《编译技术》课程设计文 档

学号：\_\_\_\_\_\_14061186\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_张 巍\_\_\_\_\_\_\_

2017 年 月 日

## 一．需求说明

### 1．文法说明

难度2所使用的是扩充的pl0文法，有整型、实型和字符型三种数据类型，包含了一维数组这种数据结构，循环语句有for循环和do-while循环两类，有条件语句if-then(-else)。具体见文法说明文档。

### 2．目标代码说明

难度2需要生成code并对code进行解释执行，参照教材给出的P-code制定了规则，其格式如下：

|  |  |  |
| --- | --- | --- |
| int instruction | int l | float a |

这里将目标码的指令声明了枚举类型变量：

enum inst{LIT, OPR,LOD, STO,CAL,INTE,JMP, JPC,RED,WRT,END};

各个指令的使用方法以及含义如下：

①LIT 0 a：将常量值加载到运行栈顶，a为常数值。

②OPR 0 a：算数运算指令和关系运算指令， a表示运算的类型，具体如下：

a==0：加法+；

a==1：减法-；

a==2：乘法\*；

a==3：除法/；

a==4：是否小于<，小于返回1；

a==5：是否小于等于<=，小于等于返回1；

a==6：是否等于=，等于返回1；

a==7：是否不等于<>，不等于返回1；

a==8：是否大于>，大于返回1；

a==9：是否大于等于>=，大于等于返回1。

③LOD 1 a：取变量（相对地址为a，层次差为1）放到数据栈栈顶。

④STO 1 a：将栈顶保存至变量（相对地址为a，层次差为1）。

⑤CAL 1 a：调用函数（入口指令地址为a，层次差为1）。

⑥CLL 1 a：调用过程（入口指令地址为a，层次差为1）。

⑦ADD 0 a：数据栈栈顶指针自加a。

⑧JMP 0 a：无条件跳转到a。

⑨JPC 0 a：条件（未通过时）跳转到a。

⑩RED 1 a：读取数据并存入变量（相对地址为a，层次差为1）。

11、WRT 0 0：将栈顶内容输出。

12、END 0 0：用于标记函数（2）、过程（1）以及程序（0）的结尾

13、LDD 0 a:数组取值，偏移在栈顶

14、SDD 0 a:数组存值，偏移在栈顶

### 3.优化方案\*

难度2没有要求进行优化

## 二．详细设计

### 1．程序结构

【从总体上描述程序的结构，文字或图示均可】

### 2．类/方法/函数功能

【描述各类/方法或函数的功能，以及关键算法】

### 3．调用依赖关系

【说明各类之间的关系，方法/函数之间的调用关系】

### 4．符号表管理方案

符号表项及其含义如下所示：

typedef struct sym{

char name[100];///名称，也可以保存一个完整的字符串

char type[20];//类型int char string float

char kind[20];//array func pro const var///这个 地方是为了这类特别的元素准备的，其他统称normal，这一位置只需在声明语句中初始化，

float value;//值//这里只记录symbol的名称和类型，若为变量或常量则以浮点数形式记录其值，字符保存的是ascii码

//对于function 和procedure,value 保存其入口指令地址

int level;///所属分程序的起始地址

int addr;///这里对过程和函数是起始地址，对其内部声明的量是相对地址。

}symbol;

符号表是symbol类型的数组，为便于管理，将程序中出现的每个量都保存在了符号表中，并在符号表中保存了其在虚拟地址空间中的地址，便于查找和解释执行。添加符号表项时，首先按照上面给出的各个成员的涵义对符号表项进行赋值，然后直接将其存入符号表。

在建立符号表时，各个元素仅按照先后顺序存入符号表，在检索符号表时为顺序查找，为了避免重名问题，检索时需要给出一个起始位置，从起始位置起通过比对符号表项的名称和类型两个元素查找所需的元素，查找的起始点是在函数或过程声明以及函数及过程调用时修改了全局的符号表基地址变量记录的，默认函数名和过程名都是唯一的。

### 5．存储分配方案

为了便于管理，在进行语法分析的过程中，为程序中出现的每个量分配虚拟的内存空间，用浮点型数组vm[1000]来保存，但是实际上除了保存常量并没有多少作用。

有关运行栈进行了如下设计：运行栈是一个浮点型数组，当新的模块被调用时，首先为其参数分配空间，将其保存，并标记当前模块为上一级模块，然后调用新的模块，为其中的变量分配数据空间，在数据栈顶进行必要的包括表达式计算、条件计算以及输入输出等操作。当被调用模块运行完毕时，清空其数据区，并清空参数所占空间。

### 6. 解释执行程序\*

此次课程设计中有关解释执行的部分主要有两个结构，一个是虚拟地址空间，用于保存各类元素的值，一个是运行栈，两者都是浮点型。解释执行的关键算法与教材上给出的编译器实例比较接近，即在进行某个模块（即某个过程或函数）的调用的时候，首先在运行栈中预留出该模块内变量所占的空间，然后再进行运算，所有运算涉及的都是栈顶的两个元素，在完成了该模块的运算后，进行必要的存储动作将数据写回地址空间，然后撤销此模块在运行栈中的数据区。解释执行的原理与汇编类似，首先找到程序的入口，然后根据实际指令进行跳转和运算。

在进行函数或过程的调用时，会首先为参数申请空间，并将参数传入运行栈，然后会跳过函数或过程的首部，为新的模块层级申请空间，跳转到相应分程序代码。有关参数不匹配的问题，程序中并为记录函数参数的个数，但是在解释执行时若发现某一变量无法从当前模块或参数表中获取时即可发现参数不匹配的错误。

对于如if语句的条件跳转情形，在进行条件的计算后，若条件不通过则跳过then分支，若条件通过则顺序执行，在then分支结尾会跳过可能存在的else分支。

对于for和while两种循环语句，循环主体均为“条件判定-循环体-步长减一-跳转到条件判定”的形式，for循环会在主体前添加步长的初始化指令。

### 7. 四元式设计\*

无需进行四元式设计

### 8. 目标代码生成方案\*

无需生成目标代码

### 9. 优化方案\*

无需进行代码优化

### 10. 出错处理

目前出错的部分会调用函数error（int a,int b），其中a为出错行数，b为错误类型，错误类型尚未全部确定现整理错误信息如下：

1、错误信息：1。

含义：标识符超长。

处理方案：照常保存标识符并报错，若标识符过长导致溢出则保留溢出之前的部分并将读取位置跳至标识符结束。

2、错误信息：2。

含义：数字溢出。

处理方案：用字符串保存整个数字，并将其标记。最终作品可能删去这一报错。

3、错误信息：3。

含义：运算符不完整，主要是针对“！=”符号。

处理方案：报错，保存不完整的运算符，将读取位置跳至不完整运算符结束。形如！！=一类的情况会读取一个“！=”符号和一个不完整符号。

4、错误信息：**4。**

含义：引号不匹配。

处理方案：将读取位置跳转至行末，保存当前引号到行末的全部内容为一个字符串，引号为单引号但内容不为字符时不在此考虑范围，会设置成另一种错误类型。

由于对文法和源码的分析尚布完善，还有很多的错误类型没有归纳出来。我将在接下来的作业中不断完善这方面内容，细化错误含义和处理方案。

11、错误信息：11。

含义：非法的数组声明，主要是由数组大小不合法，即不为整数

处理方案：报错并跳过声明部分。

## 三．操作说明

### 1．运行环境

【说明搭建运行环境的步骤】

### 2．操作步骤

【详细说明操作步骤】

## 四．测试报告

### 1．测试程序及测试结果

【给出提供的测试程序以及每个程序的测试结果，至少5个正确程序，5个错误程序，无需截屏】

### 2．测试结果分析

【说明上述测试程序对语法成分的覆盖情况】

## 五．总结感想

【说明在完成课程设计中的收获、认识和感想】

注：【】内的文字为文档模板说明，完成的作业中需去掉。

标\*的章节需根据题目的难度进行取舍。