编译原理 实验一 实验报告

高天朗 171240535

赵新榆 161120181

1.文件结构

头文件 common.h tree.h error.h

C 文件 error.c main.c

Bison 源文件 syntax.y

Flex 源文件 lexical.l

使用 Makefile 编译后即可./parser 1.cmm 查看输出

2.完成的内容

正确的代码输出语法树

识别词法错误

识别语法错误

要求 1.1 识别八进制数和十六进制数

要求 1.2 识别指数形式的浮点数

要求 1.3 识别两种形式的注释

3.其他

1)错误输出格式如下图

```
static const char *error_out[2048] = {
    [ERROR_FILEINV]   = "file '%s' invalid",

    [ERROR_MYSCOMM]  = "mysterious comment",
    [ERROR_MYSCHAR]     = "mysterious character '%s'",
    [ERROR_INVFPNUM]    = "invalid floating-point number '%s'",
    [ERROR_INVHEXNUM]   = "invalid hexadecimal number '%s'",
    [ERROR_INVOCTNUM]   = "invalid octal number '%s'",
    [ERROR_INVDECNUM]   = "invalid decimal number '%s'",
    [ERROR_FPNUM_OF]    = "floating-point number '%s' overflow",
    [ERROR_INTNUM_OF]   = "integer number '%s' overflow",

    [ERROR_SYNTAX]      = "%s",
};

static const char *error_mes[16] = {
    [0]                     = "system error: ",
    [1]                     = "error type A at line %d, col %d: ",
    [2]                     = "error type B at line %d, col %d: ",
```

2)在使用%option yylineno 的时候，发现输出行数始终为 1，而且无法表示列号。因此采用了换一种 YY_USER_ACTION 的定义并且维护 loc_t 结构体用于记录行号列号的方式。

```
//维护当前位置
#define YY_USER_ACTION                                  \
    yylloc = prev_loc;                                  \
    for (int i = 0; yytext[i] != '\0'; i++) {           \
        if (yytext[i] == '\n') {                        \
            prev_loc.line++; prev_loc.col = 1;          \
        } else {                                        \
            prev_loc.col++;                             \
        }                                               \
    }

//维护当前位置
typedef struct loc {
    int line, col;
} loc_t;
//初始化行号列号
#define LOC_INITIALIZER ((loc_t) {1, 1})
```

但是没有在 bison 的头文件中找到类似 YYSTYPE 的定义，因此手动定义了 YYLTYPE，这部分内容参考了网站 https://www.cnblogs.com/Frandy/archive/2013/04/10/parser_flex_bison_location_using.html

为了使得行号列号输出正确，又修改了 YYLLOC_DEFAULT 的实现。

```
//维护当前位置
#define YYLLOC_DEFAULT(Cur, Rhs, N) {       \
    do {                                    \
        if (N) {                            \
            (Cur) = YYRHSLOC(Rhs, 1);       \
        } else {                            \
            (Cur) = YYRHSLOC(Rhs, 0);       \
        }                                   \
    while (0)
```

3)为了识别两种形式的注释，采用了%x 的选项，这部分参考了 https://blog.csdn.net/lishichengyan/article/details/79512373

在执行下一个 BEGIN 操作之前，具有给定开始条件的规则将处于激活状态，而具有其他开始条件的规则将处于非激活状态。%x 表示独占性的开始条件，只有符合启动条件的规则才会生效。于是采用了如下方式用于识别注释。受限于报告长度，其余部分见代码。

```
%x COMMENT_1
%x COMMENT_2

"/*"                { BEGIN(COMMENT_1); }

<COMMENT_1>"*/"     { BEGIN(INITIAL); }

<COMMENT_1>(.|\n)   //清除注释

<COMMENT_1><<EOF>>  {
                        error_rec(ERROR_MYSCOMM, yylloc);
                        return 0;
                    }
```