프로그래밍 언어론 제1 과제 보고서

02분반
20202865 윤수용
20203436 이병구

파일 구성

1. ./README.txt : 코드 컴파일 및 실행 관련 설명이 담겨 있습니다.
2. ./RecursiveParser/main.cpp : 해당 프로그램의 모든 코드가 담겨 있습니다.
3. ./RecursiveParser/bin/Debug/*.txt : 프로그램 수행 확인을 위한 테스트 파일들입니다.

코드 성분

(모두 main.cpp 에 있습니다.)

- `int main(int argc, char *argv[])`
  - 프로그램 입력을 받는 함수입니다.
    - 명령인자로 파일 이름을 받아, 먼저 입력 정보를 확인합니다.
    - 파일을 열어 입력 받은 문자열을 RDP클래스 타입으로 생성자를 호출, parse함수를 호출합니다.

- `void parse()`
  - RDP class: Recursive Descent Parsing을 수행합니다.
    - 토큰 종류를 심볼 종류를 표현하는 열거형(enum) TOKEN_TYPE 으로 정의합니다.
    - 생성자로 파일을 열고, 소멸자로 파일을 닫습니다.
    - parse() 함수는 RDP클래스의 멤버 함수입니다.
    - statements() 함수로 넘어가 파싱을 진행하며, 이후 printSymbolTable() 함수에서 결과를 출력합니다.

- `void statements()`
  - <statements>를 처리합니다.
    - 세미콜론까지 읽어 먼저 lexical()함수에서 lexical analysis를 진행합니다.
    - lexical analysis를 통과했다면, statement()로 넘어갑니다. (한 문장씩)

- `void statement()`

- <statement>를 처리합니다.
    - 문장을 받아 nextToken을 검사해서 변수 할당문을 처리하고 _symTable을 업데이트 합니다.
    - _symTable에 값을 대입할 때는 expression()을 호출합니다.
    - 문제가 있는 경우 _result에 오류 메시지를 추가합니다.

- `string expression()`

- <expression>을 처리합니다.
    - term()을 호출해서 expression을 계산하고 '+(덧셈)' 또는 '-(뺄셈)'을 처리한다.

- `string term()`

- <term>을 처리합니다.
    - factor()를 호출해서 term을 계산하고 '*(곱셈)' 또는 '/(나눗셈)'을 처리합니다.

- `string factor()`

- <factor>를 처리합니다.
    - nextToken이 식별자인 경우: 변수를 처리하고 _symTable에 해당 변수를 추가합니다.
    - nextToken이 상수인 경우: 상수를 처리합니다.
    - nextToken이 '('인 경우: 괄호 안의 표현식을 계산합다.
    - 나머지 경우: 오류를 처리합니다.

- `void lexical()`

- lexical analysis를 진행합니다.
- 토큰을 읽고 종류를 결정합니다.
    - 문자를 인식하고 해당하는 nextToken값을 설정합니다.
    - 오류 처리도 겸합니다.

```
void printStatement()
```

- 파싱한 문장을 출력합니다.
  - _tokens 벡터를 순회하며 토큰들을 출력합니다.
  - _ID, _CONST, _OP 값을 문제의 조건에 맞춰 각각의 개수를 출력합니다.
  - _result에 오류/경고 메시지가 있으면 출력합니다.

```
void printSymbolTable()
```

- 심볼 테이블을 출력합니다.
  - _symTable: 맵을 순회하며 변수 이름과 값을 출력합니다.

- 크게 코드는 main() -> statements() -> statement() -> expression() -> term() -> factor() 순으로 점차 내려가되, 중간중간 lexical()를 통해 체크합니다. 이후 printStatement()를 통해 완성된 결과물을 출력합니다.

<기타 함수, 변수 목록>

```
void msgError(string msg)
void msgWarning(string msg)
```

- msgError(), msgWarning(): 오류, 경고 메시지 생성 및 결과에 메시지를 추가한다.

```
void getChar()
```

- 다음 문자를 읽습니다.
  - _ch: 현재 파싱 중인 문자

```
bool isSpace(char c)
bool isOperator(char c)
bool isOperator(string s)
bool isSpecial(char c)
```

- 문자의 공백 여부, 연산자 여부, 특수 문자 여부를 판별합니다.
  - 에러 여부, 렉시컬 분석 등에 활용되는 보조 함수들입니다.

_tokens : 파싱된 토큰들을 저장한다.

<예외 처리 방법>

- 먼저 다음 4가지 경우에 대해서 예외 처리를 해주었습니다. 정상적인 입력이 왔을 때에는 일반적으로 입력을 처리하지만, 아래의 경우와 같이 특이한 케이스의 경우에는 Warning혹은 Error로 따로 처리를 해주었습니다.
- Warning의 경우, 해당 문제를 가능한 방향으로 해결하고, 해결한 후의 결과값을 도출해서 출력합니다.
- Error의 경우, 에러 메시지를 띄우고 따로 문제해결을 시도하지 않습니다.

1. RHS 시작하는 부분 혹은 중간 부분에 연산자가 오는 경우

a. (Warning) 중복된 토큰을 제거한 뒤 다음 토큰을 불러옵니다. -> 정상적인 결과값을 출력합니다.

2. variable이 정의되지 않은 경우

a. (Error) 문장 자체는 끝까지 처리하지만 해당 Variable은 처리하지 않습니다. -> 결과값은 unknown을 출력합니다.

3. assignment가 잘못된 경우

a. (Error) 문제상황 2번과 마찬가지로, 문장 자체는 끝까지 처리하지만 해당 Variable은 처리하지 않습니다. -> 결과값은 unknown을 출력합니다.

4. 한 문장에 Warning 또는 Error가 2개 이상인 경우

a. 해당되는 모든 Warning/Error 메시지를 출력하도록 합니다. -> 결과값은 처리 가능 여부에 따라 달라집니다.