# 一、结构体定义：

## 1. 变量元素结构体

struct symbol

{

string name; //变量名

string type; //变量类型

string property; //变量属性

struct symbol \*st; //函数型成员地址

};

定义于symbol.cpp，在计算符号表时存储函数获取的形参、全局变量以及其中定义的局部变量。

## 2.函数结构体

struct func

{

string name; //函数名

string ret; //函数返回值

int num; //函数中参数个数

struct symbol st[255]; //函数符号表

};

定义于symbol.cpp，在计算符号表时存储各函数符号表以及各类信息。

## 3. token结构体

typedef struct Token

{

string re; //token原始字符串

string type; //token类型

int line; //token所在行

int colume; //token所在列

}Token;

定义于global.cpp，适用整个工程，源程序由词法分析后产生的token流中所使用的结构体。

## 4. 节点语义信息结构体

typedef struct info

{

int arrayflag; //记录变量是否为数组

int emptyflag; //记录该非终结符是否推出空

int whilenum1; //记录while入口的三地址代码值

int whilenum2; //记录while出口的三地址代码值

int ifnum1; //记录if语句真出口三地址代码值

int ifnum2; //记录if语句假出口三地址代码值

int elsenum1; //记录else入口三地址代码值

int elsenum2; //记录else出口三地址代码值

int returnflag;

}info;

定义于global.cpp，适用整个工程，用于存储由源程序生成的语法分析树中各节点的语义信息。

## 5.语法分析树节点结构体

typedef struct Gtree

{

char re[255]; //结点元素原字符串

char type[255]; //结点元素类型 非终结符为derivation 终结符为自身类型 根节点为head

int bitnum; //结点孩子节点个数

int mtnum; //如为非终结符 记录推导式编号

int nodenum;

struct Gtree \* sons[255]; //孩子节点地址数组

struct Gtree \*brother; //兄弟节点地址

struct Gtree \*father; //父亲节点地址

struct info \*info; //语义信息

}treebit, \*bit;

定义于global.cpp，适用整个工程，用于构建源程序生成的语法分析树。

## 6.栈结构体

typedef struct Stack

{

vector<string> st; //栈

int current; //当前栈中元素个数

}st;

定义于global.cpp，适用整个工程，用于ll1语法分析时的出入栈操作。

## 7. 预测分析表结构体

typedef struct Mtable

{

map<string, int>vnname; //非终结符表

map<string, int>vtname; //终结符表

int mtable[maxsize][maxsize]; //预测分析表

int vnnum; //记录非终结符个数

int vtnum; //记录终结符个数

}mt;

定义于global.cpp，适用整个工程，用于存储ll1语法的预测分析表。

# 二、全局变量定义

## global.h中全局变量(适用整个工程)

static string resource = "0 prog-resource.txt"; //源程序

static string file\_nozs = "1 prog-step1.txt"; //去注释后的文件

static string file\_clear = "2 prog-clear.txt"; //预处理完成的文件

static string file\_lex = "3-lexical.txt"; //词法分析结果存储文件

static string file\_grammar = "4 grammar.txt"; //原始语法文件

static string file\_ll1grammar = "5 ll1\_grammar.txt"; //符合ll1文法的语法文件

static string file\_ad\_grammar = "6 ad\_grammar.txt"; //处理完成的语法文件

static string file\_vn\_vt = "7 vn\_vtlist.txt"; //终结符,非终结符存储文件

static string file\_ff = "8 first\_follow.txt"; //first,follow集存储文件

static string file\_mtable = "9 mtable.txt"; //预测分析表存储文件

static string file\_showstack = "10 stack\_content.txt"; //输出栈内信息文件

static string file\_symtable = "11 symbol\_table.txt"; //符号表存储文件

static string file\_midcode = "12 middle\_code.txt"; //中间代码输出

static string file\_optimize = "13 ad\_midcode.txt"; //优化中间代码

static string period[7] = { "",

"预处理/初始化",

"词法分析",

"翻译程序",

"中间代码生成"

}; //总步骤名称列表

static string content1[10] = { "",

"去除注释",

"去除空白字符",

"消除左递归以及公共左因子",

"将语法根据|拆分",

"求vn,vt集合",

"求fisrt,follow集",

"计算预测分析表mtable"

}; //分步名称列表

static string content2 = "词法分析";

static string content3[10] = { "",

"生成语法分析树",

"生成符号表",

};

static string content4[3] = { "",

"中间代码生成",

"中间代码优化"

};

static string call = "call ";

static string param = "param ";

static string proc = "Prog ";

static int maxline;

static const int maxsize = 255;

static const string firststr = "S"; //文法开始非终结符

static fstream in, out;

static map<string, string> vttype; //终结符类型表

static vector<string> vn; //非终结符集合

static vector<string> vt; //终结符集合

//各类要使用到的存储器指针

static vector<string>::iterator iter;

static map<string, string>::iterator iter\_map;

static map<string, vector<string>>::iterator maper;

static map<string, int>::iterator map\_si;

static map<int, vector<string>> grammarmap; //存储语法的推导

static map<int, vector<string>>::iterator gmaper;

static map<string, vector<string>> firstlist; //first集

static map<string, vector<string>> followlist; //follow集

static map<string, int> emptymake; //存储所有非终结符能推导出空字的推导式编号

## grammar-analysis.cpp中全局变量

Token token;

struct Mtable mtable;

struct Stack stack;

treebit \*treenode; //语法分析树根节点

int nodenum = 0; //节点编号

map<int, vector<string>> placelistmap; //各节点存储字符串信息表

map<string, bool(\*)(treebit \*cur)> funcmap; //推导式编号与其对应的语义动作函数

map<int, string> midcodemap;

map<string, int> funcnum;

int curnum = 99; //三地址代码编号

int curtemp = 0; //临时变量个数

## symbol.cpp中全局变量

struct func functable[255]; //符号表

int funcnum = 0; //符号表中元素个数

## optimize.cpp中全局变量

map<int, vector<string>> gb; //中间代码表

map<string, double>::iterator con;

vector<int> entry; //基本块入口地址表

vector<int>::iterator it,itertemp;

int endline; //中间代码的最后一条

map<string, int> op;

# 三、函数定义

**1.global 全局函数**

//打开指定文件输入/输出 返回1表示打开成功，返回0表示打开失败

int infile(fstream &in, string filenmae);

int outfile(fstream &out, string filenmae);

//关闭in,out两个用来输入输出的文件

void closefile(fstream &file1,fstream &file2);

//输出文件内容

void readfile(string filename);

//分割线

void divideline(void);

//各过程处理以及界面输出

void precedure(string period, int num, string filein, string fileout, void(\*func)(fstream &, fstream &),string content);

//去除字符串前后多余的空格

void fix(string &str);

**2.lexical-analysis**

int isBC(char ch); //判断是否为空白字符

int Reserve(string strToken);

void Retract(fstream &source);

void lexicalanalysis(fstream &in, fstream &out); //词法分析

void fixlex(void); //修改文法

**3.optimize**

//读入所有三地址代码

void getbetter1(fstream &in);

//常量优化

void getbetter2(int start, int end);

//优化中间代码

void optimize(fstream &in, fstream &out);

**4.pretreat**

//删除程序中所有注释

void delzs(fstream &in, fstream &out);

//删除程序中所有多余的空格,tab以及回车

void delblank(fstream &in, fstream &out);

//消除左递归以及公共左因子

void delleftrecursion(fstream &in, fstream &out);

//将语法根据|拆分

void splitgrammar(fstream &in, fstream &out);

**5.symbol**

//输出符号表

void output(fstream &out);

//生成符号表

void calsymboltable(fstream &in, fstream &out);

//判断函数是否已经存在 并返回在该函数在函数名表中的位置 不存在则返回-1

int func\_exist(struct func functable[255], int funcnum, string str);

//判断变量是否出现在当前函数的符号表中 存在则返回表中位置 否则返回-1

int sym\_exist(struct func functable[255], struct symbol \*st, int num, string str);

//将当前函数名加入到当前函数表中

void pushsym(struct func functable[255], struct symbol \*st, int curnum, int num);

//将当前函数加入到函数表中 并返回该函数符号表的地址

struct symbol \* pushfunc(struct func functable[255], int &funcnum, string str, string ret);

//初始化符号表

void init\_symtable(void);

//将当前变量加入到当前函数表中

void pushsym2(struct func functable[255], struct symbol \*st, int curnum, int num, string name, string property, string type);

**6.grammar-analysis**

//初始化终结符类型表

void init\_vttype(map<string, string> &vttype);

//初始化预测分析表mtable

void init\_mtable(struct Mtable &mtable);

//判断集合中是否存在str元素

int check\_exist(vector<string> vec, string str);

//判断vtmap中是否有该vt

int checkmap(map<string, string> mp, string str);

//删除vt集中应该为vn的元素

void del(vector<string> &vec, string str);

//计算终结符集合以及非终结符集合

void cal\_vn\_vt(fstream &in, fstream &out);

//输出vector中所有信息 输出vn,vt信息至文件

void output\_vec(fstream &out);

//求first集

void cal\_first();

//求follow集

void cal\_follow();

//计算first follow集

void cal\_first\_follow(fstream &in, fstream &out);

//输出first,follow集至文件

void output\_ff(fstream &out);

//计算预测分析表

void cal\_mtable(fstream &in, fstream &out);

//初始化栈

void stack\_init(struct Stack &stack);

//元素入栈

void stack\_pushin(struct Stack &stack, string str);

//获取栈顶元素

string stack\_gettop(struct Stack stack);

//栈顶元素出栈

void stack\_popout(struct Stack &stack);

//输出栈中信息

void stack\_show(struct Stack stack, fstream &out);

//初始化语法树

void tree\_init();

//生成语法分析树

void make\_tree(fstream &in, fstream &out);

//输出中间代码

void printmidcode(fstream &out);

//初始化语义动作表

void init\_funcmap();

//中间代码生成

void midcode(fstream &in, fstream &out);

//语义动作函数

bool fomula\_1(treebit \*temp);

…

bool fomula\_82(treebit \*temp);

# 四、工程结构

**调用现有库:**

#pragma once

#include<iostream>

#include<fstream>

#include<stdio.h>

#include<math.h>

#include<stdlib.h>

#include<iomanip>

#include<cctype>

#include<string>

#include<cstring>

#include<sstream>

#include<windows.h>

#include<vector>

#include<map>

#include<limits.h>

using namespace std;

#include "pretreat.h"

#include "lexical-analysis.h"

#include "grammar-analysis.h"

#include "symbol.h"

#include "optimize.h"

Symbol.h

Pretreat.h

Optimize.h

Lexical-analysis.h

Grammar-analysis.h

Include.h

Global.h

Optimize.cpp

Symbol.cpp

Lexical-analysis.cpp

Grammar-analysis.cpp

Pretreat.cpp

Global.cpp