상명대학교 컴퓨터과학과

"EA0011: 프로그래밍 언어론" 숙제 4

제출일: 2017년 10월 30일 수업 전

언어 Expression이 있다(아래 설명 1). Scheme 언어를 사용하여 언어 Expression의 abstract syntax를 표현하는 datatype인 expression을 정의하고(아래 설명 2.1), datatype을 다루는 8개의 함수(아래 설명 2.2-2.4)를 프로그램하라. 숙제 제출물은 ① 연필로 쓴 프로그램 및 프로그램에 대한 설명, ② 작성한 프로그램 파일을 시험 프로그램 파일인 "/home/pl/hw04/test.scm"로 수행한 화면(아래 설명 3) 및 ③ 프로그래밍 서버에 저장한 프로그램 파일 "~/pl/hw04/datatype.scm"이다.

설명 1: 언어 Expression의 syntax
Expression ::= Number const-exp (num) ::= Expression+Expression add-exp (exp1 exp2) ::= Expression*Expression multi-exp (exp1 exp2) ::= (Expression) par-exp (exp1)

설명 2: 작성할 프로그램 파일 datatype.scm
1. data type expression (2점) 2. predicate 함수: const-exp?, add-exp?, multi-exp? par-exp? (8점) 3. extractor 함수: exp->num, exp->exp1, exp->exp2 (6점)

4. abstract syntax로 표현된 expression을 concrete syntax 표현된 expression으로 바꾸는 함수: exp->string (8점)

참고: 프로그램 파일 작성시 template 파일인 /home/pl/hw04/datatype.scm을 먼저 숙제 디렉토리에 복사한 후 이 파일에 추가하여 프로그램함.

설명 3. 시험 프로그램 파일 "/home/pl/hw04/test.scm"을 수행하는 과정.

```
$ mzscheme -M eopl -f datatype.scm -f /home/pl/hw04/test.scm
Welcome to MzScheme v372 [3m], Copyright (c) 2004-2007 PLT Scheme Inc
> (test)
========================================
1. Constructor test
#(struct:const-exp 7)
#(struct:const-exp 13)
#(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13))
#(struct:par-exp
  #(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13)))
#(struct:multi-exp
  #(struct:const-exp 13)
  #(struct:par-exp
    #(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13))))
#(struct:add-exp
  #(struct:par-exp
    #(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13)))
  #(struct:multi-exp
    #(struct:const-exp 13)
    #(struct:par-exp
      #(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13)))))
#(struct:multi-exp
  #(struct:multi-exp
    #(struct:const-exp 13)
    #(struct:par-exp
      #(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13))))
  #(struct:add-exp
    #(struct:par-exp
      #(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13)))
```

```
      #(struct:multi-exp
        #(struct:const-exp 13)
        #(struct:par-exp
          #(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13))))))
#(struct:par-exp
  #(struct:add-exp
    #(struct:par-exp
      #(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13)))
    #(struct:multi-exp
      #(struct:const-exp 13)
      #(struct:par-exp
        #(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13))))))
========================================
2. Predicates test
#t
#f
#t
#f
#f
#t
#f
#f
========================================
3. Extractors test
7
13
#(struct:const-exp 7)
#(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13))
#(struct:const-exp 13)
#f
#(struct:multi-exp
  #(struct:const-exp 13)
  #(struct:par-exp
    #(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13))))
#(struct:add-exp
  #(struct:par-exp
```

```
      #(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13)))
   #(struct:multi-exp
      #(struct:const-exp 13)
      #(struct:par-exp
         #(struct:add-exp #(struct:const-exp 7) #(struct:const-exp 13)))))
========================================
4. Function exp->string test
"7"
"13"
"7+13"
"(7+13)"
"13*(7+13)"
"(7+13)+13*(7+13)"
"13*(7+13)*(7+13)+13*(7+13)"
"((7+13)+13*(7+13))"
> (exit)
$
```

끝.