编译原理实验一报告

词法分析与语法分析

1190200122 袁野

一、实现功能

程序实现了对 C—源代码的词法分析和语法分析，使用了 Flex 和 Bison 工具。

1、词法分析

该程序通过在 lexical.c 中输入符合 C—语言标准的词素和正则表达式，然后对输入文件进行扫描，如果遇到符合 C—文法的词语会建立对应的语法树中的节点，否则根据其具体的错误类型将错误信息输出。

如下图 struct 为例，当读入 struct 词素时，会首先建立一个词法类型的节点，然后返回新创建的结构体。

```
"struct" {
    yylval = create_Node_1("STRUCT\0");
    return STRUCT;
}
```

当新读入的词素为 INT、FLOAT、ID 时，会额外在新建得到语法树节点上新添加对应的整数值/浮点数值/变量名称信息。

```
{INT} {
    yylval = create_Node_1("INT\0");
    yylval -> int_number = atoi(yytext);
    return INT;
}
```

部分正则表达式的定义如下：

```
IGNORE [\r \t]
CHANGELINE [\n]
INT 0|[1-9][0-9]*
FLOAT {INT}\.[0-9]+
ID [_a-zA-Z][0-9_a-zA-Z]*
RELOP >|<|>=|<=|==|!=
TYPE int|float
DIGIT [0-9]
```

词法分析器会分析所有定义过的关键字或正则表达式，当出现了未被定义过的关键字时，会认定其为非法字符。我们可以用“.”来匹配所有未被匹配过的字符，并对应输出错误信息：

```
. {
    char msg[32];
    sprintf(msg, "Mysterious character \"%s\"", yytext);
    printError('A', msg);
    Err = 1;
}
```

2、语法分析

在进行文法分析时，会根据 C—语言的文法规定进行判断，因此将 C—的文法表达式写入到 bison 的源代码中，同时写入关于出现 error 的判定，用来判断语法错误的出现及其类别。对于每一个语法单元，会将其对应的产生式右侧的符号依次当作其语

法树中的子节点。

```
ExtDefList: {
    $$ = create_Node(NULL,"ExtDefList\0",yylineno);
    $$ -> int_number=0;
}|ExtDef ExtDefList {
    $$ = create_Node($1,"ExtDefList\0",@1.first_line);
    $$ -> int_number = 2;
    $1 -> brother = $2;
}
```

而由 ε 生成的语法节点，我们仍然建立该节点，只是将其 int_number 值设为 0，用来在输出语法树时判断其是否需要输出。

```
DefList: {
    $$ = create_Node(NULL,"DefList\0",0);
    $$ -> int_number = 0;
}|Def DefList{
    $$ = create_Node($1,"DefList\0",@1.first_line);
    $1 -> brother = $2;
}
```

由空串生成的祖先节点：查询所有产生式，仅 Program:ExtDefList、ExtDefList: 满足只由空串生成祖先节点，此时可能出现祖先节点行号记录错误的问题（由于 @1.first_line 判断的节点为生成），因此进行特殊处理。

```
Program:ExtDefList{
    if($1->int_number==0)
        $$=create_Node($1,"Program\0",$1->linenumber);
    else
        $$=create_Node($1,"Program\0",@1.first_line);
};
```

关于 error 的判定，将所有可能出现的语法错误，在不会导致移入-规约冲突的情况下写入产生式中，一旦在规约过程中遇到对应类型的错误，即可输出对应的错误信息。

```
VarDec:ID{
    $$ = create_Node($1,"VarDec\0",@1.first_line);
}|VarDec LB INT RB{
    $$ = create_Node($1,"VarDec\0",@1.first_line);
    $1 -> brother = $2;
    $2 -> brother = $3;
    $3 -> brother = $4;
}| VarDec LB error RB {
    Err = 1;
    char msg[32];
    sprintf(msg, "Syntax error, near \'%c\'", yytext[0]);
    if (lastErrLineno != yylineno)
        printError('B', msg);
}
```

3、**语法树的生成**

在程序中，语法树的每一个节点均为结构体类型：

```
struct Node {
    struct Node* child;
    struct Node* brother;
    int linenumber;
    char index[15];
    short judge;
    union{
        char char_name[30];
        int int_number;
        float float_number;
    };
};
```

语法树采用儿子-兄弟储存法，child 为当前节点指向其第一个儿子的指针，brother 为当前节点指向其相邻的下一个兄弟节点的指针。linenumber 为该语法（词法）单元第一个元素所在的行号，index 为该语法（词法）元素对应的类型名称，judge 为语法/词法标识符，judge 为 0 即表示当前元素为词法单元，否则为语法单元。下方

的 char_name、int_name、float_name 为若当前单元为 ID、INT、FLOAT 时对应的名字或取值的附加信息。

对于一个符合 C—文法的源程序输出其语法树采用深度优先搜索的方式进行输出，具体代码如下：

```
void print_tree(struct Node* now,int depth) {
    if(now->judge == 0 || now->int_number) {
        for (int i=0; i<depth; ++i) printf("  ");
        print_node(now);
    }
    if (now -> child != NULL) print_tree(now -> child, depth+1);
    if (now -> brother != NULL) print_tree(now -> brother,depth);
    return;
}
```

print_node 函数即为对于一个文法单元按照实验指导书要求输出其对应信息的过程。

```
void print_node(struct Node* now){
    if (now -> judge == 0) {
        if (!strcmp (now -> index, "ID\0")) {
            printf("ID: %s\n",now -> char_name);
        } else if (!strcmp (now -> index, "TYPE\0")){
            printf("TYPE: %s\n",now -> char_name);
        } else if (!strcmp (now -> index, "INT\0")){
            printf("INT: %u\n",now->int_number);
        } else if (!strcmp(now->index,"FLOAT\0")){
            printf("FLOAT: %f\n",now->float_number);}
        else{
            printf("%s\n",now->index);
        }
    } else printf("%s (%d)\n",now->index,now->linenumber);
}
```

4、错误信息的输出

在遇到词法或者语法错误时，会通过 printErrpr 函数对其进行输出，该函数会传入对应的错误类型（A：词法错误/B：语法错误）和详细需要输出的错误信息。而每次输出错误时，会记录上一次出现错误的行号，来保证每一行的错误仅输出一个。

```
void printError(char errorType, char* msg) {
    fprintf(stderr, "Error type %c at Line %d: %s.\n", errorType, yylineno, msg);
    lastErrLineno = yylineno;
}
```

二、编译过程

通过借助网上的参考资料，构建了一个 makefile 文件来编译这个程序，命令如下：

make 编译该项目并生成 parser 可执行文件
make test 测试实验指导书中的四个必做样例
make clean 删除所有编译过程中产生的新文件