**CX语言编译器**

**软件设计说明书**

**作者:武仁超**

**版本:v1.0**

**日期:2023/06/08**

**1．介绍**

本编译器实现了对CX语言的扩展，采用类记事本的用户界面，实现了CX语言的词法分析、语法分析、出错处理、代码生成和解释，并支持查看数据栈的功能。

**2．编译器系统结构**

**2.1 编译器**

**2.1.1 CX语言语法图**

program

block

block

ident

=

statement

{

const

int

number

ident

=

bool

true

false

**;**

ident

int

ident

bool

}

statement

ident

+

block

read

+

do

**;**

ident

)

-

-

=

expression

write

ident

while

(

expression

block

repeat

)

while

(

expression

block

if

)

else

(

condition

condition

condition

expression

statement

statement

for

；

(

statement

codition

)

expression

；

statement

statement

condition

expression

odd

=

expression

!=

<=

<

>

>=

expression

expression

term

+

term

-

||

xor

term

factor

\*

factor

/

%

&&

expression

factor

!

factor

ident

number

true

false

(

(

**2.1.2判断是否符合两条限制规则**

符合。

**2.1.3 过程调用相关图**

program

condition

expression

term

factor

block

statement

**2.1.4 程序总体结构**

void build(char\* fname) //构建符号表，

void error(int n); //打印错误信息。

void getsym(); //获取当前符号。

void getch(); //获取当前字符。

void init(); //初始化编译器。

void gen(enum fct x, int y, int z); //生成目标代码，x 是操作类型，y 和 z 是操作数。

void test(bool\* s1, bool\* s2, int n); //测试是否为期望的符号。

int inset(int e, bool\* s); //测试字符 e 是否在集合 s 中。

int addset(bool\* sr, bool\* s1, bool\* s2, int n); //将集合s1和集合s2合并到集合sr中。

int subset(bool\* sr, bool\* s1, bool\* s2, int n);.//将集合s1减去集合 s2 的结果存储在集合 sr 中。

int mulset(bool\* sr, bool\* s1, bool\* s2, int n);//将集合 s1 与集合 s2 的交集存储在集合 sr 中。

void block(int lev, int tx, bool\* fsys);//分析分程序，lev 是层数，tx 是符号表指针，fsys 是语句后的符号集。

void interpret();//解释执行目标代码。

void factor(bool\* fsys, int\* ptx, int lev);//分析因子。

void term(bool\* fsys, int\* ptx, int lev);//分析项。

void condition(bool\* fsys, int\* ptx, int lev);//分析条件。

void expression(bool\* fsys, int\* ptx, int lev);//分析表达式。

void statement(bool\* fsys, int\* ptx, int lev);//分析语句。

void listall();//打印符号表中所有变量的信息。

void intdeclaration(int\* ptx, int lev, int\* pdx);//分析整型变量的声明。

void booldeclaration(int\* ptx, int lev, int\* pdx);//分析布尔型变量的声明。

void constdeclaration(int\* ptx, int lev, int\* pdx);//分析常量的声明。

int position(char\* idt, int tx);//在符号表中查找标识符，并返回其地址。

void enter(bool bconst, enum object k, int\* ptx, int lev, int\* pdx);//将一个新的标识符插入符号表中。

int base(int l, int\* s, int b);//获取相对地址为 l，层数为 b 的数据所在的绝对地址

**2.1.5 语法出错表定义**

|  |  |
| --- | --- |
| 错误号 | 错误原因 |
| 1 | 要用“=”而不是“==” |
| 2 | “=”之后必须跟随一个数 |
| 3 | “=”之后必须跟随一个布尔值 |
| 4 | 这里必须是一个“=” |
| 5 | 这里必须是一个标识符 |
| 6 | 丢了一个分号 |
| 7 | 这里等待一条语句 |
| 8 | block语句部分之后出现的不正确符号 |
| 9 | 该标识符没有说明 |
| 10 | 给常量赋值是不允许的 |
| 11 | 这里等待“{” |
| 12 | 这里等待“}” |
| 13 | 该语句跟着一个不正确的使用符号 |
| 14 | 这里等待一个关系运算符 |
| 15 | 丢了右括号 |
| 16 | 丢了左括号 |
| 17 | 表达式不能以此符号开始 |
| 18 | 数字位数超过14位 |
| 19 | 语句不完整 |
| 20 | 要用“&&”而不是“&” |
| 21 | 要用“||”而不是“|” |
| 22 | 缺少正确的常量类型声明 |
| 23 | 只有do没有while |
| 24 | 只有repeat没有until |
| 25 | Too much read |
| 26 | if缺少右括号 |

**2.2 虚拟机**

**2.2.1 虚拟机组织结构**

虚拟机由两个存储器、一个指令寄存器和三个地址寄存器组成。

程序存储器code用来存放通过编译产生的中间代码程序，它在程序解释执行过程中保持不变。数据存储器s被当成数据栈使用，所有的算术和关系操作符都从栈顶找到它的操作数，又以计算结果取而代之。

栈顶数据单元的地址用地址寄存器t标记。数据存储器s只有在代码程序被解释执行时才开始使用。

指令寄存器i含有正在解释的指令。

程序地址寄存器p含有下一条要从程序存储器取得的、被解释执行指令的地址。

基本地址寄存器b在解释执行时才用到，开始值为1.它等于正在执行的过程段在数据栈的起始地址。

（1）程序存储器code

/\* 虚拟机代码指令 \*/

enum class fct {

lit, opr, lod,

sto, jpc, ini,

jmp,

};

/\* 虚拟机代码结构 \*/

struct instruction

{

enum fct f; /\* 虚拟机代码指令 \*/

int l; /\* 引用层与声明层的层次差 \*/

int a; /\* 根据f的不同而不同 \*/

};

struct instruction code[cxmax];

（2）数据存储器s

int s[stacksize]; /\* 栈 \*/

（3）程序地址寄存器p

int p;

（4）地址寄存器t

int t;

（5）指令寄存器i

struct instruction i;

（6）基本地址寄存器b

int b;

**2.2.2 虚拟机指令格式**

1. LIT指令，把一个常数置入栈顶。
2. LOD指令，把一个变量置入栈顶。
3. STO指令，从栈顶把数置入到一个变量单元里。
4. INT指令，预留数据存储位置。
5. JMP、JPC和JEQ指令，程序转移指令。其中JMP是无条件转移指令；JPC是有条件转移指令。
6. OPR一组算数和关系运算指令。

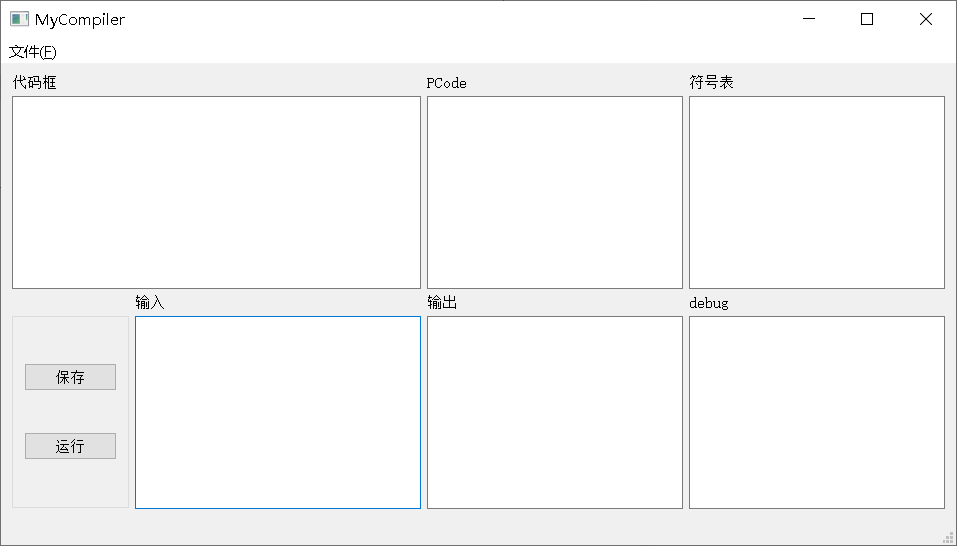
指令格式如下图所示，由3个指令分量确定。每条指令有一个指令码f，表示指令类型，一个（或两个）参数。在指令码为OPR时，参数a即是具体的操作符，决定了执行什么操作。对LIT和INT指令，参数a是一个数。对JMP、JPC和JEQ指令，参数a是一个程序地址。对LOD和STO指令，参数l和a决定了要存取的变量在数据栈的地址。

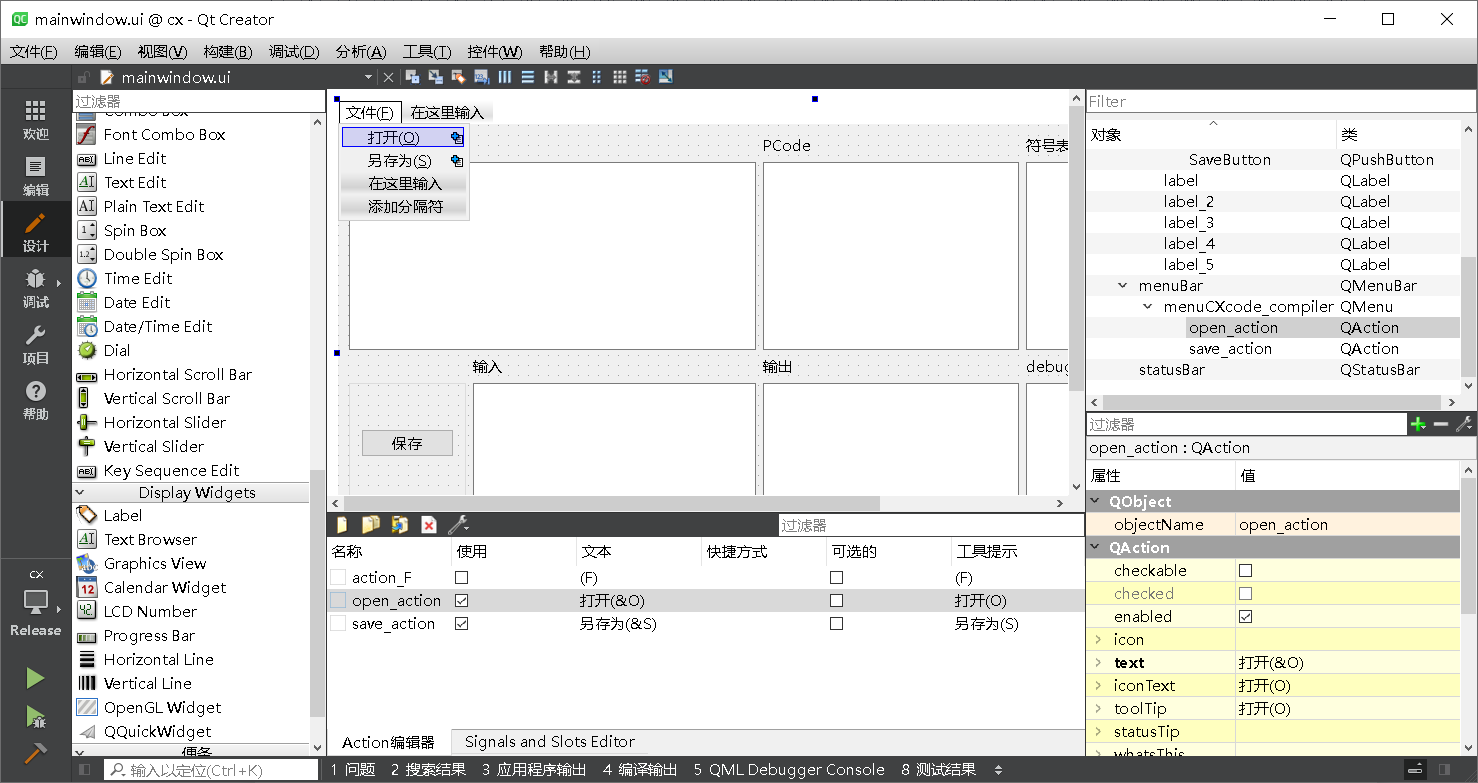
|  |  |  |
| --- | --- | --- |
| f | l | a |

**2.2.3 虚拟机指令系统及其解释**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 指令 | 实现解释的C语句 | 说明 |
| 1 | LIT 0,a | t = t + 1;  s[t] = a; | 将数a置入栈顶 |
| 2 | LOD l,a | t = t + 1;  s[t] = s[base(l,s,b) + a]; | 将l,a形成的栈地址变量值置入栈顶 |
| 3 | STO l,a | s[base(l, s, b) + a] = s[t];  t = t – 1; | 将栈顶值存到由l,a形成的栈地址变量 |
| 4 | INT 0,a | t = t + a; | 预留a个存储位置 |
| 5 | JMP 0,a | p = a; | 无条件转移 |
| 6 | JPC 0,a | if (s[t] == 0)  p = a;  t = t – 1; | 条件转移 |
| 7 | JEQ 0,a | if (s[t] != 0)  p = a;  t = t – 1; | 条件转移 |
| 8 | OPR 0,a |  | 一组算术关系运算符 |
|  | 当a=0 | t = b – 1;  p = s[t +3];  b = s[t + 3]; | 返回调用程序 |
|  | 当a=1 | s[t] = -s[t]; | 取负 |
|  | 当a=2 | t = t – 1;  s[t] = s[t] + s[t + 1]; | 相加 |
|  | 当a=3 | t = t – 1;  s[t] = s[t] - s[t + 1]; | 相减 |
|  | 当a=4 | t = t – 1;  s[t] = s[t] \* s[t + 1]; | 相乘 |
|  | 当a=5 | t = t – 1;  s[t] = s[t] / s[t + 1]; | 相除 |
|  | 当a=6 | s[t] = s[t] % 2; | 判奇偶 |
|  | 当a=7 | t = t – 1;  s[t] = s[t] % s[t + 1]; | 取余 |
|  | 当a=8 | t = t – 1;  s[t] = (s[t] == s[t + 1]); | 判相等 |
|  | 当a=9 | t = t – 1;  s[t] = (s[t] != s[t + 1]); | 判不等 |
|  | 当a=10 | t = t – 1;  s[t] = (s[t] < s[t + 1]); | 判小于 |
|  | 当a=11 | t = t – 1;  s[t] = (s[t] >= s[t + 1]); | 判大于等于 |
|  | 当a=12 | t = t – 1;  s[t] = (s[t] > s[t + 1]); | 判大于 |
|  | 当a=13 | t = t – 1;  s[t] = (s[t] <= s[t + 1]); | 判小于等于 |
|  | 当a=14 | printf("%d", s[t]);  t = t – 1; | 栈顶值输出 |
|  | 当a=15 | printf("\n"); | 输出换行符 |
|  | 当a=16 | t = t + 1;  scanf("%d", &(s[t])); | 输入 |
|  | 当a=17 | t = t – 1;  s[t] = (s[t] & s[t + 1]); | 逻辑 与 |
|  | 当a=18 | t = t – 1;  s[t] = (s[t] | s[t + 1]); | 逻辑或 |
|  | 当a=19 | s[t] = s[t] ^ 1; | 取反 |
|  | 当a=20 | t = t – 1;  s[t] = (s[t] ^ s[t + 1]); | 异或 |

**3.模块架构**





整体架构分为两个可读写文本框（代码框和输入框）、四个只读文本框（PCode、符号表、输出框和debug框）、两个PushButton和六个Label，一个“文件(&F)”,点击文件（&F），会弹出“打开（&O）”和“另存为（&S）”，以及一个GroupBox。

**4.模块功能介绍**

代码框：可读写文本框，用于显示和编辑CX源程序代码。

符号表框：只读文本框，用于显示CX程序变异后的符号表信息。

Debug框：逐行编译程序，如果没有出错，代码将输出到本debug框中，如果有错误，编译到出错行会输出出错信息。

输出框：只读文本框，用于显示CX程序运行后的结果，即存放smallC程序的输出结果。

Pcode：只读文本框，用于显示CX程序运pcode代码。

输入框：可读写文本框，用于输入。

“打开文件”open\_action：单击后弹出文件夹，用于选择导入的CX源程序文件。

“另存为”save\_action：单击后弹出文件夹，用于选择另外保存新建或者修改后源程序文件。

“保存”按钮：将代码框中的内容存入各个txt文件中。

“运行”按钮：对代码框中的源程序进行编译运行。

**5.模块接口**

private slots:

void on\_BuildButton\_clicked();

void on\_SaveButton\_clicked();

void openActionSlot();

void saveActionSlot();

MainWindow::MainWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::MainWindow)

{

ui->setupUi(this);

this->setWindowTitle("MyCompiler");

//建立连接

connect(ui->open\_action, &QAction::triggered, this, &MainWindow::openActionSlot);

connect(ui->save\_action, &QAction::triggered, this, &MainWindow::saveActionSlot);

}

MainWindow::~MainWindow()

{

delete ui;

}

//打开文件

void MainWindow::openActionSlot()

{

QString fileName= QFileDialog::getOpenFileName(this, "选择一个文件",

QCoreApplication::applicationFilePath(), "\*.txt");

if(fileName.isEmpty())

{

QMessageBox::warning(this,"警告","请选择一个文件");

}

else

{

//qDebug() <<fileName;

QFile file(fileName); //创建文件对象

file.open(QIODevice::ReadOnly);

QByteArray ba = file.readAll();

ui->CxCode->setText(QString(ba));

file.close();

}

}

//另存为

void MainWindow::saveActionSlot()

{

QString fileName = QFileDialog::getSaveFileName(this, "选择一个文件",

QCoreApplication::applicationFilePath());

if(fileName.isEmpty())

{

QMessageBox::warning(this,"waring","please choose a file");

}

else {

QFile file(fileName);

file.open(QIODevice::WriteOnly);

//ui->CxCode-toPlaintext();

QByteArray ba;

ba.append(ui->CxCode->toPlainText());

file.write(ba);

file.close();

}

}

void MainWindow::savefilein(QString path){

QString data;

QFile file(path);

//以文本方式打开

if( file.open(QIODevice::WriteOnly | QIODevice::Text) )

{

QTextStream out(&file); //IO设备对象的地址对其进行初始化

out << ui->CxCode->toPlainText() << endl; //输出

file.close();

}

}

void MainWindow::savefileinput(QString path){

QString data;

QFile file(path);

//以文本方式打开

if( file.open(QIODevice::WriteOnly | QIODevice::Text) )

{

QTextStream out(&file); //IO设备对象的地址对其进行初始化

out << ui->InputEdit->toPlainText() << endl; //输出

file.close();

}

}

void MainWindow::openfilein(QString path){

QString data;

QFile file(path); //path是文件的路径

if(!file.open(QIODevice::ReadOnly | QIODevice::Text))

{

qDebug()<<"文件未打开！";

}

while(!file.atEnd())

{

QByteArray array = file.readLine();

QString str(array);

// qDebug()<< str;

data.append(str);

}

ui->CxCode->setText(data);

}

void MainWindow::openfileout(QString path){

QString data;

QFile file(path); //path是文件的路径

if(!file.open(QIODevice::ReadOnly | QIODevice::Text))

{

qDebug()<<"文件未打开！";

}

while(!file.atEnd())

{

QByteArray array = file.readLine();

QString str(array);

// qDebug()<< str;

data.append(str);

}

ui->OutputEdit->setText(data);

}

void MainWindow::openfilecode(QString path){

QString data;

QFile file(path); //path是文件的路径

if(!file.open(QIODevice::ReadOnly | QIODevice::Text))

{

qDebug()<<"文件未打开！";

}

while(!file.atEnd())

{

QByteArray array = file.readLine();

QString str(array);

// qDebug()<< str;

data.append(str);

}

ui->PcodeEdit->setText(data);

}

void MainWindow::openfiletable(QString path){

QString data;

QFile file(path); //path是文件的路径

if(!file.open(QIODevice::ReadOnly | QIODevice::Text))

{

qDebug()<<"文件未打开！";

}

while(!file.atEnd())

{

QByteArray array = file.readLine();

QString str(array);

// qDebug()<< str;

data.append(str);

}

ui->TableEdit->setText(data);

}

void MainWindow::openfileinput(QString path){

QString data;

QFile file(path); //path是文件的路径

if(!file.open(QIODevice::ReadOnly | QIODevice::Text))

{

qDebug()<<"文件未打开！";

}

while(!file.atEnd())

{

QByteArray array = file.readLine();

QString str(array);

// qDebug()<< str;

data.append(str);

}

ui->InputEdit->setText(data);

}

void MainWindow::openfileres(QString path){

QString data;

QFile file(path); //path是文件的路径

if(!file.open(QIODevice::ReadOnly | QIODevice::Text))

{

qDebug()<<"文件未打开！";

}

while(!file.atEnd())

{

QByteArray array = file.readLine();

QString str(array);

// qDebug()<< str;

data.append(str);

}

ui->ResultEdit->setText(data);

}

void MainWindow::resload()

{

QLatin1String qsres("fresult.txt");

QString ffres(qsres);

this->openfileres(ffres);

}

void MainWindow::load()

{

QLatin1String qsin("test.txt");

QString ffin(qsin);

QLatin1String qsout("foutput.txt");

QString ffout(qsout);

QLatin1String qscode("fcode.txt");

QString ffcode(qscode);

QLatin1String qstable("ftable.txt");

QString fftable(qstable);

QLatin1String qsinput("finput.txt");

QString ffinput(qsinput);

this->openfilein(ffin);

this->openfileout(ffout);

this->openfilecode(ffcode);

this->openfiletable(fftable);

this->openfileinput(ffinput);

}

//运行按钮

void MainWindow::on\_BuildButton\_clicked()

{

//char path[100] = "test.txt";

build("test.txt");

load();

resload();

}

//保存按钮

void MainWindow::on\_SaveButton\_clicked()

{

QLatin1String qsin("test.txt");

QString ffin(qsin);

this->savefilein(ffin);

QLatin1String qsinput("finput.txt");

QString ffinput(qsinput);

this->savefileinput(ffinput);

//load();

}

**6．全局数据结构、常量和变量**

#include<string.h>

#define bool int

#define true 1

#define false 0

#define norw 16 /\* 保留字个数 \*/

#define txmax 100 /\* 符号表容量 \*/

#define nmax 14 /\* 数字的最大位数 \*/

#define al 10 /\* 标识符的最大长度 \*/

#define maxerr 30 /\* 允许的最多错误数 \*/

#define amax 2048 /\* 地址上界\*/

#define levmax 3 /\* 最大允许过程嵌套声明层数\*/

#define cxmax 200 /\* 最多的虚拟机代码数 \*/

#define stacksize 500 /\* 运行时数据栈元素最多为500个 \*/

/\* 符号 \*/

enum symbol {

nul, ident, number, truesym, falsesym,

plus, minus, times, slash, lss,

leq, gtr, geq, eql, neq,

becomes, orsym, andsym, notsym, semicolon,

lparen, rparen, lbrace, rbrace, mod,

xorsym, oddsym, incre, decre, intsym,

boolsym, ifsym, elsesym, whilesym, writesym,

readsym, forsym, dosym, constsym, repeatsym,

untilsym, period,

};

#define symnum 42

/\* 符号表中的类型 \*/

enum object {

integer,

booltype,

};

/\* 虚拟机代码指令 \*/

enum fct {

lit, opr, lod,

sto, ini, jmp,

jpc, jeq,

};

#define fctnum 8

/\* 虚拟机代码结构 \*/

struct instruction

{

enum fct f; /\* 虚拟机代码指令 \*/

int l; /\* 引用层与声明层的层次差 \*/

int a; /\* 根据f的不同而不同 \*/

};

/\* 符号表结构 \*/

struct tablestruct

{

char name[al]; /\* 名字 \*/

enum object kind; /\* 类型：integer或booltype \*/

bool bconst; /\* 是否为const \*/

int val; /\* 数值，仅const使用 \*/

int level; /\* 所处层，仅const不使用 \*/

int adr; /\* 地址，仅const不使用 \*/

};

struct tablestruct table[txmax]; /\* 符号表 \*/

FILE\* fin; /\* 输入源文件 \*/

FILE\* ftable; /\* 输出符号表 \*/

FILE\* fcode; /\* 输出虚拟机代码 \*/

FILE\* foutput; /\* 输出文件及出错示意（如有错）、各行对应的生成代码首地址（如无错） \*/

FILE\* fresult; /\* 输出执行结果 \*/

char fname[al];

int err; /\* 错误计数器 \*/

char ch; /\* 存放当前读取的字符，getch 使用 \*/

enum symbol sym; /\* 当前的符号 \*/

char id[al+1]; /\* 当前ident，多出的一个字节用于存放0 \*/

int num; /\* 当前number \*/

int cc, ll; /\* getch使用的计数器，cc表示当前字符(ch)的位置 \*/

int cx; /\* 虚拟机代码指针, 取值范围[0, cxmax-1]\*/

int dx; /\* 记录数据分配的相对地址 \*/

char line[81]; /\* 读取行缓冲区 \*/

char a[al+1]; /\* 临时符号，多出的一个字节用于存放0 \*/

struct instruction code[cxmax]; /\* 存放虚拟机代码的数组 \*/

char word[norw][al]; /\* 保留字 \*/

enum symbol wsym[norw]; /\* 保留字对应的符号值 \*/

enum symbol ssym[256]; /\* 单字符的符号值 \*/

char mnemonic[fctnum][5]; /\* 虚拟机代码指令名称 \*/

bool declbegsys[symnum]; /\* 表示声明开始的符号集合 \*/

bool statbegsys[symnum]; /\* 表示语句开始的符号集合 \*/

bool facbegsys[symnum]; /\* 表示因子开始的符号集合 \*/

int rel;

char errormsg[50][50];

int fx; /\* record the index of read fscanf \*/

int maxfx;

int readtimes; /\* the times of read \*/

int preread[stacksize]; /\* record the pre read \*/

**7.函数原型**

|  |  |
| --- | --- |
| 函数原型 | void build(char\* fname); |
| 参数描述 | 代码文件名 |
| 函数描述 | 编译一段代码 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void init(); |
| 参数描述 | 无 |
| 函数描述 | 初始化 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | int inset(int e, bool\* s) |
| 参数描述 | \*s为数组指针，整型数e是数组中一个元素 |
| 函数描述 | 判断元素e是否在数组s中 |
| 返回值 | 1/0 |

|  |  |
| --- | --- |
| 函数原型 | int addset(bool\* sr, bool\* s1, bool\* s2, int n) |
| 参数描述 | \*sr \*s1 \*s2为数组指针，整型数n为数组长度 |
| 函数描述 | 指针s1所指数组和s2所指数组进行或运算并保存至sr所指数组中 |
| 返回值 | 0 |

|  |  |
| --- | --- |
| 函数原型 | int subset(bool\* sr, bool\* s1, bool\* s2, int n) |
| 参数描述 | \*sr \*s1 \*s2为数组指针，整型数n为数组长度 |
| 函数描述 | 指针s1所指数组和s2所指数组进行差运算并保存至sr所指数组中 |
| 返回值 | 0 |

|  |  |
| --- | --- |
| 函数原型 | int mulset(bool\* sr, bool\* s1, bool\* s2, int n) |
| 参数描述 | \*sr \*s1 \*s2为数组指针，整型数n为数组长度 |
| 函数描述 | 指针s1所指数组和s2所指数组进行交运算并保存至sr所指数组中 |
| 返回值 | 0 |

|  |  |
| --- | --- |
| 函数原型 | void listall() |
| 参数描述 | 无 |
| 函数描述 | 输出所有目标代码 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | int base(int l, int\* s, int b) |
| 参数描述 | l为当前函数层次，s为运行时数据栈指针，b为当前过程基址 |
| 函数描述 | 通过过程基址求上l层过程的基址 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void error(int n); |
| 参数描述 | 整数参数n，表示错误号 |
| 函数描述 | 出错处理，打印出错位置和错误编码 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void getsym(); |
| 参数描述 | 无 |
| 函数描述 | 词法分析，获取一个符号 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void getch(); |
| 参数描述 | 无 |
| 函数描述 | 读取一个字符 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void gen(enum fct x, int y, int z); |
| 参数描述 | 虚拟机代码x，y，z  x表示操作码的枚举类型 fct，  y表示层差的整数类型 l，  z表示偏移地址的整数类型 a。 |
| 函数描述 | 生成虚拟机代码指令 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void test(bool\* s1, bool\* s2, int n); |
| 参数描述 | s1: 需要的单词集合  s2: 如果不是需要的单词，在某一出错状态时，可恢复语法分析继续正常工作的补充单词符号集合  n: 错误号 |
| 函数描述 | 测试当前符号是否合法 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void block(int lev, int tx, bool\* fsys); |
| 参数描述 | lev: 当前分程序所在层  tx: 符号表当前尾指针  fsys: 当前模块后继符号集合 |
| 函数描述 | 编译程序主体 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void enter(bool bconst, enum object k, int\* ptx, int lev, int\* pdx)； |
| 参数描述 | k ： 标识符的种类为const，var或procedure；  Ptx：符号表尾指针的指针，为了可以改变符号表尾指针的值  lev：标识符所在的层次  pdx:dx为当前应分配的变量的相对地址，分配后要增加1 |
| 函数描述 | 用于将标识符插入符号表中 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | int position(char\* id, int tx) |
| 参数描述 | Id:要查找的名字  Tx:当前符号表尾指针 |
| 函数描述 | 查找标识符在符号表中的位置，从tx开始倒序查找标识符 |
| 返回值 | 找到则返回在符号表中的位置，否则返回0 |

|  |  |
| --- | --- |
| 函数原型 | void constdeclaration(int\* ptx, int lev, int\* pdx); |
| 参数描述 | ptx: 符号表尾指针的指针，为了可以改变符号表尾指针的值  lev: 标识符所在的层次  pdx: dx为当前应分配的变量的相对地址，分配后要增加1 |
| 函数描述 | 常量声明处理 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void intdeclaration(int\* ptx,int lev,int\* pdx) |
| 参数描述 | 它接受三个参数：一个表示符号表的当前位置的指针 ptx，一个表示当前嵌套层数的整数类型 lev，以及一个表示当前数据空间相对地址的指针 pdx |
| 函数描述 | 用于处理整型变量的声明 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void booldeclaration(int\* ptx,int lev,int\* pdx) |
| 参数描述 | 接受三个参数：一个表示符号表的当前位置的指针 ptx，一个表示当前嵌套层数的整数类型 lev，以及一个表示当前数据空间相对地址的指针 pdx。 |
| 函数描述 | 用于处理布尔型变量的声明 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void listall() |
| 参数描述 | 无 |
| 函数描述 | 输出所有目标代码 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void statement(bool\* fsys, int\* ptx, int lev); |
| 参数描述 | ptx: 符号表尾指针的指针，为了可以改变符号表尾指针的值  lev: 标识符所在的层次  fsys: 后继符号集合 |
| 函数描述 | 语句处理 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void expression(bool\* fsys, int\* ptx, int lev); |
| 参数描述 | ptx: 符号表尾指针的指针，为了可以改变符号表尾指针的值  lev: 标识符所在的层次  fsys: 后继符号集合 |
| 函数描述 | 表达式处理 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void term(bool\* fsys, int\* ptx, int lev) |
| 参数描述 | lev表示当前分程序所在层，ptx为符号表尾指针的指针，fsys为当前模块跟随符集合 |
| 函数描述 | 对文法中的term模块进行处理 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void factor(bool\* fsys, int\* ptx, int lev) |
| 参数描述 | lev表示当前分程序所在层，ptx为符号表尾指针的指针，fsys为当前模块跟随符集合 |
| 函数描述 | 对文法中的factor模块进行处理 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void condition(bool\* fsys, int\* ptx, int lev); |
| 参数描述 | ptx: 符号表尾指针的指针，为了可以改变符号表尾指针的值  lev: 标识符所在的层次  fsys: 后继符号集合 |
| 函数描述 | 条件处理 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void interpret() |
| 参数描述 | 无 |
| 函数描述 | 解释程序 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | int base(int l, int\* s, int b) |
| 参数描述 | l为当前函数层次，s为运行时数据栈指针，b为当前过程基址 |
| 函数描述 | 通过过程基址求上l层过程的基址 |
| 返回值 | 无 |

|  |  |
| --- | --- |
| 函数原型 | void debug() |
| 参数描述 | 无 |
| 函数描述 | 调试程序 |
| 返回值 | 无 |