

Principles of Programming, Fall 2011

Practice 8

String, Mutual recursion, Exception catch

Programming Research Lab, @SNU

Seungjung Lee, Youngseok Lee

November 14, 2011

The Objectives of this practice:

- Learn to print basic types of Ocaml in screen.
- Learn to handle a string.
- Learn to handle an exception.

1. Print basic types in screen.

In Ocaml, a prefix of functions which print a string in screen is `print_`.

```
let a =  
print_string "print string."  
print_newline()  
print_int 4;  
7
```

Let's compile the above code and execute it. `print_newline` has no arguments. But in order to represent a function execution, you must attach `()` which implies unit. Three functions that start with `print_` return unit type. `;` executes instructions which return unit in turn and returns a value which is a result of the last instruction. So a value of `a` would be 7.

2. Handling string

`^` concatenates two strings. The following string is "`s = 4` means that s equals 4".

```
"s = " ^ (string_of_int 4) ^ " means that s equals 4"
```

3. Exception catch

In case of an exception, not terminate but catch the exception and execute relevant instructions. Compile the following code and input `b` as 0.

```

let s = try
string_of_int(
let a=read_int() in
let b=read_int() in
a/b)
with Division_by_zero -> "error"
let _ = print_string (s^"\n")

```

Exercise

1. Let's make a function `string_of_intlist` that changes an int list to a string.

`string_of_intlist : int list → string`

This function takes an integer list and outputs a string like the folloing format.

(1, 3, 7)

Using pattern matching and `^`, you can easily make this function.

```

match ilist with
| [] -> ...      (* If there is no element *)
| [hd] -> ...    (* If there is only one element *)
| hd::tl -> ...  (* If there are two or more elements *)

```

2. Mutual recursive function

Mutual recursion is a form of recursion where two functions are defined in terms of each other. At then, you have to use `and` instead of `let rec` while defining second function. Define two functions which are `isEven` and `is Odd`. Do not mod procedure and make as mutual recursive function.

```

let rec isEven num =
...
and isOdd num =
...

let _ =
print_string(string_of_bool (isEven 3)) (* print false in screen *)

```

3. Simple caculator

Make a function `calc` which takes the following type `exp` and returns string including the relevent expression and the answer.

`calc : exp → string`

```
type exp = NUM of int
|ADD of exp * exp
|MUL of exp * exp
|DIV of exp * exp
```

In case of division by 0, catch the exception and return “Div by zero”.

```
# let _ = print_string (calc (ADD ((NUM 1),(MUL ((NUM 1), (NUM 4))))))
(1 + (1 * 4)) = 5
# let _ = print_string (calc (DIV ((NUM 2),(NUM 0))))
Div by zero
```