lab1 实验报告

乔羿童 21311111

编译和测试方法

1. `cd Code/`：进入源代码和 `Makefile` 文件所在的路径下
2. `make`：使用 `make` 命令对程序代码进行编译构建
3. `./parser ../Test/xxx.cmm`：对指定的 C--代码 `xxx.cmm` 进行分析，注意测试文件的路径要正确。也可以修改 `Makefile` 文件中的 `test` 部分的内容，然后使用 `make test` 命令进行测试。第二种方式可以同时输出多个测试文件的测试结果。
   注：如果直接使用二进制文件parser测试遇到权限问题，使用chmod +x ./parser命令后再测试即可。

代码部分

注：对于某些语法错误，本人进行了错误信息的定制以满足输出样式。但是原有默认输出的报错语法信息不会被覆盖，因此测试时会存在重复冗余的报错信息！（如程序 `2.cmm` 和程序 `testcase_3`）（如非法浮点数同时报出词法和语法错误）但本人实现的程序的特点就是在符合语法产生式含义的情况下尽可能满足输出错误信息的要求。

本人在所给示例代码的基础上进行修改。经过测试，原有示例代码已经可以成功通过大部分用例，并且**用例中语法树的构建输出都不存在问题**。下面本人将对不能正常通过的用例进行代码补充并给出思路详解。**对于 `Test` 文件中的测试用例 `xx.cmm`，本人已经将结果输出到 `Test/out` 文件夹中**。

1. 程序 `2.cmm`：错误信息定位与补充

示例代码已经能够成功定位到错误的类型和行号并成功输出，本人通过对语法产生式的分析定位到了这两处错误，并增添 `yyerror` 中的语句信息，可以使得结果符合预期的样例输出（但**存在重复的报错信息**）。

```
1 Stmt :| IF LP Exp RP error ELSE Stmt  { $$ = createNode("Error", ENUM_SYN_NULL,
2           @$.first_line, 0, NULL);  yyerror("Missing \";\"."); yyerrok; }
3 Exp : | Exp LB error RB    { $$ = createNode("Error", ENUM_SYN_NULL,
4           @$.first_line, 0, NULL);  yyerror("Missing \"]\"");  yyerrok; }
```

2. 程序 `6.cmm`： 识别非法的八进制数和十六进制数

该用例出现的问题是存在非法的八进制、十六进制数，但是词法分析器不能够识别相应的内容（`09` 和 `0x3G`）为非法的八进制数和十六进制数，而是 `0` 和 `9`，导致语法错误。 本人的思路是在词法分析器中增添新的 `RE` 来识别这两种非法的数字，词法分析器识别到这种 `RE` 时，即刻报出错误。

```
OCT_ILLEGAL 0[0-7]*[89]+[0-9]*
HEX_ILLEGAL 0[xX][0-9a-fA-F]*[g-zG-Z]+[0-9a-zA-Z]*
{HEX_ILLEGAL}   {printf("Error type A at Line %d: Illegal hexadecimal number \'%
{OCT_ILLEGAL}   {printf("Error type A at Line %d: Illegal octal number \'%s\'.\n
```

3. 程序 `8.cmm`：识别非法的浮点数

该用例的解决思路同理程序 `6.cmm`，新增 `RE` 来识别错误的浮点数即可。本人给出了多种非法浮点数的情况，进行错误全面覆盖。

```
digit}*"."{digit}+[eE]  {printf("Error type A at Line %d: Illegal floating
point number \"%s\".\n", yylineno, yytext); lexError++;}
"."{digit}+    {digit}+"."   {digit}+"."{digit}*[eE]    // ..打印错误信息同上
{digit}+[eE][+-]?{digit}*    "."[eE][+-]?{digit}+  // ..(共定义6种非法float)
```

下面对 `External_test` 文件夹中的用例的解决进行叙述（同样，原示例代码中已经可以成功通过的不再赘述）：

4. 程序 `testcase_3` ：定位全部语法错误

直接进行测试时，仅有第 1 处语法错误 `struct V_stack test_stack == 1;` 能够被报出。 在变量定义 `VarDec` 处增添如下语句，第 4 处语法错误 `array[4+i] = array[5 */ 12];` 可以被成功定位。

```
VarDec: | error RB               { $$ = createNode("Error", ENUM_SYN_NULL,
          @$.first_line, 0, NULL); yyerror("Invalid expression."); yyerrok; }
```

之后多次尝试定位 `--` 和 `exit` 处的语法错误： `int extra_d = (d * 3) / (4 + 2) * (d -= 5);` 和 `exit(-1;`，尝试过新增 `token` `EXIT` 来定位 `exit`，但是并未成功。且 `Exp MINUS error` 处的语法错误也并没有被识别，于是本人判断是原示例代码中**高层产生式的 error 将底层的 error"屏蔽"**，导致错误信息无法输出。检查过后，本人**将下面的高层语法错误进行注释**，原有错误均可以被成功定位。

```
ExtDecList : /*  | VarDec error COMMA ExtDecList  { $$ = createNode("Error",
                    ENUM_SYN_NULL, @$.first_line , 0, NULL); yyerrok; } */
```

`ExtDecList` 表示**零个或多个**对一个变量的定义 `VarDec`，因此就会**将测试程序中的很多行的变量定义均包含进去**，导致底层的 `error` 无法被输出。 将其注释掉之后就可以定位到错误，可以选择是

否使用自定义的 `yyerror` 语句进行报错信息的打印（`error SEMI` 处的错误也许并不意味着缺失 `"）"`，因此本人没有定义专门的错误信息）。

```
Stmt: | error SEMI       {$$ = createNode("Error",NUM_SYN_NULL,
                              @$.first_line, 0, NULL);yyerrok; }
Exp:  | Exp MINUS error  { $$ = createNode("Error", ENUM_SYN_NULL,
     @$.first_line, 0, NULL);  yyerror("Invalid expression."); yyerrok; }
```

由于将高层 error 进行注释，会出现 `while` 语句错误识别的情况，因此本人将源代码中 `Exp : ID LP error RP` 修改为 `Exp : WHILE LP error RP` ， `while` 循环处的不应该有的语法错误便不会被错误输出。

5. 程序 `testcase_14` ： 定位不完整的 `/* */` 注释

运行此用例后程序不会终止，是注释 `/* */` 读取的问题，代码中最后未出现 `*/` 。以下是匹配到 `/*` 时会运行的代码，可见当源代码中不出现 `*/` 时， `while` 循环永远不终止，陷入死循环。

```
    char a = input();  char b = input();
    while (!(a == '*' && b == '/')) { a = b; b = input(); }
```

查阅 Flex 文档，得知 `input` 函数在遇到文件结尾 `EOF` 时返回 `0` ，据此可以为上述添加终止条件来避免死循环。

```
"/*"  { char a = input();  char b = input();
        while (!(a == '*' && b == '/') && b != 0) { a = b; b = input();}
        if(b == 0) {return MISSING_ANNOTATION_RIGHT; } }
```

当读取到文件末尾时发现没有收注释 `"*/"` 符号，词法分析器将返回给语法分析器错误标识 `token` ： `MISSING_ANNOTATION_RIGHT` ，在语法分析器可以识别到这个 `token` ，从而进行错误信息输出。

```
Stmt: | MISSING_ANNOTATION_RIGHT  { $$ = createNode("Error", ENUM_SYN_NULL,
           @$.first_line, 0, NULL); yyerror("Incomplet annotation."); yyerrok; }
```

注：所提供的样例输出第 33 行注释不完整，由于源程序 34 行存在空行，会导致实际输出有所不同！（因为进行注释完整性的判断需要读取到整个文件的末尾，所输出的错误信息的行号也就会随着文件的结束行号而变动）