COOL 语言词法分析器开发报告

Compiler Principle Assignment

姓名: 朱彦渊，姚文顶
学号: 20238132022，20238132007
班级: 物联网 2 班

2025 年 10 月 30 日

摘要

本文档详细记录了 COOL (Classroom Object-Oriented Language) 词法分析器的设计与实现过程。报告首先深入阐述 Flex 词法分析器的工作原理，包括有限状态自动机理论、模式匹配机制和状态转换过程；然后详细说明词法规则的实现方法，涵盖关键字、标识符、字符串、注释及错误处理；最后通过完整的测试验证，包括集成测试，展示词法分析器与编译器其他组件的协作以及最终程序的正确运行。

0.1 开发环境

0.1.1 硬件配置

- **CPU**: Intel Core i5-12600KF @ 3.70GHz
- **内存**: 16GBx2 DDR4
- **硬盘**: 1024GB SSD

0.1.2 软件环境

- **操作系统**: Kubuntu 24.04.2 LTS x86_64
- **内核版本**: 6.11.0-29-generic
- **Flex 版本**: flex 2.6.4
- **G++ 版本**: g++ (Ubuntu 13.3.0-6ubuntu2 24.04) 13.3.0
- **Make 版本**: GNU Make 4.3
- **其他工具**: SPIM Version 6.5 of January 4, 2003. 编辑器: Kate

0.1.3 项目目录结构

```
~/Desktop/student-dist/assignments/PA2
|-- cool.flex
|-- test.cl
|-- lexer
|-- cool-lex.cc
`-- Makefile
```

0.1.4 环境配置过程

没啥好配置的，相关组件都在先前就已经添加到环境变量中，但是作业提供的 mycoolc 是 csh 脚本，需要修改。(因为我是 bash) 由于修改后把./parser 等换成了 parser, 所以无需使用符号链接。修改后的 mycoolc 如下:

```
./lexer "$@" | parser "$@" | semant "$@" | cgen "$@"
```

Listing 1: COOL 编译器驱动脚本

代码 1 展示了编译器的管道流程。

1 Flex 词法分析器原理

1.1 Flex 工作流程

Flex 工作流程如图 1 图 2 所示:

1.2 有限状态自动机（FSA）原理

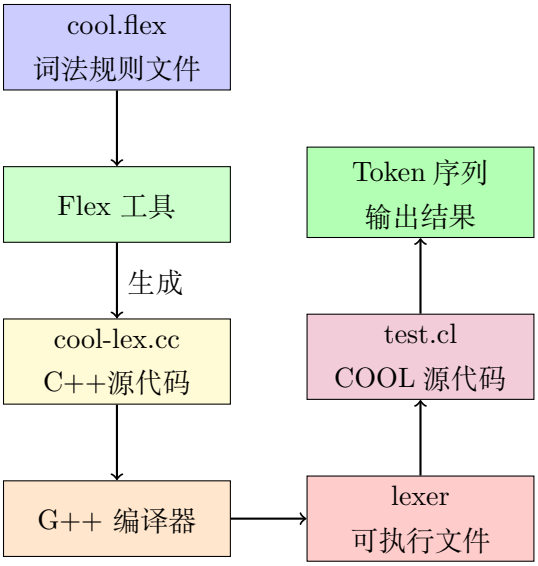

图 1: Flex 词法分析器生成与工作流程（工程上）

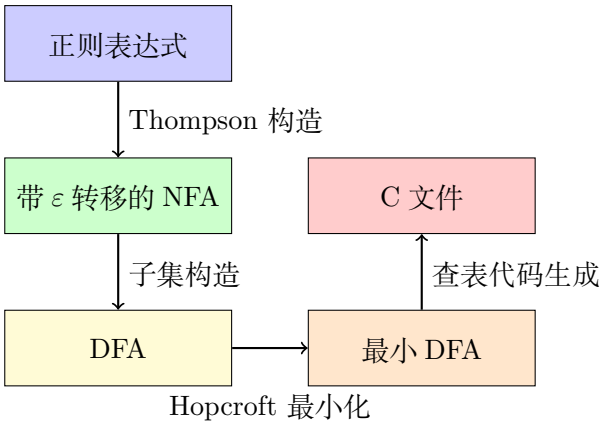

图 2: Flex 词法分析器生成与工作流程（原理上）

有限状态自动机（Finite State Automaton, FSA）是词法分析器的理论基础，它是描述正则语言的数学模型。

1.2.1 正则表达式的基本概念

基本原子

符号	含义	举例
`.`	换行外任意单字符	`a.c` 匹配 `abc`、`aqc` 等
`[abc]`	字符类：a 或 b 或 c	`[abc]`
`[^0-9]`	否定字符类：非数字	`[^0-9]+`

量词（重复）

符号	含义	举例
`*`	0 次或多次	`a*`
`+`	1 次或多次	`\d+`
`?`	0 或 1 次（可有可无）	`colou?r`
`{n,m}`	n 到 m 次	`a{2,5}`
`{n,}`	至少 n 次	`[0-9]{3,}`

断言（零宽）

符号	说明
`^`	行首（不在 `[...]` 内时）
`$`	行尾
`\b`	单词边界
`\B`	非单词边界

分组与分支

符号	说明
`(...)`	捕获组；整体当原子
`(?:...)`	非捕获组（Flex 不支持）
`|`	分支“或”
`\1 \2`	反向引用（Flex 不支持）

```
<span> <b> This is a test. </b> </span>
```

图 3: 使用正则表达式<.+?>捕获 html 标签

实战例子（捕获 html 标签和 ipv4 地址） 如果使用<.+>会捕获一整行，因为正则表达式默认的是最长匹配原则，加上? 后改为惰性匹配，从而成功捕获到标签。

```
255.234.0.1
0.0.0.0
192.168.0.1
1.2.3
256.0.2.1
```

图 4: 使用正则表达式捕获 ipv4 地址

```
\b((25[0-5]|2[0-4]\d|[0-1]?\d\d?)\.){3}((25[0-5]|2[0-4]\
    d|[0-1]?\d\d?))\b
```

Listing 2: 捕获 ipv4 地址所用的正则表达式

ipv4 地址由 4 段组成，前面三段可以看作是 0 到 255 的数字加上”.”，后面一段为 0 到 255 的数字。如果前两个数字为 25, 那么第三个数字的范围为 0 到 5; 如果第一个数字为 2 且第二个数字为 0 到 4 则第三位为任意数字; 如果第一位数字为 0 到 1 则后两位为任意数字，由于不一定为三位数，所以加上”?“表示可有可无。使用`\b` 表示单词边界 (word boudary)。Flex 没有 `\b`，可用首尾上下文代替：`(?^|[\w])word($|[\w])`，其中 `[\w]` 等价于 `[A-Za-z0-9_]`。

使用了 https://regex101.com/来测试正则表达式。

1.2.2 有限状态自动机基本概念

定义： 有限状态自动机是一个五元组 $M = (Q, \Sigma, \delta, q_0, F)$，其中：

- Q：有限状态集合
- Σ：有限输入字母表
- δ：状态转移函数，$\delta : Q \times \Sigma \to Q$
- q_0：初始状态，$q_0 \in Q$
- F：接受状态集合，$F \subseteq Q$

关键概念说明：

- **状态（State）**：系统在某一时刻的配置
- **转换（Transition）**：在输入符号驱动下的状态变化
- **接受状态（Accepting State）**：标识成功识别的模式

1.2.3 NFA 与 DFA 的区别

表 1: NFA 与 DFA 对比

特性	NFA（非确定有限自动机）	DFA（确定有限自动机）
状态转移	多值函数	单值函数
ε 转移	允许	不允许
同一输入的多路径	允许	不允许
实现复杂度	较高	较低
运行效率	较低	较高

数学定义区别：

- **NFA：** $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$
- **DFA：** $\delta : Q \times \Sigma \to Q$

1.2.4 Flex 将正则表达式转换为 NFA

Flex 使用 Thompson 构造法将正则表达式转换为 NFA。Thompson 构造法通过以下五个基本构件组合成整个正则表达式的 NFA：(和搭积木差不多)

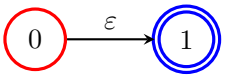

图 5: 空串 (ε) 的 NFA 构造

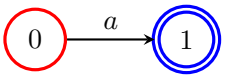

图 6: 单个字符 (a) 的 NFA 构造

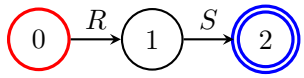

图 7: 连接操作 (RS) 的 NFA 构造

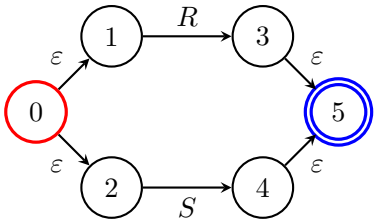

图 8: 选择操作 $(R|S)$ 的 NFA 构造

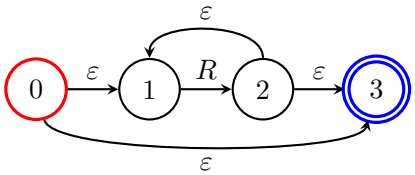

图 9: 克林闭包 (R^*) 的 NFA 构造

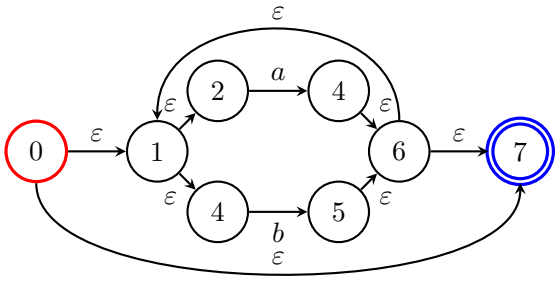

图 10: 例子：(a|b)*的 NFA 构造

1.2.5 从 NFA 到 DFA 的子集构造法

子集构造法（Subset Construction）将 NFA 转换为等价的 DFA：

算法步骤：

1. **计算 ε-闭包**：从状态 q 出发，仅通过 ε 转移能到达的所有状态集合
2. **初始状态**：$Q^0_{DFA} = \varepsilon\text{-closure}(q^0_{NFA})$
3. **状态转移**：对于每个状态集合 S 和每个输入符号 a：
   - $T = \bigcup_{q \in S} \delta_{NFA}(q, a)$
   - $\delta_{DFA}(S, a) = \varepsilon\text{-closure}(T)$
4. **接受状态**：任何包含 NFA 接受状态的 DFA 状态都是接受状态

总结：从初始状态开始求闭包，得到起始状态。然后对于字母表中的每个字符 a，计算 $move(T, a)$，即从 T 中的所有状态出发，通过输入 a 能够到达的状态集合，然后再计算 $\varepsilon closure(move(T, a))$，这是新的 DFA 状态。重复上述步骤直到不出现新状态为止。使用状态表来记录 DFA 的所有状态和转移关系。

示例：将前述 (a|b)*(图 10) NFA 转换为 DFA：

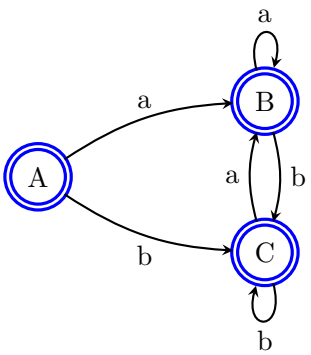

图 11: $(a|b)*$ 的 DFA（由图 10 中的 NFA 转换）

状态转移表和形式化定义 该 DFA 可以形式化定义为五元组 $(Q, \Sigma, \delta, q_0, F)$：

- $Q = \{A, B, C\}$ (状态集合)
- $\Sigma = \{a, b\}$ (字母表)
- $\delta : Q \times \Sigma \to Q$ (转移函数，见表 2)

当前状态	输入 a	输入 b
A	B	C
B	B	C
C	B	C

表 2: DFA 状态转移表

DFA 状态	对应的 NFA 状态集合	接受状态
A	$\{0, 1, 2, 4, 7\}$	是
B	$\{1, 2, 3, 4, 6, 7\}$	是
C	$\{1, 2, 4, 5, 6, 7\}$	是

表 3: DFA 状态详细说明

- $q_0 = A$ (起始状态)
- $F = \{A, B, C\}$ (接受状态集合)

1.2.6 DFA 的最小化过程

DFA 最小化通过合并等价状态来减少状态数量:

算法步骤 (Hopcroft 算法):

1. **初始划分**: 将状态划分为接受状态和非接受状态
2. **迭代细化**: 对于每个划分 P 和每个输入符号 a, 如果状态 p, q 在 a 上的转移目标属于不同的划分, 则分离 p 和 q
3. **终止条件**: 划分不再变化

最小化优势:

- 减少内存占用
- 提高匹配速度
- 简化自动机结构

1.2.7 为什么 DFA 比 NFA 效率更高

运行时间复杂度:

- **NFA:** $O(|Q|^2 \times |s|)$, 其中 $|s|$ 是输入字符串长度
- **DFA:** $O(|s|)$, 每个输入字符只需一次状态转移

效率差异原因:

1. **确定性**: DFA 每个状态-输入对只有唯一的下一状态, 无需回溯
2. **无 ε 转移**: DFA 不需要处理空转移, 减少状态维护开销
3. **简单实现**: DFA 可以用简单的查表法实现

说白了就是 NFA 路径太多需要不断穷举和回溯。以及 NFA 使用的数据结构是图, 需要动态分配内存, 硬件不友好; DFA 使用的数据结构是表, 顺序访问, 硬件友好。

1.2.8 识别整数的自动机示例

考虑正则表达式: `[0-9]+`, 用于识别非负整数。首先将 `[0-9]+`转变为 `[0-9][0-9]*`, 然后使用 Thompson 构造法将其转变为 NFA:

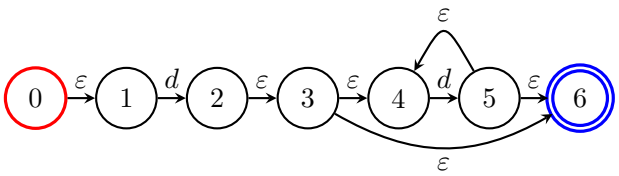

图 12: $[0-9]+$ 的 Thompson NFA 构造, 图中的 d 表示数字 digit, 即 $[0-9]$.

接着开始求闭包, 得到如下状态表:

当前状态	输入 digit
A	B
B	C
C	C

表 4: DFA 状态转移表

1.3 模式匹配机制

DFA 状态	对应的 NFA 状态集合	接受状态
A	$\{0, 1\}$	否
B	$\{2, 3, 4, 6\}$	是
C	$\{4, 5, 6\}$	是

表 5: DFA 状态详细说明

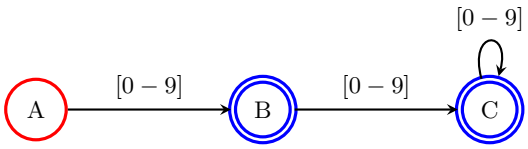

图 13: $[0-9]+$ 的 DFA

1.3.1 最长匹配原则（Longest Match）

```
%%
"+"        { return PLUS; }
"++"       { return PLUS2; }     // 匹配2个字符
"+++"      { return PLUS3; }         // 匹配3个字符
%%
```

Listing 3: 示例规则 1

当输入”+++” 的时候，根据最长匹配原则，return PLUS3.

1.3.2 规则优先级 (First Match)

先定义的规则优先。也就是说要先定义关键字再定义标识符。如果先定义标识符，flex 会把所有的关键字都识别成标识符。规则优先级是处理二义性的关键。

当输入”class” 时，为什么匹配 CLASS 而不是标识符?

因为关键字的定义在标识符之前，优先匹配关键字。

```
%%
/* 先定义关键字 */
"if"       { return IF; }
"else"     { return ELSE; }
"while"    { return WHILE; }
"class"    { return CLASS; }
"public"   { return PUBLIC; }

/* 再定义标识符 */
[a-zA-Z_][a-zA-Z0-9_]* {
    printf("标识符: %s\n", yytext);
    return IDENTIFIER;
}
%%
```

Listing 4: 示例规则 2

1.3.3 变量解释

yytext 用于储存匹配的文本，比如在代码 4 中将输出匹配到的标识符。yyleng 是匹配长度，储存匹配到的字符串的长度。

1.4 状态与状态转换

Flex 允许定义多个状态来处理不同的上下文中的不同词法规则。INITIAL 状态是默认状态，不需要显式声明，所有规则默认都在这个状态下生效，所有没指定状态的规则都属于 INITIAL。独占状态用%x 声明，进入独占状态后，只有当前状态的规则有效，用于完全切换上下文，比如注释，字符串等。包容状态用%s 声明，进入包容状态后，当前状态和 INITIAL 状态的规则都有效，用于拓展上下文。

为什么需要状态?

没有状态无法分辨同一符号的不同语义。以字符串和注释处理为例，我就想在字符串里面输出”(*)”，如果没有状态，读取到”(*” 的时候编译器会认为注释开始了; 同理，当我在注释用使用”” 的时候也会引起编译器误判。

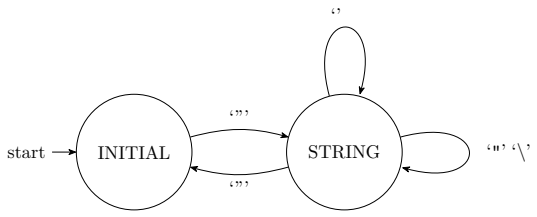

图 14: STRING 和 INITIAL 状态转换图 (简化版)

Flex 状态的基本语法如下:

```
<STATE_NAME>pattern {
    action
}
```

Listing 5: Flex 状态语法

2 实现细节

本节简要说明词法规则的实现思路。完整代码见附录。

2.1 关键字与标识符

基本要求：

- 关键字（如 class、if、while 等）是大小写不敏感的
- 布尔常量 true 和 false 的首字母必须小写
- TYPE_ID 以大写字母开头, OBJECT_ID 以小写字母开头
- 整数常量为数字序列

```
%%
[cC][lL][aA][sS][sS]        { return (CLASS); }
[eE][lL][sS][eE]            { return (ELSE); }
[fF][iI]                    { return (FI); }
%%
```

Listing 6: 大小不敏感的实现

直接为每个字符定义所有可能的大小写组合来实现大小写不敏感。注意 [cC] 就已经能匹配 c 或者 C，[c|C] 的写法是错误的，会匹配 c,|,C。同理布尔常量的匹配只需要使首字母小写其他字母大小写不敏感即可。如何处理关键字与标识符的优先级问题已经在前文说明，不再赘述。

2.2 字符串处理

基本要求：

- 字符串以双引号开始和结束，不能跨行
- 支持转义字符：\n, \t, \b, \f, \", \\
- 最大长度 1024 字符，不能包含空字符
- 需要使用 Flex 状态机（%x STRING）处理

在 INITIAL 状态下读取到", BEGIN(STRING), 然后在 STRING 状态下读到" 则 BEGIN(INITIAL)。这种写法要求当你要在字符串里面使用" 的时候必须使用\", 即不能嵌套引号，要不然可能导致解析错误。对于不能跨行的要求，只要在 STRING 状态下读到 EOF 直接报错就行。对于转义字符的要求，将转义字符写入缓冲区即可。对于最大长度的要求，只需要验证缓冲区的字符串长度就行。

2.3 操作符与注释

基本要求：

- 多字符操作符：<-、<=、=>
- 单行注释：--到行尾
- 多行注释：(* *)，支持嵌套
- 忽略空白字符，换行时更新行号

对于多字操作符，依据最长匹配原则和最先定义原则来处理，将多字操作符定义于单字操作符前面。对于嵌套注释，本质上和处理 STRING 没区别，但是需要对其进行计数，每次读入 (*就将注释深度+1, 读到*）就将注释深度-1, 只有注释深度为 0 才允许切回 INITIAL 而不是读到一次*）直接回到 INITIAL.（类似于 Stack）

2.4 错误处理

需要检测的错误：

- 未闭合的字符串、字符串过长、字符串中的空字符
- EOF 在字符串或注释中
- 未匹配的注释结束符*)
- 源代码中的非法字符

未闭合的字符串：读取到\n。注意回车键换行在 Flex 中是\n，而转移换行在 Flex 中是\\n。未匹配的注释结束符*) 只能是结束符号比开始符号多，要不然就是未闭合了，所以当在 INITIAL 状态下读取到游离的结束符就可以报这个错误了。非法字符，也就是前面所有规则都不匹配，使用. 捕获，根据先定义优先的规则，非法字符的捕获必须放在最后。

3 测试与验证

为了验证词法分析器的正确性，我设计了多个测试用例，并使用了项目提供的测试工具。

3.1 基础功能测试

测试目标：验证关键字、标识符、常量、操作符的正确识别。

测试用例（test_basic.cl）：

```
class Main {
    x : Int <- 42;
    flag : Bool <- true;
    main() : Int { x };
};
```

Listing 7: 基础功能测试

测试命令：

```
$ ./lexer test_basic.cl
```

实际输出结果：

```
#name "test_basic.cl"
#1 CLASS
#1 TYPEID Main
#1 '{'
#2 OBJECTID x
#2 ':'
#2 TYPEID Int
#2 ASSIGN
#2 INT_CONST 42
#2 ';'
#3 OBJECTID flag
#3 ':'
#3 TYPEID Bool
#3 ASSIGN
#3 BOOL_CONST true
#3 ';'
#4 OBJECTID main
#4 '('
#4 ')'
#4 ':'
#4 TYPEID Int
#4 '{'
#4 OBJECTID x
#4 '}'
#4 ';'
#5 '}'
#5 ';'
```

3.2 字符串与注释测试

测试目标：验证字符串转义字符、嵌套注释、各种错误检测。

测试用例（test_string.cl）：

```
(* 测试注释 (* 嵌套注释 *) *)
class Test {
    str1 : String <- "Hello\nWorld";  -- 转义字符
    str2 : String <- "Quote\"Test\"";
};
```

Listing 8: 字符串测试

测试命令与输出：

```
$ ./lexer test_string.cl
```

```
#name "test_string.cl"
#2 CLASS
#2 TYPEID Test
#2 '{'
#3 OBJECTID str1
#3 ':'
#3 TYPEID String
#3 ASSIGN
#3 STR_CONST "Hello\nWorld"
#3 ';'
#4 OBJECTID str2
#4 ':'
#4 TYPEID String
#4 ASSIGN
#4 STR_CONST "Quote\"Test\""
#4 ';'
#5 '}'
#5 ';'
```

3.3 错误处理测试

测试目标：验证各种错误情况的检测和报告。

测试 1：未闭合字符串

测试代码：

```
class Main { str : String <- "unclosed
```

输出：

```
#name "unclosed.cl"
#1 CLASS
#1 TYPEID Main
#1 '{'
#1 OBJECTID str
#1 ':'
#1 TYPEID String
#1 ASSIGN
#2 ERROR "Unterminated string constant"
```

测试 2：未匹配的注释结束符

测试代码：`class Main { x : Int; *) }`

输出：

```
#name "unclosed.cl"
#1 CLASS
#1 TYPEID Main
#1 '{'
#1 OBJECTID x
#1 ':'
#1 TYPEID Int
#1 ';'
#1 ERROR "Unmatched *)"
#1 '}'
```

测试 3：EOF 在注释中

测试代码：`(* comment without closing`

输出：

```
#name "unclosed.cl"
#2 ERROR "EOF in comment"
```

3.4 集成测试

测试目标：验证词法分析器能与编译器其他阶段（语法分析、语义分析、代码生成）正确协作，最终生成可运行的 MIPS 汇编代码。

测试程序（hello.cl）：

```
class Main inherits IO {
    main() : Object {
        out_string("Hello, COOL!\n")
    };
};
```

Listing 9: 集成测试程序

编译过程：

```
$ ./mycoolc hello.cl
```

运行结果：

```
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus
(larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: ../lib/trap.handler
Hello, COOL!
COOL program successfully executed
```

测试程序（queue.cl）（没错就是上个 PA）：

```
-- 队列节点
class Node inherits Object{
  data : Int;
  next : Node;

  init(d : Int, n : Node) : Node {
    {
      data <- d;
      next <- n;
      self;
    }
  };

  setNext( n : Node) : Object{
    {
      next <- n;
      self;
    }
  };

  getData() : Int{
    data
  };

  getNext() : Node{
    next
  };

};

```

```
--队列
class Queue inherits IO{
  head : Node;
  tail : Node;
  size : Int;
  init() :  Queue{
    {
      size <- 0;
      self;
    }
  };

  isEmpty() : Bool {
    isvoid head
  };

  enqueue (data : Int) : Object {
    {
    let n : Node in
    let newNode : Node <- (new Node).init(data, n) in
    {
      if isEmpty() then{
        head <- newNode;
        tail <- newNode;
      }
      else{
        tail.setNext(newNode);
        tail <- newNode;
      }fi;

      size <- size + 1;
    };
    }
  };

  dequeue () : Int{
    if isEmpty() then
      {
        out_string("Error! Can't dequeue an empty queue
     !\n");
        0;
      }
    else{
      let data2rm : Int <- head.getData() in {
        head <- head.getNext();
        size <- size - 1;

        data2rm;
      };
    }
    fi
  };


  front () : Int{
    if isEmpty() then
      {
        out_string("Error! Empty queue!\n");
        0;
      }
    else{
      head.getData();
    }
    fi
  };


  print () : Object {
    if isEmpty() then{
      out_string("empty queue!\n");
    }
    else{
      let curr : Node <- head in{
        out_string("the elemnets in the queue:\n");
        while not (isvoid curr) loop{
          out_int(curr.getData());
          out_string(" ");
          curr <- curr.getNext();
        }pool;
        out_string("\n");
        out_string("size of the queue: ");
        out_int(size);
        out_string("\n");
      };
    } fi

  };
};

-- test
class Main inherits IO {
  main() : Object {
    let q : Queue <- new Queue in {

      if q.isEmpty() then
        out_string("empty queue.\n")
      else
        out_string("not empty.\n")
      fi;

      out_string("enqueued : 1,2,3\n");
      q.enqueue(1);
      q.enqueue(2);
      q.enqueue(3);
      q.print();

      out_string("the front of the queue is : ");
      out_int(q.front());
      out_string("\n");
      q.print();

      q.dequeue();
      out_string("after dequeue: ");
      q.print();


    }
  };
```

```
148 };
```

Listing 10: 集成测试程序-队列

编译过程：

```
$ ./mycoolc queue.cl
```

运行结果：

```
SPIM Version 6.5 of January 4, 2003
Copyright 1990-2003 by James R. Larus
 (larus@cs.wisc.edu).
All Rights Reserved.
See the file README for a full
copyright notice.
Loaded: ../lib/trap.handler
empty queue.
enqueued : 1,2,3
the elemnets in the queue:
1 2 3
size of the queue: 3
the front of the queue is : 1
the elemnets in the queue:
1 2 3
size of the queue: 3
after dequeue: the elemnets in the queue:
2 3
size of the queue: 2
COOL program successfully executed
```

测试结论：

词法分析器成功识别了所有 Token，编译器顺利完成了语法分析、语义分析和代码生成，生成的 MIPS 汇编代码在 SPIM 模拟器上正确执行，输出了预期结果。这证明词法分析器的实现是正确和完整的。

4 遇到的问题与解决方案

对缓冲区不是很了解，反正加了%option noyywrap 就能跑了，能跑就行。以及依旧没办法写中文注释，我不知道为什么，反正写了就跑不了。还有关于 BEGIN() 怎么切换状态，没找到 BEGIN() 在哪个头定义的，问 ai 说在 lex.yy.c 里面，然而并没有看到这个文件。

5 总结

通过本次实验，我深入理解了词法分析的理论基础和 Flex 工具的工作原理。从有限状态自动机的理论出发，理解了 Flex 如何将正则表达式转换为高效的 DFA，如何进行模式匹配和状态转换。在实践中，我成功实现了一个功能完整且健壮的 COOL 语言词法分析器，特别是掌握了状态管理机制在处理复杂词法结构时的应用。通过完整的集成测试，我验证了词法分析器能够与编译器其他组件正确协作，最终生成可执行的 MIPS 代码。这次实验让我对编译器前端有了全面而深刻的认识。

A 附录：cool.flex 完整源码

```
1
2  %{
3  #include <cool-parse.h>
4  #include <stringtab.h>
5  #include <utilities.h>
6
7  /* The compiler assumes these identifiers. */
8  #define yylval cool_yylval
9  #define yylex  cool_yylex
10
11 /* Max size of string constants */
12 #define MAX_STR_CONST 1025
13 #define YY_NO_UNPUT /* keep g++ happy */
14
15 extern FILE *fin; /* we read from this file */
16
17
18 /* define YY_INPUT so we read from the FILE fin:
19  * This change makes it possible to use this scanner in
20  * the Cool compiler.
21  */
22 #undef YY_INPUT
23 #define YY_INPUT(buf,result,max_size) \
24     if ( (result = fread( (char*)buf, sizeof(char), max_size, fin)) < 0) \
25         YY_FATAL_ERROR( "read() in flex scanner failed");
26
27 char string_buf[MAX_STR_CONST]; /* to assemble string constants */
28 char *string_buf_ptr;
29
30 extern int curr_lineno;
31 extern int verbose_flag;
32 extern YYSTYPE cool_yylval;
33
34 static int comment_depth = 0;
35
36 /*
37  *  Add Your own definitions here
38  */
39
40 %}
41
42 /*
43  * Define names for regular expressions here.
```

```
 */


%option noyywrap

DARROW          =>
ASSIGN          <-
LE              <=
DIGIT           [0-9]
LOWER           [a-z]
UPPER           [A-Z]
LETTER          [a-zA-Z]
ALNUM           [a-zA-Z0-9_]
WHITESPACE      [ \t\r\f\v]

%x COMMENT
%x STRING

%%

"(*"    {
    BEGIN(COMMENT);
    comment_depth = 1;
}

<COMMENT>"(*"   { comment_depth++; }

<COMMENT>"*)"   {
    comment_depth--;
    if (comment_depth == 0) BEGIN(INITIAL);
}

<COMMENT>\n     { curr_lineno++; }

<COMMENT><<EOF>> {
    cool_yylval.error_msg = "EOF in comment";
    BEGIN(INITIAL);
    return ERROR;
}

<COMMENT>.      { }

"*)"    {
    cool_yylval.error_msg = "Unmatched *)";
    return ERROR;
}

"--".*          { }

{DARROW}        { return (DARROW); }
{ASSIGN}        { return (ASSIGN); }
{LE}            { return (LE); }

[cC][lL][aA][sS][sS]        { return (CLASS); }
[eE][lL][sS][eE]            { return (ELSE); }
[fF][iI]                    { return (FI); }
[iI][fF]                    { return (IF); }
[wW][hH][iI][lL][eE]        { return (WHILE); }
[iI][nN]                    { return (IN); }
[iI][nN][hH][eE][rR][iI][tT][sS]    { return (INHERITS); }
[iI][sS][vV][oO][iI][dD]        { return (ISVOID); }
[lL][eE][tT]                { return (LET); }
[lL][oO][oO][pP]            { return (LOOP); }
[pP][oO][oO][lL]            { return (POOL); }
[tT][hH][eE][nN]            { return (THEN); }
[cC][aA][sS][eE]            { return (CASE); }
[eE][sS][aA][cC]            { return (ESAC); }
[nN][eE][wW]                { return (NEW); }
[oO][fF]                    { return (OF); }
[nN][oO][tT]                { return (NOT); }

t[rR][uU][eE]               {
    cool_yylval.boolean = 1;
    return (BOOL_CONST);
}

f[aA][lL][sS][eE]           {
    cool_yylval.boolean = 0;
    return (BOOL_CONST);
}

{UPPER}{ALNUM}*     {
    cool_yylval.symbol = stringtable.add_string(yytext);
    return (TYPEID);
}

{LOWER}{ALNUM}*     {
    cool_yylval.symbol = stringtable.add_string(yytext);
    return (OBJECTID);
}

{DIGIT}+            {
    cool_yylval.symbol = stringtable.add_string(yytext);
    return (INT_CONST);
}

\"                  {
    BEGIN(STRING);
    string_buf_ptr = string_buf;
}

<STRING>\"          {
    BEGIN(INITIAL);
    *string_buf_ptr = '\0';
    if (string_buf_ptr - string_buf >= MAX_STR_CONST) {
        cool_yylval.error_msg = "String constant too long";
        return ERROR;
    }
    cool_yylval.symbol = stringtable.add_string(string_buf);
    return (STR_CONST);
}

<STRING>\n          {
    curr_lineno++;
    BEGIN(INITIAL);
    cool_yylval.error_msg = "Unterminated string constant";
    return ERROR;
}

<STRING><<EOF>>     {
    BEGIN(INITIAL);
    cool_yylval.error_msg = "EOF in string constant";
    return ERROR;
}

<STRING>\\\n        {
    curr_lineno++;
    if (string_buf_ptr - string_buf < MAX_STR_CONST - 1) {
        *string_buf_ptr++ = '\n';
    }
}

<STRING>\\n         {
    if (string_buf_ptr - string_buf < MAX_STR_CONST - 1) {
        *string_buf_ptr++ = '\n';
    }
}

<STRING>\\t         {
    if (string_buf_ptr - string_buf < MAX_STR_CONST - 1) {
        *string_buf_ptr++ = '\t';
    }
}

<STRING>\\b         {
    if (string_buf_ptr - string_buf < MAX_STR_CONST - 1) {
        *string_buf_ptr++ = '\b';
    }
}

<STRING>\\f         {
    if (string_buf_ptr - string_buf < MAX_STR_CONST - 1) {
        *string_buf_ptr++ = '\f';
    }
}

<STRING>\\[^\0]     {
    if (string_buf_ptr - string_buf < MAX_STR_CONST - 1) {
        *string_buf_ptr++ = yytext[1];
    }
}
```

```
206
207
208 <STRING>.           {
209     if (yytext[0] == '\0') {
210         cool_yylval.error_msg = "Null character in string";
211         BEGIN(INITIAL);
212         return ERROR;
213     }
214     if (string_buf_ptr - string_buf >= MAX_STR_CONST - 1) {
215         cool_yylval.error_msg = "String constant too long";
216         BEGIN(INITIAL);
217         return ERROR;
218     }
219     *string_buf_ptr++ = yytext[0];
220 }
221
222 "+"                 { return '+'; }
223 "-"                 { return '-'; }
224 "*"                 { return '*'; }
225 "/"                 { return '/'; }
226 "~"                 { return '~'; }
227 "<"                 { return '<'; }
228 "="                 { return '='; }
229 "."                 { return '.'; }
230 "@"                 { return '@'; }
231 ","                 { return ','; }
232 ":"                 { return ':'; }
233 ";"                 { return ';'; }
234 "("                 { return '('; }
235 ")"                 { return ')'; }
236 "{"                 { return '{'; }
237 "}"                 { return '}'; }
238
239 {WHITESPACE}+       { }
240 \n                  { curr_lineno++; }
241
242 .                   {
243     cool_yylval.error_msg = yytext;
244     return ERROR;
245 }
246
247 %%
```