ここまで
  (let* (;typeに入っているのは (stx:spec_st 'intか'void ポジション)
         (type (stx:declaration_st-type-spec st))
         (declarator-list (stx:declaration_st-declarator-list st))
         ;objのlistを作成する.
         (obj-list (map*
                    (lambda (x) (make-obj-from-decl x (stx:spec_st-type type) lev))
                    declarator-list)))
    ;意味解析上のエラーがないかは外側でチェックするのでここでは実装しなくて良い.
    (stx:declaration_st type obj-list)))

;envはnodeclの場合あり.
(define (analy-compstate st lev env func-tag)
  (cond ((stx:null_statement_st? st)
         st)
        ((stx:assign_exp_st? st)
         (stx:assign_exp_st (analy-compstate (stx:assign_exp_st-dest st) lev env func-tag)
                            (analy-compstate(stx:assign_exp_st-src st) lev env func-tag)
                            (stx:assign_exp_st-pos st)))
        ((stx:logic_exp_st? st)
         (stx:logic_exp_st (stx:logic_exp_st-log-ope st)
                           (analy-compstate (stx:logic_exp_st-op1 st) lev env func-tag)
                           (analy-compstate (stx:logic_exp_st-op2 st) lev env func-tag)
                           (stx:logic_exp_st-pos st)))
        ((stx:rel_exp_st? st)
         (stx:rel_exp_st (stx:rel_exp_st-rel-ope st)
                         (analy-compstate (stx:rel_exp_st-op1 st) lev env func-tag)
                         (analy-compstate (stx:rel_exp_st-op2 st) lev env func-tag)
                         (stx:rel_exp_st-pos st)))
        ((stx:alge_exp_st? st)
         (stx:alge_exp_st (stx:alge_exp_st-alge-ope st)
                          (analy-compstate (stx:alge_exp_st-op1 st) lev env func-tag)
                          (analy-compstate (stx:alge_exp_st-op2 st) lev env func-tag)
                          (stx:alge_exp_st-pos st)))
        ((stx:unary_exp_st? st)
         (let* ((normal-out (stx:unary_exp_st (stx:unary_exp_st-mark st)
                                              (analy-compstate
                                               (stx:unary_exp_st-op st) lev env func-tag)
                                              (stx:unary_exp_st-pos st)))
                (pos (stx:unary_exp_st-pos st)))
           ;*で表された配列参照式を判別する.
           (cond ((equal? 'ast (stx:unary_exp_st-mark st))
                  (cond ((stx:exp_in_paren_st? (stx:unary_exp_st-op st))
                         (cond ((stx:alge_exp_st? (stx:exp_in_paren_st-exp
                                                   (stx:unary_exp_st-op st)))
                                (cond ((equal? 'add
                                               (stx:alge_exp_st-alge-ope
                                                (stx:exp_in_paren_st-exp
                                                 (stx:unary_exp_st-op st))))
                                       ;ここまでで*(x + y)の形までが保証される.
                                       (analy-compstate
                                        (stx:array_var_st
                                         (stx:alge_exp_st-op1
                                          (stx:exp_in_paren_st-exp
                                           (stx:unary_exp_st-op st)))
                                         (stx:alge_exp_st-op2
                                          (stx:exp_in_paren_st-exp
                                           (stx:unary_exp_st-op st)))
                                         pos) lev env func-tag))
                                      (else normal-out)))
```

```
                                (else normal-out)))
                        (else normal-out)))
                (else normal-out))))
        ((stx:constant_st? st) st)
        ((stx:exp_with_semi_st? st)
         (stx:exp_with_semi_st
          (analy-compstate (stx:exp_with_semi_st-exp st) lev env func-tag)))
        ((stx:exp_in_paren_st? st)
         (stx:exp_in_paren_st
          (analy-compstate (stx:exp_in_paren_st-exp st) lev env func-tag)))
        ((stx:if_else_st? st)
         (stx:if_else_st
          (analy-compstate (stx:if_else_st-cond-exp st) lev env func-tag)
          (analy-compstate (stx:if_else_st-state st) lev env func-tag)
          (analy-compstate (stx:if_else_st-else-state st) lev env func-tag)
          (stx:if_else_st-if-pos st)
          (stx:if_else_st-else-pos st)))
        ((stx:while_st? st)
         (stx:while_st
          (analy-compstate (stx:while_st-cond-exp st) lev env func-tag)
          (analy-compound_st (stx:while_st-statement st) lev env func-tag)
          (stx:while_st-pos st)))
        ((stx:return_st? st)
         (stx:sem_return_st
          (analy-compstate (stx:return_st-exp st) lev env func-tag)
          (stx:return_st-pos st)
          func-tag))
        ((or (stx:compound_st? st)
             (stx:compound_dec_st? st)
             (stx:compound_sta_st? st)
             (stx:compound_null_st? st))
         (analy-compound_st st lev env func-tag))
        ;チェック時は環境に'nodeclが入ることがあることに注意.
        ((stx:func_st? st)
         (stx:func_st (check-func-ref st lev env)
                      (cond ((equal? 'nopara (stx:func_st-para st)) 'nopara)
                            (else (map (lambda (x)
                                         (analy-compstate x lev env func-tag))
                                       (flatten (stx:func_st-para st)))))))
        ((or (stx:id_st? st)
             (stx:id_ast_st? st)
             (stx:array_var_st? st))
         ;stはid_stもしくはid_ast_st
         ;levはcompound-statementのレベル
         ;envは大域環境(objのlist)
         (check-var-ref st lev
                        (append (cond ((equal? 'nopara para-env) '())
                                      (else para-env)) env)))
        ;デバグ用.(本来はどんなプログラムを書いてもこの分岐には入らないはず.)
        (else (error "AN UNEXPECTED STRUCTURE! CONDITION ERROR IN ANALY-COMPSTATE FOR" st))
        ))

(define (sem-analyze-tree t)
  (define (sem-analyze-struct st)
    (cond ((stx:declaration_st? st) (analy-declaration_st st 0))
          ((stx:func_proto_st? st) (analy-func_proto_st st))
          ((stx:func_def_st? st) (analy-func_def_st st))
          (else
           (error
            "SYNTAX ERROR! AN EXPECTED ARGUMENT >
            DECLARATION/FUNCTION PROTOTYPE/FUNCTION DEFINITION."))))
  (let* ((out-tree (map* sem-analyze-struct t)))
    (set! env '())
    out-tree))



;テスト
#;(begin
    (define p101 (open-input-file "kadai01.c"))
    (port-count-lines! p101)
    (sem-analyze-tree (k08:parse-port p101)))
```

リスト 8: 式の形の検査を行う処理 analy-form.rkt

```
#lang racket
(require (prefix-in k08: "kadai08.rkt"))
```

```
(require (prefix-in sem: "semantic-analy.rkt"))
(require (prefix-in stx: "mysyntax.rkt"))
(require parser-tools/lex
         (prefix-in : parser-tools/lex-sre)
         parser-tools/yacc)
(require "myenv.rkt")
(provide (all-defined-out))

(define (analy-form t)
  (begin (map form-check t)
         (display "OK! THIS PROGRAM IS IN A CORRECT FORM.")))

(define (form-check st)
  (cond
    ((not (list? st)) (list st))
    ((stx:declaration_st? st) #t)
    ((stx:func_declarator_st? st) #t)
    ((stx:func_proto_st? st) #t)
    ((stx:func_def_st? st)
     (cond ((eq? 'nostat (stx:compound_st-statement-list
                          (stx:func_def_st-compound-state-list st)))
            #t)
           (else
            (map form-check (stx:compound_st-statement-list
                             (stx:func_def_st-compound-state-list st))))))
    ((stx:assign_exp_st? st)
     (begin (if (or (stx:unary_exp_st? (stx:assign_exp_st-dest st))
                    (cond ((obj? (stx:assign_exp_st-dest st))
                           (and (eq? 'var (obj-kind (stx:assign_exp_st-dest st)))
                                (eq? type_array? (obj-type (stx:assign_exp_st-dest st)))))
                          (else #f)))
                #t
                (error (format "ERROR! AN INVALID ASSIGN EXPRESSION FORM AT ~a"
                               (stx:assign_exp_st-pos st))))
            (map form-check (stx:assign_exp_st-src st))))
    ((stx:logic_exp_st? st)
     (begin (map form-check (stx:logic_exp_st-op1 st))
            (map form-check (stx:logic_exp_st-op2 st))))
    ((stx:rel_exp_st? st)
     (begin (map form-check (stx:logic_exp_st-op1 st))
            (map form-check (stx:logic_exp_st-op2 st))))
    ((stx:alge_exp_st? st)
     (begin (map form-check (stx:logic_exp_st-op1 st))
            (map form-check (stx:logic_exp_st-op2 st))))
    ((stx:spec_st? st) #t)
    ((stx:unary_exp_st? st)
     (cond ((obj? (stx:unary_exp_st-op st))
            (cond ((eq? 'var (obj-kind (stx:unary_exp_st-op st)))
                   #t)
                  (else (error (format "ERROR! AN INVALID & FORM AT ~a"
                                       (stx:unary_exp_st st))))))
           (else (error (format "ERROR! AN INVALID & FORM AT ~a"
                                (stx:unary_exp_st st))))))
    ((stx:constant_st? st) #t)
    ((stx:null_statement_st? st) #t)
    ((stx:exp_in_paren_st? st)
     (map form-check (stx:exp_in_paren_st-exp st)))
    ((stx:if_else_st? st)
     (begin (map form-check (stx:if_else_st-cond-exp st))
            (map form-check (stx:if_else_st-else-state st))))
    ((stx:while_st? st)
     (begin (map form-check (stx:while_st-cond-exp st))
            (map form-check (stx:while_st-statement st))))
    ((stx:sem_return_st? st)
     (map form-check (stx:return_st-exp st)))
    ((stx:compound_st? st)
     (map form-check (stx:compound_st-statement-list st)))
    ((stx:func_st? st) #t)
    ((obj? st) #t)
    ((position? st) #t)
    ;デバグ用エラー発生（実際にはどのようなプログラムを読み込んでもこの分岐には入らないはず.）
    (else (error "ERROR! UNEXPECTED STRUCTURES IN AN ARGUMENT OF ANALY-FORM." st))))


;テスト
;(define p (open-input-file "test01.c"))
;(port-count-lines! p)
;;(sem:sem-analyze-tree (k08:parse-port p))
;(analy-form (sem:sem-analyze-tree (k08:parse-port p)))
```

リスト 9: 式の型の検査を行う処理 analy-type.rkt

```
#lang racket
(require (prefix-in k08: "kadai08.rkt"))
(require (prefix-in sem: "semantic-analy.rkt"))
(require (prefix-in stx: "mysyntax.rkt"))
(require parser-tools/lex
         (prefix-in : parser-tools/lex-sre)
         parser-tools/yacc)
(require "myenv.rkt")
(provide (all-defined-out))
;myenv.rkt内で定義
;(struct obj (name lev kind type)#:transparent)

(define (analy-type t)
  (begin (map check-type t)
         (display "OK! THIS PROGRAM IS WELL TYPED.")))

;引数は構造体
;戻り値は'well-typed
(define (check-type st)
  (cond
    ((list? st) (map check-type st))
    ((stx:declaration_st? st)
     (let* ((decl-obj-list (stx:declaration_st-declarator-list st)))
       (map (lambda (x)
              (cond
                ;宣言した変数が配列の時
                ;void型、voidポインタ型はエラー
                ((type_array? (obj-type x))
                 (cond ((or (equal? 'void
                                    (type_array-type (obj-type x)))
                            (equal? (type_pointer 'pointer 'void)
                                    (type_array-type (obj-type x))))
                        (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
                                       (obj-name x)(obj-pos x))))))
                ;宣言した変数が配列でない時
                ;void型、voidポインタ型はエラー
                (else (cond ((equal? (type_pointer 'pointer 'void) (obj-type x))
                             (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
                                            (obj-name x)(obj-pos x))))
                            ((equal? 'void (obj-type x))
                             (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
                                            (obj-name x)(obj-pos x))))
                            (else 'well-typed)))))
            decl-obj-list)
       'well-typed))
    ((stx:func_proto_st? st)
     (let* ((func-declarator (stx:func_proto_st-func-declarator-st st))
            (func-para-list (stx:func_declarator_st-para-list func-declarator))
            (func-obj (stx:func_declarator_st-name func-declarator))
            (func-type (obj-type func-obj))
            (func-out-type (type_fun-out func-type)))
       ;関数プロトタイプの
       ;戻り値がvoidポインタはエラー
       ;パラメータがvoid型、voidポインタはエラー
       (cond ;戻り値がvoidのポインタ型であるとき
         ((equal? (type_pointer 'pointer 'void )
                  func-out-type)
          (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
                         (obj-name func-obj)(obj-pos func-obj))))
         ;パラメターががvoid型、voidポインタ型であるとき
         ;chcek-type-paraはこれらの型の以上がパラメータの中に無いかどうかを判定する.
         ((equal? 'well-typed (check-type-para func-para-list))
          'well-typed)
         (else 'well-typed))))
    ((stx:func_def_st? st)
     ;戻り値がvoidポインタはエラー
     ;パラメータがvoid型、voidポインタはエラー
     (let* ((func-declarator (stx:func_def_st-func-declarator-st st))
            (func-para-list (stx:func_declarator_st-para-list func-declarator))
            (func-obj (stx:func_declarator_st-name func-declarator))
            (func-type (obj-type func-obj))
            (func-out-type (type_fun-out func-type))
            (func-compound-state (stx:compound_st-statement-list
                                  (stx:func_def_st-compound-state-list st)))
            (func-compound-decl (stx:compound_st-declaration-list
                                 (stx:func_def_st-compound-state-list st))))
       (cond ((and (equal? 'well-typed
                           (cond ((equal? (type_pointer 'pointer 'void )
```

```
                                        func-out-type)
                                 (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
                                                (obj-name func-obj)(obj-pos func-obj))))
                                ((equal? 'well-typed (check-type-para func-para-list))
                                 'well-typed)
                                (else 'well-typed)))
                    ;begin文の前者がうまく実行されれば#tが出力される.
                    ;そうでなければ勝手にエラーで止まる.
                    (begin
                      (cond ((equal? 'nodecl func-compound-decl) 'well-typed)
                            (else (map check-type func-compound-decl)))
                      (cond ((equal? 'nostat func-compound-state) 'well-typed)
                            (else (map check-type func-compound-state)))
                      #t))
                'well-typed)
               (else (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
                                    (obj-name func-obj)(obj-pos func-obj)))))))
     ((stx:null_statement_st? st) 'well-typed)
     ((stx:exp_in_paren_st? st)
      (check-type (stx:exp_in_paren_st-exp st)))
     ((stx:sem_return_st? st)
      (cond ((equal? (stx:sem_return_st-exp st) 'noreturn)
             'well-typed)
            ((type-void? (stx:sem_return_st-exp st))
             (error (format "ERROR NOT WELL TYPED RETURN-STATEMENT AT ~a"
                            (stx:sem_return_st-pos st))))
            ((and (type-int? (stx:sem_return_st-exp st))
                  (equal? 'int (type_fun-out (obj-type (stx:sem_return_st-tag st)))))
             'well-typed)
            ((and (type-intp? (stx:sem_return_st-exp st))
                  (equal? (type_pointer 'pointer 'int)
                          (type_fun-out (obj-type (stx:sem_return_st-tag st)))))
             'well-typed)
            ((and (type-intp? (stx:sem_return_st-exp st))
                  (equal? (type_pointer 'pointer 'int)
                          (type_fun-out (obj-type (stx:sem_return_st-tag st)))))
             'well-typed)
            (else (error (format "ERROR NOT WELL TYPED RETURN-STATEMENT AT ~a"
                                 (stx:sem_return_st-pos st))))))
     ((stx:if_else_st? st)
      (cond ((and (type-int? (stx:if_else_st-cond-exp st))
                  (equal? 'well-typed (check-type (stx:if_else_st-state st)))
                  (equal? 'well-typed (check-type (stx:if_else_st-else-state st))))
             'well-typed)
            (else (error (format "ERROR NOT WELL TYPED IF-STATEMENT AT ~a"
                                 (stx:if_else_st-if-pos st))))))
     ((stx:while_st? st)
      (cond ((and (type-int? (stx:while_st-cond-exp st))
                  (equal? 'well-typed (check-type (stx:while_st-statement st))))
             'well-typed)
            (else (error (format "ERROR NOT WELL TYPED WHILE-STATEMENT AT ~a"
                                 (stx:while_st-pos st))))))
     ((stx:compound_st? st)
      (begin
        (cond ((equal? 'nostat (stx:compound_st-statement-list st)) 'well-typed)
              (else (map check-type (stx:compound_st-statement-list st))))
        (cond ((equal? 'nodecl (stx:compound_st-declaration-list st)) 'well-typed)
              (else (map check-type (stx:compound_st-declaration-list st))))
        ;各要素はerrorか'well-typedを返すので、mapが実行されれば
        ;必然的にlistの要素は'well-typedになっている.
        'well-typed))
     (else (cond ((or (equal? 'int (type st))
                      (equal? (type_pointer 'pointer 'int) (type st))
                      (equal? (type_pointer 'pointer (type_pointer 'pointer 'int)) (type st))
                      (equal? 'void (type st)))
                  'well-typed)
                 (else (error "ERROR NOT WELL TYPED" st))))))

;型は'int、(type_pointer 'pointer 'int)、(type_pointer 'pointer (type_pointer 'pointer 'int))
(define (sametype? x y)
  (equal? (type x) (type y)))
(define (type-int? x)
  (equal? 'int (type x)))
(define (type-intp? x)
  (equal? (type_pointer 'pointer 'int) (type x)))
(define (type-intpp? x)
  (equal? (type_pointer 'pointer (type_pointer 'pointer 'int)) (type x)))
(define (type-void? x)
  (equal? 'void (type x)))
```

```
159
160 ;objのlistもしくは(list 'nopara)を受け取って
161 ;エラーすなわち
162 ;パラメターの中にvoid型、voidポインタ型がなければ
163 ;well-typedを返す.
164 (define (check-type-para para-list)
165   (cond ((equal? 'nopara para-list) 'well-typed)
166         (else (map (lambda (x)
167                      (let ((x-type (obj-type x)))
168                        (begin (cond ((or (equal? 'void x-type)
169                                          (equal? (type_pointer 'pointer 'void) x-type))
170                                      (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
171                                                     (obj-name x)(obj-pos x))))
172                                     (else 'well-typed))
173                               'well-typed)))
174                    para-list))))
175
176 (define (type st)
177   (cond
178     ((stx:sem_return_st? st) (type (stx:sem_return_st-exp st)))
179     ((stx:exp_in_paren_st? st) (type (stx:exp_in_paren_st-exp st)))
180     ((stx:assign_exp_st? st)
181      (let* ((type-dest (type (stx:assign_exp_st-dest st)))
182             (type-src (type (stx:assign_exp_st-src st))))
183        (cond ((equal? type-dest type-src)
184               type-dest)
185              (else (error (format "ERROR NOT WELL TYPED '=' AT ~a ~a"
186                                   (stx:assign_exp_st-pos st) st))))))
187     ((stx:logic_exp_st? st)
188      (let* ((type-op1 (type (stx:logic_exp_st-op1 st)))
189             (type-op2 (type (stx:logic_exp_st-op2 st))))
190        (cond ((and (type-int? type-op1)
191                    (type-int? type-op2))
192               'int)
193              (else (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
194                                   (cond ((equal? 'or (stx:logic_exp_st-log-ope st)) '||)
195                                         ((equal? 'or (stx:logic_exp_st-log-ope st)) '&&))
196                                   (stx:logic_exp_st-pos st)))))))
197
198     ((stx:rel_exp_st? st)
199      (let* ((type-op1 (type (stx:rel_exp_st-op1 st)))
200             (type-op2 (type (stx:rel_exp_st-op2 st))))
201        (cond ((equal? type-op1 type-op2)
202               'int)
203              (else (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
204                                   (cond ((equal? 'equal (stx:rel_exp_st-rel-ope st)) '==)
205                                         ((equal? 'not (stx:rel_exp_st-rel-ope st)) '!=)
206                                         ((equal? 'less (stx:rel_exp_st-rel-ope st)) '<)
207                                         ((equal? 'and_less (stx:rel_exp_st-rel-ope st)) '<=)
208                                         ((equal? 'more (stx:rel_exp_st-rel-ope st)) '>)
209                                         ((equal? 'and_more (stx:rel_exp_st-rel-ope st)) '>=))
210                                   (stx:rel_exp_st-pos st)))))))
211     ((stx:alge_exp_st? st)
212      (let* ((type-op1 (stx:alge_exp_st-op1 st))
213             (type-op2 (stx:alge_exp_st-op2 st))
214             (ope (stx:alge_exp_st-alge-ope st))
215             (pos (stx:alge_exp_st-pos st)))
216        (cond ((equal? 'add ope)
217               (cond ((and (type-int? type-op1)
218                           (type-int? type-op2))
219                      'int)
220                     ((and (type-intp? type-op1)
221                           (type-int? type-op2))
222                      (type_pointer 'pointer 'int))
223                     ((and (type-int? type-op1)
224                           (type-intp? type-op2))
225                      (type_pointer 'pointer 'int))
226                     ((and (type-intpp? type-op1)
227                           (type-int? type-op2))
228                      (type_pointer 'pointer (type_pointer 'pointer 'int)))
229                     ((and (type-int? type-op1)
230                           (type-intpp? type-op2))
231                      (type_pointer 'pointer (type_pointer 'pointer 'int)))
232                     (else (error (format "ERROR NOT WELL TYPED '+' AT ~a" pos)))))
233              ((equal? 'sub ope)
234               (cond ((and (type-intp? type-op1)
235                           (type-int? type-op2))
236                      (type_pointer 'pointer 'int))
237                     ((and (type-intpp? type-op1)
238                           (type-int? type-op2))
```

```
                    (type_pointer 'pointer (type_pointer 'pointer 'int)))
                   ((and (type-int? type-op1)
                         (type-int? type-op2)) 'int)
                   (else (error (format "ERROR NOT WELL TYPED '-' AT ~a" pos)))))
            ((or (equal? 'mul ope)
                 (equal? 'div ope))
             (cond ((and (type-int? type-op1)
                         (type-int? type-op2))
                    'int)
                   (else (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
                                        (cond ((equal? 'mul ope) '*)
                                              ((equal? 'div ope) '/))
                                        pos)))))
            ;デバグ用.(本来はどんなプログラムを書いてもこの分岐には入らないはず.)
            (else (error "ERROR NOT WELL TYPED" st)))))
    ((stx:unary_exp_st? st)
     (let* ((mark (stx:unary_exp_st-mark st))
            (op (stx:unary_exp_st-op st))
            (type-op (type op))
            (pos (stx:unary_exp_st-pos st)))
       (cond ((equal? 'amp mark)
              (cond ((equal? 'int op) 'int)
                    (else (error (format "ERROR NOT WELL TYPED '&' AT ~a" pos)))))
             ((equal? 'ast mark)
              (cond ((equal? (type_pointer 'pointer 'int) type-op) 'int)
                    ((equal? (type_pointer 'pointer (type_pointer 'pointer 'int)) type-op)
                     (type_pointer 'pointer 'int))
                    (else (error (format "ERROR NOT WELL TYPED '*' AT ~a" st)))))
            ;デバグ用.(本来はどんなプログラムを書いてもこの分岐には入らないはず.)
            (else (error "ERROR NOT WELL TYPED" st)))))
    ((stx:constant_st? st) 'int)
    ((stx:func_st? st)
     (let* (;funcを表すobj
            (func-ref (stx:func_st-name st))
            ;funcを表すobj中のtype_fun
            (func-type-fun (obj-type func-ref))
            ;funcを表すobj中のtyp_fun内のパラメータの型のlist
            ;もしくは'nopara
            (func-in-list (type_fun-in func-type-fun))
            ;type_fun中の関数の戻り値
            ;(type_pointer 'pointer 'int)か'intか'void
            (func-out (type_fun-out func-type-fun))
            ;パラメータのobjのlistもしくは'nopara
            (func-para (stx:func_st-para st))
            (pos (obj-pos func-ref))
            (func-name (obj-name func-ref)))
       (cond ((equal? (map (lambda (x)
                             (cond
                               ((equal? 'nopara x) 'nopara)
                               (else (type x))))
                           func-para)
                      func-in-list)
              (cond
                ;関数の戻り値として許されるのは
                ;int、intのポインタ型、void型
                ((equal? 'int func-out) 'int)
                ((equal? 'void func-out) 'void)
                ((equal? (type_pointer 'pointer 'int) func-out) (type_pointer 'pointer 'int))
                (else (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
                                     func-name pos)))))
             (else (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
                                  func-name pos))))))
    ((obj? st)
     (let* ((type-obj (obj-type st)))
       (cond
         ;配列型のとき
         ((type_array? type-obj)
          ;ポインタ型として許されるのはintのみ
          (cond ((equal? 'int (type_array-type type-obj)) 'int)
                (else (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
                                     (obj-name st) (obj-pos st))))))
         ;配列型でないとき
         ;許されるのはint、intのポインタ型のみ
         (else
          (cond ((equal? 'int type-obj) 'int)
                ((equal? (type_pointer 'pointer 'int) type-obj) (type_pointer 'pointer 'int))
                (else (error (format "ERROR NOT WELL TYPED '~a' AT ~a"
                                     (obj-name st) (obj-pos st)))))))))))
```

```
319
320
321
322 ;テスト
323 ;(define p100 (open-input-file "kadai01.c"))
324 ;(port-count-lines! p100)
325 ;;(sem:sem-analyze-tree (k08:parse-port p100))
326 ;(analy-type (sem:sem-analyze-tree (k08:parse-port p100)))
```

3 実行結果

これらのプログラムを使って次のテストプログラムを解析した.

リスト 10: テストプログラム

```
1  int comp_num(int a, int b);
2  int sort_array[8];
3
4
5  int comp_num(int a, int b){
6      if(a > b) return 0;
7      if(a < b) return 1;
8      if(a == b) return 2;
9  }
10
11
12 int main(){
13     int i;
14     int j;
15     int h;
16
17     sort_array[0] = 6;
18     sort_array[1] = 4;
19     sort_array[2] = 2;
20     sort_array[3] = 5;
21     sort_array[4] = 7;
22     sort_array[5] = 8;
23     sort_array[6] = 1;
24     sort_array[7] = 3;
25
26
27     for(j = 8; j > 1; j = (j-1)){
28         for(i = 0; i < (j-1); i = (i+1)){
29             if(comp_num(sort_array[i+1],sort_array[i])){
30                 h = sort_array[i+1];
31                 sort_array[i+1] = sort_array[i];
32                 sort_array[i] = h;
33             }
34
35         }
36     }
37     i = 0;
38     while(8 > i){
39         if(i != 7){
40         print(sort_array[i]);
41         }
42         else(print(sort_array[7]));
43         i = (i+1);
44     }
45
46 }
```

実行結果は次のようになった.

リスト 11: 課題 8 の実行結果

```
1  #lang racket
2  ;;;;;;;;;;;;;;;;;;;;;;;;;;;以下が課題8の実行結果です;;;;;;;;;;;;;;;;;;;;;;;;;;;.
3  (list*
4   (func_proto_st
5    (spec_st 'void 'print-proto)
6    (func_declarator_st
```

```
   'print
   (para_declaration_st (spec_st 'int 'print-proto) (id_st 'v 'print-proto))
   'print-proto))
  (cons
   (cons
    (func_proto_st
     (spec_st 'int (position 1 1 0))
     (func_declarator_st
      'comp_num
      (cons
       (para_declaration_st (spec_st 'int (position 14 1 13)) (id_st 'a (position 18 1 17)))
       (para_declaration_st (spec_st 'int (position 21 1 20)) (id_st 'b (position 25 1 24))))
      (position 5 1 4)))
    (declaration_st
     (spec_st 'int (position 29 2 0))
     (declarator_st (array_st 'sort_array 8 (position 33 2 4)))))
   (func_def_st
    (spec_st 'int (position 50 5 0))
    (func_declarator_st
     'comp_num
     (cons
      (para_declaration_st (spec_st 'int (position 63 5 13)) (id_st 'a (position 67 5 17)))
      (para_declaration_st (spec_st 'int (position 70 5 20)) (id_st 'b (position 74 5 24))))
     (position 54 5 4))
    (compound_sta_st
     (cons
      (cons
       (if_else_st
        (rel_exp_st 'more (id_st 'a (position 85 6 7)) (id_st 'b (position 89 6 11)) (position 87 6 9))
        (return_st (constant_st 0 (position 99 6 21)) (position 92 6 14))
        (null_statement_st 'null)
        (position 82 6 4)
        'syntax-sygar)
       (if_else_st
        (rel_exp_st
         'less
         (id_st 'a (position 109 7 7))
         (id_st 'b (position 113 7 11))
         (position 111 7 9))
        (return_st (constant_st 1 (position 123 7 21)) (position 116 7 14))
        (null_statement_st 'null)
        (position 106 7 4)
        'syntax-sygar))
      (if_else_st
       (rel_exp_st
        'equal
        (id_st 'a (position 133 8 7))
        (id_st 'b (position 138 8 12))
        (position 135 8 9))
       (return_st (constant_st 2 (position 148 8 22)) (position 141 8 15))
       (null_statement_st 'null)
       (position 130 8 4)
       'syntax-sygar)))))
  (func_def_st
   (spec_st 'int (position 155 12 0))
   (func_declarator_null_st 'main (position 159 12 4))
   (compound_st
    (cons
     (cons
      (declaration_st
       (spec_st 'int (position 171 13 4))
       (declarator_st (id_st 'i (position 175 13 8))))
      (declaration_st
       (spec_st 'int (position 182 14 4))
       (declarator_st (id_st 'j (position 186 14 8)))))
     (declaration_st
      (spec_st 'int (position 193 15 4))
      (declarator_st (id_st 'h (position 197 15 8)))))
    (cons
     (cons
      (cons
       (cons
        (cons
         (cons
          (cons
           (cons
            (cons
             (assign_exp_st
              (unary_exp_st
```

```
              'ast
              (exp_in_paren_st
               (alge_exp_st
                'add
                (id_st 'sort_array (position 205 17 4))
                (constant_st 0 (position 216 17 15))
                'syntax-sugar))
              'syntax-sugar)
             (constant_st 6 (position 221 17 20))
             (position 219 17 18))
            (assign_exp_st
             (unary_exp_st
              'ast
              (exp_in_paren_st
               (alge_exp_st
                'add
                (id_st 'sort_array (position 228 18 4))
                (constant_st 1 (position 239 18 15))
                'syntax-sugar))
              'syntax-sugar)
             (constant_st 4 (position 244 18 20))
             (position 242 18 18)))
           (assign_exp_st
            (unary_exp_st
             'ast
             (exp_in_paren_st
              (alge_exp_st
               'add
               (id_st 'sort_array (position 251 19 4))
               (constant_st 2 (position 262 19 15))
               'syntax-sugar))
             'syntax-sugar)
            (constant_st 2 (position 267 19 20))
            (position 265 19 18)))
          (assign_exp_st
           (unary_exp_st
            'ast
            (exp_in_paren_st
             (alge_exp_st
              'add
              (id_st 'sort_array (position 274 20 4))
              (constant_st 3 (position 285 20 15))
              'syntax-sugar))
            'syntax-sugar)
           (constant_st 5 (position 290 20 20))
           (position 288 20 18)))
         (assign_exp_st
          (unary_exp_st
           'ast
           (exp_in_paren_st
            (alge_exp_st
             'add
             (id_st 'sort_array (position 297 21 4))
             (constant_st 4 (position 308 21 15))
             'syntax-sugar))
           'syntax-sugar)
          (constant_st 7 (position 313 21 20))
          (position 311 21 18)))
        (assign_exp_st
         (unary_exp_st
          'ast
          (exp_in_paren_st
           (alge_exp_st
            'add
            (id_st 'sort_array (position 320 22 4))
            (constant_st 5 (position 331 22 15))
            'syntax-sugar))
          'syntax-sugar)
         (constant_st 8 (position 336 22 20))
         (position 334 22 18)))
       (assign_exp_st
        (unary_exp_st
         'ast
         (exp_in_paren_st
          (alge_exp_st
           'add
           (id_st 'sort_array (position 343 23 4))
           (constant_st 6 (position 354 23 15))
           'syntax-sugar))
         'syntax-sugar)
```

```
          (constant_st 1 (position 359 23 20))
          (position 357 23 18)))
        (assign_exp_st
         (unary_exp_st
          'ast
          (exp_in_paren_st
           (alge_exp_st
            'add
            (id_st 'sort_array (position 366 24 4))
            (constant_st 7 (position 377 24 15))
            'syntax-sugar))
          'syntax-sugar)
         (constant_st 3 (position 382 24 20))
         (position 380 24 18)))
       (compound_sta_st
        (cons
         (assign_exp_st
          (id_st 'j (position 399 27 8))
          (constant_st 8 (position 403 27 12))
          (position 401 27 10))
         (while_st
          (rel_exp_st
           'more
           (id_st 'j (position 406 27 15))
           (constant_st 1 (position 410 27 19))
           (position 408 27 17))
          (compound_sta_st
           (cons
            (compound_sta_st
             (compound_sta_st
              (cons
               (assign_exp_st
                (id_st 'i (position 437 28 12))
                (constant_st 0 (position 441 28 16))
                (position 439 28 14))
               (while_st
                (rel_exp_st
                 'less
                 (id_st 'i (position 444 28 19))
                 (exp_in_paren_st
                  (alge_exp_st
                   'sub
                   (id_st 'j (position 449 28 24))
                   (constant_st 1 (position 451 28 26))
                   (position 450 28 25)))
                 (position 446 28 21))
                (compound_sta_st
                 (cons
                  (compound_sta_st
                   (if_else_st
                    (func_st
                     'comp_num
                     (cons
                      (unary_exp_st
                       'ast
                       (exp_in_paren_st
                        (alge_exp_st
                         'add
                         (id_st 'sort_array (position 491 29 24))
                         (alge_exp_st
                          'add
                          (id_st 'i (position 502 29 35))
                          (constant_st 1 (position 504 29 37))
                          (position 503 29 36))
                         'syntax-sugar))
                       'syntax-sugar)
                      (unary_exp_st
                       'ast
                       (exp_in_paren_st
                        (alge_exp_st
                         'add
                         (id_st 'sort_array (position 507 29 40))
                         (id_st 'i (position 518 29 51))
                         'syntax-sugar))
                       'syntax-sugar)))
                    (compound_sta_st
                     (cons
                      (cons
                       (assign_exp_st
                        (id_st 'h (position 540 30 16))
```

```
                      (unary_exp_st
                       'ast
                       (exp_in_paren_st
                        (alge_exp_st
                         'add
                         (id_st 'sort_array (position 544 30 20))
                         (alge_exp_st
                          'add
                          (id_st 'i (position 555 30 31))
                          (constant_st 1 (position 557 30 33))
                          (position 556 30 32))
                         'syntax-sugar))
                       'syntax-sugar)
                      (position 542 30 18))
                     (assign_exp_st
                      (unary_exp_st
                       'ast
                       (exp_in_paren_st
                        (alge_exp_st
                         'add
                         (id_st 'sort_array (position 577 31 16))
                         (alge_exp_st
                          'add
                          (id_st 'i (position 588 31 27))
                          (constant_st 1 (position 590 31 29))
                          (position 589 31 28))
                         'syntax-sugar))
                       'syntax-sugar)
                      (unary_exp_st
                       'ast
                       (exp_in_paren_st
                        (alge_exp_st
                         'add
                         (id_st 'sort_array (position 595 31 34))
                         (id_st 'i (position 606 31 45))
                         'syntax-sugar))
                       'syntax-sugar)
                      (position 593 31 32)))
                    (assign_exp_st
                     (unary_exp_st
                      'ast
                      (exp_in_paren_st
                       (alge_exp_st
                        'add
                        (id_st 'sort_array (position 626 32 16))
                        (id_st 'i (position 637 32 27))
                        'syntax-sugar))
                      'syntax-sugar)
                     (id_st 'h (position 642 32 32))
                     (position 640 32 30))))
                  (null_statement_st 'null)
                  (position 479 29 12)
                  'syntax-sygar))
                (assign_exp_st
                 (id_st 'i (position 455 28 30))
                 (exp_in_paren_st
                  (alge_exp_st
                   'add
                   (id_st 'i (position 460 28 35))
                   (constant_st 1 (position 462 28 37))
                   (position 461 28 36)))
                  (position 457 28 32))))
               'syntax-sugar))))
           (assign_exp_st
            (id_st 'j (position 413 27 22))
            (exp_in_paren_st
             (alge_exp_st
              'sub
              (id_st 'j (position 418 27 27))
              (constant_st 1 (position 420 27 29))
              (position 419 27 28)))
             (position 415 27 24))))
         'syntax-sugar))))
    (assign_exp_st
     (id_st 'i (position 692 37 4))
     (constant_st 0 (position 696 37 8))
     (position 694 37 6)))
   (while_st
    (rel_exp_st
     'more
```

```
       (constant_st 8 (position 709 38 10))
       (id_st 'i (position 713 38 14))
       (position 711 38 12))
      (compound_sta_st
       (cons
        (if_else_st
         (rel_exp_st
          'not
          (id_st 'i (position 728 39 11))
          (constant_st 7 (position 733 39 16))
          (position 730 39 13))
         (compound_sta_st
          (func_st
           'print
           (unary_exp_st
            'ast
            (exp_in_paren_st
             (alge_exp_st
              'add
              (id_st 'sort_array (position 751 40 14))
              (id_st 'i (position 762 40 25))
              'syntax-sugar))
             'syntax-sugar)))
         (exp_in_paren_st
          (func_st
           'print
           (unary_exp_st
            'ast
            (exp_in_paren_st
             (alge_exp_st
              'add
              (id_st 'sort_array (position 796 42 19))
              (constant_st 7 (position 807 42 30))
              'syntax-sugar))
             'syntax-sugar)))
         (position 725 39 8)
         (position 785 42 8))
        (assign_exp_st
         (id_st 'i (position 821 43 8))
         (exp_in_paren_st
          (alge_exp_st
           'add
           (id_st 'i (position 826 43 13))
           (constant_st 1 (position 828 43 15))
           (position 827 43 14)))
         (position 823 43 10))))
      (position 703 38 4))))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;以下が意味解析の実行結果です;;;;;;;;;;;;;;;;;;;;;;;;;;.
OK! CRRECT PARAMETERS OF FUNCTION PROTOTYPE
OK! CRRECT FUNCTION PROTOTYPE OF 'comp_num'.
OK! CRRECT PARAMETERS OF FUNCTION PROTOTYPE
OK! CRRECT FUNCTION PROTOTYPE OF 'comp_num'.
OK! CRRECT FUNCTION PROTOTYPE OF 'comp_num'.
OK! CRRECT FUNCTION PROTOTYPE OF 'comp_num'.
OK! CRRECT PARAMETERS OF FUNCTION PROTOTYPE
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! CRRECT FUNCTION PROTOTYPE OF 'main'.
OK! CRRECT FUNCTION PROTOTYPE OF 'main'.
OK! CRRECT FUNCTION PROTOTYPE OF 'main'.
OK! CRRECT FUNCTION PROTOTYPE OF 'main'.
OK! CRRECT PARAMETERS OF FUNCTION PROTOTYPE.
OK! CORRENCT DECLARATIONS IN COMPONUND STATEMENT!
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
(list
 (func_proto_st
  (spec_st 'void 'print-proto)
  (func_declarator_st
   (obj 'print 0 'proto (type_fun 'fun 'void '(int)) 'print-proto)
   (list (obj 'v 1 'parm 'int 'print-proto))
   'print-proto))
```

```
(func_proto_st
 (spec_st 'int (position 1 1 0))
 (func_declarator_st
  (obj 'comp_num 0 'proto (type_fun 'fun 'int '(int int)) (position 5 1 4))
  (list (obj 'a 1 'parm 'int (position 18 1 17)) (obj 'b 1 'parm 'int (position 25 1 24)))
  (position 5 1 4)))
(declaration_st
 (spec_st 'int (position 29 2 0))
 (list (obj 'sort_array 0 'var (type_array 'int 8) (position 33 2 4))))
(func_def_st
 (spec_st 'int (position 50 5 0))
 (func_declarator_st
  (obj 'comp_num 0 'fun (type_fun 'fun 'int '(int int)) (position 54 5 4))
  (list (obj 'a 1 'parm 'int (position 67 5 17)) (obj 'b 1 'parm 'int (position 74 5 24)))
  (position 54 5 4))
 (compound_st
  'nodecl
  (list
   (if_else_st
    (rel_exp_st
     'more
     (obj 'a 1 'parm 'int (position 67 5 17))
     (obj 'b 1 'parm 'int (position 74 5 24))
     (position 87 6 9))
    (sem_return_st
     (constant_st 0 (position 99 6 21))
     (position 92 6 14)
     (obj 'comp_num 0 'fun (type_fun 'fun 'int '(int int)) (position 54 5 4)))
    (null_statement_st 'null)
    (position 82 6 4)
    'syntax-sygar)
   (if_else_st
    (rel_exp_st
     'less
     (obj 'a 1 'parm 'int (position 67 5 17))
     (obj 'b 1 'parm 'int (position 74 5 24))
     (position 111 7 9))
    (sem_return_st
     (constant_st 1 (position 123 7 21))
     (position 116 7 14)
     (obj 'comp_num 0 'fun (type_fun 'fun 'int '(int int)) (position 54 5 4)))
    (null_statement_st 'null)
    (position 106 7 4)
    'syntax-sygar)
   (if_else_st
    (rel_exp_st
     'equal
     (obj 'a 1 'parm 'int (position 67 5 17))
     (obj 'b 1 'parm 'int (position 74 5 24))
     (position 135 8 9))
    (sem_return_st
     (constant_st 2 (position 148 8 22))
     (position 141 8 15)
     (obj 'comp_num 0 'fun (type_fun 'fun 'int '(int int)) (position 54 5 4)))
    (null_statement_st 'null)
    (position 130 8 4)
    'syntax-sygar))))
(func_def_st
 (spec_st 'int (position 155 12 0))
 (func_declarator_st
  (obj 'main 0 'fun (type_fun 'fun 'int 'nopara) (position 159 12 4))
  'nopara
  (position 159 12 4))
 (compound_st
  (list
   (declaration_st
    (spec_st 'int (position 171 13 4))
    (list (obj 'i 2 'var 'int (position 175 13 8))))
   (declaration_st
    (spec_st 'int (position 182 14 4))
    (list (obj 'j 2 'var 'int (position 186 14 8))))
   (declaration_st
    (spec_st 'int (position 193 15 4))
    (list (obj 'h 2 'var 'int (position 197 15 8)))))
  (list
   (assign_exp_st
    (obj 'sort_array 0 'var (type_array 'int (constant_st 0 (position 216 17 15))) (position 33 2 4))
    (constant_st 6 (position 221 17 20))
    (position 219 17 18))
   (assign_exp_st
```

```
      (obj 'sort_array 0 'var (type_array 'int (constant_st 1 (position 239 18 15))) (position 33 2 4))
      (constant_st 4 (position 244 18 20))
      (position 242 18 18))
     (assign_exp_st
      (obj 'sort_array 0 'var (type_array 'int (constant_st 2 (position 262 19 15))) (position 33 2 4))
      (constant_st 2 (position 267 19 20))
      (position 265 19 18))
     (assign_exp_st
      (obj 'sort_array 0 'var (type_array 'int (constant_st 3 (position 285 20 15))) (position 33 2 4))
      (constant_st 5 (position 290 20 20))
      (position 288 20 18))
     (assign_exp_st
      (obj 'sort_array 0 'var (type_array 'int (constant_st 4 (position 308 21 15))) (position 33 2 4))
      (constant_st 7 (position 313 21 20))
      (position 311 21 18))
     (assign_exp_st
      (obj 'sort_array 0 'var (type_array 'int (constant_st 5 (position 331 22 15))) (position 33 2 4))
      (constant_st 8 (position 336 22 20))
      (position 334 22 18))
     (assign_exp_st
      (obj 'sort_array 0 'var (type_array 'int (constant_st 6 (position 354 23 15))) (position 33 2 4))
      (constant_st 1 (position 359 23 20))
      (position 357 23 18))
     (assign_exp_st
      (obj 'sort_array 0 'var (type_array 'int (constant_st 7 (position 377 24 15))) (position 33 2 4))
      (constant_st 3 (position 382 24 20))
      (position 380 24 18))
     (compound_st
      'nodecl
      (list
       (assign_exp_st
        (obj 'j 2 'var 'int (position 186 14 8))
        (constant_st 8 (position 403 27 12))
        (position 401 27 10))
       (while_st
        (rel_exp_st
         'more
         (obj 'j 2 'var 'int (position 186 14 8))
         (constant_st 1 (position 410 27 19))
         (position 408 27 17))
        (compound_st
         'nodecl
         (list
          (compound_st
           'nodecl
           (list
            (compound_st
             'nodecl
             (list
              (assign_exp_st
               (obj 'i 2 'var 'int (position 175 13 8))
               (constant_st 0 (position 441 28 16))
               (position 439 28 14))
              (while_st
               (rel_exp_st
                'less
                (obj 'i 2 'var 'int (position 175 13 8))
                (exp_in_paren_st
                 (alge_exp_st
                  'sub
                  (obj 'j 2 'var 'int (position 186 14 8))
                  (constant_st 1 (position 451 28 26))
                  (position 450 28 25)))
                (position 446 28 21))
               (compound_st
                'nodecl
                (list
                 (compound_st
                  'nodecl
                  (list
                   (if_else_st
                    (func_st
                     (obj 'comp_num 0 'fun (type_fun 'fun 'int '(int int)) (position 54 5 4))
                     (list
                      (obj
                       'sort_array
                       0
                       'var
                       (type_array
                        'int
```

```
                    (alge_exp_st
                     'add
                     (id_st 'i (position 502 29 35))
                     (constant_st 1 (position 504 29 37))
                     (position 503 29 36)))
                   (position 33 2 4))
                  (obj
                   'sort_array
                   0
                   'var
                   (type_array 'int (id_st 'i (position 518 29 51)))
                   (position 33 2 4))))
                (compound_st
                 'nodecl
                 (list
                  (assign_exp_st
                   (obj 'h 2 'var 'int (position 197 15 8))
                   (obj
                    'sort_array
                    0
                    'var
                    (type_array
                     'int
                     (alge_exp_st
                      'add
                      (id_st 'i (position 555 30 31))
                      (constant_st 1 (position 557 30 33))
                      (position 556 30 32)))
                    (position 33 2 4))
                   (position 542 30 18))
                  (assign_exp_st
                   (obj
                    'sort_array
                    0
                    'var
                    (type_array
                     'int
                     (alge_exp_st
                      'add
                      (id_st 'i (position 588 31 27))
                      (constant_st 1 (position 590 31 29))
                      (position 589 31 28)))
                    (position 33 2 4))
                   (obj
                    'sort_array
                    0
                    'var
                    (type_array 'int (id_st 'i (position 606 31 45)))
                    (position 33 2 4))
                   (position 593 31 32))
                  (assign_exp_st
                   (obj
                    'sort_array
                    0
                    'var
                    (type_array 'int (id_st 'i (position 637 32 27)))
                    (position 33 2 4))
                   (obj 'h 2 'var 'int (position 197 15 8))
                   (position 640 32 30))))
                (null_statement_st 'null)
                (position 479 29 12)
                'syntax-sygar)))
              (assign_exp_st
               (obj 'i 2 'var 'int (position 175 13 8))
               (exp_in_paren_st
                (alge_exp_st
                 'add
                 (obj 'i 2 'var 'int (position 175 13 8))
                 (constant_st 1 (position 462 28 37))
                 (position 461 28 36)))
               (position 457 28 32))))
            'syntax-sugar)))))
       (assign_exp_st
        (obj 'j 2 'var 'int (position 186 14 8))
        (exp_in_paren_st
         (alge_exp_st
          'sub
          (obj 'j 2 'var 'int (position 186 14 8))
          (constant_st 1 (position 420 27 29))
          (position 419 27 28)))
```

```
           (position 415 27 24))))
        'syntax-sugar)))
     (assign_exp_st
      (obj 'i 2 'var 'int (position 175 13 8))
      (constant_st 0 (position 696 37 8))
      (position 694 37 6))
     (while_st
      (rel_exp_st
       'more
       (constant_st 8 (position 709 38 10))
       (obj 'i 2 'var 'int (position 175 13 8))
       (position 711 38 12))
      (compound_st
       'nodecl
       (list
        (if_else_st
         (rel_exp_st
          'not
          (obj 'i 2 'var 'int (position 175 13 8))
          (constant_st 7 (position 733 39 16))
          (position 730 39 13))
         (compound_st
          'nodecl
          (list
           (func_st
            (obj 'print 0 'proto (type_fun 'fun 'void '(int)) 'print-proto)
            (list
             (obj
              'sort_array
              0
              'var
              (type_array 'int (id_st 'i (position 762 40 25)))
              (position 33 2 4))))))
         (exp_in_paren_st
          (func_st
           (obj 'print 0 'proto (type_fun 'fun 'void '(int)) 'print-proto)
           (list
            (obj
             'sort_array
             0
             'var
             (type_array 'int (constant_st 7 (position 807 42 30)))
             (position 33 2 4)))))
         (position 725 39 8)
         (position 785 42 8))
        (assign_exp_st
         (obj 'i 2 'var 'int (position 175 13 8))
         (exp_in_paren_st
          (alge_exp_st
           'add
           (obj 'i 2 'var 'int (position 175 13 8))
           (constant_st 1 (position 828 43 15))
           (position 827 43 14)))
         (position 823 43 10))))
      (position 703 38 4))))))


;;;;;;;;;;;;;;;;;;;;;;;;;;;;以下が形検査の実行結果です;;;;;;;;;;;;;;;;;;;;;;;;;;.
OK! CRRECT PARAMETERS OF FUNCTION PROTOTYPE
OK! CRRECT FUNCTION PROTOTYPE OF 'comp_num'.
OK! CRRECT PARAMETERS OF FUNCTION PROTOTYPE
OK! CRRECT FUNCTION PROTOTYPE OF 'comp_num'.
OK! CRRECT FUNCTION PROTOTYPE OF 'comp_num'.
OK! CRRECT FUNCTION PROTOTYPE OF 'comp_num'.
OK! CRRECT PARAMETERS OF FUNCTION PROTOTYPE
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! CRRECT FUNCTION PROTOTYPE OF 'main'.
OK! CRRECT FUNCTION PROTOTYPE OF 'main'.
OK! CRRECT FUNCTION PROTOTYPE OF 'main'.
OK! CRRECT FUNCTION PROTOTYPE OF 'main'.
OK! CRRECT PARAMETERS OF FUNCTION PROTOTYPE.
OK! CORRENCT DECLARATIONS IN COMPONUND STATEMENT!
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
```

```
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! THIS PROGRAM IS IN A CORRECT FORM.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;以下が型検査の実行結果です;;;;;;;;;;;;;;;;;;;;;;;;;;;;.
OK! CRRECT PARAMETERS OF FUNCTION PROTOTYPE
OK! CRRECT FUNCTION PROTOTYPE OF 'comp_num'.
OK! CRRECT PARAMETERS OF FUNCTION PROTOTYPE
OK! CRRECT FUNCTION PROTOTYPE OF 'comp_num'.
OK! CRRECT FUNCTION PROTOTYPE OF 'comp_num'.
OK! CRRECT FUNCTION PROTOTYPE OF 'comp_num'.
OK! CRRECT PARAMETERS OF FUNCTION PROTOTYPE
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! CRRECT FUNCTION PROTOTYPE OF 'main'.
OK! CRRECT FUNCTION PROTOTYPE OF 'main'.
OK! CRRECT FUNCTION PROTOTYPE OF 'main'.
OK! CRRECT FUNCTION PROTOTYPE OF 'main'.
OK! CRRECT PARAMETERS OF FUNCTION PROTOTYPE.
OK! CORRENCT DECLARATIONS IN COMPONUND STATEMENT!
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! NO DECLARATIONS IN COMPONUND STATEMENT
OK! THIS PROGRAM IS WELL TYPED.
>
```

4 感想

今回の実験では前回作成したソースコードを文字列として受け取りそれを抽象構文木に変換して構造体の形で表示するというプログラムを改良し、シンタックスシュガーを識別して抽象構文木を作り直し、それを意味解析、形解析、型解析を適用するという処理を行うプログラムを作成したが、予想以上に苦労した. そもそもプログラム自体は提出期限内に完成したものだと思い込んでいるとバグが見つかり、大幅な改変をせざるを得なくなってしまった. その際に自分の書いたコード中で何を行っている処理なのかをすぐに把握できずにかなり苦労したので、これからは let 文で使う値を先に全て取り出してコメントに書いておくなど、工夫の余地も大きいと感じた.