

프로그래밍 언어 개론

과제 보고서

[PL00]HW5_201601980_KimSeongHan

개발환경 : Ubuntu

제출일 (2021-04-07)

201601980 / 김성한(00분반)

<과제 설명>

Homework

실습을 확장하여 정수와 변수를 인식하는 어휘분석기를 만든다.
+ 정수 및 변수를 인식하는 FA를 그래프형태로 보고서에 첨부하시오.

```
(* 정수 = -[0-9]+ | [0-9]+, 변수 = [a-z][a-zA-Z0-9'_]* *)
type t =
| Int of int
| Var of string

let char_to_int c = (* convert a character to an integer : use the given function *)
let char_to_str c = (* convert a character to a string : use the given function *)

let lex chars = (* write your code *)
(* lex : char list -> t *)

(* example *)
let a = lex['0','1'] (* Int 1 *)
let b = lex['-','7','9'] (* Int -79 *)
let c = lex['a','1','T'] (* Var a1T *)
let d = lex['x','_','\'] (* Var x\_ *) (* \ is an escape character for ' *)
let e = lex['V','a','1'] (* exception : failwith "Not a valid integer or a valid variable" *)
    프로그래밍언어개론
```

[문제 HomeWork]

● 이번 과제는 주어진 정규 표현식을 Finite Automata(유한 오토마타)로 표현하고 해당 오토마타를 직접 Ocaml로 구현해보는 과제이다.

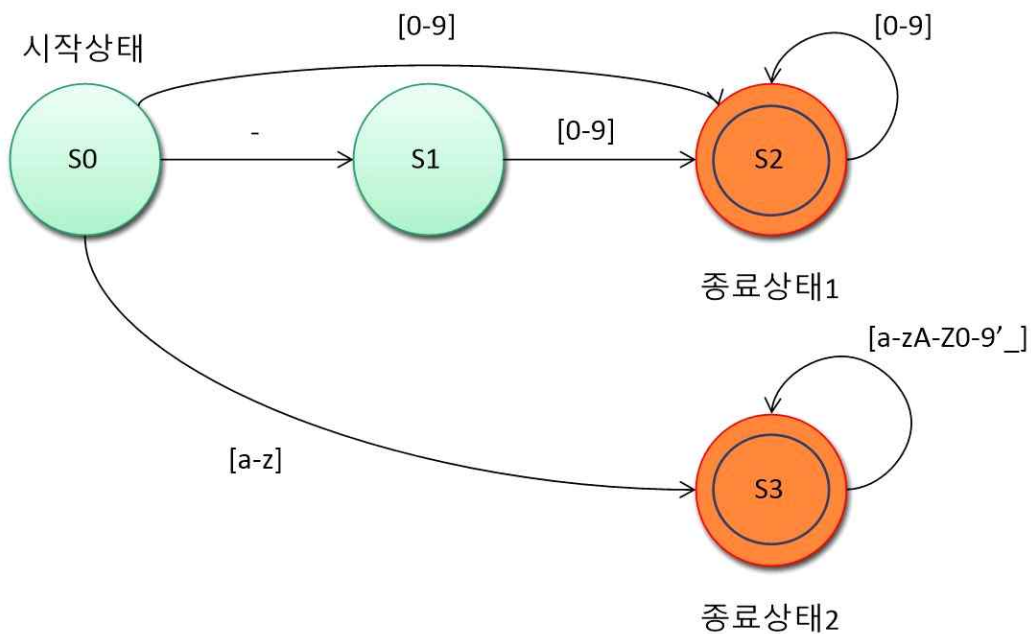
● 입력으로 주어질 문자열들의 정규표현식은

(정수 = $-[0-9]^+ \mid [0-9]^+$, 변수 = $[a-z][a-zA-Z0-9'_]^*$) 로 표현된다.

● 즉, 0~9까지의 수들 중 최소 한자리 이상의 숫자로 이루어진 정수 및 소문자로 시작하면서 a~z, A~Z, 0~9, '(작은따옴표), _(언더바)를 포함하는 최소 한자리 이상인 문자열을 입력받게 된다. 이를 바탕으로 먼저 FA를 그려보았고 해당 FA를 이용하여 함수구상을 시작하였다.

<Finite Automata - 유한 오토마타>

201601980 김성한



[homework의 정규표현식을 표현한 유한 오토마타]

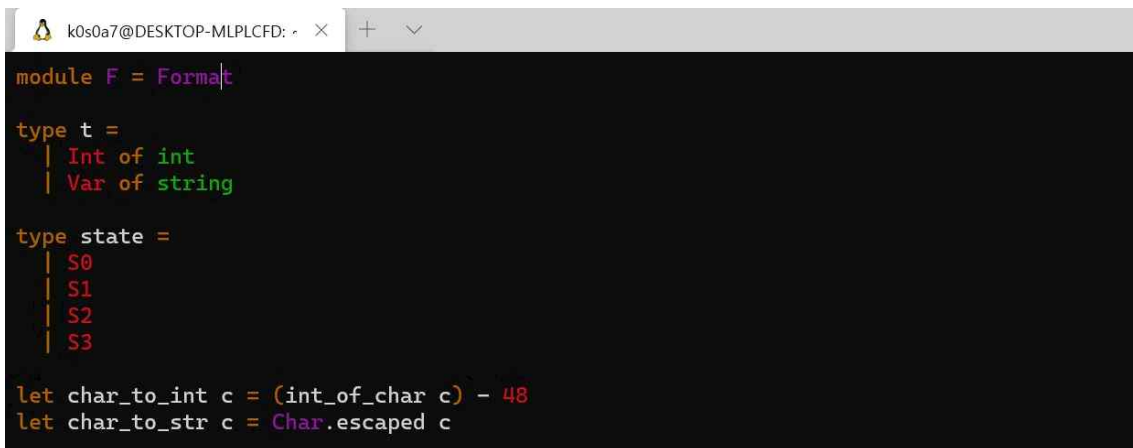
● 입력될 문자열의 표현식

(정수 = $-[0-9]^+ \mid [0-9]^+$, 변수 = $[a-z][a-zA-Z0-9'_-]^*$) 을

위의 사진처럼 유한 오토마타로 표현하였다. 총 4개의 state가 필요하며 그 중 S2와 S3는 종료 상태로서 각각의 조건에 맞는 S2는 정수형, S3는 문자열을 받으면 해당 상태에서 종료를 하게 된다.

● 이를 기반으로 함수를 구상할 때 state간의 움직임을 확인하기 쉬웠으며, 어떤 조건이 필요한지도 깔끔하게 정리되어 이번 과제 구현에 도움을 준 것 같다.

<코드 구현 및 실행 결과 화면>



```
module F = Format

type t =
  | Int of int
  | Var of string

type state =
  | S0
  | S1
  | S2
  | S3

let char_to_int c = (int_of_char c) - 48
let char_to_str c = Char.escaped c
```

[lexer.ml의 초반부]

- lexer.ml에는 초기 Format 모듈과 데이터 타입들이 선언되어있다. 이때 state는 4개로 앞선 내용에 포함한 FA의 state개수와 같다.
- 또한 두가지 메소드를 정의하였는데 char_to_int 메소드는 인자로 들어온 character형 c의 아스키코드 값에서 48을 빼줌으로써 int로 변환을 한다.
- char_to_str 메소드는 Char의 메소드인 escaped를 이용하여 해당 인자를 문자열로 바꿔준다.

```

(* lex : char list -> t *)
let lex chars = (* write your code *)
  let rec lex_impl state chars v =
    match state with
    | S0 ->
      begin
        match chars with
        | h::t ->
          begin
            match h with
            | '-' ->
              begin
                match lex_impl S1 t v with
                | Int v -> Int(-v)
                | _ -> failwith "Not a valid integer or a valid variable"
              end
            | '0' .. '9' -> lex_impl S2 t (v @ [h])
            | 'a' .. 'z' -> lex_impl S3 t (v @ [h])
            | _ -> failwith "Not a valid integer or a valid variable"
          end
        | [] -> failwith "Not a valid integer or a valid variable"
      end
  end

```

[lex 함수의 시작 및 S0 pattern matching]

- lex 함수는 초기 char형 list인 chars를 인자로 받는다.
- 내부구현을 위해 let rec lex_impl state chars v 형태의 내부 재귀함수를 선언하였다.
- 내부함수 impl의 타입은 주어진 자료에서 변형해도 된다고 하여
state -> charlist -> charlist -> t 로 구상을 하였다.
- 전체적인 내부 동작은 상태와 chars 및 빈 리스트를 인자로 가지고 시작하여 마지막에 t형 결과를 반환하는 함수이다.
- pattern matching을 살펴보면 초기 state가 S0인 경우 chars로부터 원소 하나를 떼어내어 다시 matching 한다. 이때 h가 '-' 인 경우에는 재귀함수 결과(이때 재귀함수의 상태는 S1으로 향한다.)를 matching 하여 Int 타입이라면 -v를 반환하고, 만일 아니라면 failwith으로 exception 처리를 해준다.
- h가 0 ~ 9 사이의 숫자형 문자라면 S2로 상태를 이동하고 v에는 h를 추가해준다.
- h가 a ~ z 사이의 알파벳형 문자라면 S3로 상태를 이동하고 v에는 h를 추가해준다.
- 만일 어디에도 해당되지 않는다면 exception 처리를 해주고 h에 대한 matching을 종료한다.
- state 즉, S0에서 matching할 문자열이 없는 빈 리스트의 경우라면 해당 정규표현식에는 맞지 않으므로 chars의 리스트가 비어있는 경우의 matching으로 예외처리를 해주었다.

```

|S1 ->
    begin
        match chars with
        |h::t ->
            begin
                match h with
                |'0' .. '9' -> lex_impl S2 t (v @ [h;])
                | _ -> failwith "Not a valid integer or a valid variable"
            end
        |[] -> failwith "Not a valid integer or a valid variable"
    end

```

[lex 함수의 S1 pattern matching]

- S1의 상태는 초기 S0에서 pattern matching시에 맨앞에 '-' 부호가 있는경우이다. 만일 해당 chars에 남아있는 것이 0 ~ 9 사이의 숫자형 문자라면 S2로 상태를 옮겨준다. 이때 h는 v에 저장한다.

- 하지만 0 ~ 9 사이의 문자가 아니라면 예외처리를 해준다. 또한 chars가 빈리스트인 경우 failwith을 이용하여 예외처리를 해준다.

```

|S2 ->
  begin
    match chars with
    |h::t ->
      begin
        match h with
        |'0' .. '9' -> lex_impl S2 t (v @ [h;])
        | _ -> failwith "Not a valid integer or a valid variable"
      end
    |[] ->
      begin
        let rec for_Int charlist k =
          match charlist with
          | h::t -> for_Int t (10 * k + (char_to_int h))
          | [] -> k
        in
          Int (for_Int v 0)
        end
      end
  end

```

[lex 함수의 S2 pattern matching]

- S2는 종료조건이다. 타입은 정수형 문자열의 종료 조건이며 숫자를 리턴해줘야 한다.
- chars에서 h를 떼어 비교하는데 0~9 인 경우 S2의 상태에서 재귀를 반복한다. 만일 0~9사이의 숫자가 아니라면 에러를 발생시킨다. 이렇게 재귀를 반복하면 숫자들이 v 에 쌓이게 된다.
- 만일 chars 가 빈 리스트인 경우라면 여기서는 해당 문자열을 Int형으로 반환해야한다. 그 이유는 아래의 사진을 통해서 알 수 있다.

```

match lex_impl S1 t v with
|Int v -> Int(-v)
| _ -> failwith "Not a valid integer or a valid variable"

```

[lex 함수의 S0에서 첫 문자가 '-'인 경우]

- 초기 S0 상태에서 음수인 경우 S1으로 옮겨가고 S1에서 S2로 상태가 옮겨가게 된다. 이때 반환되는 Int 형 결과를 받아 음수로 바꿔주는 부분을 위해서이다.
- 다시 S2상태의 빈리스트인 pattern을 보면 재귀함수를 하나 더 선언한 것을 볼 수 있다. 또한 재귀문의 stack 오버플로우를 염두해 tail-recursive하게 작성하였다.
- 해당 for_Int 재귀문의 타입은 charlist -> int -> int 이다. 초기 charlist를 pattern matching 하여 원소가 존재하면 하나씩 1에 자리에 더해주고 기존에 저장된 값들은 10씩 곱하여 왼쪽으로 쉬프트를 진행한다(10진수 기준). charlist에 더 이상 원소가 없다면 int 형 k를 반환하고, 이후 Int(for_Int v 0)를 선언하여 해당 함수를 선언함과 동시에 결과값을 Int형을 형변환을 해준다. 이렇게 되면 S0에 -를 추가해주는 부분에서 pattern matching을 진행할 수 있다.

```

|S3 ->
begin
  match chars with
  |h::t ->
    begin
      match h with
      |'a' .. 'z' -> lex_impl S3 t (v @ [h;])
      |'A' .. 'Z' -> lex_impl S3 t (v @ [h;])
      |'0' .. '9' -> lex_impl S3 t (v @ [h;])
      |'\'' -> lex_impl S3 t (v @ [h;])
      |'_' -> lex_impl S3 t (v @ [h;])
      |_ -> failwith "Not a valid integer or a valid variable"
    end
  |[] ->
    begin
      let rec for_Var charlist k =
        match charlist with
        |h::t -> for_Var t (k ^ (char_to_str h))
        |[] -> k
      in
      Var (for_Var v "")
    end
end

```

[lex 함수의 S3 pattern matching]

- S3상태도 S2와 마찬가지로 종료조건이다 . 하지만 S3는 S2와 다르게 Var 타입의 결과를 반환 해야한다.

- 초기 chars의 원소 h를 확인하여 a ~ z, A ~ Z, 0 ~ 9, \', _를 포함하면 S3상태에서 재귀를 반복할 수 있지만 해당 문자들에 포함이 안된다면 에러를 발생시켜준다.

- S3상태에서 chars에 더 이상 원소가 없을 때는 S2에서와 마찬가지로 v (charlist)에 쌓인 문자들을 문자열로 합쳐줘야한다. 이를 위해 재귀함수 for_Var을 선언하였으며 타입은 charlist -> string 이 된다. for_Var 함수도 tail-recursive하게 구현 하였으며 초기 charlist에 대하여 pattern matching을 진행하고 원소가 있따면 ^(concat) 연산자를 통해 원소인 h 와 그전에 쌓인 문자열 k를 붙여준다.(k는 초기상태가 비어있는 문자열이다.)
^ 연산은 두 string에 대한 연산이기 때문에 한글자씩 char_to_str 메소드를 이용해 문자를 문자열로 바꿔주었다.

- 그 후 charlist 가 비워질 때까지 반복하여 만일 리스트가 빈다면 문자열 k를 반환한다.

- 함수 작성을 마친 뒤 in을 선언 후 Var (for_Var v "")를 선언하였다. 이렇게 되면 리스트 v에 쌓여있던 문자열들이 string 형태로 반환되고 Var()를 통해 Var 형으로 변환된다.


```

    in
    lex_impl S0 chars []

let pp fmt v =
  match v with
  | Int i -> F.fprintf fmt "Int %d" i
  | Var x -> F.fprintf fmt "Var %s" x

```

[lexer.ml의 후반부]

- 그렇게 lex_impl에 대한 작성을 마친 후 래퍼함수에서의 호출을 위해 in을 써준뒤 그 밑에서 lex_impl 함수를 호출한다. 이때 초기 상태는 S0 이며 lex 의 인자로 들어온 chars를 넣어주고 v에는 [] (빈 리스트)를 할당한다.
- pp 함수는 formatter와 v를 인자로 하여 v의 타입이 Int인지 Var인지 확인하여 해당하는 문장을 출력해주는 함수이다.

```

module F = Format

(* Test cases *)
let _ =
  let a = Lexer.lex ['1'] in (* a = Int 1 *)
  let b = Lexer.lex ['5'; '7'; '9'] in (* b = Int 579 *)
  let c = Lexer.lex ['0'; '1'; '4'] in (* c = Int 14 *)
  let d = Lexer.lex ['-'; '4'; '3'] in (* d = Int -43 *)
  let e = Lexer.lex ['v'; 'a'; 'r'] in (* e = Var var *)
  let f = Lexer.lex ['x'; '_'; 'z'] in (* f = Var x_z *)
  let g = Lexer.lex ['t'; 'M'; '\\'] in (* g = Var tM\\' *)
  let h = Lexer.lex ['x'; '0'; 'Y'; '_'; '\\'] in (* h = x0Y_\\' *)
  let _ = F.printf "a = %a\n" Lexer.pp a in
  let _ = F.printf "b = %a\n" Lexer.pp b in
  let _ = F.printf "c = %a\n" Lexer.pp c in
  let _ = F.printf "d = %a\n" Lexer.pp d in
  let _ = F.printf "e = %a\n" Lexer.pp e in
  let _ = F.printf "f = %a\n" Lexer.pp f in
  let _ = F.printf "g = %a\n" Lexer.pp g in
  let _ = F.printf "h = %a\n" Lexer.pp h in
  try
    let _ = Lexer.lex ['V'; 'a'; 'r'] in ()
  with Failure e ->
    F.printf "Exception occurs : %s\n" e (* Exception occurs : Not a valid integer or a valid variable*)

```

[main.ml 의 내부코드]

```

k0s0a7@DESKTOP-MLPLCFD:~/week5/hw$ dune exec ./main.exe
a = Int 1
b = Int 579
c = Int 14
d = Int -43
e = Var var
f = Var x_z
g = Var tM\\'
h = Var x0Y_\\'
Exception occurs : Not a valid integer or a valid variable
k0s0a7@DESKTOP-MLPLCFD:~/week5/hw$

```

[main.exe 실행 결과]

- 과제와 함께 주어진 testcase가 포함된 main.ml의 코드와 main.exe를 실행하였을 때 결과이다.
- 해당하는 문자들에 대하여 올바른 Int형 정수와 Var형 string을 반환하고 있다.
- 또한 마지막에는 try-with 구문을 사용하여 exception을 출력해주는 모습을 확인할 수 있었다.

<과제 구현 중 어려웠던 부분 및 해결 과정>

```
k0s0a7@DESKTOP-MLPLCFD: ~$  
module F = Format  
  
(* Test cases *)  
let _ =  
  let a = Lexer.lex ['1'] in (* a = Int 1 *)  
  let b = Lexer.lex ['5';'7';'9'] in (* b = Int 579 *)  
  let c = Lexer.lex ['0';'1';'4'] in (* c = Int 14 *)  
  let d = Lexer.lex ['-';'4';'3'] in (* d = Int -43 *)  
  let e = Lexer.lex ['v';'a';'r'] in (* e = Var var *)  
  let f = Lexer.lex ['x';'_' ;'z'] in (* f = Var x_z *)  
  let g = Lexer.lex ['t';'M';'\'] in (* g = Var tM\ *)  
  let h = Lexer.lex ['x';'0';'Y';'_' ;'\'] in (* h = x0Y_\ *)  
  let _ = F.printf "a = %a\n" Lexer.pp a in  
  let _ = F.printf "b = %a\n" Lexer.pp b in  
  let _ = F.printf "c = %a\n" Lexer.pp c in  
  let _ = F.printf "d = %a\n" Lexer.pp d in  
  let _ = F.printf "e = %a\n" Lexer.pp e in  
  let _ = F.printf "f = %a\n" Lexer.pp f in  
  let _ = F.printf "g = %a\n" Lexer.pp g in  
  let _ = F.printf "h = %a\n" Lexer.pp h in  
  try  
    let _ = Lexer.lex ['V';'a';'r'] in ()  
  with Failure e ->  
    F.printf "Exception occurs : %s\n" e (* Exception occurs : Not a valid integer or a valid variable*)
```

[main.ml 의 내부코드]

```
k0s0a7@DESKTOP-MLPLCFD:~/week5/hw$ dune exec ./main.exe  
Fatal error: exception Failure("Not a valid integer or a valid variable")  
k0s0a7@DESKTOP-MLPLCFD:~/week5/hw$ vi main.ml
```

[main.exe 실행 결과 - Fatal error]

● 이번 과제에선 로직 자체를 구현하는 데에는 크게 문제가 없었다. 다만 구현 중 Fatal error가 결과로 출력되었는데, 본인은 해당 error가 failwith으로 예외처리가 잘 안되었다는 의미로 받아들여 애를 먹었다. 검색도 해보았지만 잘나오지 않아 질문을 하였는데 확인해보니 정말 스스로가 무지했다는 것을 깨달았다.

● 결론적으로 해당 error는 failwith을 통해 출력되는 error 문구이다. 이전과제에서도 여러번 작성한 적이 있는데 왜 인지하지 못하고 있었을까.. 많이 반성하게 되었다. 이에 대한 내용을 듣고 올바른 testcase에서 failwith으로 처리되는 부분이 있구나 하면서 살펴본 결과 S3상태에서 pattern matching 부분에 오류를 확인한 후 고쳐줬다. 이후 과제를 완성했다.

● 앞으로 기억하고 다시는 같은 실수를 반복하면 안되겠다.

<최종 제출 코드 및 결과 화면>

```
module F = Format
type t =
  | Int of int
  | Var of string
type state =
  | S0
  | S1
  | S2
  | S3
let char_to_int c = (int_of_char c) - 48
let char_to_str c = Char.escaped c
(* lex : char list -> t *)
let lex chars = (* write your code *)
  let rec lex_impl state chars v =
    match state with
    | S0 ->
      begin
        match chars with
        | h::t ->
          begin
            match h with
            | '-' ->
              begin
                match lex_impl S1 t v with
                | Int v -> Int(-v)
                | _ -> failwith "Not a valid integer or a valid variable"
              end
            | '0' .. '9' -> lex_impl S2 t (v @ [h])
            | 'a' .. 'z' -> lex_impl S3 t (v @ [h])
            | _ -> failwith "Not a valid integer or a valid variable"
          end
        | [] -> failwith "Not a valid integer or a valid variable"
      end
    | S1 ->
      begin
        match chars with
        | h::t ->
          begin
            match h with
            | '0' .. '9' -> lex_impl S2 t (v @ [h])
            | _ -> failwith "Not a valid integer or a valid variable"
          end
        | [] -> failwith "Not a valid integer or a valid variable"
      end
    | S2 ->
      begin
        match chars with
        | h::t ->
          begin
            match h with
            | '0' .. '9' -> lex_impl S2 t (v @ [h])
            | _ -> failwith "Not a valid integer or a valid variable"
          end
        | [] ->
          begin
            let rec for_Int charlist k =
              match charlist with
              | h::t -> for_Int t (10 * k + (char_to_int h))
              | [] -> k
            in
            Int (for_Int v 0)
          end
      end
    | S3 ->
      begin
        match chars with
        | h::t ->
          begin
            match h with
            | 'a' .. 'z' -> lex_impl S3 t (v @ [h])
            | 'A' .. 'Z' -> lex_impl S3 t (v @ [h])
            | '0' .. '9' -> lex_impl S3 t (v @ [h])
            | '[' -> lex_impl S2 t (v @ [h])
            | '.' -> lex_impl S3 t (v @ [h])
            | _ -> failwith "Not a valid integer or a valid variable"
          end
        | [] ->
          begin
            let rec for_Var charlist k =
              match charlist with
              | h::t -> for_Var t (k ^ (char_to_str h))
              | [] -> k
            in
            Var (for_Var v "")
          end
      end
    end
  in
  lex_impl S0 chars []
let pp fmt v =
  match v with
  | Int i -> F.printf fmt "Int %d" i
  | Var x -> F.printf fmt "Var %s" x
```

98,38 Bot

[전체 lexer.ml 코드의 모습]

```

module F = Format

(* Test cases *)
let _ =
  let a = Lexer.lex ['1'] in (* a = Int 1 *)
  let b = Lexer.lex ['5'; '7'; '9'] in (* b = Int 579 *)
  let c = Lexer.lex ['0'; '1'; '4'] in (* c = Int 14 *)
  let d = Lexer.lex ['-'; '4'; '3'] in (* d = Int -43 *)
  let e = Lexer.lex ['v'; 'a'; 'r'] in (* e = Var var *)
  let f = Lexer.lex ['x'; '_'; 'z'] in (* f = Var x_z *)
  let g = Lexer.lex ['t'; 'M'; '\\'] in (* g = Var tM\\' *)
  let h = Lexer.lex ['x'; '0'; 'Y'; '_'; '\\'] in (* h = x0Y_\\' *)
  let _ = F.printf "a = %a\n" Lexer.pp a in
  let _ = F.printf "b = %a\n" Lexer.pp b in
  let _ = F.printf "c = %a\n" Lexer.pp c in
  let _ = F.printf "d = %a\n" Lexer.pp d in
  let _ = F.printf "e = %a\n" Lexer.pp e in
  let _ = F.printf "f = %a\n" Lexer.pp f in
  let _ = F.printf "g = %a\n" Lexer.pp g in
  let _ = F.printf "h = %a\n" Lexer.pp h in
  try
    let _ = Lexer.lex ['V'; 'a'; 'r'] in ()
  with Failure e ->
    F.printf "Exception occurs : %s\n" e (* Exception occurs : Not a valid integer or a valid variable*)

```

[전체 main.ml 코드의 모습]

```

k0s0a7@DESKTOP-MLPLCFD:~/week5/hw$ dune exec ./main.exe
a = Int 1
b = Int 579
c = Int 14
d = Int -43
e = Var var
f = Var x_z
g = Var tM\\'
h = Var x0Y_\\'
Exception occurs : Not a valid integer or a valid variable
k0s0a7@DESKTOP-MLPLCFD:~/week5/hw$

```

[main.exe 실행 결과]

- 과제로 제출할 lexer.ml의 전체모습과 main.ml 그리고 main.exe의 실행결과이다.
- lexer.ml은 내부 lex 함수에 의해 정규표현식에 맞게 들어온 문자열을 Int 혹은 Var 형의 타입으로 변환하여 출력해주는 기능을 담당한다.
- main문의 testcase를 맞게 출력하고 있으며 예외처리도 잘 수행됨을 확인할 수 있었다.

<새로 고찰한 내용 및 느낀점>

● 이번 과제에서 failwith 과 관련하여 착각을 하면 계속해서 그곳에 몰두하여 원하는 결과를 찾지 못한 스스로를 보면서 다시는 잊지 않는 것도 중요하지만 한번 숨을 고르고 차근차근 다시 한번 보는 습관이 필요하다는 것을 깨달았다. 가장 결정적으로는 Fatal error가 failwith의 결과인지 모르고 있었다는 점에서 조금 충격을 받았다. 답변메일을 받고 꽤 가만히 앉아 생각했던 것 같다. 생각해보면 이전 과제에서도 failwith을 종종 이용하였는데 이번 과제에서 잘못 알고 있었던 지식을 확실하게 배우게 된 것 같다. 코드를 조금 더 침착하게 해석하고 성급하지 않도록 조심해야겠다는 생각이 들었다. 조금은 더 분발해야겠다.