|  |
| --- |
| Programming Project #2  <수식 인터프리터 개발> |

|  |  |
| --- | --- |
| 과목 | 컴파일러 |
| 교수님 | 류기열 교수님 |
| 학과 | 수학과 |
| 학번 | 201421155 |
| 이름 | 이안규 |

****

1. 서론

* 과제소개

이번과제는 Recursive Descent Parsing 기법을 이용해서 lexical analysis를 실행한 토큰들의 syntax analysis를 실행한다. Syntax analysis에서 error가 없는 경우 구성된 syntax tree를 이용하여 계산 결과까지 출력하는 것이 이번 과제이다.

* 이번과제에서 구현해야 하는 recursive-descent parsing과 recursive-descent parsing으로 syntax tree를 생성하는 것과 생성된 syntax tree를 이용하여 계산결과를 출력하는 것까지 모든 부분이 구현되었다.

1. 문제 분석

* Grammar rule 분석

미리 주어진 grammar for expressions은

A -> id A’ | F’ T’ E’

A’ -> = A | T’ E’

E -> T E’

E’ -> + T E’ | ε

T -> F T’

T’ -> \* F T’ | ε

F -> id | F’

F’ -> ( E ) | inum | fnum | - F 이다.

여기서 문제에서 주어진 minus 연산과 divide 연산을 grammar rule( E’ -> - T E’

T’ -> / F T’)에 추가 하였다.

|  |  |
| --- | --- |
| Rules | tasks |
| A -> id A’ | “Id” tree node를 만들어서 A’의 parameter로 전달 후 A’에서 반환되는 tree node를 A에서 반환한다. |
| A -> F’ T’ E’ | F’에서 반환되는 tree node를 T’의 parameter로 전달해주고 T’에서 반환되는 tree node를 E’의 parameter로 전달 후 반환되는 tree node를 A에서 반환한다. |
| A’ -> = A | Parameter로 전달받은 tree node(left child)와 A에서 반환된 tree node(right child)를 두 child로 하는 = tree node를 만들어서 반환한다. |
| A’ -> T’ E’ | Parameter로 전달받은 tree node를 T’의 parameter로 전달하고 T’에서 반환되는 tree node를 E’의 parameter로 전달 후 E’에서 반환되는 tree node를 반환한다. |
| E -> T E’ | T가 반환한 tree node를 E’의 parameter로 전달하고 E’가 반환한 tree node를 반환한다. |
| E’ -> + T E’ | Parameter로 전달받은 tree node(left child)와 T에서 반환된 tree node(right child)를 두 child로 하는 + tree node를 만들어 E’의 parameter로 전달하고 우측의 E’에서 반환한 tree node를 반환한다. |
| E’ -> - T E’ | Parameter로 전달받은 tree node(left child)와 T에서 반환된 tree node(right child)를 두 child로 하는 - tree node를 만들어 E’의 parameter로 전달하고 우측의 E’에서 반환한 tree node를 반환한다. |
| E’ -> ε | Parameter로 전달받은 tree node를 반환한다. |
| T -> F T’ | F가 반환한 tree node를 T’의 parameter로 전달하고 T’가 반환한 tree node를 반환한다. |
| T’ -> \* F T’ | Parameter로 전달받은 tree node(left child)와 F에서 반환된 tree node(right child)를 두 child로 하는 \* tree node를 만들어 T’의 parameter로 전달하고 우측의 T’에서 반환한 tree node를 반환한다. |
| T’ -> / F T’ | Parameter로 전달받은 tree node(left child)와 F에서 반환된 tree node(right child)를 두 child로 하는 / tree node를 만들어 T’의 parameter로 전달하고 우측의 T’에서 반환한 tree node를 반환한다. |
| T’ -> ε | Parameter로 전달받은 tree node를 반환한다. |
| F -> id | Id tree node를 만들어 반환한다. |
| F -> F’ | F’에서 반환한 tree node를 반환한다. |
| F’ -> ( E ) | E가 반환한 tree node를 반환한다. |
| F’ -> inum | Integer constant tree node를 만들어 반환한다. |
| F’ -> fnum | Real constant tree node를 만들어 반환한다. |
| F’ -> - F | F에서 반환된 tree node(left child)를 child로 하는 – tree node를 만들어 반환된 tree node를 반환한다. |

* Recursive-descent parsing을 이용한 수식 계산기의 기본 개념정리

Recursive-descent parsing을 거치면서 생성된 syntax tree의 경우 tree node가 operator인 경우 양쪽 child를 operator로 계산하는 결과를 반환하고 tree node가 operator가 아닌 id, integer constant, real constant인 경우 해당하는 값을 반환하도록 recursive한 형태로 수식 계산기 함수를 구성한다. 하지만 minus operator의 경우 양쪽 child가 존재하지 않는 한쪽 child만 존재하는 경우가 있기 때문에 그 경우를 확인하여 계산하도록 구성해준다.

1. 설계

* 주요 자료구조

Recursive-descent parsing 도중 syntax tree를 만들기 위해 tree의 자료구조를 사용했다. Syntax tree의 각 노드를 생성하기 위해 Tree \* mkTree(TOKEN type, char\*val ) 함수를 정의 하였고 생성된 syntax tree로 계산을 하기 위한 double cal(Tree \* rtnode) 함수를 recursive한 형태로 정의 하였다. 또한 생성된 tree의 메모리를 free하기 위한 void Free\_tree(Tree \* node) 함수를 정의 하였다.

* 모듈 설명

|  |  |
| --- | --- |
| **Lexical analysis** | |
| void do\_lexical\_analysis( ); | Lexical analysis를 실행한다. |
| void yyerror( ); | Lexical error에 대한 메시지를 출력한다. |
| void initialize\_symbol\_table( ); | 심볼 테이블을 초기화 한다. |
| bool is\_symbol\_duplicated(char \* target); | 심볼 테이블에 중복된 심볼이 있는 지 체크한다. |
| int find\_symbol(char\* target); | 심볼 테이블에 일치하는 심볼의 인덱스를 반환한다. |
| void print\_token\_list( ); | 토큰 리스트를 출력한다. |
| void save\_token(TOKEN token, char\* value); | 토큰 리스트에 토큰의 종류와 토큰의 값을 저장한다. |
| void save\_symbol(char\* symbol); | 심볼 테이블 심볼과 심볼의 값을 저장한다. |
| void addEnd( ); | Lexical analysis로 생성된 토큰리스트의 마지막에 마지막을 지정하는 문자를 저장한다. |
| typedef enum { ID=1, INT, REAL, PLUS, MINUS, MUL, DIV, ASSIGN, LP, RP} TOKEN; | 토큰들을 선언한 enum |
| typedef struct{  TOKEN token;  char value[TOKEN\_VALUE\_MAX+1];  }TOKEN\_LIST; | 토큰의 타입을 정의하는 token과 토큰의 값을 저장하는 value를 갖는 구조체 TOKEN\_LIST를 정의한다. |
| typedef struct{  char symbol[SYMBOL\_MAX+1];  TOKEN type;  union{  int integer\_constant;  double real\_constant;  }value;  int set\_up;  }SYMBOL\_TABLE; | 심볼의 이름을 저장하는 symbol과 심볼의 타입을 정의하는 type, 심볼의 값을 저장하는 value, 심볼이 초기화 되었는지를 저장하는 set\_up을 갖는 구조체 SYMBOL\_TABLE을 정의한다. |

|  |  |
| --- | --- |
| **Syntax analysis** | |
| Tree\* A( ); | A의 syntax analysis후 반환된 tree node를 반환한다. |
| Tree\* A\_(Tree \* a\_ ); | 전달받은 tree node로 A’의 syntax analysis후 반환된 tree node를 반환한다. |
| Tree\* E( ); | E의 syntax analysis후 반환된 tree node를 반환한다. |
| Tree\* E\_(Tree \* e\_ ); | 전달받은 tree node로 E’의 syntax analysis후 반환된 tree node를 반환한다. |
| Tree\* T( ); | T의 syntax analysis후 반환된 tree node를 반환한다. |
| Tree\* T\_(Tree \* t\_ ); | 전달받은 tree node로 T’의 syntax analysis후 반환된 tree node를 반환한다. |
| Tree\* F( ); | F의 syntax analysis후 반환 된 tree node를 반환한다. |
| Tree\* F\_( ); | F’의 syntax analysis후 반환 된 tree node를 반환한다. |
| Tree\* mkTree(TOKEN type, char\* vl ); | Parameter로 전달받은 토큰의 타입과 토큰의 값이 들어간 tree node를 반환한다. |
| int not\_defined(TOKEN\_LIST t[ ] ); | 생성된 토큰리스트를 parameter로 전달받아 id가 초기화 되어있는지 확인한다. |
| void match( int look\_ahd ); | Syntax analysis 중 non-terminal이 생성될 때 전달받은 look\_ahd 와 토큰리스트의 토큰의 타입과 동일한지 확인하고 token\_list의 index를 가리키는 index\_location을 1 증가시킨다. |
| void syntax\_error( ); | Syntax error를 확인후 syntax error메시지를 출력하고 syntax error를 판단하는 변수 s\_error를 1로 초기화 해준다. |
| void do\_syntax\_analysis( ); | Syntax analysis을 진행한다. |
| void Free\_tree( Tree\* node ); | 생성된 syntax tree node에 할당된 메모리를 해제해준다. |
| typedef struct TreeNode{  int token\_type;  char token\_value[TOKEN\_VALUE\_MAX+1];  struct TreeNode \* left;  struct TreeNode \* right;  }Tree; | 토큰의 타입과 token의 이름 양쪽 tree node를 가리키는 변수를 갖는 구조체 Tree를 정의한다. |

|  |  |
| --- | --- |
| **Syntax tree calculator** | |
| double cal(Tree\* rtnode) | 생성된 syntax tree를 전달받아 계산해여 값을 반환해준다. 함수 형태는 recursive한 형태로 구성한다.   1. Tree node가 id(심볼 테이블에 해당하는 값), int constant, real constant 인 경우 값을 그대로 반환한다. 2. Tree node가 + 인 경우 양쪽 child의 합을 반환한다. 3. Tree node가 – 인 경우 4. 왼쪽 child만 존재하는 경우 왼쪽 child를 계산한 값에 -를 붙인 값을 반환한다. 5. 양쪽 child가 존재하는 경우 왼쪽child에서 오른쪽 child를 뺀 값을 반환한다. 6. Tree node가 \* 인 경우 양쪽 노드의 곱을 반환한다. 7. Tree node가 / 인 경우 왼쪽 child에서 오른쪽 child를 나눈 값을 반환한다. 8. Tree node가 = 인 경우 왼쪽child의 id의 값을 symbol table에서 오른쪽 child의 값으로 초기화 시켜주고 오른쪽 child값을 반환한다. |

1. 수행결과

