VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Nguyên Lý Ngôn Ngữ Lập Trình (PPL)

PPL1 - HK242

Task 1 - LEXER

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

TP. HỒ CHÍ MINH, THÁNG 1/2025

Mục lục

1 Thiết kế lexer ngôn ngữ TienJS

1.1 Program structure

Đoạn code

```
number @array = <1, 0, -1>;
func number foo(a : number, b : bool):
    a <- a + 1;
    for (number c <- 2; > 3; #2) {
        if c = 2 {
            b <- [0]@array * 2 > a;
            continue;
        }
    }
    return a;
endfunc

func void main():
    number a <- 0;
    a = a ** call foo <- a, T;
    call printNumber <- a;
endfunc
```

1.2 Code Lexical structure trong file main/VoTien.g4

```
//! ------------------------- Lexical structure ---------------------- TODO KeyWord

// TODO KeyWord

// TODO Operators

// TODO Separators

// TODO Identifiers

// TODO Literal

// TODO SKIP

// TODO ERROR
ERROR_CHAR: . {raise ErrorToken(self.text)};
UNCLOSE_STRING:
ILLEGAL_ESCAPE:
//!  ------------------------- end Lexical structure ------------------ //
```

1.3 Lexical structure

1.3.1 Characters set

Một chương trình TienJS là một dãy các ký tự thuộc bộ ký tự ASCII. Khoảng trắng (`' '`), tab (`'\t'`), backspace (`'\b'`), form feed (`'\f'`), carriage return (`'\r'`), và newline (`'\n'`) đều được coi là các ký tự trống (whitespace). Trong TienJS, ký tự `'\n'` được sử dụng làm ký tự xuống dòng.

Định nghĩa về dòng này có thể được dùng để xác định số dòng do trình biên dịch TienJS tạo ra.

1.3.2 Program comment

Có hai loại chú thích (comment) trong TienJS: chú thích khối (block) và chú thích dòng (line).

Chú thích khối bắt đầu bằng `/*` và bỏ qua tất cả các ký tự (trừ khi gặp cuối tệp – EOF) cho đến khi gặp `*/`.

Chú thích dòng bỏ qua tất cả các ký tự kể từ `//` đến cuối dòng hiện tại, nghĩa là bỏ qua cho đến khi gặp dấu xuống dòng hoặc cuối tệp.

```
/* This is a block comment, that
may span in many lines*/
a <- 5; //this is a line comment
```

Các quy tắc sau đây được áp dụng trong TienJS:

- Chú thích không được lồng nhau.
- Cặp `/*` và `*/` không mang ý nghĩa đặc biệt trong bất kỳ chú thích dòng nào.
- `//` không mang ý nghĩa đặc biệt trong bất kỳ chú thích khối nào.

Ví dụ:

```
/* This is a block comment so // has no meaning here */
//This is a line comment so /* has no meaning here
```

1.3.3 Identifiers

Các **định danh (identifiers)** được dùng để đặt tên cho biến, hằng, lớp, phương thức và tham số. Mỗi định danh phải bắt đầu bằng một chữ cái (A–Z hoặc a–z) hoặc dấu gạch dưới (_), và có thể chứa chữ cái, dấu gạch dưới, hoặc chữ số (0–9).

TienJS là ngôn ngữ **phân biệt chữ hoa/thường (case-sensitive)**, do đó các định danh sau đây được xem là khác nhau: `PrintLn`, `println`, và `PRINTLN`.

Định danh bắt đầu bằng ký tự @ (At identifiers) là một loại đặc biệt, dùng để **khai báo các biến toàn cục (global variables)** trong một chương trình. Định danh này bắt đầu bằng ký tự `@` và theo sau là một dãy **không rỗng** các chữ cái, dấu gạch dưới và chữ số. Ví dụ: `@globalVar1`, `@_config`, v.v.

1.3.4 Keywords

Trong TienJS, từ khóa phải bắt đầu bằng một chữ cái thường `(a-z)`. Danh sách các từ khóa hợp lệ trong TienJS như sau:

```
T         continue   F      if
else      for        bool   number
return    string     bool   func
endfunc   call       bool   func
```

1.3.5 Operators

Dưới đây là danh sách các toán tử hợp lệ:

```
+    -   *    =
>    <   **   #
<-
```

Ý nghĩa của các toán tử này sẽ được giải thích trong các phần tiếp theo.

1.3.6 Separators

Dưới đây là danh sách các Separators hợp lệ:

```
(  )  [  ]
<  ,  ;  :
{  }  >
```

1.3.7 Number literal

Trong TienJS, kiểu `number` bao gồm cả **số nguyên (integer)** và **số thực (float)**:

- **Số nguyên (integer):**
  - Luôn được biểu diễn ở dạng thập phân (cơ số 10).
  - Là một chuỗi các chữ số (`0-9`) có độ dài ít nhất 1.
  - Ví dụ: `0, 100, 255, 2500`.
- **Số thực (float):**
  - Gồm ba phần: *phần số nguyên*, *phần thập phân* và *phần mũ*.
  - *Phần số nguyên*: một hoặc nhiều chữ số.
  - *Phần thập phân*: một dấu chấm (`.`) có thể kèm các chữ số phía sau.
  - *Phần mũ*: bắt đầu bằng `E` hoặc `e`, tùy chọn `+` hoặc `-`, rồi đến một dãy không rỗng các chữ số.
  - Phần thập phân hoặc phần mũ có thể được lược bỏ, nhưng **không thể lược bỏ cả hai** cùng lúc.
  - Ví dụ hợp lệ: `9.0, 12e8, 1., 0.33E-3, 128e+42`.
  - Ví dụ không hợp lệ: `.12` (thiếu phần số nguyên), `143e` (không có chữ số sau `e`).

1.3.8 Boolean Litera

Một **boolean literal** là một trong hai giá trị: `T` hoặc `F`. Các literal này thuộc kiểu dữ liệu `bool`.

1.3.9 String Literals

Một **string literal** bao gồm 0 hoặc nhiều ký tự, được đặt trong cặp dấu ngoặc kép (`" "`). Có thể sử dụng các *escape sequence* (liệt kê bên dưới) để biểu diễn các ký tự đặc biệt trong chuỗi.

Nếu xuất hiện ký tự xuống dòng hoặc ký tự `EOF` bên trong một **string literal**, điều đó sẽ gây lỗi biên dịch (compile-time error).

Các escape sequence được hỗ trợ:

- `\b` backspace
- `\f` form feed
- `\r` carriage return
- `\n` newline
- `\t` horizontal tab
- `\"` dấu ngoặc kép (")
- `\\` dấu gạch chéo ngược (\)

Ví dụ hợp lệ về chuỗi:

```
"This is a string containing tab \t"
"He asked me: \"Where is John?\""
```

Mỗi **string literal** đều thuộc kiểu dữ liệu `string`.

1.3.10 các lỗi

- **ERROR_TOKEN** với `<unrecognized char>` (lexeme): xảy ra khi bộ phân tích từ (lexer) phát hiện một ký tự không được nhận dạng.
- **UNCLOSE_STRING** với `<unclosed string>` (lexeme): xảy ra khi lexer phát hiện một chuỗi không kết thúc (unterminated string). Lưu ý rằng `<unclosed string>` không bao gồm dấu ngoặc kép mở đầu (").
- **ILLEGAL_ESCAPE** với `<wrong string>` (lexeme): xảy ra khi lexer phát hiện một escape sequence không hợp lệ trong chuỗi. `<wrong string>` được lấy từ phần đầu chuỗi (không bao gồm dấu ngoặc kép mở) cho đến vị trí escape không hợp lệ.

```
# main/lexererr.py
class LexerError(ABC, Exception):
    pass
class ErrorToken(LexerError):
    def __init__(self, s):
        self.message = "Error Token " + s
class UncloseString(LexerError):
    def __init__(self, s):
        self.message = "Unclosed String: " + s
class IllegalEscape(LexerError):
    def __init__(self, s):
        self.message = "Illegal Escape In String: " + s
```