2023春季编译原理实验一报告

201220113 贺保定

功能介绍

本次实验完成了语法分析和词法分析。对于格式正确的c--源代码，生成抽象语法树并向屏幕打印；对于存在词法或语法错误的代码，分类给出错误提示。根据任务编号，除必做内容外，最终得到的分析器还能对指数形式的浮点数进行识别。

词法分析

根据给出的c--词法，为每种词法单元书写正则表达式。为了与语法分析配合，识别到词法单元时需要采取合适的动作。

1. 识别到空白符[\r\t\ ]：不采取动作，即丢弃此单元；
2. 识别到换行符\n：执行维护行号的操作；
3. 识别到可能为指数形式浮点数的单元：判断是否为合法的浮点数。若不合法，报告A型错误；若合法，处理步骤同4；
4. 识别到其他合法的单元：为yylval的type_node成员赋值，创建对应的语法树节点，返回词法单元；
5. 其他情况：匹配到通配符' . '时，认为存在非法字符，报告A型错误。

正确执行词法分析还需要注意单元的优先级。例如，lexical.l源文件中ID单元置于TYPE等单元后，避免关键字int、float、if等被误识别为标识符。

除了正确性外，为了便于维护，对代码结构应当进行适当优化。容易发现，大多数词法单元的动作是相似的，区别仅在于创建的语法树结点和返回词法单元的名称。所以，可以用以下带参数的宏来简化代码：

```
    #define process_token(name) \
        char* str = strdup(yytext); \
        yylval.type_node = create_node(#name, str, 0, NULL, 1,
yylloc.first_line); \
        return name;
```

其中#name可以将name转为字面字符串常量。这样一来，对于大部分词法单元，例如`if`，只要填入`process_token(IF)`，就可以完成词法动作。

关于指数形式的浮点数错误处理，我选择将其交给语法分析处理。词法分析仅识别合法的浮点数。所以，当c--源代码中存在不合法的浮点数时，通常会报告类型为B的错误。

抽象语法树生成

根据bison的格式规范，书写好c--的CFG。然后在每个产生式后填入合适的语义动作，即可完成语法树的生成。

首先介绍语法树节点的格式。语法树节点的内容是一个格式如下的结构体：

```
    typedef struct {
        char* name; // 节点名称
        union data {
            char* s;
            int i;
            float f;
        } value; // 终结符节点的值。int型使用成员i，float型使用成员f，其他单元使用s
        int child_num; //子节点个数
        struct Node** child; // 依次存储子节点位置的数组
        int is_terminal; // 标识节点是否是一个终结符
        int line; // 语法单元的行号
    } Node;
```

每个合法的产生式执行的语义动作是类似的。例如，对于产生式`Program: ExtDefList;`采取如下动作：

```
    Node** child=(Node**)malloc(sizeof(Node*)); // 申请空间
    child[0]=$1; // 链接子节点
    $$=create_node("Program",NULL,1,child, 0, @$.first_line); // 赋值
```

Program具有一个子节点，所以先申请能够存储一个子节点位置的空间。然后在该空间中依次存储子节点的位置。最后维护当前节点的值$$，调用`create_node`函数，创建具有完备信息的节点。

除书写产生式和语义动作外，还需要定义操作符的优先级和结合性，使得语法树结构正确。

最后，在根节点Program处，检查标志变量legal。legal初始为1，会在检测到语法错误时被赋值为0。若legal最终仍为1，说明程序合法，可以打印语法树。`print_ast`函数对语法树进行先序遍历，按层次以合适的缩进量依次打印节点名称，若为int和float字面量还需打印其值。另外，对于非终结符，如果它没有子节点，说明应用了ϵ产生式，此时不需打印该节点的信息。

语法错误处理

本次实验中使用了以下几种处理语法错误的策略：

1. 使用分号、右括号等作为错误恢复的同步符，书写形如`error SEMI`的产生式；
2. 对于更具意义的语法单元，可以将语法单元替换为`error`，书写形如`IF LP error RP Stmt`的产生式；
3. 分号、右括号等分隔符本身也可能缺失，所以还需要类似`LC DefList StmtList error`的产生式。

以上就是语法错误恢复的基本思路。由于程序员出错的类型多种多样，最终实现的解析器不能保证对所有类型的错误都进行了足够完美的错误恢复，但是对于实验要求提供的样例而言，都能正确报告错误类型和错误位置。