# 实验报告

## *第一部分——词法（.l文件）*

这一部分基本上就是对着lab1改了一下，主要改动就是加上了return以传递参数。没有太多好说的....

## *第二部分—— 语法（.y文件）*

### 语法规则

这一部分我们首先要实现语法规则，大部分照着文档抄就完事了，但是由于Bison不支持[ ]和{ }这两个正则表达式，所以我们需要做出一些改变，基本思路如下：

- 对[ ]：假设式子A → [ B ] C，可以改编成A → C | A → BC
- 对{ }：假设式子A → { B } C，可以改编成A → DC | A → C ， D → DB | D → B

if-else是一个比较难实现的式子，这里我如下实现，其中LOWER_THAN_ELSE和ELSE都是%nonassoc属性的

```
Stmt : IF '(' Cond ')' Stmt %prec LOWER_THAN_ELSE
     | IF '(' Cond ')' Stmt ELSE Stmt
```

同时为了错误处理，我在一些关键的式子用error代替正常的符号，并使用了如下yyerror函数实现报错：

```
void yyerror(const char *s){
    fprintf(stderr, "Error at line %d: %s\n", lineno, s);
}
```

### 输出处理

然后我们需要输出两个文件，即Detail.txt 和Tree.dot。

前者很简单，我们只需在每一个式子中将对应式子输出到指定文件夹中，如下列代码所示：

```
CompUnit: OtherCompUnit{
    fprintf(fDetail, "CompUnits -> OtherCompUnit\n");
    }
    | CompUnit OtherCompUnit{
    fprintf(fDetail, "CompUnits -> CompUnit OtherCompUnit\n");
    }
```

后者则比较困难。根据观察Detail.txt我们可以发现，Bison使用对应式子归约的顺序恰好和语法树后续遍历的顺序相同。所以想要输出Tree.dot，就需要通过后序遍历的顺序建立语法树，然后通过前序遍历的方式输出语法树规则到Tree.dot。

## 建树

我原本准备给非终结符赋予一个自定义的属性node*来建树，但是我并没有成功地在%union中导入我自定义的数据结构，所以只能退而使用一个全局数组来装所有的node。同时给非终结符赋予int属性，以这个int值为数组的序。

代码如下，假设遇到一条规约式：A → BC，则此时B和C一定已经存在于nodes向量中，即有nodes[ B.val ]和nodes[ C.val ] 。此时我们新添A后，

- nodes[ A.val ].label = A代表的符号
- nodes[ A.val ]. n_children = 2
- nodes[ A.val ].children[ 0 ] = B.val　　nodes[ A.val ].children[ 1 ] = C.val
- CHD_NO和Node_NO是前序输出的时候用的，建树的时候尚且用不到。

```
typedef struct Node {
    char* label;                    // 节点标签
    int n_children;                 // 子节点数量
    int children[20];               // 子节点索引列表
    int CHD_NO;                    //画图的时候判断是父亲的第几个儿子。
    int Node_NO;                   //画图的时候判断父亲是谁。
} Node;

Node nodes[MAX_NODES];          // 全局节点数组
int cnt = 0;                    // 全局节点计数器

int newNode(const char* label) {
    int idx = cnt++;
    nodes[idx].label = strdup(label);
    nodes[idx].n_children = 0;
    return idx;
}

void addChild(int parent_idx, int child_idx) {
    Node* parent = &nodes[parent_idx];
    if (parent->n_children < MAX_NODES) {
        parent->children[parent->n_children++] = child_idx;
    }
}

//规约式中的应用
ConstDecl : CONST INT ConstDef ';'{
    fprintf(fDetail, "ConstDecl -> const int ConstDef ;\n");
    $$ = newNode("ConstDecl");
    addChild($$, newNode("const"));
    addChild($$, newNode("int"));
    addChild($$, $3);
    addChild($$, newNode("\\;"));
    }
```

## 前序遍历语法树并输出

由于是后序建树的，所以很显然树的根节点是nodes[cnt-1]，然后通过drawTree前序遍历。

遍历的逻辑分为三部分

首先遍历当前节点的的子节点，把这些子节点整合为一个大节点，并为它们在Tree.dot中创建node，使用一个全局变量treeCnt作为它们Tree.dot中的node编号。同时，虽然合并成大节点了，仍需要用CHD_NO和Node_NO分别记录子节点是当前节点的第几个孩子，以及当前节点在Tree.dot中的node编号。

然后输出一行使当前节点的父节点连接到当前节点。

最后遍历子节点，若有孩子（即是非终结符）则继续调用drawTree。

```
int treeCnt = 0;

void drawTree(int tar){
    fprintf(ft, "node%d[label = \"", ++treeCnt);
    for(int i=0;i<nodes[tar].n_children;i++){
        nodes[nodes[tar].children[i]].CHD_NO = i;
        nodes[nodes[tar].children[i]].Node_NO = treeCnt;
        fprintf(ft, "<f%d> %s",i,nodes[nodes[tar].children[i]].label);
        if(i == nodes[tar].n_children - 1)
            fprintf(ft, "\"];\n");
        else
            fprintf(ft, "|");

    }

    fprintf(ft, "\"node%d\":f%d-=>\"node%d\";\n",nodes[tar].Node_NO,nodes[tar].CHD_NO,treeCnt);

    for(int i=0;i<nodes[tar].n_children;i++){
        if(nodes[nodes[tar].children[i]].n_children > 0)
            drawTree(nodes[tar].children[i]);
    }
}

int main(int argc, char *argv[]) {
    yyin = fopen(argv[1], "r");

    fDetail = fopen("Detail.txt","w");
    ft = fopen("Tree.dot", "w");


    if (yyin == NULL) {
        perror("Error opening file");
        exit(1);
    }
    yyparse();
    fprintf(ft, "digraph \" \"{\n");
    fprintf(ft, "node [shape = record,height=.1]\n");
    if(!cnt){
        fprintf(ft, "node0[label = \"<f0> NULL\"];\n");
        fprintf(ft, "}");
        fclose(fDetail);
        fclose(yyin);
        return 0;
    }

    fprintf(ft, "node0[label = \"<f0> CompUnit\"];\n");
    nodes[cnt-1].CHD_NO = 0;
    nodes[cnt-1].Node_NO = 0;
    drawTree(cnt-1);
    fprintf(ft, "}");
```

```
    fclose(fDetail);
    fclose(yyin);
    return 0;
}
```

然后便写完了，调用Dot画图就行了，好耶(∠・ω< )⌒★!