형식언어 MiniC Scanner 구현

제출일	2024년 6월 3일	학과	컴퓨터공학전공
과목	형식언어_02	학번	2020112736
담당교수	송수환 교수님	이름	안성현

▪ Scanner.java 코드 분석 (추가 구현 코드 포함)

• next()

주어진 입력에서 해당하는 다음 토큰을 찾고 반환하는 함수이다. 문자를 읽어들인 후, 그 문자에 해당하는 토큰이나 그 다음에 특정 문자가 나왔을 때 가능한 토큰이 나오게 문자를 받아 토큰으로 리턴한다.

다음 문자가 알파벳이나 '_'에 해당하는 경우, 이후의 문자들은 식별자나 키워드에 해당하므로 concat() 함수로 문자열을 이어 String spelling에 저장하고 키워드 토큰으로 반환한다. 다음 문자가 숫자이거나 '.'인 경우는 이후의 문자들은 int, double형에 해당하기 때문에 숫자, '.'(소수점), 'e', '+', '-'(부동소수점)의 집합으로 묶어 concat() 함수를 통해 숫자들을 연결한다. 이러면 소수점, 숏폼, 부동소수점, 정수 모두 하나의 토큰으로 연결하게 된다. String number로 받은 후 '.'의 포함 여부에 따라 정수 리터럴 토큰, 소수 리터럴 토큰으로 나누어 토큰을 리턴한다.

공백문자를 받았을 경우는 무시하고 다음 문자를 받아오고, '/'을 받았을 경우, 그 다음 문자를 받아와서 그 문자에 따라 그에 맞는 토큰을 반환한다. ('/='은 나누어 할당, '/'은 그냥 나누기, '/*'는 여러 줄 주석, '//'은 한 줄 주석)

작은 따옴표, 큰 따옴표가 나왔을 때에는 charLiteral, stringLiteral로 리턴될 수 있기 때문에 그에 맞는 작업을 한다. (Token.java, TokenType.java에도 새로운 토큰 형식을 생성해준다. 처리하는 방식은 intLiteral을 처리하는 방식과 비슷하게 하였다.) 작은 따옴표는 char이기 때문에 바로 다음 문자 하나만 받아 charLiteral 토큰으로 리턴해주고, 큰 따옴표는 문자열(string)이므로, 매개 변수가 없는 concat() 함수(기존 함수를 오버로딩)를 불러와 문자열을 연결해서 stringLiteral 토큰으로 리턴한다. 매개 변수가 없는 concat() 함수는 자세히 후술한다.

그 밖에 eofCh, ';', ',', 괄호들은 그 토큰으로 리턴, '+', '-', '*', '%', '&', '|', '=', '<', '>', '!' 문자는 '/'와 비슷한 방식으로 연산자 및 기호 처리하여 리턴한다. 추가로 추가 구현해야되는 case, default 키워드에서 쓰이는 ':' 기호도 그에 맞는 토큰을 Token.java, TokenType.java에 새로 만들어 리턴한다. 이외에 해당하는 토큰이 없는 경우 에러를 출력하는 함수를 호출한다.

• concat()

concat() 함수의 경우 오버로딩을 통해 매개변수가 있는 경우와 없는 경우에 대해 다르게 작동하도록 하였다.

매개 변수가 있는 경우 매개 변수 집합에 해당하는 문자들을 대상으로 문자열을 연결하여 리턴한다. nextChar()를 통해 문자를 계속 인식받아 이어 붙이는데 이것을 매개 변수 집합에 해당하는 문자가 나오지 않을 때 까지 반복한다.

매개 변수가 없는 경우 "~~~"와 같은 stringLiteral 토큰을 리턴하기 위해 문자열을 연결하는 함수로, StringBuilder 클래스를 통해 일반적인 경우에는 문자를 받아 문자열에 추가, 이스케이프 문자(줄바꿈, 탭 등등)가 나올 경우 그것을 인식해서 문자열에 추가하는 방식이다. 이것은 string이 끝나는 큰 따옴표가 나올때 까지 반복하는데, '₩"'과 같이 큰 따옴표 자체가 string에 포함된 경우에는 이스케이프 문자들을 처리하는 방식과 똑같이 해주고 큰 따옴표 하나만 나왔을 때의 경우를 string이 끝난 경우로 인식하였다.

• error()

오류 처리를 위한 함수이다. 줄 번호와 함께 에러 문구를 출력한다. 또한, 에러 문구

출력 후 다음 토큰 인식을 시도해야 하는 조건에 따라 프로그램 종료를 하지 않고 에러가 발생한 메서드를 조금 수정하여 다음 토큰을 읽도록 하였다.

- **Token.java와 TokenType.java에서의 추가 구현 설명**

  Token.java에 새로 인식되어야 할 토큰들을 선언한다. 또한, intLiteral에 대한 토큰을 리턴하는 방식과 비슷한 방식으로 double, char, stringLiteral에 대한 토큰을 리턴하는 함수도 추가해주었다.

  TokenType.java에는 추가된 토큰들을 마찬가지로 추가해주었다. Switch, Case들과 같은 키워드들의 경우 Eof위에 추가하여 키워드로 인식할 수 있게 해주었고, 나머지 토큰들은 밑에 부분에 추가해주었다.

▪ 확장된 내용을 인식하는 miniC 코드에 대한 결과

- **입력 (extend.mc)**

```
void main()
{
  // string, char, double literal examples
  char chr;
  string str1;
  double num1, num2, num3, num4;
  chr = 'A';
  str1 = "a₩nbc₩"d₩te";
  num1 = 32.612;
  num2 = .123;
  num3 = 123.;
  num4 = 12.345678e+5
  // 'for' examples
  for (int i = 0; i < 10; i ++) {
    num3 += i;
  }
  // 'do' example
  do {
    num3 += 10
  } while (num3 < 180);
  // switch, case, break, default examples
  switch (chr) {
    case 'A':
      num1 = 32.;
      break;
    case 'B':
      num1 = 33.;
      break;
    default:
      num1 = 34.;
```

```
        break;
    }
    test:
        if (num1 == 34) {
            write(num1);
        }
    // 'goto' example
    goto test;
}
```

- **출력 (extend.mc)**

```
Begin scanning... programs/extend.mc

void
Identifier  main
(
)
{
char
Identifier  chr
;
string
Identifier  str1
;
double
Identifier  num1
,
Identifier  num2
,
Identifier  num3
,
Identifier  num4
;
Identifier  chr
=
CharLiteral A
;
Identifier  str1
=
StringLiteral   a
bc"d    e
;
Identifier  num1
=
DoubleLiteral   32.612
;
Identifier  num2
=
DoubleLiteral   .123
;
Identifier  num3
=
DoubleLiteral   123.
;
Identifier  num4
=
DoubleLiteral   12.345678e+5
for
(
int
Identifier  i
=
IntLiteral  0
;
Identifier  i
<
IntLiteral  10
;
Identifier  i
++
)
{
Identifier  num3
+=
Identifier  i
;
}
do
{
Identifier  num3
+=
IntLiteral  10
}
while
(
Identifier  num3
<
IntLiteral  180
)
;
switch
(
Identifier  chr
)
{
case
CharLiteral A
:
Identifier  num1
=
DoubleLiteral   32.
;
break
;
case
CharLiteral B
:
Identifier  num1
=
DoubleLiteral   33.
;
break
;
default
:
Identifier  num1
=
DoubleLiteral   34.
;
break
;
}
Identifier  test
:
if
(
Identifier  num1
==
IntLiteral  34
)
{
Identifier  write
(
Identifier  num1
;
}
goto
Identifier  test
;
}

Process finished with exit code 0
```

확장된 내용들에 대해서도 토큰 인식이 잘 되는 것을 확인할 수 있다. stringLiteral의 경우 이스케이프 문자도 인식하여 바르게 출력되었다. doubleLiteral의 경우 숏폼, 부동소수점도 마찬가지로 잘 인식하였다. 그 밖에 switch, case, default 등과 같은 키워드들과 double, char, String과 같은 변수 선언 부분의 키워드들도 바르게 인식되어 출력되었고, 주석 문구도 주석으로 처리되었음을 알 수 있다.

▪ 기존의 miniC 코드에 대한 Scanner 실습 후 결과 분석

아래의 출력 화면들에서도 알 수 있듯이, Scanning을 시작하는 것을 알리는 문구와 함께, scanning이 시작된다. 식별자들은 identifier 키워드로 분류되어 출력되었고, ';', '(', ')', '{', '}' 과 같은 기호들도 다른 식별자같은것으로 인식하지 않고 각각 그에 맞는 토큰으로 분류되어 올바르게 출력된다. '*', '+', '-', '++', '%=', '*=', '--=', '/='등과 같은 연산자들도 종류나 문자의 개수는 다르지만, 각각 해당하는 토큰으로 맞게 분류되어 출력된다. 그 밖에 키워드들(else, return 등등)과 intLiteral도 해당하는 토큰에 맞게 인식되어 출력이 되었음을 알 수 있다.

```
Begin scanning... programs/comment.mc

void
Identifier  main
(
)
{
int
Identifier  i
;
Identifier  i
=
IntLiteral  100
;
Identifier  write
(
Identifier  i
)
;
}
```

그림 7 comment.mc

```
Begin scanning... programs/ext.mc

int
Identifier  x
;
void
Identifier  main
(
)
{
Identifier  func
(
)
;
Identifier  write
(
Identifier  x
)
;
}
void
Identifier  func
(
)
{
Identifier  x
=
IntLiteral  100
;
}
```

그림 8 ext.mc

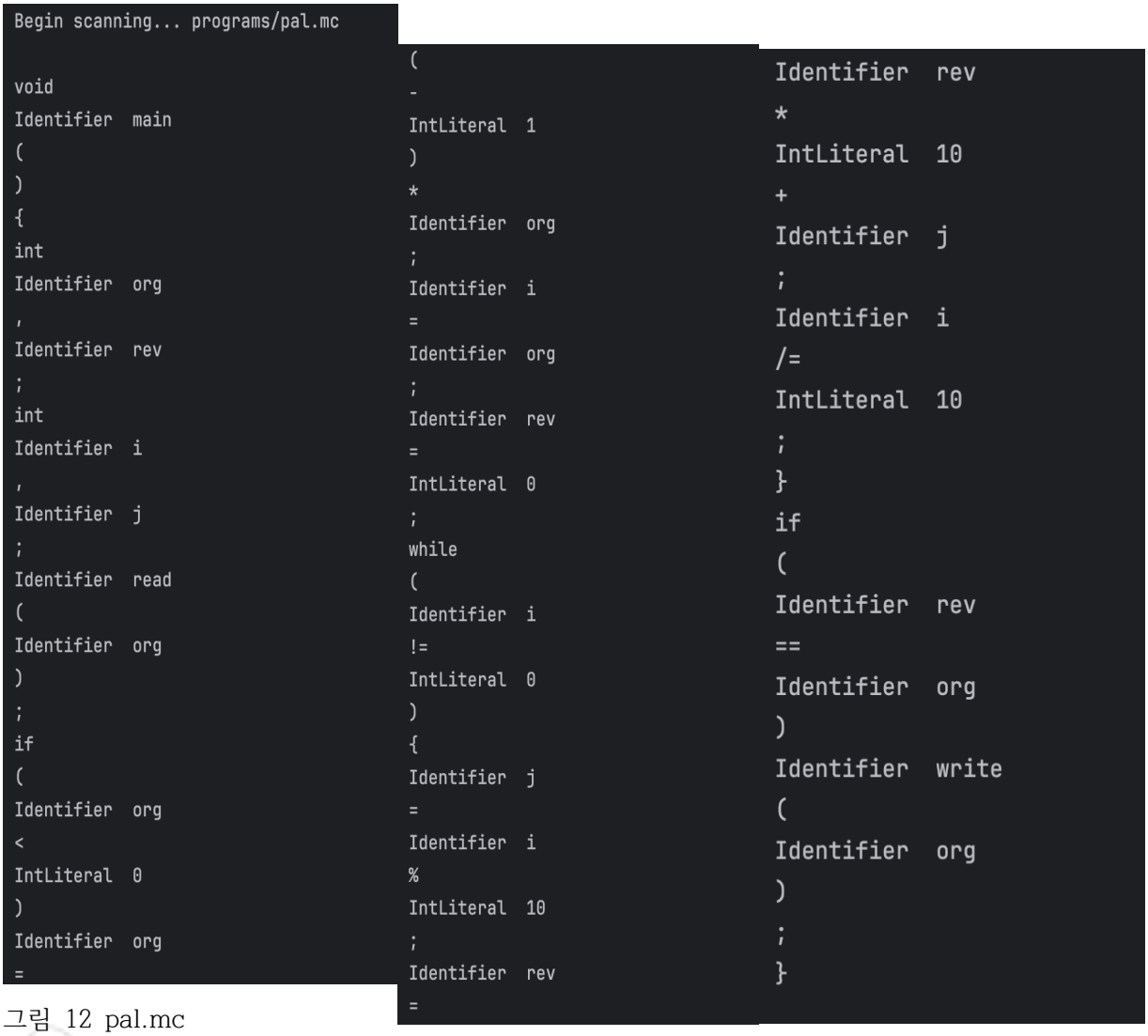
```
Begin scanning... programs/factorial.mc

void
Identifier  main
(
)
{
int
Identifier  n
,
Identifier  f
;
Identifier  read
(
Identifier  n
)
;
Identifier  write
(
Identifier  n
)
;
Identifier  f
=
Identifier  factorial
(
Identifier  n
)
{
if
(
Identifier  n
==
IntLiteral  1
)
return
IntLiteral  1
;
else
return
Identifier  n
*
Identifier  factorial
(
Identifier  n
-
IntLiteral  1
)
;
}
```

그림 9 factorial.mc

```
void
Identifier  main
(
)
{
int
Identifier  x
;
Identifier  x
=
IntLiteral  333
;
Identifier  x
++
;
Identifier  write
(
Identifier  x
)
;
Identifier  x
%=
IntLiteral  10
;
Identifier  write
(
Identifier  x
)
;
}
```

그림 11 mod.mc



```
Begin scanning... programs/pal.mc

void
Identifier  main
(
)
{
int
Identifier  org
,
Identifier  rev
;
int
Identifier  i
,
Identifier  j
;
Identifier  read
(
Identifier  org
)
;
if
(
Identifier  org
<
IntLiteral  0
)
Identifier  org
=
(
-
IntLiteral  1
)
*
Identifier  org
;
Identifier  i
=
Identifier  org
;
Identifier  rev
=
IntLiteral  0
;
while
(
Identifier  i
!=
IntLiteral  0
)
{
Identifier  j
=
Identifier  i
%
IntLiteral  10
;
Identifier  rev
=
Identifier  rev
*
IntLiteral  10
+
Identifier  j
;
Identifier  i
/=
IntLiteral  10
;
}
if
(
Identifier  rev
==
Identifier  org
)
Identifier  write
(
Identifier  org
)
;
}
```

그림 12 pal.mc

```
Begin scanning... programs/perfect.mc

const
int
Identifier  max
=
IntLiteral  500
;
void
Identifier  main
(
)
{
int
Identifier  i
,
Identifier  j
,
Identifier  k
;
int
Identifier  rem
,
Identifier  sum
;
Identifier  i
=
IntLiteral  2
;
```

```
while
(
Identifier  i
<=
Identifier  max
)
{
Identifier  sum
=
IntLiteral  0
;
Identifier  k
=
Identifier  i
/
IntLiteral  2
;
Identifier  j
=
IntLiteral  1
;
while
(
Identifier  j
<=
Identifier  k
)
{
Identifier  rem
=
```

```
Identifier  i
%
Identifier  j
;
if
(
Identifier  rem
==
IntLiteral  0
)
Identifier  sum
+=
Identifier  j
;
++
Identifier  j
;
}
if
(
Identifier  i
==
Identifier  sum
)
Identifier  write
(
Identifier  i
)
;
++
```

```
Identifier  i
;
}
}
```

그림 18 perfect.mc

```
Begin scanning... programs/prime.mc

const
int
Identifier  max
=
IntLiteral  100
;
void
Identifier  main
(
)
{
int
Identifier  i
,
Identifier  j
,
Identifier  k
;
int
Identifier  rem
,
Identifier  prime
;
Identifier  i
=
IntLiteral  2
;
while
(
Identifier  i
<=
Identifier  max
)
{
Identifier  prime
=
IntLiteral  1
;
Identifier  k
=
Identifier  i
/
IntLiteral  2
;
Identifier  j
=
IntLiteral  2
;
while
(
Identifier  j
<=
Identifier  k
)
{
Identifier  rem
=
Identifier  i
%
Identifier  j
;
if
(
Identifier  rem
==
IntLiteral  0
)
Identifier  prime
=
IntLiteral  0
;
++
Identifier  j
;
}
if
(
Identifier  prime
==
IntLiteral  1
)
Identifier  write
(
Identifier  i
)
;
++
Identifier  i
;
}
}
```

그림 22 prime.mc

```
Begin scanning... programs/retval.mc

int
Identifier  sum
(
int
Identifier  n
,
int
Identifier  m
)
{
int
Identifier  i
;
Identifier  i
=
Identifier  n
+
Identifier  m
;
return
Identifier  i
;
}
void
Identifier  main
(
)
{
int
Identifier  x
;
Identifier  x
=
Identifier  sum
(
IntLiteral  10
,
IntLiteral  20
)
;
Identifier  write
(
Identifier  x
)
;
}
```

그림 24 retval.mc

```
Begin scanning... programs/test.mc

const
Identifier  max
=
IntLiteral  100
;
int
Identifier  sum
(
int
Identifier  n
,
int
Identifier  m
)
{
int
Identifier  i
;
Identifier  write
(
Identifier  n
)
;
Identifier  write
(
Identifier  m
)
;
Identifier  i
=
Identifier  n
+
Identifier  m
+
Identifier  max
;
if
(
Identifier  i
==
IntLiteral  1
)
Identifier  i
=
IntLiteral  100
;
return
Identifier  i
;
}
void
Identifier  main
(
)
{
int
Identifier  x
;
Identifier  write
(
Identifier  max
)
;
Identifier  x
=
IntLiteral  333
;
Identifier  x
++
;
Identifier  write
(
Identifier  x
)
;
Identifier  x
=
Identifier  sum
(
IntLiteral  10
,
IntLiteral  20
)
;
Identifier  write
(
Identifier  x
)
;
}
```

그림 27 test.mc