

情報科学実験レポート

Sprolog

6321120 横溝 尚也

提出日：7月20日（水）

1 回答した問題と提出するソースファイルのファイル名

解答した問題問題 1～問題 8、

ファイル名

問題 1 : test_kadai1.ml

問題 4 : test_kadai4.ml

問題 5 : test_kadai5.ml

問題 6 : test_kadai6.ml

問題 7 : test_kadai7.ml

問題 8 (すべて完成したときのプログラム) : test_kadai8.ml

2 Prolog の字句解析と構文解析

2.1 問題 1

問題 1 では字句解析のプログラムにおいて指定の行のプログラムを行った。14～17、26～37 行目が上のプログラムで埋めた箇所である。

以下が字句解析のプログラムである。

(*実験課題*)

```
1 module Lexer = struct
2   type token = CID of string | VID of string | NUM of string
3             | TO | IS | QUIT | OPEN | EOF | ONE of char
```

(*課題 1*)

```
4 module P = Printf
5 exception End_of_system
7 let _ISTREAM = ref stdin
8 let ch = ref []
9 let read () = match !ch with [] -> input_char !_ISTREAM
10                | h::rest -> (ch := rest; h)
11 let unread c = ch := c::!ch
12 let lookahead () = try let c = read () in unread c; c with End_of_file -> '$'
13 let rec integer i =
```

(*課題 1 : 埋めた*)

```
14   let c = lookahead () in
15   if (c >= '0' && c <= '9') then
16     integer (i^(Char.escaped (read ())))
17   else i
18 and identifier id =
19   let c = lookahead () in
20   if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') ||
21       (c >= '0' && c <= '9') || c == '_') then
22     identifier (id^(Char.escaped (read ())))
23   else id
24 and native_token () =
25   let c = lookahead () in
```

(*課題 1 : 埋めた*)

```
26   if (c >= 'a' && c <= 'z') then
27     let id = identifier "" in
28     match id with
```

```

29     "is" -> IS
30     | "quit" -> QUIT
31     | "open" -> OPEN
32     | _ -> CID (id)
33     else if (c >= 'A' && c <= 'Z') then VID (identifier "")
34     else if (c >= '0' && c <= '9') then NUM (integer "")
35     else if (c = ':' ) then let x = read () in
36         if (lookahead () = '-' ) then let y = read () in TO
37         else ONE (':' )
38     else ONE (read ())
39 and gettoken () =
40     try
41         let token = native_token () in
42         match token with
43         | ONE ' ' -> gettoken ()
44         | ONE '\t' -> gettoken ()
45         | ONE '\n' -> gettoken ()
46         | _ -> token
47     with End_of_file -> EOF

48 let print_token tk =
49     match tk with
50     | (CID i) -> P.printf "CID(%s)" i
51     | (VID i) -> P.printf "VID(%s)" i
52     | (NUM i) -> P.printf "NUM(%s)" i
53     | (TO) -> P.printf ":-"
54     | (QUIT) -> P.printf "quit"
55     | (OPEN) -> P.printf "open"
56     | (IS) -> P.printf "is"
57     | (EOF) -> P.printf "eof"
58     | (ONE c) -> P.printf "ONE(%c)" c

59 end

```

問題1の実行結果を確認する際のファイル kadail.ml は、module Lexer (1～59行目) と、run() 関数で構成した。また、run() 関数は課題のスライドに記載されているものには、module Parser 内の関数も使用しているため、独自に実装した。問題1用の run() 関数は以下である。

```

let rec run () =
    flush stdout;
    let rlt = gettoken () in

```

```

match rlt with
  (ONE '$') -> raise End_of_system
  | _ -> (print_token rlt; P.printf "\n"; run())

```

字句解析 (module Lexer 内) を run() 関数を用いて実行した結果は以下のようになる。

```

# #use "kadail.ml";;
File "kadail.ml", line 38, characters 31-32:
Warning 26: unused variable x.
File "kadail.ml", line 39, characters 39-40:
Warning 26: unused variable y.
module Lexer :
sig
  type token =
    | CID of string
    | VID of string
    | NUM of string
    | TO
    | IS
    | QUIT
    | OPEN
    | EOF
    | ONE of char
  module P = Printf
  exception End_of_system
  val count : int ref
  val _ISTREAM : in_channel ref
  val ch : char list ref
  val read : unit -> char
  val unread : char -> unit
  val lookahead : unit -> char
  val integer : string -> string
  val identifier : string -> string
  val native_token : unit -> token
  val gettoken : unit -> token
  val print_token : token -> unit
  val run : unit -> 'a
end
# open Lexer;;
# run ();;
I am 19 years old$
VID(I)
CID(am)
NUM(19)
CID(years)
CID(old)
Exception: Lexer.End_of_system.
#

```

図1 run 関数による字句解析の実行結果

2.2 問題 2

問題 2 では左再帰の形をしている文法 `terms, args` を右再帰の形に変更した。問題 3 で変更した文法を使用しているがここにも変更後の文法を明記する。

```
terms -> term terms'
terms' -> , terms'

args -> arg args'
args' -> , args'
```

2.3 問題 3

問題 3 では構文解析のプログラムにおいて指定のプログラムを行った。79 行目～81 行目 (`to_opt`)、91～94(`term`)、95～98(`terms`)、99(`predicate`)、100～103(`args`)、104～120(`expr`)、121～123(`tail_opt`)、127～130(`list_opt`)、131～135(`id`) が上のプログラムで埋めた箇所である。

以下が構文解析のプログラムである。

```
(*課題3*)
60 module Parser = struct
61   module L = Lexer
62   (* module E = Evaluator *)
63   let tok = ref (L.ONE ' ')
64   let getToken () = L.getToken ()
65   let advance () = (tok := getToken(); L.print_token (!tok))
66   exception Syntax_error
67   let error () = raise Syntax_error
68   let check t = match !tok with
69     | L.CID _ -> if (t = (L.CID "")) then () else error()
70     | L.VID _ -> if (t = (L.VID "")) then () else error()
71     | L.NUM _ -> if (t = (L.NUM "")) then () else error()
72     | tk -> if (tk=t) then () else error()
73   let eat t = (check t; advance())
74   let rec clauses() = match !tok with
75     | L.EOF -> ()
76     | _ -> (clause(); clauses())
77   and clause() = match !tok with
78     | L.ONE '(' -> (term(); eat(L.ONE ' '))
79     | _ -> (predicate(); to_opt(); eat(L.ONE ' '))
80   (*課題3：埋めた*)
```

```

79 and to_opt() = match !tok with
80     L.TO -> (eat(L.TO); terms())
81     | _ -> ()
82 and command() = match !tok with
83     L.QUIT -> exit 0
84     | L.OPEN -> (eat(L.OPEN);
85                   match !tok with
86                       L.CID s -> (eat(L.CID ""); check (L.ONE ' ');
87                                   L._ISTREAM := open_in (s^".pl");
88                                   advance(); clauses(); close_in (!L._ISTREAM))
89                       | _ -> error())
90     | _ -> (term(); check(L.ONE ' '))
(*課題3：埋めた*)
91 and term() = match !tok with
92     L.ONE '(' -> (eat(L.ONE '('); term(); eat(L.ONE ')') )
94     | _ -> (predicate())
(*課題3：埋めた*)
95 and terms() = (term(); terms'())
96 and terms'() = match !tok with
97     L.ONE ',', -> (eat(L.ONE ',',); term(); terms'() )
98     | _ -> ()
(*課題3：埋めた*)
99 and predicate() = (eat(L.CID ""); eat(L.ONE '('); args(); eat(L.ONE ')') )
(*課題3：埋めた*)
100 and args() = (expr(); args'())
101 and args'() = match !tok with
102     L.ONE ',', -> (eat(L.ONE ',',); expr(); args'() )
103     | _ -> ()
(*課題3：埋めた*)
104 and expr() = (eat(L.ONE '('); expr(); eat(L.ONE ')') )
(*課題3：埋めた*)
121 and tail_opt() = match !tok with
122     L.ONE '(' -> (eat(L.ONE '('); args(); eat(L.ONE ')') )
123     | _ -> ()
124 and list() = match !tok with
125     L.ONE ']' -> ()
126     | _ -> (expr(); list_opt())
(*課題3：埋めた*)
127 and list_opt() = match !tok with
128     L.ONE '|' -> (eat(L.ONE '|'); id())

```

```

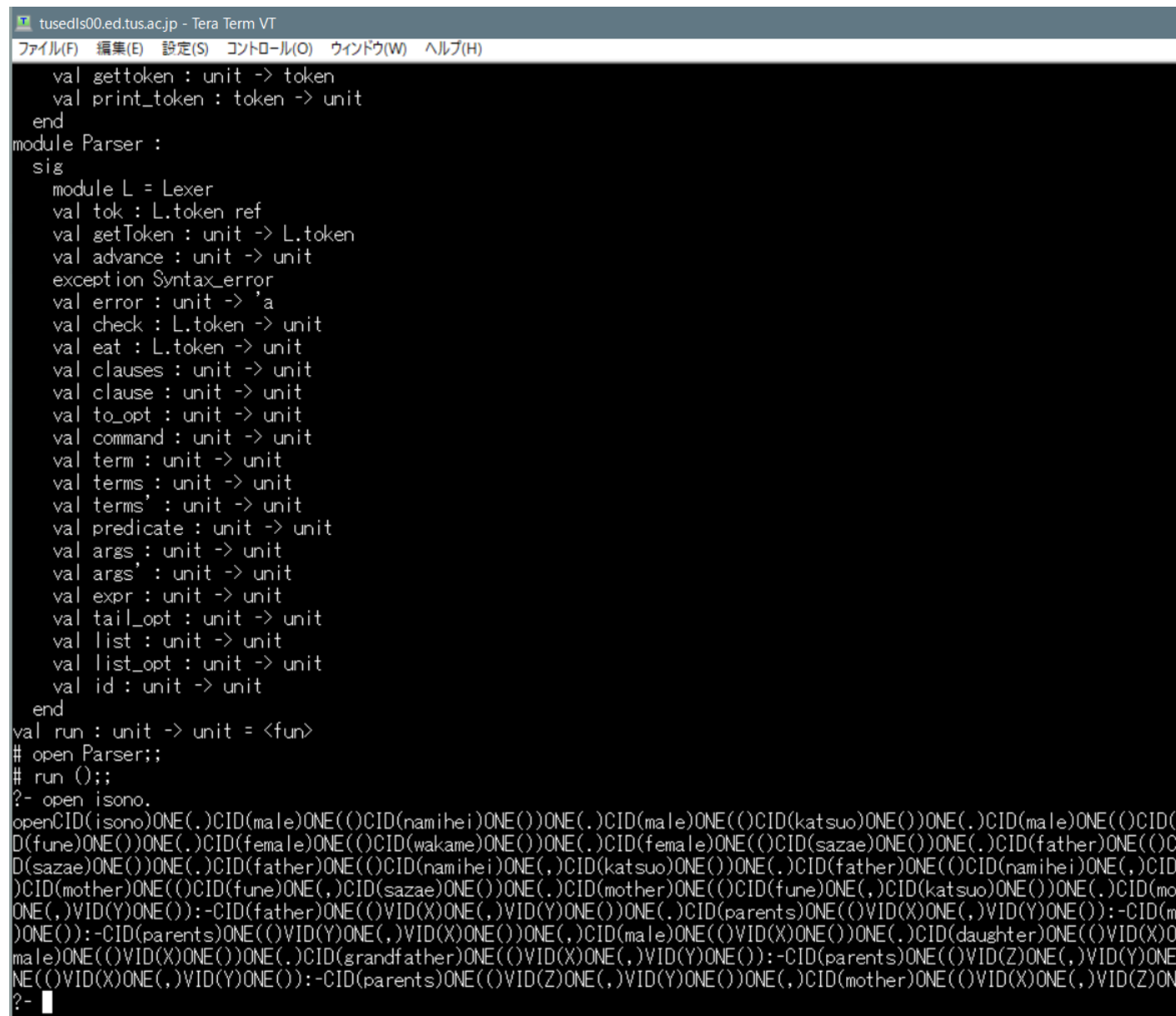
129   | L.ONE ',' -> (eat(L.ONE ',')); list()
130   | _ -> ()
(*課題3：埋めた*)
131 and id() = match !tok with
132     L.CID s -> (eat(L.CID ""))
133   | L.VID s -> (eat(L.VID ""))
134   | L.NUM n -> (eat(L.NUM ""))
135   | _ -> error ()
136end

137 let rec run() =
138   print_string "?- ";
139   while true do
140     flush stdout; Lexer._ISTREAM := stdin;
141     Parser.advance(); Parser.command(); print_string "\n?- "
142   done

```


2.4 問題 4

問題 4 ではここまでで作成したプログラムを使用して isono.pl がうまく実行されるかを確認する。その様子が以下ようになった。



```
tusedls00.ed.tus.ac.jp - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウインドウ(W) ヘルプ(H)

val gettoken : unit -> token
val print_token : token -> unit
end
module Parser :
sig
  module L = Lexer
  val tok : L.token ref
  val getToken : unit -> L.token
  val advance : unit -> unit
  exception Syntax_error
  val error : unit -> 'a
  val check : L.token -> unit
  val eat : L.token -> unit
  val clauses : unit -> unit
  val clause : unit -> unit
  val to_opt : unit -> unit
  val command : unit -> unit
  val term : unit -> unit
  val terms : unit -> unit
  val terms' : unit -> unit
  val predicate : unit -> unit
  val args : unit -> unit
  val args' : unit -> unit
  val expr : unit -> unit
  val tail_opt : unit -> unit
  val list : unit -> unit
  val list_opt : unit -> unit
  val id : unit -> unit
end
val run : unit -> unit = <fun>
# open Parser;;
# run ();;
?- open isono.
openCID(isono)ONE(.CID(male)ONE(.CID(namihei)ONE())ONE(.CID(male)ONE(.CID(katsuo)ONE())ONE(.CID(male)ONE(.CID(fune)ONE())ONE(.CID(female)ONE(.CID(wakame)ONE())ONE(.CID(female)ONE(.CID(sazae)ONE())ONE(.CID(father)ONE(.CID(sazae)ONE())ONE(.CID(father)ONE(.CID(namihei)ONE(.CID(katsuo)ONE())ONE(.CID(father)ONE(.CID(namihei)ONE(.CID(mother)ONE(.CID(fune)ONE(.CID(sazae)ONE())ONE(.CID(mother)ONE(.CID(fune)ONE(.CID(katsuo)ONE())ONE(.CID(mo
ONE(.VID(Y)ONE())):-CID(father)ONE(.VID(X)ONE(.VID(Y)ONE())ONE(.CID(parents)ONE(.VID(X)ONE(.VID(Y)ONE())):-CID(m
)ONE())):-CID(parents)ONE(.VID(Y)ONE(.VID(X)ONE())ONE(.CID(male)ONE(.VID(X)ONE())ONE(.CID(daughter)ONE(.VID(X)O
male)ONE(.VID(X)ONE())ONE(.CID(grandfather)ONE(.VID(X)ONE(.VID(Y)ONE())):-CID(parents)ONE(.VID(Z)ONE(.VID(Y)ONE
NE(.VID(X)ONE(.VID(Y)ONE())):-CID(parents)ONE(.VID(Z)ONE(.VID(Y)ONE())ONE(.CID(mother)ONE(.VID(X)ONE(.VID(Z)ON
?- 
```

図 2 isono.pl によるふるまい確認

2.5 問題 5

問題 5 では複数の述語を記述できるようにプログラムを改造した。

プログラムを修正した箇所は構文解析 (問題 3) の 90 行目のみである。90 行目 `command` 関数の `term()`;
を `terms()`; の変更した。comannd 関数の修正部分が以下である。

```
and command() = match !tok with
  L.QUIT -> exit 0
| L.OPEN -> (eat(L.OPEN);
  match !tok with
    L.CID s -> (eat(L.CID ""); check (L.ONE ' ');
      L._ISTREAM := open_in (s^".pl");
      advance(); clauses(); close_in (!L._ISTREAM))
  | _ -> error())
| _ -> (terms(); check(L.ONE ' '))
(*課題5：複数の述語定義できるように term を terms に修正*)
```

修正後のプログラムで複数の述語を入力したときのふるまいが以下となる。



```
val id : unit -> unit
end
val run : unit -> unit = <fun>
# open Parser;;
# run();;
?- male(X), female(Y), mother(Z).
CID(male)ONE(())VID(X)ONE(())ONE(,)CID(female)ONE(())VID(Y)ONE(())ONE(,)CID(mother)ONE(())VID(Z)ONE(())ONE(,)
?-
```

図 3 複数の述語を記述したときの実行結果

2.6 問題6

問題6では通常のエラーの出力を自分で記述し、そこにエラーが発生している行数を印字する。

まずエラーが発生した際の行数をカウントするために字句解析プログラムに変更を加えた。以下が問題1で作成した字句解析プログラムに変更を加えたプログラムであるが、変更した箇所は6, 45行目のみである。

```
4  module P = Printf
5  exception End_of_system
6  let count = ref 1 (*課題6：行数カウントのため修正*)
7  let _ISTREAM = ref stdin
8  let ch = ref []
9  let read () = match !ch with [] -> input_char !_ISTREAM
10                 | h::rest -> (ch := rest; h)
11  let unread c = ch := c::!ch
12  let lookahead () = try let c = read () in unread c; c with End_of_file -> '$'
13  let rec integer i =
14    let c = lookahead () in
15    if (c >= '0' && c <= '9') then
16      integer (i^(Char.escaped (read ())))
17    else i
18  and identifier id =
19    let c = lookahead () in
20    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) ||
21       (c >= '0' && c <= '9') || c == '_' then
22      identifier (id^(Char.escaped (read ())))
23    else id
24  and native_token () =
25    let c = lookahead () in
26    if (c >= 'a' && c <= 'z') then
27      let id = identifier "" in
28      match id with
29      | "is" -> IS
30      | "quit" -> QUIT
31      | "open" -> OPEN
32      | _ -> CID (id)
33    else if (c >= 'A' && c <= 'Z') then VID (identifier "")
34    else if (c >= '0' && c <= '9') then NUM (integer "")
35    else if (c = ':') then let x = read () in
36      if (lookahead () = '-') then let y = read () in TO
```

```

37     else ONE (':')
38     else ONE (read ())
39 and gettoken () =
40     try
41         let token = native_token () in
42         match token with
43             ONE ' ' -> gettoken ()
44         | ONE '\t' -> gettoken ()
45         | ONE '\n' -> count := !count + 1 ; gettoken () (*課題6：行数カウントのため修正*)
46         | _ -> token
47     with End_of_file -> EOF
48 let print_token tk =
49     match tk with
50         (CID i) -> P.printf "CID(%s)" i
51     | (VID i) -> P.printf "VID(%s)" i
52     | (NUM i) -> P.printf "NUM(%s)" i
53     | (TO) -> P.printf ":-"
54     | (QUIT) -> P.printf "quit"
55     | (OPEN) -> P.printf "open"
56     | (IS) -> P.printf "is"
57     | (EOF) -> P.printf "eof"
58     | (ONE c) -> P.printf "ONE(%c)" c
59 end

```

改行した回数をカウントする機能はプログラムできた。エラーを出力するプログラムを自分で実装したいため、今までの `run()` 関数から新たな `run'()` 関数を定義する。以下が `run'()` のプログラムである。このプログラムを構文解析器の後ろに付けた。

```

137 let rec run() =
138     print_string "?- ";
139     while true do
140         flush stdout; Lexer._ISTREAM := stdin;
141         Parser.advance(); Parser.command(); print_string "\n?- "
142     done

```

`isono.pl` の中身をあえて8行目にエラーの起きるようなプログラムに変更し、出力結果を確認したのが以下である。

```

tusedls00.ed.tus.ac.jp - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
tusedls08$ cat isono.pl
male(namihei). male(katsuo). male(tara). male(masuo).

female(fune). female(wakame). female(sazae).
father(masuo, tara). father(namihei, sazae). father(namihei, katsuo).
father(namihei, wakame).
mother(sazae, tara). mother(fune, sazae). mother(fune, katsuo).
mother(fune, wakame).
エラーです

parents(X, Y) :- father(X, Y). parents(X, Y) :- mother(X, Y).
son(X, Y) :- parents(Y, X), male(X).
daughter(X, Y) :- parents(Y, X), female(X).
grandfather(X, Y) :- parents(Z, Y), father(X, Z).
grandmother(X, Y) :- parents(Z, Y), mother(X, Z).
tusedls08$ ocaml
OCaml version 4.05.0

#

```

図4 isono.pl の変更後の中身

```

?- open isono.
openCID(isono)ONE(.)CID(male)ONE(.)CID(namihei)ONE(.)ONE(.)CID(male)ONE(.)CID(katsuo)ONE(.)ONE(.)CID(male)ONE(.)CID(
D(fune)ONE(.)ONE(.)CID(female)ONE(.)CID(wakame)ONE(.)ONE(.)CID(female)ONE(.)CID(sazae)ONE(.)ONE(.)CID(father)ONE(.)C
D(sazae)ONE(.)ONE(.)CID(father)ONE(.)CID(namihei)ONE(.)CID(katsuo)ONE(.)ONE(.)CID(father)ONE(.)CID(namihei)ONE(.)CID
)CID(mother)ONE(.)CID(fune)ONE(.)CID(sazae)ONE(.)ONE(.)CID(mother)ONE(.)CID(fune)ONE(.)CID(katsuo)ONE(.)ONE(.)CID(mo
文法エラー:8行目
?-- : unit = ()
#

```

図5 isono.pl によるエラー出力の確認

2.7 問題 7

問題 7 では算術計算できるように文法を拡張した。文法解析の部分拡張したが、変更したのは 104~122 行目 (expr の拡張) のみである。左再帰の文法があるが、右再帰に変更する方法は課題 2 と全く同じであるので説明は省略する。拡張したプログラムが以下である。

```
60 module Parser = struct
61   module L = Lexer
62   (* module E = Evaluator *)
63   let tok = ref (L.ONE ' ')
64   let getToken () = L.gettoken ()
65   let advance () = (tok := getToken(); L.print_token (!tok))
66   exception Syntax_error
67   let error () = raise Syntax_error
68   let check t = match !tok with
69     | L.CID _-> if (t = (L.CID "")) then () else error()
70     | L.VID _-> if (t = (L.VID "")) then () else error()
71     | L.NUM _-> if (t = (L.NUM "")) then () else error()
72     | tk -> if (tk=t) then () else error()
73   let eat t = (check t; advance())
74   let rec clauses() = match !tok with
75     | L.EOF -> ()
76     | _ -> (clause(); clauses())
77   and clause() = match !tok with
78     | L.ONE '(' -> (term(); eat(L.ONE '.'))
79     | _ -> (predicate(); to_opt(); eat(L.ONE '.'))
80   and to_opt() = match !tok with
81     | L.TO -> (eat(L.TO); terms())
82     | _ -> ()
83   and command() = match !tok with
84     | L.QUIT -> exit 0
85     | L.OPEN -> (eat(L.OPEN);
86       match !tok with
87         | L.CID s -> (eat(L.CID ""); check (L.ONE '.'));
88         | L._ISTREAM := open_in (s^".pl");
89         | _ -> error()
90         | _ -> (advance(); clauses(); close_in (!L._ISTREAM))
91     | _ -> (term(); check(L.ONE '.'))
92   and term() = match !tok with
```

```

92     L.ONE '(' -> (eat(L.ONE '('); term(); eat(L.ONE ')')) )
93   | _ -> (predicate())
94 and terms() = (term(); terms'())
95 and terms'() = match !tok with
96   L.ONE ',' -> (eat(L.ONE ','); term(); terms'() )
97   | _ -> ()
98 and predicate() = (eat(L.CID ""); eat(L.ONE '('); args(); eat(L.ONE ')')) )
99 and args() = (expr(); args'())
100 and args'() = match !tok with
101   L.ONE ',' -> (eat(L.ONE ','); expr(); args'() )
102   | _ -> ()
103 (*課題7: expr を拡張*)
104 and expr() = arithmexp()
105 and arithmexp() = (arithmterm() ; arithmexp'()) (*右再帰に変更し記述*)
106 and arithmexp'() = match !tok with
107   L.ONE '+' -> (eat(L.ONE '+'); arithmterm(); arithmexp'())
108   | L.ONE '-' -> (eat(L.ONE '-'); arithmterm(); arithmexp'())
109   | _ -> ()
110 and arithmterm() = (arithmfactor(); arithmterm'())
111 and arithmterm'() = match !tok with
112   L.ONE '*' -> (eat(L.ONE '*'); arithmfactor(); arithmterm'())
113   | L.ONE '/' -> (eat(L.ONE '/'); arithmfactor(); arithmterm'())
114   | _ -> ()
115 and arithmfactor() = match !tok with
116   | L.ONE '(' -> (eat(L.ONE '('); arithmexp (); eat(L.ONE ')'))
117   | L.ONE '-' -> (eat(L.ONE '-'); arithmexp () )
118   | L.ONE '[' -> (eat(L.ONE '['); list(); eat(L.ONE ')'))
119   | L.CID s -> (eat(L.CID ""); tail_opt() )
120   | L.VID s -> (eat(L.VID ""))
121   | L.NUM n -> (eat (L.NUM ""))
122   | _ -> error()
123 and tail_opt() = match !tok with
124   L.ONE '(' -> (eat(L.ONE '('); args(); eat(L.ONE ')')) )
125   | _ -> ()
126 and list() = match !tok with
127   L.ONE ']' -> ()
128   | _ -> (expr(); list_opt())
129 and list_opt() = match !tok with
130   L.ONE '|' -> (eat(L.ONE '|'); id())
131   | L.ONE ',' -> (eat(L.ONE ','); list())

```

```

132 | _ -> ()
133 and id() = match !tok with
134   L.CID s -> (eat(L.CID ""))
135   | L.VID s -> (eat(L.VID ""))
136   | L.NUM n -> (eat(L.NUM ""))
137   | _ -> error ()
138end

137 let rec run() =
138   print_string "?- ";
139   while true do
140     flush stdout; Lexer._ISTREAM := stdin;
141     Parser.advance(); Parser.command(); print_string "\n?- "
142   done

```

算術式を可能としたプログラムでの事項結果は以下のようになる。また、課題5で複数の述語を表せるようにしたので複数の算術式も表せている。

```

var run : unit -> unit = fun () ->
# open Parser;;
# run'();;
?- calculation(1 + 1).
CID(calculation)ONE(())NUM(1)ONE(+)NUM(1)ONE(())ONE(.)
?- calculation(2 * 5).
CID(calculation)ONE(())NUM(2)ONE(*)NUM(5)ONE(())ONE(.)
?- callculation(7 - 6) , calculation(4 + 9).
CID(callculation)ONE(())NUM(7)ONE(-)NUM(6)ONE(())ONE(,)CID(calculation)ONE(())NUM(4)ONE(+)NUM(9)ONE(())ONE(.)
?-

```

図6 算術式を加えた実行結果

2.8 問題 8

問題 8 は変数への代入を記述する。変更部分は構文解析プログラムの term (91~94 行目) である。以下のプログラムの 94 行目に新たに文法を記述することで変数代入が可能となった。

```
91     and term() = match !tok with
92       L.ONE '(' -> (eat(L.ONE '('); term(); eat(L.ONE ')')) )
93       (*課題 8 : 変数への代入を加入を修正*)
94       | L.VID s -> (eat(L.VID "");eat(L.IS) ;arithmexp())
95       | _ -> (predicate())
```

変数への代入を記述できているのか確かめるための構文は、スライドにあるものを使用した。

実行結果を用いて確認するとき run 関数起動後にキーボード入力で変数への代入式を記述すると構文エラーが発生し、式をファイルに入れて open fail することでエラーはなくなった。これは今回使用している構文の形に当てはまらないからではないかと考えた。変数代入を可能としたプログラムの実行結果は以下のようになる。

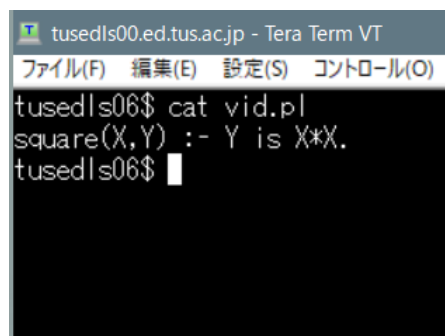


図 7 変数代入を記述したファイルの中身

```

end
module Parser :
sig
  module L = Lexer
  val tok : L.token ref
  val getToken : unit -> L.token
  val advance : unit -> unit
  exception Syntax_error
  val error : unit -> 'a
  val check : L.token -> unit
  val eat : L.token -> unit
  val clauses : unit -> unit
  val clause : unit -> unit
  val to_opt : unit -> unit
  val command : unit -> unit
  val term : unit -> unit
  val terms : unit -> unit
  val terms' : unit -> unit
  val predicate : unit -> unit
  val args : unit -> unit
  val args' : unit -> unit
  val expr : unit -> unit
  val arithmexp : unit -> unit
  val arithmexp' : unit -> unit
  val arithmterm : unit -> unit
  val arithmterm' : unit -> unit
  val arithmfactor : unit -> unit
  val tail_opt : unit -> unit
  val list : unit -> unit
  val list_opt : unit -> unit
  val id : unit -> unit
end
val run : unit -> unit = <fun>
val run' : unit -> unit = <fun>
# open Parser;;
# run'();;
?- open vid.
openCID(vid)ONE(. )CID(square)ONE(( )VID(X)ONE(. )VID(Y)ONE(. ):-VID(Y)isVID(X)ONE(*)VID(X)ONE(. )eof
?-

```

図8 変数の代入の実行結果

最後に問題8まですべての機能が備わったプログラム（test_kadai8.ml）をこのレポートの最後に乗せておく。

(*実験課題*)

```

module Lexer = struct
  type token = CID of string | VID of string | NUM of string
              | TO | IS | QUIT | OPEN | EOF | ONE of char

```

(*課題1*)

```

module P = Printf
exception End_of_system
let count = ref 1 (*課題6: 行数カウント*)
let _ISTREAM = ref stdin
let ch = ref []
let read () = match !ch with [] -> input_char !_ISTREAM
                | h::rest -> (ch := rest; h)
let unread c = ch := c::!ch
let lookahead () = try let c = read () in unread c; c with End_of_file -> '$'
let rec integer i =
  let c = lookahead () in
  if (c >= '0' && c <= '9') then

```

```

        integer (i^(Char.escaped (read ())))
    else i
and identifier id =
    let c = lookahead () in
    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') ||
        (c >= '0' && c <= '9') || c == '_') then
        identifier (id^(Char.escaped (read ())))
    else id
and native_token () =
    let c = lookahead () in
    if (c >= 'a' && c <= 'z') then
        let id = identifier "" in
        match id with
            "is" -> IS
        | "quit" -> QUIT
        | "open" -> OPEN
        | _ -> CID (id)
    else if (c >= 'A' && c <= 'Z') then VID (identifier "")
    else if (c >= '0' && c <= '9') then NUM (integer "")
    else if (c == ':') then let x = read () in
        if (lookahead () == '-') then let y = read () in TO
        else ONE (':')
    else ONE (read ())
and gettoken () =
    try
        let token = native_token () in
        match token with
            ONE ' ' -> gettoken ()
        | ONE '\t' -> gettoken ()
        | ONE '\n' -> count := !count + 1 ; gettoken () (*課題6: 行数カウント*)
        | _ -> token
    with End_of_file -> EOF

let print_token tk =
    match tk with
        (CID i) -> P.printf "CID(%s)" i
    | (VID i) -> P.printf "VID(%s)" i
    | (NUM i) -> P.printf "NUM(%s)" i
    | (TO) -> P.printf ":-"
    | (QUIT) -> P.printf "quit"

```

```

| (OPEN) -> P.printf "open"
| (IS) -> P.printf "is"
| (EOF) -> P.printf "eof"
| (ONE c) -> P.printf "ONE(%c)" c

end

```

(*課題3*)

```

module Parser = struct
  module L = Lexer
  (* module E = Evaluator *)
  let tok = ref (L.ONE ' ')
  let getToken () = L.gettoken ()
  let advance () = (tok := getToken(); L.print_token (!tok))
  exception Syntax_error
  let error () = raise Syntax_error
  let check t = match !tok with
    L.CID _-> if (t = (L.CID "")) then () else error()
  | L.VID _-> if (t = (L.VID "")) then () else error()
  | L.NUM _-> if (t = (L.NUM "")) then () else error()
  | tk -> if (tk=t) then () else error()
  let eat t = (check t; advance())
  let rec clauses() = match !tok with
    L.EOF -> ()
  | _ -> (clause(); clauses())
  and clause() = match !tok with
    L.ONE '(' -> (term(); eat(L.ONE '.'))
  | _ -> (predicate(); to_opt(); eat(L.ONE '.'))
  and to_opt() = match !tok with
    L.TO -> (eat(L.TO); terms())
  | _ -> ()
  and command() = match !tok with
    L.QUIT -> exit 0
  | L.OPEN -> (eat(L.OPEN);
    match !tok with
      L.CID s -> (eat(L.CID ""); check (L.ONE '.'));
      L._ISTREAM := open_in (s^".pl");
      advance(); clauses(); close_in (!L._ISTREAM))
    | _ -> error())
  | _ -> (terms(); check(L.ONE '.'))

```

(*課題5 term -> terms に修正*)

```
and term() = match !tok with
  L.ONE '(' -> (eat(L.ONE '('); term(); eat(L.ONE ')')) )
```

(*課題8: term 部分の変更*)

```
  | L.VID s -> (eat(L.VID ""); eat(L.IS); arithmexp())
  | _ -> (predicate())
and terms() = (term(); terms'())
and terms'() = match !tok with
  L.ONE ',' -> (eat(L.ONE ','); term(); terms'() )
  | _ -> ()
and predicate() = (eat(L.CID ""); eat(L.ONE '('); args(); eat(L.ONE ')')) )
and args() = (expr(); args'())
and args'() = match !tok with
  L.ONE ',' -> (eat(L.ONE ','); expr(); args'() )
  | _ -> ()
```

(*課題7: 算術計算できるように修正*)

```
and expr() = arithmexp()
and arithmexp() = (arithmterm() ; arithmexp'())
and arithmexp'() = match !tok with
  L.ONE '+' -> (eat(L.ONE '+'); arithmterm(); arithmexp'())
  | L.ONE '-' -> (eat(L.ONE '-'); arithmterm(); arithmexp'())
  | _ -> ()
and arithmterm() = (arithmfactor(); arithmterm'())
and arithmterm'() = match !tok with
  L.ONE '*' -> (eat(L.ONE '*'); arithmfactor(); arithmterm'())
  | L.ONE '/' -> (eat(L.ONE '/'); arithmfactor(); arithmterm'())
  | _ -> ()
and arithmfactor() = match !tok with
  | L.ONE '(' -> (eat(L.ONE '('); arithmexp (); eat(L.ONE ')'))
  | L.ONE '-' -> (eat(L.ONE '-'); arithmexp (); )
  | L.ONE '[' -> (eat(L.ONE '['); list(); eat(L.ONE ']'))
  | L.CID s -> (eat(L.CID ""); tail_opt() )
  | L.VID s -> (eat(L.VID ""))
  | L.NUM n -> (eat (L.NUM ""))
  | _ -> error()
and tail_opt() = match !tok with
  L.ONE '(' -> (eat(L.ONE '('); args(); eat(L.ONE ')')) )
  | _ -> ()
and list() = match !tok with
  L.ONE ']' -> ()
```

```

    | _ -> (expr(); list_opt())
and list_opt() = match !tok with
    L.ONE '|' -> (eat(L.ONE '|'); id())
  | L.ONE ',' -> (eat(L.ONE ','); list())
  | _ -> ()
and id() = match !tok with
    L.CID s -> (eat(L.CID ""))
  | L.VID s -> (eat(L.VID ""))
  | L.NUM n -> (eat(L.NUM ""))
  | _ -> error ()
end

let rec run() =
  print_string "?- ";
  while true do
    flush stdout; Lexer._ISTREAM := stdin;
    Parser.advance(); Parser.command(); print_string "\n?- "
  done

let run' () = (*課題6エラー出力できるように run' () を追加*)
  try let c = run () in c
  with Parser.Syntax_error -> print_string "\n"; print_string "文法エラー:" ; Printf.printf ("%d") !
  行目"; print_string "\n?-"

```