2022 봄학기

KNU 4471.043 컴파일러 설계 과제 4

고상기

2022년 4월 18일 0시

- 이번 과제는 3, 4주차에 배운 내용을 기반으로 Decaf 언어의 어휘 분석기를 구현하는 것입니다.
- Decaf 언어는 객체지향 언어로 상속성 및 캡슐화를 지원하는 특징을 가지며 C++/Java와 많은 유사성을 갖고 있습니다. 다음은 Decaf 언어의 문법을 EBNF(Extended Backus-Naur Form) 형식으로 기술한 것입니다.

⟨Program⟩ ::= ⟨Decl⟩$^+$

⟨Decl⟩ ::= ⟨VariableDecl⟩ | ⟨FunctionDecl⟩ | ⟨ClassDecl⟩ | ⟨InterfaceDecl⟩

⟨VariableDecl⟩ ::= ⟨Variable⟩ ;

⟨Type⟩ ::= `int` | `double` | `bool` | `string` | `id` | ⟨Type⟩ [ ]

⟨FunctionDecl⟩ ::= ⟨Type⟩ `id` (⟨Formals⟩) ⟨StmtBlock⟩ |
`void` `id` (⟨Formals⟩) ⟨StmtBlock⟩

⟨ClassDecl⟩ ::= `class` `id` [`extends` `id`] [`implements` `id`$^+$,] { ⟨Field⟩* }

⟨Field⟩ ::= ⟨VariableDecl⟩ | ⟨FunctionDecl⟩

⟨InterfaceDecl⟩ ::= `interface` `id` { ⟨Prototype⟩* }

⟨Prototype⟩ ::= ⟨Type⟩ `id` (⟨Formals⟩) ; | `void` `id` (⟨Formals⟩) ;

⟨StmtBlock⟩ ::= { ⟨VariableDecl⟩* ⟨Stmt⟩* }

⟨Stmt⟩ ::= ⟨Expr⟩ ; | ⟨IfStmt⟩ | ⟨WhileStmt⟩ | ⟨ForStmt⟩ | ⟨BreakStmt⟩ |
⟨ReturnStmt⟩ | ⟨PrintStmt⟩ | ⟨StmtBlock⟩

⟨IfStmt⟩ ::= `if` (⟨Expr⟩) ⟨Stmt⟩ [`else` ⟨Stmt⟩]

⟨WhileStmt⟩ ::= `while` (⟨Expr⟩) ⟨Stmt⟩

⟨ForStmt⟩ ::= `for` (⟨Expr⟩ ; ⟨Expr⟩ ; ⟨Expr⟩) ⟨Stmt⟩

⟨ReturnStmt⟩ ::= `return` ⟨Expr⟩ ;

⟨BreakStmt⟩ ::= `break` ;

⟨PrintStmt⟩ ::= `Print` (⟨Expr⟩$^+$,) ;

⟨Expr⟩ ::= ⟨LValue⟩ = ⟨Expr⟩ | ⟨Constant⟩ | ⟨LValue⟩ | `this` | ⟨Call⟩ |
− ⟨Expr⟩ | ! ⟨Expr⟩ | ⟨Expr⟩ ⟨BinOp⟩ ⟨Expr⟩ | (⟨Expr⟩)
`ReadInteger` ( ) | `ReadLine` ( ) | `new` `id` | `NewArray` (⟨Expr⟩ , ⟨Type⟩)

⟨LValue⟩ ::= `id` | ⟨Expr⟩ . `id` | ⟨Expr⟩ [⟨Expr⟩]

⟨BinOp⟩ ::= ⟨ArithOp⟩ | ⟨RelOp⟩ | ⟨EqOp⟩ | ⟨CondOp⟩

⟨ArithOp⟩ ::= `+` | `-` | `*` | `/` | `%`

⟨RelOp⟩ ::= `<` | `>` | `<=` | `>=`

⟨EqOp⟩ ::= `==` | `!=`

⟨CondOp⟩ ::= `&&` | `||`

⟨Call⟩ ::= `id` ($\left[\right.$ ⟨Expr⟩$^+$, $\left.\right]$) | ⟨Expr⟩ . `id` ($\left[\right.$ ⟨Expr⟩$^+$, $\left.\right]$)

⟨Constant⟩ ::= `intConst` | `doubleConst` | `boolConst` | `stringConst` | `null`

- 다음은 위 EBNF 문법에서 사용된 표기법에 대한 추가적인 설명입니다.
  - ⟨knu⟩ : knu라는 이름의 논터미널 기호입니다.
  - `knu` : knu라는 이름의 터미널 문자열입니다.
  - $\left[x\right]$: x가 존재할 수도 있고 없을 수도 있습니다. 참고로 '[' 기호와 ']' 기호는 EBNF 문법 기호가 아닌 터미널 기호입니다.
  - x^* : x가 0번 또는 여러번 반복되어 사용될 수 있습니다.
  - x^+ : x가 한 번 이상 반복되어 활용될 수 있습니다.
  - x^+, : x가 쉼표 ',' 기호로 구분되어 한 번 이상 반복되어 활용될 수 있습니다.
- Decaf 언어에 대한 좀 더 자세한 내용은 다음 URL을 참조하세요: `https://web.stanford.edu/class/archive/cs/cs143/cs143.1128/handouts/030%20Decaf%20Specification.pdf`
- 다음은 Decaf 언어에서 사용되는 토큰들의 종류입니다.
  - 예약어(keywords 또는 reserved words): `void int double bool string class interface null this extends implements for while if else return break new true false NewArray Print ReadLine ReadInteger`
    - 예약어 `NewArray, Print, ReadInteger, ReadLine`은 미리 정의된 Decaf 내장함수의 이름으로 배열 생성, 입력, 출력 등의 기능을 수행합니다.
  - 식별자(identifiers): 위 문법에서는 `id`로 표현되어 있으며 식별자는 4주차 강의자료 6페이지에 나온대로 첫 번째 기호가 영문자 소문자나 대문자 또는 밑줄(underscore)로 시작하고 두 번째 기호부터는 영문자 소문자나 대문자, 숫자, 밑줄이 올 수 있습니다. 식별자의 최대 길이는 31이며 대소문자를 구별합니다. 예를 들어, `while`은 반복문을 위한 예약어이지만 `WHILE`이나 `While`은 식별자로 사용될 수 있습니다.
  - 상수(constants): Decaf에서는 아래와 같이 네 가지 자료형의 상수를 지원합니다.
    1. 정수형(integer): 위 문법에는 `intConst`로 표현되어 있습니다. Decaf는 정수 값을 4바이트로 저장하기 때문에 -2147483648(-2^{31})부터 2147483647($2^{31}-1$)까지의 값을 가질 수 있습니다. 만약 토큰이 `0x`로 시작한다면 이 토큰은 16진수로 표현된 정수를 나타낼 수 있으며 이 때는 0부터 9까지의 숫자 뿐만 아니라 16진수를 표현하기 위한 알파벳 기호 `a, b, c, d, e, f, A, B, C, D, E, F`도 이어서 사용될 수 있습니다.

2. 실수형(real number): 위 문법에는 `doubleConst`로 표현되어 있습니다. 실수형 상수의 패턴은 4주차 강의자료 8페이지에 기술되어 있습니다.
3. 문자열(string): 문자열 상수는 큰따옴표(double quotation mark)로 둘러싸인 임의의 문자열(줄바꿈 문자와 큰따옴표는 제외)로 표현됩니다. 문자열 상수는 반드시 한 줄에서 정의되어야 합니다.
4. 불(Boolean): 불 상수는 `true` 또는 `false` 두 개의 예약어로 구성됩니다.

– 연산자(operators): `+ - * / % < <= > >= = == != && || !`

– 구분자(punctuation symbols): `; , . [ ] ( ) { }`

- 입력 파일에는 한 줄 주석과 범위 주석 두 가지 방식의 주석이 포함될 수 있습니다. 어휘 분석기는 주석 부분을 무시하고 아래의 토큰들만 인식하여 출력합니다.
  – 한 줄 주석은 `//` 로 시작되어 줄바꿈이 될 때까지 그 뒷 부분은 모두 주석으로 인식됩니다.
  – 범위 주석으로는 정규 표현식 $/*(a+*^{+}b)^{*}(*^{+}/)$으로 기술 가능한 주석이 포함될 수 있으며 여러 줄을 주석으로 만들 수도 있습니다. (이 때, a는 $*$를 제외한 모든 문자, b는 $*$와 $/$를 제외한 모든 문자를 의미합니다.)

- 다음은 Decaf 언어로 작성된 예시 프로그램(tests/samples/program2.decaf)입니다.

```
int a;

void main() {
   int b;
   int a;
   int d;

   d = 2 + 3 * 4 - 6;
   b = 3;
   a = b + 2;
}
```

우리가 작성한 프로그램은 위 입력 파일에 대해 아래와 같은 결과를 출력합니다. 출력할 정보는 토큰 문자열(어휘항목), 토큰의 줄 번호와 열 번호, 그리고 토큰의 값입니다. 식별자의 경우 프로그램을 어휘 분석하면서 각 식별자들을 순서대로 기호표에 저장 후 1번부터 차례대로 번호를 매겨 번호 값을 출력하고, 상수의 경우 상수 값을 출력하면 됩니다.

```
int             line 1 cols 1-3 is T_Int
a               line 1 cols 5-5 is T_Identifier (token address: 1)
;               line 1 cols 6-6 is ';'
void            line 3 cols 1-4 is T_Void
main            line 3 cols 6-9 is T_Identifier (token address: 2)
(               line 3 cols 10-10 is '('
)               line 3 cols 11-11 is ')'
{               line 3 cols 13-13 is '{'
int             line 4 cols 4-6 is T_Int
b               line 4 cols 8-8 is T_Identifier (token address: 3)
;               line 4 cols 9-9 is ';'
int             line 5 cols 4-6 is T_Int
```

```
a               line 5 cols 8-8 is T_Identifier (token address: 1)
;               line 5 cols 9-9 is ';'
int             line 6 cols 4-6 is T_Int
d               line 6 cols 8-8 is T_Identifier (token address: 4)
;               line 6 cols 9-9 is ';'
d               line 8 cols 4-4 is T_Identifier (token address: 4)
=               line 8 cols 6-6 is '='
2               line 8 cols 8-8 is T_IntConstant (token value: 2)
+               line 8 cols 10-10 is '+'
3               line 8 cols 12-12 is T_IntConstant (token value: 3)
*               line 8 cols 14-14 is '*'
4               line 8 cols 16-16 is T_IntConstant (token value: 4)
-               line 8 cols 18-18 is '-'
6               line 8 cols 20-20 is T_IntConstant (token value: 6)
;               line 8 cols 21-21 is ';'
b               line 9 cols 4-4 is T_Identifier (token address: 3)
=               line 9 cols 6-6 is '='
3               line 9 cols 8-8 is T_IntConstant (token value: 3)
;               line 9 cols 9-9 is ';'
a               line 10 cols 4-4 is T_Identifier (token address: 1)
=               line 10 cols 6-6 is '='
b               line 10 cols 8-8 is T_Identifier (token address: 3)
+               line 10 cols 10-10 is '+'
2               line 10 cols 12-12 is T_IntConstant (token value: 2)
;               line 10 cols 13-13 is ';'
}               line 11 cols 1-1 is '}'
```

- 아래(tests/samples/comment.frag)와 같이 완성된 프로그램이 아니라 어휘 분석기의 동작을 테스트하기 위해 만든 코드 조각들도 제공됩니다.

```
// KNU is the best!!!!
// here is a comment
/* here is a simple comment */
/* here
is
a
multi-line
comment
*/

/* here are some nasty comments: */
/* ///// /// */
/*******/

/*   /* an almost-nested comment */
"all done!"
```

위 코드 조각에 대한 출력은 아래와 같아야 합니다.

```
"all done!"    line 16 cols 1-11 is T_StringConstant (token value: "all done!")
```

- 제공된 Java 스켈레톤 코드(src/knu/compiler/Main.java)는 완성된 Decaf 언어 어휘 분석기에서 아래 부분에 대한 코드가 제거된 코드입니다. 여러분들이 이 부분을 구현하여 Decaf 언어 어휘 분석기를 완성한 후 코드를 제출하시면 됩니다.
  - 16진수로 입력된 정수형 상수 처리
  - 실수형 상수 처리
  - 식별자 처리
  - 주석 처리
- 예시 프로그램(tests/samples/*.decaf 또는 tests/samples/*.frag)들과 여러분들이 작성한 코드가 출력해야 하는 예시 출력(tests/samples/*.out)들은 압축 파일에 포함되어 있으니 확인하시고 과제를 수행하시기 바랍니다.