Programming Methodology
Instructor: Kyuseok Shim

Project #3: Scheme Interpreter
Due Date: 0:0 a.m. between 2007-11-18 & 2007-11-19

Introduction

이 프로젝트는 수업시간에 배운 Scheme 언어의 Interpreter를 C++로 구현하는 것으로, 모든 기능을 구현하는 것이 아니라 그 일부만을 구현한다. 구현해야 할 부분은 사용자가 Console에서 Scheme 명령어를 입력하면 이를 parsing해서 Symbol table(Hash table)과 Node array에 입력하는 부분이며 최종적으로 Node array에 저장된 Scheme 명령어를 출력하도록 한다.

Implementation

1. 사용자가 Input으로 콘솔에 한 줄 혹은 여러 줄을 입력하면 이를 parsing해서 Symbol table(Hash table)과 Node array에 삽입한 다음 Output으로 출력하는 프로그램을 구현한다.
2. 사용자가 file을 이용하여 scheme을 입력하면 그 결과를 file로 출력한다는 프로그램을 구현한다.

• cdr과 car을 제외한 연산자들은 symbol로 취급하도록 구현하도록 한다. 즉, (+ 1 2)를 입력하더라도 3을 출력하는 것이 아니라 list에 (+ 1 2)가 입력되어 있는 형태를 출력해야 한다. cdr과 car은 구현이 되어야 한다. (car (+ 1 2))를 입력한다면 car가 저장되어있는 node가 (cdr (+ 1 2))를 입력한다면 (1 2)가 저장되어 있는 node가 출력되도록 구현한다.
• quotation과 lambda는 preprocessing이 되도록 구현한다. 즉, 아래와 같이 자동으로 명령어를 바꾸어서 처리하도록 구현한다.
- quotation mark가 있는 경우
  ‘(1 2 3) => (QUOTE (1 2 3))
- 함수를 정의할 경우
  (define (sum x y) (+ x y)) => (define sum (lambda (x y) (+ x y)))

Input & Output

실행 파일을 시작하면 입력프롬프트가 나타나고 괄호가 정확히 입력될 때까지 입력받도록 한다. 두 줄 이상에 걸쳐 입력을 받을 수도 있다. 아무 내용도 없다면 loop에서 빠져 나오고, 괄호가 맞지 않다면 에러를 출력하도록 한다. 정상적으로 입력이 되었다면 입력된 내용이 저장된 구조체를 출력한 후 다시 입력 상태가 된다

>scheme.exe

input>...

(결과)…

input>

실행 파일 뒤에 파일을 덧붙여 실행한다면 파일 안의 내용의 결과를 파일로 출력한다. 입력 파일에는 여러 개의 scheme 명령어가 들어갈 수 있다. 단, 빈줄('₩n'만 존재)은 없다고 가정한다. 출력 파일은 프롬프트입력시 결과와 동일한 형태로 출력한다.

>scheme.exe input.txt output.txt

(input.txt 안의 명령어를 실행하여 output.txt에 결과 저장)

Node array에 저장된 내용은 입력한 형태와 tree 형태로 출력하도록 한다. 입력한 형태는 car, cdr의 경우는 evaluation을 해서 출력, 나머지 경우는 그대로 출력한다. Tree의 형태는 아래의 실행 예시를 참고한다. 한 줄에 하나의 node을 출력하되 출력하려는 tree의 node의 depth가 깊어질수록 node의 앞에 일정한 크기의 빈 공간을 더 넣는다. 즉, 같은 depth의 node를 표시할 때는 앞의 빈칸의 개수가 같다.

Tree의 node가 1)Node array의 element를 가리키면 (Array Index, Left, Right)의 형태로 출력하도록 한다. 여기서 Array Index는 현재 element의 Node Array의 index, Left는 tree상에서 왼쪽 child에 해당하는 node의 index이고 Right는 오른쪽 child에 해당하는 node의 index를 나타낸다.(해당하는 child가 없으면 NULL로 표시한다.). Tree의 node가 2)Hash Table의 element를 가리키면 (Hash Index, symbol)의 형태로 출력한다. 여기서, Hash Index는 hash table의 index를 가리키고, symbol은 해당 hash table의 element에 저장된 symbol을 가리킨다. Node Array의 크기는 무한하다고 가정한다.(즉, 충분히 큰 크기의 array를 사용한다.)

문법적으로 사용된 괄호는 hash table 에 저장되지 않고 Node array상에서 사용되지 않는 node들의 list인 free list를 유지한다.

프로그램 실행 예시

Node array나 Hash table의 index값은 변할 수 있다.

```
> scheme.exe
input> (define x 3)          // 정상 입력
(define x 3)                 // 입력한 형태로 출력
(11, -1756, 12)              // 각각의 node를 출력
   (-1756, define)
   (12, -121, 13)
      (-121, x)
      (13, -52, 0)
         (-52, 3)
         (0, NULL)
input> (define five 4))      // 괄호에 문제가 있는 경우
error!!
input> (define x             // 두 줄 이상에 걸쳐 입력하는 경우
input> 3)
(define x 3)                 // 입력한 형태로 출력
(11, -1756, 12)              // 각각의 node를 출력
   (-1756, define)
   (12, -121, 13)
      (-121, x)
      (13, -52, 0)
         (-52, 3)
         (0, NULL)
input> '( 1 2 )              // '가 있을 경우
(QUOTE (1 2))                // 입력한 형태를 preprocessing한 형태로 출력
(21, -872, 22)               // 각각의 node를 출력
   (-872, QUOTE)
   (22, 23, 0)
      (23, -321, 0)
         (-321, 1)
         (24, -75, 0)
            (-75, 2)
            (0, NULL)
      (0, NULL)
```

```
input> (define (sum x y) (+ x y))// 함수를 정의하는 경우
(define sum ( lambda (x y) (+ x y)))    // preprocessing한 것 출력
(35, -755, 36)                    // node 출력
   (-755, define)
   (36, -390, sum)
      (-390, sum)
      (37, 38, 0)
          (38, -354, 39)
             (-354, lambda)
             (39, 40, 42)
                (40, -264, 41)
                   (-264, x)
                   (41, -111, 0)
                       (-111, y)
                       (0, NULL)
                (42, 43, 0)     // 이후 생략
                  ...
input> ( car ( + 1 2) )    // car
+                          // car를 실행한 부분 출력
(-1302, +)                 // node출력
input> ( cdr ( + 1 2) )    // cdr
(1 2)                      // cdr를 실행한 부분 출력
(108, -1111, 109)          // node 출력
    (-1111, 1)
    (109, -989, 0)
         (-989, 2)
         (0, NULL)
input>                     // 입력이 없으면 종료
>

> scheme.exe  input.txt  output.txt      // input과 output을 입력할 경우
>
```

참고

1. Scheme interpreter의 pseudo-code는 게시된 pdf 파일을 참고하도록 한다.

2. hash table

hash table을 구현하는 방법에는 여러 가지 있으나 여기서는 open address 방법을 이용하도록 한다.

다음 skeleton code를 참고한다. 기능만 동일하다면 아래 코드와 전혀 다르게 구현하더라도 무방하다.

※클래스 이름, 함수 이름, 함수 인자 종류 등은 임의로 변경가능

```
#define       HASH_SIZE     10001  // Hashtable의 크기, 아주 큰 수로 둔다

class HashEntry {
public :
       char *symbol;        // key로 사용
       int value;           // node의 index나 다른 hash value를 가리킴
};

class HashTable {
private:
       HashEntry table[HASH_SIZE];       // hash table
       int hashFunc(char *symbol);       // hash function
public :
       HashTable();               // Constructor
       ~HashTable();              // Destructor
       int insert(char *symbol);  // Hashtable에 symbol을 삽입
       int getIndex(char *symbol);// symbol에 해당하는 Hashvalue를 얻음
       char* getSymbol(int index);// Hashtable의 symbol을 얻음
       int setValue(int index, int value); // Hash Entry의 value 설정
       int getValue(int index);   // Hash Entry의 value 얻음
};
```

3. hash function

hash function에는 제약이 없다. 가능한 1:1대응에 가까울수록 좋은 hash function이라고 할 수 있다. hash function의 한 예는 다음과 같다.

f() : 각각의 char 의 ASCII 값의 제곱의 합을 Hashtable의 크기로 나눈 나머지

ex) f(“ABC”) = (65^2+ 66^2+ 67^2)%10001 = 3070

제출사항

1. 제출 방법 및 내용

   1) 프로젝트 구현 파일 제출

      - 파일은 ee 홈페이지에 과제 제출을 이용하여 제출한다.
      - 소스 파일과 컴파일을 할 수 있는 파일과 실행파일을 압축하여 제출한다.
      - **각 제출 파일 이름은 다음의 형식을 따른다.**

        pro_3_학번.zip

        예) pro_3_2007-12345.zip

      - 각 소스 파일 상단에 **E-mail주소와 작성자 학번 이름**을 적는다.

   2) 보고서 제출

      간단한 설명과 discussion을 하드카피로 제출한다.

2. 제출 일시

   파일 제출 마감일 : 2007년 11월 18일과 19일 사이 새벽 0:00

   - 구현 파일 제출 시각은 홈페이지 과제 제출 시각에 준한다

   보고서 제출 마감일 : 2007년 11월 19일 오전 12:00(정오)

   302동 516-2호 앞 과제제출함

채점 규칙

몇 가지 다양한 scheme들에 대해서 프로그램이 정확히 동작하는 지 확인하여 채점

딜레이는 하루에 10%씩 감점

Copy 발견 시 모든 숙제 0점 처리