Compilers Scanner

Team members

成員	貢獻
資工三 108590043 李永祺	虛擬機操作,範例查詢
資工三 108590045 廖永誠	程式碼邏輯撰寫

Environment

- 使用Ubuntu20.04.1
- 需有以下flex套件

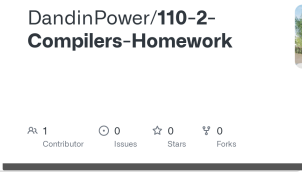
```
sudo apt-get install flex
```

- Github地址

110-2-Compilers-Homework/hw2 at main · DandinPower/110-2-Compilers-Homework

Contribute to DandinPower/110-2-Compilers-Homework development by creating an account on GitHub.

https://github.com/DandinPower/110-2-Compilers-Homework/tree/main/hw2



Target Language

- 使用Standford CS143課程所提供的Coolc作為本次作業的target language
- 該語言的Document

http://web.stanford.edu/class/cs143/materials/cool-manual.pdf

Lex Structure

- Integers
  - 0-9組成的整數
- strings
  - 被包括在" "裡的字串
- Keywords
  - 如class,else,false,fi,if,while....的預留關鍵字
- Whitespace
  - \n\f\r\t\v

一些基本的架構

- Classes

```
class <type> [ inherits <type> ] {
  <feature_list>
};
```

- Attributes

```
<id> : <type> [ <- <expr> ];
```

- Methods

```
<id>(<id> : <type>,...,<id> : <type>): <type> { <expr> };
```

- Assign

```
<id> <- <expr>
```

- Dispatch

```
<expr>.<id>(<expr>,...,<expr>)
```

- Conditions

```
if <expr> then <expr> else <expr> fi
```

- Loops

```
while <expr> loop <expr> pool
```

- Blocks

```
{ <expr>; ... <expr>; }
```

- Let

```
let <id1> : <type1> [ <- <expr1> ], ..., <idn> : <typen> [ <- <exprn> ] in <expr>
```

- Case

```
case <expr0> of
  <id1> : <type1> => <expr1>;
  . . .
  <idn> : <typen> => <exprn>;
esac
```

- New

```
new <type>
```

- Isvoid

```
isvoid expr
```

- Comparison

```
expr1 <op> expr2
```

使用說明

- 編譯scanner

```
flex cool-token.l
gcc -o scanner.out lex.yy.c
```

- 使用方法一

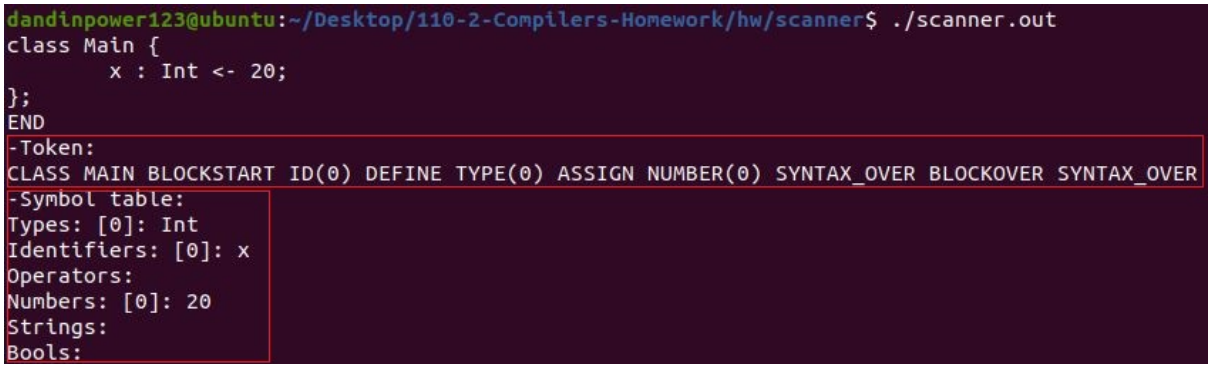
```
./scanner.out
```

  - 之後即可輸入直到輸入錯誤的情況或輸入EOF

- 使用方法二

```
./scanner.out < test.txt
```

輸出說明



- 這邊以一個簡單的範例來示範

```
class Main {
    x : Int <- 20;
};
```

- 輸出分為Token轉換以及Symbol Table

```
-Token:
CLASS MAIN BLOCKSTART ID(0) DEFINE TYPE(0) ASSIGN NUMBER(0) SYNTAX_OVER BLOCKOVER SYNTAX_OVER
```

```
-Symbol table:
Types: [0]: Int
Identifiers: [0]: x
Operators:
Numbers: [0]: 20
Strings:
Bools:
```

錯誤判斷

- 錯誤時會中斷scanner並拋錯
- 情況一
  - 錯誤情況可觀察標紅的地方註解的結尾符號沒有對應的開始符號

```
class StackCommand {

   getChar(): String {
      "Called from base class"
   };

   execute(node: StackNode): StackNode {
      let ret: StackNode in {
         (new IO).out_string("Undefined execution!\n");
         ret;*)
      }
   };

   getNumber(): Int {
      0
   };
};
```

```
ERROR:unmatched (*
-Token:
CLASS TYPE(0) BLOCKSTART ID(0) ITEMSTART ITEMOVER DEFINE TYPE(1) BLOCKSTART STRI
ITEMSTART NEW TYPE(3) ITEMOVER DOT ID(4) ID(5) ITEMSTART STRING(1) ITEMOVER SYNT
-Symbol table:
Types: [0]: StackCommand,[1]: String,[2]: StackNode,[3]: IO
Identifiers: [0]: getChar,[1]: execute,[2]: node,[3]: ret,[4]: out,[5]: string
Operators:
Numbers:
Strings: [0]: "Called from base class",[1]: "Undefined execution!\n"
Bools:
```

- 情況二
  - 錯誤情況可觀察標紅的地方註解的部分包含了檔案結束EOF符號

```
class StackCommand {

   getChar(): String {
```

```
      "Called from base class"
   };

   execute(node: StackNode): StackNode {
      let ret: StackNode in {
         (new IO).out_string("Undefined execution!\n");
         ret;
      }
   };

   getNumber(): Int {
      0
   };
};(* Hello
world
```

```
ERROR:EOF in comment body
-Token:
CLASS TYPE(0) BLOCKSTART ID(0) ITEMSTART ITEMOVER DEFINE TYPE(1) BLOCKSTART STRING(0) BLOCKOVER S
ITEMSTART NEW TYPE(3) ITEMOVER DOT ID(4) ID(5) ITEMSTART STRING(1) ITEMOVER SYNTAX_OVER SYNTAX_OV
SYNTAX_OVER BLOCKOVER SYNTAX_OVER
-Symbol table:
Types: [0]: StackCommand,[1]: String,[2]: StackNode,[3]: IO,[4]: Int
Identifiers: [0]: getChar,[1]: execute,[2]: node,[3]: ret,[4]: out,[5]: string,[6]: getNumber
Operators:
Numbers: [0]: 0
Strings: [0]: "Called from base class",[1]: "Undefined execution!\n"
Bools:
```

- 情況三
  - 錯誤情況可觀察標紅的地方出現了未知的token

```
class StackCommand {

   getChar(): String {
      "Called from base class"
   };

   execute(node: StackNode): % StackNode {
      let ret: StackNode in {
         (new IO).out_string("Undefined execution!\n");
         ret;
      }
   };

   getNumber(): Int {
      0
   };
};
```

```
ERROR:unmatched token
-Token:
CLASS TYPE(0) BLOCKSTART ID(0) ITEMSTART ITEMOVER DEFINE TYPE(1)
-Symbol table:
Types: [0]: StackCommand,[1]: String,[2]: StackNode
Identifiers: [0]: getChar,[1]: execute,[2]: node
Operators:
Numbers:
Strings: [0]: "Called from base class"
Bools:
```