编译原理实验一报告

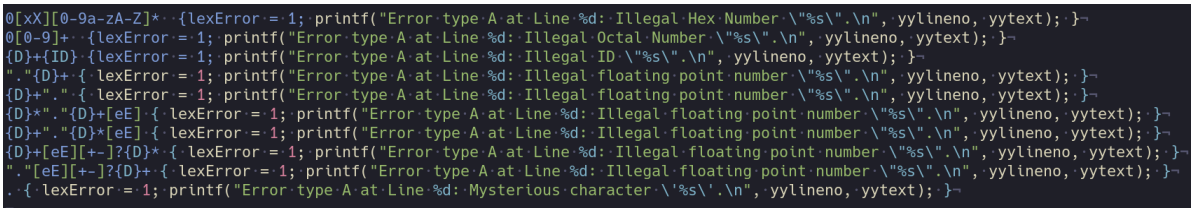
郑昊卓 1190200609

1. 程序功能

程序利用flex与bison实现了指导书中的对与C--语言的语法分析器。

程序在有参数的情况下会去读取参数中的文件，否则就去读取标准输入。在输入没有错误的情况下，程序会打印AST,否则程序会打印错误信息及其所在的行号。

1.1 词法分析中的具体错误



在flex文件底部加入特别设计的正则表达式可以处理特定类型的错误，例如以数字开头的ID，不正确的八进制，十六进制，浮点数。

1.2 处理嵌套注释

在flex规则中加入

```
"/*" { comment(); }
```

可以在遇到`/*`的时候调用comment函数，函数不断获取输入，跳过字符，并记录遇到的`/*`个数，在遇到`*/`的时候计数减一，最后在等于零的时候跳出处理，如果遇到文件末尾还是没有闭合，就报告Endless Comment并退出。

```
void comment(void)
{
  char c, prev = 0;
  int comment = 1;
  while ((c = input()) != 0)      /* (EOF maps to 0) */
  {
    if (c == '*' && prev == '/') {
      comment++;
    }
    if (c == '/' && prev == '*'){
      comment--;
    }
    if (comment == 0) {
      return;
    }
    prev = c;
  }
  lexError = 1;
  printf("Error type A at Line %d: Endless Comment\n", yylineno);
}
```

1.3 处理Exp的优先级问题

为了处理表达式的优先级问题，我们将表达式的规则按照优先级从低到高的顺序拆分：

```
AndExp:
      /* RelationalExp { $$ = newInternalNode(@$.first_line, "AndExp", 1, $1); } */
      RelationalExp { $$ = $1; }
      | AndExp AND RelationalExp { $$ = newInternalNode(@$.first_line, "Exp", 3, $1, $2, $3); }
      ;

OrExp:
     /* AndExp { $$ = newInternalNode(@$.first_line, "OrExp", 1, $1); } */
     AndExp { $$ = $1; }
     | OrExp OR AndExp { $$ = newInternalNode(@$.first_line, "Exp", 3, $1, $2, $3); }
     ;


AssignmentExp:
            /* OrExp { $$ = newInternalNode(@$.first_line, "AssignmentExp", 1, $1); } */
            OrExp { $$ = $1; }
            | PostfixExp ASSIGNOP AssignmentExp { $$ = newInternalNode(@$.first_line, "Exp", 3, $1, $2, $3); }
            ;

Exp:
   AssignmentExp { $$ = $1; }
   | Exp COMMA AssignmentExp { $$ = newInternalNode(@$.first_line, "Exp", 3, $1, $2, $3); }
   ;
```

以此类推，这样就可以灵活的解决最后的优先级产生的冲突，但是这样可能会产生多层的表达式嵌套，所以在构造语法树的时候需要特殊处理，在直接归约为下一个优先级的Exp时不记录这一层的节点信息，如果有具体的表达式才记录。

1.4 构建语法树

```
typedef struct astNode {
  int lineNum;
  int startColumn;
  int endColumn;
  NodeType type;
  char* name;
  char* val;
  struct astNode** child;
  unsigned int child_count;
} ASTNode;
```

语法树的定义如下：

其中存储了语法树单元的行号等位置信息，元素的类型，名字，值，以及一个子节点的数组。

对于终结符，编写了newTokenNode函数，对于非终结符，我们定义了newInternalNode函数：

```
pASTNode newTokenNode(int lineNum, int startColumn, int endColumn,
NodeType type,char* tokName, char* tokText);
pASTNode newInternalNode(int lineNum, char* tokName, int argc, ...);
```

其中newInternalNode函数使用的是变长参数，以处理不同数量子节点的AST节点。

1.5 打印语法树

在主函数中，有如下代码

```
if (!lexError && !synError) {
    printASTTree(root, 0);
}
```

其中lexError和synError是定义的标志，在程序调用错误处理的时候会被置为1。在无错误的时候进行打印语法树的操作。

打印语法树就是递归地先序遍历树的每一个节点，通过函数参数记录深度，以便操作打印不同数量的空格区分深度。在打印的时候判断节点的类型，以针对不同的类型进行特殊处理。

2. 如何编译

程序使用cmake来管理项目源文件，文件如下：

```
cmake_minimum_required(VERSION 3.21)
project(lab1)

set(CMAKE_CXX_STANDARD 14)
```

```
add_custom_command(
  OUTPUT ../lex.yy.c
  COMMAND flex --header=lex.yy.h cmm.l
  DEPENDS cmm.l
  WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
)

add_custom_command(
  OUTPUT ../cmm.tab.c
  OUTPUT ../cmm.tab.h
  COMMAND bison -d -Wcounterexamples cmm.y
  DEPENDS cmm.y
  WORKING_DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR}
)

aux_source_directory(. SRC)

add_executable(lab1 ${SRC})
```

其中add_custom_command指令可以自动检测我们是否更改了flex与bison源文件，如果更改了就可以自动运行flex与bison重新生成c语言源文件。

在项目根目录下运行

```
mkdir build && cd build
cmake ..
make
```

就可以编译项目。