程式說明文件

此程式為中原大學「程式語言」課程之 Project。

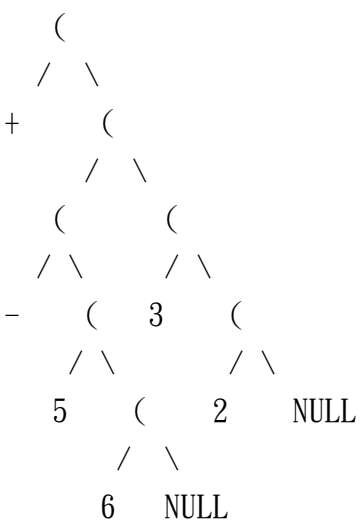
一、Syntax:

```
<S-exp> ::= <ATOM>
          | LEFT-PAREN <S-exp> { <S-exp> } [ DOT <S-exp> ]
            RIGHT-PAREN
          | QUOTE <S-exp>
<ATOM>  ::= SYMBOL | INT | FLOAT | STRING
           | NIL | T | LEFT-PAREN RIGHT-PAREN
```

(S1 S2 S3 S4 . S5)
= (S1 . (S2 . (S3 . (S4 . S5))))
Ex : (list 2 3) = (list .(2 .(3)))

二、樹狀結構

舉例而言,(+ (- 5 6) 3 2) 的樹狀結構為。
(+ (- 5 6) 3 2) = (+ .((- .(5 .(6))) .(3 .(2))))
後者為樹狀結構, “.” 以右子樹代表



碰到 root or 左子節點的”(” 時, 檢查該點的左子樹的指令以及右子樹的 argument 是否有無 error 等等,再進行計算。

三、指令(括號內的數字指的是這個 function 可接受的 argument 的數目):

1. '(1) :功能與 quote 相同, ' 前不需要括弧。
2. quote (1) :quote 之後的 argument 不作計算。

```
input: (quote (3 (4 5)))  or   '(3 (4 5))
output:( 3
         ( 4
           5
         )
       )
```

3. cons (2):將第二個 argument 移至右子樹。

```
input: (cons 3 4)
output:( 3
         .
         4
       )
```

4. define (>=2) :宣告變數, 若配合 lambda 可宣告函式。

```
input: (define x '((3 4) 5)) ;變數
       (define f ( lambda (x) (+ x x c)  ) );函式
```

5. list (>=0):列出 argument。

```
input: (list 3 4)
output:( 3
         4
       )
```

6. car (1):抓取右子樹的左子樹。

```
input: (car '(3 4))
output: 3
```

7. cdr (1):抓取右子樹的右子樹。

```
input: (cdr '((3 4) 5)  )
output:( 5
       )
```

8. atom? (1):檢測 argument 是否為 atom。
   input: (atom? 3)
   output: #t

9. pair? (1):檢測 argument 是否為 pair。
10. list? (1):檢測 argument 是否為 list。
11. null? (1):檢測 argument 是否為 nil。
12. integer? (1):檢測 argument 是否為 integer。
13. real? (1):檢測 argument 是否為實數。
14. number? (1):檢測 argument 是否為數字。
15. string? (1):檢測 argument 是否為 string。
16. boolean? (1):檢測 argument 是否為 boolean 值。
17. symbol? (1):檢測 argument 是否為 symbol。
18. + (>=2):加法計算。
   Input: (+ 3 7 10 25)
   Output : 45

19. - (>=2):減法計算。
20. * (>=2):乘法計算。
21. / (>=2):除法計算。

22. not (1):反相器。
   input: (not #t)
   output: nil

23. and (>=2):and 閘
24. or (>=2):or 閘
25. > (>=2):檢測 argument 之間放入”>”後, 是否正確。
   Input: (> 3 2)
   Output: #t

26. < (>=2):檢測 argument 之間放入”<”後, 是否正確
27. >= (>=2):檢測 argument 之間放入”>=”後, 是否正確
28. <= (>=2):檢測 argument 之間放入”<=”後, 是否正確
29. = (>=2):檢測 argument 之間放入”=”後, 是否正確
30. string-append (>=2):連接字串
   input:(string-append "Hello," " there!"
   " Wait!")

output: "Hello, there! Wait!"

31. string>? (>=2):比較字串字元大小
    input: (string>? "az" "aw")
    output: #t

32. string<? (>=2) :比較字串字元大小
33. string=? (>=2) :比較字串字元大小
34. eqv? (2):memory 的擺放位置是否相同
    input1: (eqv? 3 3)
    output1:#t
    input2: (define a 3)
            (eqv? a 3)
    Output2: nil

35. equal? (2):比較 2 個 argument 是否相同
    input:(define a '(3 4))
          (equal? a '(3 4))
    Output: #t

36. begin (>=2):輸出最後一個 argument
    input: (begin 3 4 5)
    output: 5

37. if (2 or 3):條件式
    input: (if #t (begin 3 4 5) (begin 6 7))
    output: 5

38. cond (>=1):多個條件式
    input: (cond ((> 3 4) 'bad)
                  ((> 4 5) 'bad)
                  (else "What happened?"))
    Output: "What happened?"

39. clean-environment (0):清除已宣告變數
40. exit (0):離開

41. let(>=2)
  格式: (let (...) .....)

其中(…) 可宣告區域變數, 回傳最後一個 argument 的計算結果。

```
Input: ( let ( (x 3) (y '(1 2 3))
         )
         (cons 1 '(4 5))
         (cons x (cdr y))
         )
Output:( 3
         2
         3
         )
```

42. lambda (>=2)

格式: (lambda (zero-or-more-symbols) one-or-more-S-expressions)

宣告一個”無名的 function”, 可搭配 define 讓 function 擁有名字。

```
Input1: ( lambda () (+ c 5) )
Output1: #<procedure lambda>
Input2: ( ( lambda () (+ 5 5) (+ 5 6) )  )
Output2: 11
```

四、函式介紹:

void PutbackToken(SStr token):將 token 放回 input。

void Movetofront(SStr token, int loc):將字串的字元往前一格。

Void GetNextToken(SStr token1):取得下一個 token。

Void Type(SStr token, SStr type):區別 token 的種類。

bool IsAtom(SStr type):判別 token 的 type 是否為 ATOM。

void Addwait(NeedRP * head):等待右括弧 linked list 的 wait 值增加, 通常碰到 “(“ or “ \’ ” 時才會進入此函式。

void NewwaitRP():(再輸入指令時, 程式會將指令格式轉換為單一的格式
Ex : (list 2 3) = (list .(2 .(3)))

2 者指令的意思相同,後者為樹的儲存方式)

由於添加了 “(” , 因此需要添加新的右括弧 “)” 。

void Subwait(NeedRp head):等待右括弧 linked list 的 wait 值減少,
通常碰到 “) ” 時才會進入此函式。

bool ChwaitRP(NeedRp head):查看是否 head 的 wait<=0。

void DealArriveRP(): 結合 ChwaitRP 以確認是否該將”)”放至 input。

bool TypeNlasttype(SStr type): (建樹過程)檢查前一個 token 的 type
若為” RIGHT-PAREN” or ATOM 中的
type , 則需在現在的 token 與上一個
token 之間添加 “.(“ 。

void NewTnode(BTree &head, BTree &Bnode, bool buildleft, SStr token,
SStr type)
:新增樹的 node , (如 head != null),當 buildleft = true, 將 node 建立
在 Bnode 的左子樹; 反之,則建在右子樹。

void Printspace(int spacenum):用於輸出的縮排。

void Preorder(BTree head, bool isrightchild, int spacenum)
:前序走訪樹並依序輸出。

Quoter(BTree &tree):quote 以後的 S-exp 皆不計算, 故需利用此函示將
quote 以及 quote 以下的 nodes 標記。

IsSexp(BTree &head, BTree Bnode, SStr token, SStr type, bool hrp,
bool hdot)
:利用遞迴檢測 syntax error

CleanTree(BTree &head)
:執行完 input 後將樹清除

Checkexit(BTree head)
:查看指令是否為 exit
CleanWaitingRP(NeedRp &head)
:清除等待加入”)” linked list。

void Fota(SStr &num1, float num2)
:浮點數 num2 轉為字串 num1。

bool IsatomT(BTree head)
:檢查樹的左子節點是否為 atom tree(配合指令 “atom?”)。

bool Isdottedpair(BTree head)
:檢查樹是否為 dotted pair。

bool Islist2(BTree head)
:檢查樹是否為 list。

bool Ispair(BTree head)
:檢查樹的左子節點是否為 pair。

bool Isnull(BTree head)
:檢查樹的左子節點是否為 nil

bool IsInteger(BTree head)
:檢查樹的左子節點是否為 integer

bool Isstring(BTree head)
:檢查樹的左子節點是否為 string

bool Issymbol(BTree head)
:檢查樹的左子節點是否為 symbol

bool Isboolean(BTree head)
:檢查樹的左子節點是否為 T or NIL

bool Isnumber(BTree head)
:檢查樹的左子節點是否為數字

bool IsInstruction(SStr token)
:檢查 token 是否為指令

void CopyTree(BTree head, BTree &head2)

:複製樹

void Address_plus(SStr name, BTree head)
:將樹的每個節點做 name 的標記, eqv 指令會使用到該標記

Void NewdefinedArea(AreavariableList &deList, DefinitionList
node_betweenArea)
:利用 linked list 的方式新增放置變數的區域, 全域變數的區域 linker =
NULL

Void NewDefinition(AreavariableList &gdeList, SStr name, BTree head,
bool isfunction)
:宣告新的變數並賦予該變數代表的「樹」。

void InstructionDefinition()
:將內建指令存入 defined list。

int AtomNum(BTree head)
:算出同 level 的 s-exp 總數 。

bool CheckifinDeList_S(SStr token, BTree &head, AreavariableList
deList, DefinitionList nownode)
:查詢 token 是否在 defined list , 若有, 拿取該 token 代表的樹。

bool CheckifisFunctioninDeList(SStr token, AreavariableList deList,
DefinitionList nownode)
:查詢 token 是否為已被宣告的 function。

void ProceduretoIns(BTree head)
:令 xx 為指令, 此函式將#<procedure xx> 轉換為 xx。

void ChecknameinDeList(SStr name, SStr &originname, AreavariableList
deList, DefinitionList nownode)
:查詢某 defined symbol 最初始的地址

bool Isequal(BTree head1, BTree head2)
:比較兩棵樹是否相同

```
void CleanDeList( DefinitionList &deList1 );
```
:清除 defined symbol

```
void DeleteTopAreavariableList( AreavariableList &gdeList )
```
:移除最近的 area variable

```
void Clean_environment( AreavariableList &gdeList )
```
:移除所有變數

```
bool Iseqv( BTree head1, BTree head2 )
```
:檢查 2 棵樹的地址是否相同

```
bool Argumentnum( int mix, int max, BTree head, SStr instructionname,
                  bool needprint )
```
:檢測函式的 argument 數量是否正確

```
void BeCommendhead( BTree &definehead, SStr name )
```
:建立一棵指令為“name”字串的樹

```
void NewTemplambda( BTree parameter, BTree statement )
```
:新增函式至函式 list。

```
void CleanTemplambdaList( TemplambdaList &gTemplambdaList )
```
:清除函式 list。

```
bool NodeshaveQuote( BTree tree )
```
:檢測樹中是否含有 quote。

```
bool Preorder_Evalu( BTree linker, BTree &head, bool isfirstlevel,
                     AreavariableList nowdeList,
                     DefinitionList nownode,
                     bool infunction, bool thelastargument,
                     bool incondition, bool intestcondition );
```
:執行指令, 並檢測是否有計算的 error。