

# MLIR 编译框架的使用与探索

吴非 519021910924

0.实验环境： Unbuntu 20.04 5.13.0-40-generic

1.实验过程：

首先编写[词法分析器](#)：

查看要求：变量名以字母开头；变量名由字母、数字和下划线组成；变量名中有数字时，数字应该位于变量名末尾。

能够识别“return”、“def”和“var”三个关键字

可以看到这两个情况都是以字母开头的。根据提示查看 isalpha 函数：判断是否是字母。isalnum 判断是否是字母和数字。

查看后面的官方代码，以 if 判断开头，因此我也通过 if (isalpha(lastChar))进入标识符|关键字的判断逻辑，这样就满足了[变量名以字母开头](#)的要求。之后进入 while 循环逐个读取判断[是否是字母、数组或下划线](#)，通过 bool 类型的 lastDigit 判断[是否如果出现数字后之后的字符是数字](#)，如果不是那么直接返回它，然后打印错误：

```
(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_1.tiny -emit=ast
Parse error (5, 8): expected 'identified' after 'var' declaration but has Token 95 '_'
Parse error (5, 8): expected 'nothing' at end of module but has Token 95 '_'
```

如果 while 循环结束没有执行这句返回错的标识符，那么就是三种关键字或者一个合法的标识符。直接先用 if 判断关键字即可。注释比较详细。

其次是[语法分析器](#)：

观察 TODO 的位置，首先 parseVarDeclaration 处主要写的是判断逻辑有三个 TODO，观察附近代码，parseDeclaration 中就有 tok\_var, tok\_identifier 的判断，直接复制即可。值得注意的是需要 loc 变量，搜索即可发现其他位置有这句 auto loc = lexer.getLastLocation();而此函数是得到现在 token 的开始位置的，显然这里直接也是复制这句就行。最后一个 TODO 位于 parseType()，而此函数在上述的第三个 TODO 处被调用，可以观察到这个函数是检查[2][3]或者<2,3>等的合法性的（并返回一个 type，存储具体的 shape，type->shape.push\_back）。

同样我根据对<>的代码模仿编写[]的:不同于<>，[2][3]需要判断一个[]和数字后改变一个 bool 变量 ismedium 从而辨别此时位于第一个[]还是第二个[]。其整体判断逻辑是 while 循环判断是否是数字，是数字后加入 type，得到下一个 token 后在当前循环判断是否是[]。

由于整个循环的条件是判断数字，循环内部单独处理[]的情况，因此如果出现%\*等奇怪的字符，while 循环会直接结束，而最后这个字符如果不是[]（[]中没有数字也合法），会报错：return parseError<VarType>("[", "to end type");注释也比较详细。

最后是[优化部分](#)：

阅读提示，其思想是，A=Transpose(Transpose(B)),A 相当于代码中的 op，

通过 getOperand 得到 A 的操作数，或者说是输入：(Transpose(B)，对应代码中的变量 transpose\_input)，对 transpose\_input 调用 getDefiningOp 得到它的操作符 Transpose（如果是  $A = \text{Transpose}(B * C)$ ，此时得到的就是  $*$ ），之后 step 2 对它用了一个智能指针的转换：dyn\_cast\_or\_null<TransposeOp>，就是如果得到的这个操作符是 TransposeOp，那么返回这个指针，否则返回 0。这样就实现了两个嵌套转置的判断。判断成功后进入 step 3，通过 replaceOp 操作把 op，也就是 A 整体替换为 Transpose(B) 的操作数，也就是 B 即可。

## 2.实验结果:

test1:

```
(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_1.tiny -emit=ast
Parse error (5, 8): expected 'identified' after 'var' declaration but has Token 95 '-'
Parse error (5, 8): expected 'nothing' at end of module but has Token 95 '-'
```

test2:

```
(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_2.tiny -emit=ast
Parse error (5, 8): expected 'identified' after 'var' declaration but has Token 99 'c'
Parse error (5, 8): expected 'nothing' at end of module but has Token 99 'c'
```

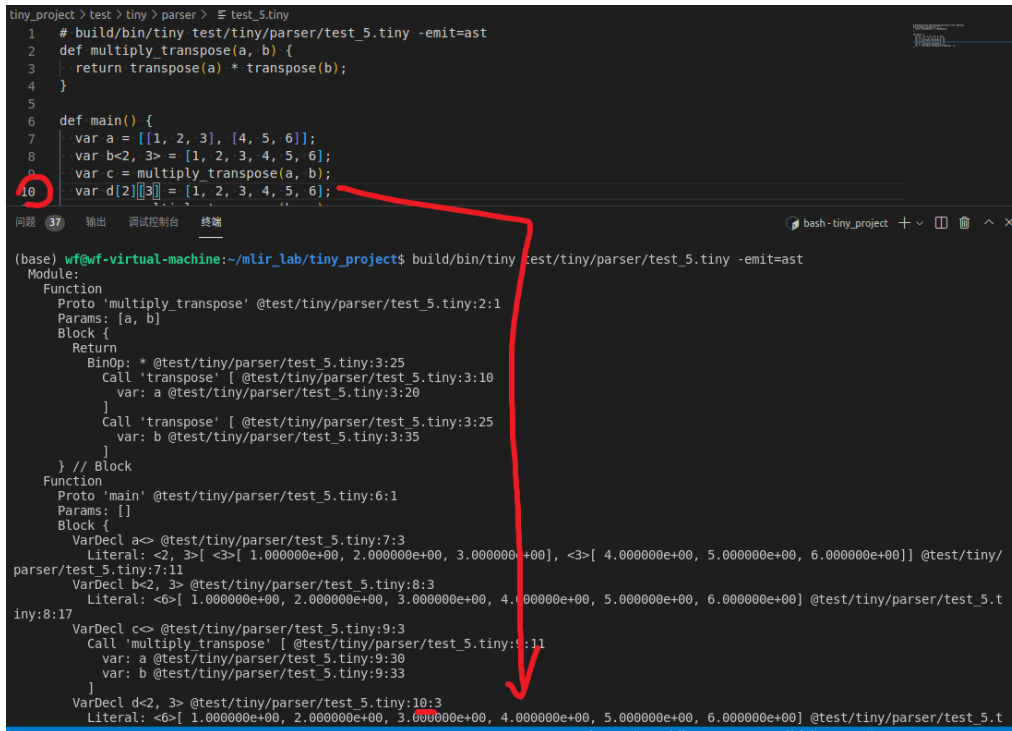
test3:

```
(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_3.tiny -emit=ast
Parse error (5, 8): expected 'identified' after 'var' declaration but has Token 99 'c'
Parse error (5, 8): expected 'nothing' at end of module but has Token 99 'c'
```

test4:

```
(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_4.tiny -emit=ast
Parse error (5, 9): expected 'identified' after 'var' declaration but has Token 95 '_'
Parse error (5, 9): expected 'nothing' at end of module but has Token 95 '_'
```

test5:



```
tiny_project > test > tiny > parser > test_5.tiny
1 # build/bin/tiny test/tiny/parser/test_5.tiny -emit=ast
2 def multiply_transpose(a, b) {
3   return transpose(a) * transpose(b);
4 }
5
6 def main() {
7   var a = [[1, 2, 3], [4, 5, 6]];
8   var b<2, 3> = [1, 2, 3, 4, 5, 6];
9   var c = multiply_transpose(a, b);
10  var d[2][3] = [1, 2, 3, 4, 5, 6];
11 }

(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_5.tiny -emit=ast
Module:
  Function
    Proto 'multiply_transpose' @test/tiny/parser/test_5.tiny:2:1
    Params: [a, b]
    Block {
      Return
        BinOp: * @test/tiny/parser/test_5.tiny:3:25
          Call 'transpose' [ @test/tiny/parser/test_5.tiny:3:10
            var: a @test/tiny/parser/test_5.tiny:3:20
          ]
          Call 'transpose' [ @test/tiny/parser/test_5.tiny:3:25
            var: b @test/tiny/parser/test_5.tiny:3:35
          ]
        } // Block
    } // Function
  Proto 'main' @test/tiny/parser/test_5.tiny:6:1
  Params: []
  Block {
    VarDecl a<> @test/tiny/parser/test_5.tiny:7:3
      Literal: <2, 3>[ <3>[ 1.000000e+00, 2.000000e+00, 3.000000e+00], <3>[ 4.000000e+00, 5.000000e+00, 6.000000e+00]] @test/tiny/
      parser/test_5.tiny:7:11
    VarDecl b<2, 3> @test/tiny/parser/test_5.tiny:8:3
      Literal: <6>[ 1.000000e+00, 2.000000e+00, 3.000000e+00, 4.000000e+00, 5.000000e+00, 6.000000e+00] @test/tiny/parser/test_5.t
    iny:8:17
    VarDecl c<> @test/tiny/parser/test_5.tiny:9:3
      Call 'multiply_transpose' [ @test/tiny/parser/test_5.tiny:::11
        var: a @test/tiny/parser/test_5.tiny:9:30
        var: b @test/tiny/parser/test_5.tiny:9:33
      ]
    VarDecl d<2, 3> @test/tiny/parser/test_5.tiny:10:3
      Literal: <6>[ 1.000000e+00, 2.000000e+00, 3.000000e+00, 4.000000e+00, 5.000000e+00, 6.000000e+00] @test/tiny/parser/test_5.t
```

```
tiny_project > test > tiny > parser > test_5.tiny
4 }
5
6 def main() {
7   var a = [[1, 2, 3], [4, 5, 6]];
8   var b<2, 3> = [1, 2, 3, 4, 5, 6];
9   var c = multiply_transpose(a, b);
10  var d[2][3] = [1, 2, 3, 4, 5, 6];
11  var e = multiply_transpose(b, c);
12  var f = multiply_transpose(transpose(a), c);
13 }
14

(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_5.tiny -emit=ast
Parse error (10, 7): expected 'identified' after 'var' declaration but has Token -7
Parse error (10, 7): expected 'nothing' at end of module but has Token -7
(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$
```

```
5
6 def main() {
7   var a = [[1, 2, 3], [4, 5, 6]];
8   var b<2, 3> = [1, 2, 3, 4, 5, 6];
9   var c = multiply_transpose(a, b);
10  var d[2][3] = [1, 2, 3, 4, 5, 6];
11  var e = multiply_transpose(b, c);
12  var f = multiply_transpose(transpose(a), c);
13 }
14

(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_5.tiny -emit=ast
Parse error (10, 10): expected ']' to end type but has Token 62 '>'
Parse error (10, 10): expected 'nothing' at end of module but has Token 62 '>'
(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$
```

```
5
6 def main() {
7   var a = [[1, 2, 3], [4, 5, 6]];
8   var b<2, 3> = [1, 2, 3, 4, 5, 6];
9   var c = multiply_transpose(a, b);
10  var d[2][3> = [1, 2, 3, 4, 5, 6];
11  var e = multiply_transpose(b, c);
12  var f = multiply_transpose(transpose(a), c);
13 }
14
```

base) wf@wf-virtual-machine:~/mlir\_lab/tiny\_project\$ build/bin/tiny test/tiny/parser/test\_5.tiny -emit=ast  
arse error (10, 13): expected ']' to end type but has Token 62 '>'  
arse error (10, 13): expected 'nothing' at end of module but has Token 62 '>'  
base) wf@wf-virtual-machine:~/mlir\_lab/tiny\_project\$

test6:

```
(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_6.tiny -emit=jit  
1.000000 8.000000 27.000000  
64.000000 125.000000 216.000000
```

test7:

未优化:

opt:

```
(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_7.tiny -emit=mlir -opt  
module {  
  tiny.func @main() {  
    %0 = tiny.constant dense<[[1.000000e+00, 2.000000e+00, 3.000000e+00], [4.000000e+00, 5.000000e+00, 6.000000e+00]]> : tensor<2x3xf64  
4> %1 = tiny.transpose(%0 : tensor<2x3xf64>) to tensor<3x2xf64>  
%2 = tiny.transpose(%1 : tensor<3x2xf64>) to tensor<2x3xf64>  
tiny.print %2 : tensor<2x3xf64>  
tiny.return  
}  
}
```

可以看到经过了两次转置

jit:

```
(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_7.tiny -emit=jit  
1.000000 2.000000 3.000000  
4.000000 5.000000 6.000000
```

优化后:

opt:

```
(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_7.tiny -emit=mlir -opt  
module {  
  tiny.func @main() {  
    %0 = tiny.constant dense<[[1.000000e+00, 2.000000e+00, 3.000000e+00], [4.000000e+00, 5.000000e+00, 6.000000e+00]]> : tensor<2x3xf6  
4> tiny.print %0 : tensor<2x3xf64>  
tiny.return  
}  
}
```

可以看到没有多余的两次转置了。

jit:

```
(base) wf@wf-virtual-machine:~/mlir_lab/tiny_project$ build/bin/tiny test/tiny/parser/test_7.tiny -emit=jit  
1.000000 2.000000 3.000000  
4.000000 5.000000 6.000000
```

矩阵结果与未优化一致。

### 3.总结:

本实验通过对 MLIR 编译框架中的词法分析，语法分析，转置冗余消除等代码的补充，锻炼了自己的代码能力，加深了对词法和语法分析器等的理解。

对 MLIR 的理解：提供一套中间模块，这个中间模块有两个作用：1. 对接不同的软件框架；2. 对接软件框架和硬件芯片。同时比较新颖的概念：方言，有方言内部和方言间的转换，这样提高了泛化和通用能力。

#### 4.分工：

由吴非同学单独完成。

联系方式：邮箱：[legendary@sjtu.edu.cn](mailto:legendary@sjtu.edu.cn) 手机：19821286671

#### 5.个人建议：

个人感觉任务难度和代码量都不是很大，可以一个人完成，小组作业懂得都懂。