编译原理实验一：C--语言的词法分析与语法分析

软件工程 范兆基 20331011 819402765@qq.com

一、词法分析(lexical.l)

1. 实现了终结符号、注释、空白的匹配。

```
delim [ \t]
whitespace {delim}+
letter [A-Za-z_]
digit [0-9]


COMMENT_SINGLE \/\/[^\n]*
COMMENT_MULTI "/*"([^\*]|(\*)*[^\*\/])*(\*)*"*/"

DEC 0|[1-9]{digit}*
OCT 0[0-7]+
HEX 0[xX]([0-9A-F]|[0-9a-f])+
FLOAT {digit}+\.{digit}+|{digit}+\.{digit}*([eE][+-]?{digit}+)?

ID  {letter}({letter}|{digit})*
RELOP ==|>=|<=|!=|>|<

TYPE int|float
```

2. 匹配到注释或空白，则跳过

```
{COMMENT_SINGLE} {
    // printf("COMMENT_SINGLE:\n%s\n",yytext);
    // printf("%d\n",yylloc.first_line);
}
{COMMENT_MULTI} {
    int len = strlen(yytext);
    for(int i = 2; i < len-1; i++)
    {
        if(yytext[i]=='*' && yytext[i+1]=='/' && i!=len-2)
        {
            printf("error_multi_line:line%d:\n",yylineno);
            exit(1);
        }
    }
    // printf("COMMENT_MULTI:\n%s\n",yytext);
}

{delim} {}
```

3. 匹配到终结符号，将匹配结果返回至bison：各关键字、类型、整数(八/十/十六进制)常量、浮点数常量、标识符(ID)

```
{TYPE} {
    yylval.type_node = createNode(yylineno,TYPE,yytext,0,0);
    return TYPE;
    }
{RELOP} {return RELOP;}

// 省略。。。

"while" {yylval.type_node = createNode(yylineno,WHILE,NULL,yylineno,0);return WHILE;}


{DEC} {
    // 省略。。。
    return INT;

}

{OCT} {
    // 省略。。。
    return INT;
}

{HEX} {
    // 省略。。。
    return INT;
}

{FLOAT} {
    // 省略。。。
    return FLOAT;

}

{ID} {
    // 省略。。。
    return ID;
}
```

二、语法分析(syntax.y)

1. 声明：
   a. 各符号的类型：type_node类型为指向语法树节点的指针
   b. 部分符号的结合性：解决大部分二义性

```
%start Program
%locations

%token <type_node> INT;
%token  <type_node> FLOAT;
%token <type_node> ID SEMI COMMA
%token <type_node> TYPE
%token <type_node> LC RC
%token <type_node> STRUCT RETURN IF ELSE WHILE



%right <type_node> ASSIGNOP
%left <type_node> OR
%left <type_node> AND
%left <type_node> RELOP
%left <type_node> PLUS MINUS
%left <type_node> STAR DIV
%right <type_node> NOT
%left <type_node> LP RP LB RB DOT
%nonassoc LOWER_THAN_ELSE
%nonassoc ELSE

%type<type_node> Program ExtDecList ExtDef ExtDefList Specifier StructSpecifier OptTag
            Tag VarDec FunDec VarList ParamDec CompSt StmtList Stmt DefList Def Exp Args DecList Dec
```

2. 根据附录A给出的产生式进行代码书写，其中绝大部分是完全复刻，但是其中的if-else的移入-规约冲突需要进行处理，提高移入的优先级

```
%nonassoc LOWER_THAN_ELSE
%nonassoc ELSE

// 省略。。

Stmt :
// 省略。。。
| IF LP Exp RP Stmt %prec LOWER_THAN_ELSE {。。。}
| IF LP Exp RP Stmt ELSE Stmt {。。。}
;
```

三、语法树(tree.h、tree.c)

1. 构建语法树节点

```
// 用于存储常量的数值的共用体
typedef union
{
    int type_int;
    double type_double;
}Type;

// 语法树树节点
struct Node
{
    int tag;// 标记节点类型，对应各种符号
    char* name;// 存储类型名称或者ID名称
    struct Node *kids[10];// 子节点
    int kid_num;// 子节点数目
    Type value;// 存储常量数值
    int level;// 节点高度，在遍历树时使用
    int row;// 节点行号
    int if_empty;// 是否为空节点，此处空节点是为了解决X->ε这类产生式在语法树便利时产生的Bug
};

// 用于标记节点类型枚举类型
/* ******************************************
   需要注意的是：此处只对非终结符分配，先前在syntax.y文件中对终结符进行了token声明。
   bison会创建一个枚举类型yytokentype，从258开始为终结符分配数值，所以此处不必再次分配
********************** */
enum yyNTtype
{
    Program=10000, ExtDecList, ExtDef, ExtDefList, Specifier, StructSpecifier,
    OptTag, Tag, VarDec, FunDec, VarList, ParamDec, CompSt, StmtList, Stmt,
    DefList, Def, Exp, Args, DecList, Dec

};
```

2. 创建语法树节点

```
struct Node *createNode(int r,int tag, char *text,int i,double d)
{
    struct Node *nd=(struct Node*) malloc(sizeof(struct Node));
    nd->kid_num=0;
    nd->tag=tag;
    nd->name=NULL;
    nd->row = r;
    nd->if_empty = 0;
    switch(tag)
    {
        case INT:
            nd->value.type_int = i;// 存储整数数值
            break;
        case FLOAT:
            nd->value.type_double = d;// 存储浮点数数值

            break;
        case ID:
        case TYPE:
            // 存储名字
            nd->name=(char*)malloc(sizeof(char)*strlen(text));
            strcpy(nd->name,text);
            break;
        default:
            break;
    }
    return nd;
}
```

3. 遍历打印语法树

```
// 打印某个节点信息
void treePrintLevel(struct Node *nd)
{
    // 省略。。。
}

// 遍历语法树

// 利用栈遍历
struct Node* stack[500];// top is null
void stack_print(struct Node *nd)
{
    // 省略。。。
}

// 递归遍历
// 递归
void treePrint_2(struct Node *nd)
{
    if(nd==NULL)
    {
        printf("poniter is null\n");
        return;
    }
    int cur_level = nd->level;
    treePrintLevel(nd);
    int kid_num = nd->kid_num;
    int i;
    for( i = 0; i < kid_num; i++)
    {
        nd->kids[i]->level = cur_level+1;
        treePrint_2(nd->kids[i]);
    }
}

// 递归入口
void recursion_print(struct Node *nd)
{
    if(nd==NULL)
    {
        printf("poniter is null\n");
        return;
    }
    nd->level = 0;
    treePrint_2(nd);
}
```

四、语法树与语法分析结合

1. 修改标号类型

```
%union
{
    struct Node *type_node;
}
```

2. 在lexical.l与syntax.y中为符号创建节点并连接

```
// lexical.l中例子
// 创建叶子结点
";" {yylval.type_node = createNode(yylineno,SEMI,NULL,yylineno,0);return SEMI;}

// syntax.y中例子
// 创建内部节点，并且连接子节点(除非是空)
ExtDefList : {$$=createNode(0,ExtDefList,NULL,0,0);$$->if_empty=1;}
| ExtDef ExtDefList {$$ = createNode($1->row,ExtDefList,NULL,0,0);$$->kid_num = 2;$$->kids[0] = $1;$$->kids[1] = $2;}
;
```

五、错误处理

错误处理部分未能很好处理

1. 词法错误：遇到未知词

```
. {
    fail = 1;
    printf("Error type A at Line %d:Mysterious character \'%s\'\n",yylineno,yytext);
}
```

2. 语法错误：仅通知第一处语法错误发生在哪里

```
// 重写yyerror()
void yyerror(char* msg)
{
    fail = 1;
    extern int yylineno;        // defined and maintained in lex
    extern char *yytext;        // defined and maintained in lex
    fprintf(stderr, "Error type B at line %d: %s at symbol '%s'\n", yylineno, msg, yytext);
}
```

六、特别注意

在该代码作业中，需要对原makefile文件进行修改才可使用make进行编译得到可运行文件parser
或可依次输入以下指令得到parser文件

```
flex lexical.l
bison -d -v syntax.y
gcc lex.yy.c syntax.tab.c main.c tree.c -o parser
```