18231047 王肇凯

语法分析

最初设计

采用递归子程序法进行自顶而下的分析。在调用子程序前先读入一个token，然后每个子程序内通过读入token以及递归调用其他子程序来分析一种非终结符号，并将语法成分输出到文件中。

分析文法得知不存在左递归，为了避免回溯，采用了预读的方法，即在多个选择间存在冲突时提前读1~3个token进行判断出唯一选择，并回退到预读前的token，然后调用该选择的子程序。

当读入的终结符号与预期不同时，由于避免了回溯，因此直接产生异常，保存至错误表中。

其中注意到`<有返回值函数调用>`与`<无返回值函数调用>`的语法完全一致，需要根据语义区分。因此建立简单符号表（等到语义分析时再加以完善），在函数定义时在符号表中添加 “函数标识符 — 有无返回值” 的映射，在需要区分函数调用时通过标识符查表，从而选择有返回值或无返回值函数调用。

实现与完善

设计`Grammar`类进行语法分析工作。在初始化时使用词法分析步骤的结果保存至`tokens`作为输入，并指定是否输出分析结果至文件。使用时，调用其`analyze()`方法进行语法分析主过程。

将符号表`stmTable`等全局变量作为该类的成员变量。

考虑到预读后需要回退至预读前的位置，因此将待输出内容按行保存至`output_str`中，在回退时删去上一个词法分析的输出行。

成员变量`tk`，`sym`，`pos`分别保存当前读到的词法分析结果、结果的词法成分、在所有词法分析tokens中的位置。

方法`next_sym()`，`retract()`，`error()`，`output()`分别进行读入token、预读结束后回退、存储错误、输出语法成分。

各个递归子程序作为方法保存在类中。

```
class Grammar {
public:
    vector<LexResults> tokens;
    vector<Error> errors;
    map<string, SymTableItem> symTable;
    vector<string> output_str;

    bool save_to_file;
    LexResults tk{INVALID, INVALID, -1, -1, -1};
    int pos = 0;
    string sym = "";

    void error(const string &expected);
    int next_sym();
    void retract();
    vector<GrammarResults> analyze(const char
*out_path);
    Grammar(vector<LexResults> t, bool save) :
tokens(std::move(t)), save_to_file(save) {};
    void output(const string &name);

    void Program();
    void ConstDeclare();
    void ConstDef();
    void UnsignedInt();
    void Int();
```

```
    void Identifier();
    void DeclareHead();
    void Const();
    void VariableDeclare();
    void VariableDef();;
    void TypeIdentifier();
    void SharedFuncDef();
    void RetFuncDef();
    void NonRetFuncDef();
    void CompoundStmt();
    void ParaList();
    void Main();
    void Expr();
    void Item();
    void Factor();
    void Stmt();
    void AssignStmt();
    void ConditionStmt();
    void Condition();
    void LoopStmt();
    void PaceLength();
    void CaseStmt();
    void CaseList();
    void CaseSubStmt();
    void Default();
    void SharedFuncCall();
    void RetFuncCall();
    void NonRetFuncCall();
    void ValueParaList();
    void StmtList();
    void ReadStmt();
    void WriteStmt();
    void ReturnStmt();
};
```

`analyze()`只需打开关闭输出文件流、读入第一个token、调用`<程序>`子程序即可。

`<数字>`，`<标识符>`等基础的非终结符号在词法分析时已经进行过判断，因此不必写子程序。

每个递归子程序在调用前需要先使用`next_sym()`读入一个token，然后根据右部各选择的首符号进行选择（必要时采用预读），对于非终结符号调用其子程序，终结符号则判断是否与预期一致，不一致则报错。