《编译技术》课程设计文 档

学号：\_\_\_\_\_13061049\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_季豪\_\_\_\_\_\_\_\_\_

2015年11月19日

## 一．需求说明

### 1．文法说明

**1) 获取的文法（低难度PL/0）**

1.<程序>::=<分程序>.

2.<分程序>::=[<常量说明部分>][<变量说明部分>][<过程说明部分>]<复合语句>

3.<常量说明部分>::=const<常量定义>{,<常量定义>};

4.<常量定义>::=<标识符>=<无符号整数>

5.<无符号整数>::=<非零数字>{<数字>}

6.<标识符>::=<字母>{<字母>|<数字>}

7.<变量说明部分>::=var<标识符>{, <标识符>};

8.<过程说明部分>::=<过程首部><分程序>{;<过程说明部分>};

9.<过程首部>::=procedure<标识符>;

10.<语句>::=<赋值语句>|<条件语句>|<当循环语句>|<过程调用语句>|<复合语句>|<读语句>|<写语句>|<空>

11.<赋值语句>::=<标识符> := <表达式>

12.<表达式>::= [+|-]<项>{<加法运算符><项>}

13.<项>::=<因子>{<乘法运算符><因子>}

14.<因子>::=<标识符>|<无符号整数>| '('<表达式>')'

15.<加法运算符>::=+|-

16.<乘法运算符>::=\*|/

17.<条件>::=<表达式><关系运算符><表达式>|odd<表达式>

18.<关系运算符>::==|<>|<|<=|>|>=

19.<条件语句>::=if<条件>then<语句>[else<语句>]

20.<当循环语句>::=while<条件>do<语句>

21.<过程调用语句>::=call<标识符>

22.<复合语句>::=begin<语句>{;<语句>}end

23.<读语句>::=read '('<标识符>{, <标识符>}')'

24.<写语句>::=write '('<表达式>{, <表达式>}')'

25.<字母>::=a|b|c|d…x|y|z|A|B...|Z

26.<数字>::=0|<非零数字>

27.<非零数字>::=1|2|3…8|9

附加说明：标识符不区分大小写字母

1. **改写扩展**

由于原文法不存在左递归，因此不对文法进行改写

1. **文法解读**

|  |  |  |
| --- | --- | --- |
| 规则 | 分析 | 范例 |
| 1.<程序>::=<分程序>. | 程序主体是分程序，“.”相当于结束符号. | begin  end. |
| 2.<分程序>::=[<常量说明部分>][<变量说明部分>][<过程说明部分>]<复合语句> | 分程序严格按照顺序由常量说明部分，变量说明部分，过程说明部分，以及复合语句组成。  常量说明，变量说明，过程说明部分是【可有可无的】，复合语句是【必须存在的】 | const a=1,b=2;  var x,y;  begin  x:=a;  y:=b  end |
| 3.<常量说明部分> ::= const<常量定义>{,<常量定义>}; | const a=1,b=2,c=3;这种格式是正确的  const a=1; const b=2;这种是错误的 | const a=1,b=2; |
| 4.<常量定义>::=<标识符>=<无符号整数> | 常量定义的时候必须带<无符号整数>,并且用的是“=”而不是“:=” | a=1 |
| 5.<无符号整数>::=<非零数字>{<数字>} | 这里的无符号整数不允许0开头，并且是不包含0的。  我觉得这里的文法改动略微有些不妥，虽然改动文法有助于锻炼同学们的能力，但是把{1,2,3,4......}作为无符号整数的定义与常识相悖，一些无助于知识技能掌握的逻辑问题可能会阻碍同学们的学习进程，我建议以后学弟学妹的教学这里可以改成：  <无符号整数>::=<非零数字>{<数字>}|0 | 正确：1,2,10,876  错误：0,000003,0010 |
| 6.<标识符>::=<字母>{<字母>|<数字>} | 标识符必须是是字母打头，后面可以跟数字或字母，按照附加说明，标识符不区分大小写字母 | 正确：a,b,a1,p5p  错误：3c,25,5b2 |
| 7.<变量说明部分>::=var<标识符>{, <标识符>}; | 与常量说明类似  var x,y,z;这种格式是正确的  vart x; var y;这种是错误的  并且只能跟标识符，不能给标识符赋值。 | var x,y,z; |
| 8.<过程说明部分>::=<过程首部><分程序>{;<过程说明部分>}; | 过程说明部分右递归 | procedure test;  begin  end; |
| 9.<过程首部>::=procedure<标识符>; | 与常量、变量说明类似，procedure后跟一个标识符和“;” | procedure test; |
| 10.<语句>::=<赋值语句>|<条件语句>|<当循环语句>|<过程调用语句>|<复合语句>|<读语句>|<写语句>|<空> | 语句包括赋值语句、条件语句、当循环语句、过程调用语句、复合语句、读语句、写语句、空语句 | x:=1;  if x>2 then  x:=3  else  x:=4;  while x>1 do  call dosth;  begin  read(x);  write(2)  end;  ; |
| 11.<赋值语句>::=<标识符> := <表达式> | 注意赋值语句本身是没有“;”的，另一点由于无符号整数规则的缘故x:=0也是不符合规则的 | x:=1  x:=1-1 |
| 12.<表达式>::= [+|-]<项>{<加法运算符><项>} | 项的前面只能带一个加法运算符，形如++x,+-y是不符合规则的 | -2+3\*5 |
| 13.<项>::=<因子>{<乘法运算符><因子>} | 项由因子和乘法运算符组成 | 2\*3/4 |
| 14.<因子>::=<标识符>|<无符号整数>| '('<表达式>')' | 因子可以是标识符，也可以是无符号整数，也可是括号括起来的表达式。 | x  2  (1+3) |
| 15.<加法运算符>::=+|- | 加法运算符包括+、- |  |
| 16.<乘法运算符>::=\*|/ | 乘法运算符包括\*、/ |  |
| 17.<条件>::=<表达式><关系运算符><表达式>|odd<表达式> | 条件分为两种，一种是二元的<表达式><关系运算符><表达式>，一种是一元的odd<表达式>，odd是判奇数 | x>3  odd 7 |
| 18.<关系运算符>::==|<>|<|<=|>|>= | 关系运算符包括等于、不等于、小于、小于等于、大于、大于等于 |  |
| 19.<条件语句>::=if<条件>then<语句>[else<语句>] | 条件语句 if 条件 后面必须跟 then 语句，else 语句可有可无，并且then 语句后面不跟“;” | if x>2 then  x:=3  else  x:=4 |
| 20.<当循环语句>::=while<条件>do<语句> | 当循环语句 while 条件后面必须跟 do 语句 | while(x>1) do  write(1) |
| 21.<过程调用语句>::=call<标识符> | 过程调用语句以 call打头 后面跟标识符 | call dosth |
| 22.<复合语句>::=begin<语句>{;<语句>}end | 复合语句以begin为开头，end为结尾，中间语句以“;”作分割，但是最后一条语句和end之间没有“;”，如果有，视为最后还有一句空语句 | begin  read(x);  write(2)  end |
| 23.<读语句>::=read '('<标识符>{, <标识符>}')' | 读语句以read开头，括号里面跟标识符，以“,”分割。 | read(x,y,z) |
| 24.<写语句>::=write '('<表达式>{, <表达式>}')' | 与读语句类似，以write开头，括号里面跟表达式，以“,”分割 | write(x,x+y+z) |
| 25.<字母>::=a|b|c|d…x|y|z|A|B...|Z | 字母包括a~z和A~Z，根据附加说明，标识符不区分大小写字母。 |  |
| 26.<数字>::=0|<非零数字> | 数字包括0、1、2、3、4、5、6、7、8、9 |  |
| 27.<非零数字>::=1|2|3…8|9 | 非零数字包括1、2、3、4、5、6、7、8、9 |  |

### 2．目标代码说明

**1)文法生成的目标代码是p-code码，由interpret()函数解释执行**

enumfct {

lit, opr, lod, sto, cal, Int, jmp, jpc,red,wrt

};

Typedef struct {

enumfct f;

int l;

int a;

} instruction;

**2)目标代码指令集**

|  |  |
| --- | --- |
| 指令 | 主要功能 |
| **lit** | 把常量值放到运行栈栈顶 |
| **opr** | 运算操作 |
| **lod** | 找到变量的地址，并存入栈顶 |
| **sto** | 将数据栈顶内容保存入变量 |
| **cal** | 过程调用 |
| **int** | 开辟新的内存单元 |
| **jmp** | 无条件跳转 |
| **jpc** | 有条件跳转 |
| **red** | 读入数据 |
| **wrt** | 输出数据 |

## 二．详细设计

### 1．程序结构

**1)函数定义层次结构**

main()

    error()//错误处理

    getsym()//词法分析

        getch()

    gen()//生成P-CODE

    test()//测试当前单词符号是否合法

    block()//分程序分析

        enter()

        position()

        constdeclaration()

        vardeclaration()

        listcode()

        statement()//语句分析处理

            expression()

                term()

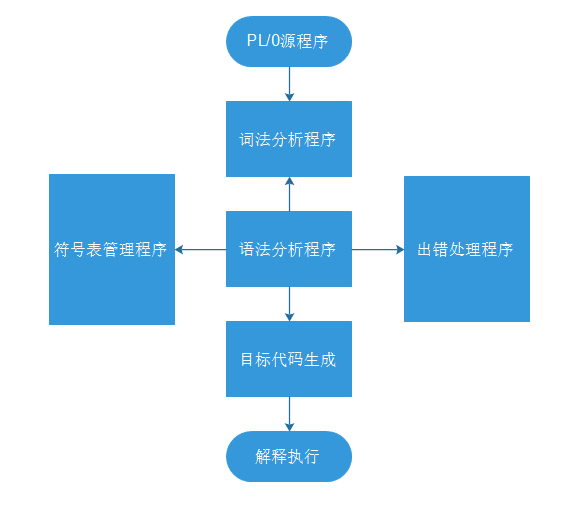
                    factor()

            condition()

    interpret()//P-code解释执行

        base()

**2)系统层次结构**



### 2．类/方法/函数功能

**1)函数定义解释**

|  |  |
| --- | --- |
| 函数 | 主要功能 |
| main | 主程序 |
| error | 错误处理 |
| getsym | 词法分析 |
| getch | 取字符 |
| gen | 生成P-code指令 |
| test | 测试当前单词符号是否合法 |
| block | 分程序分析处理 |
| enter | 登记符号表 |
| position | 查找标识符在符号表中的位置 |
| constdeclaration | 常量定义处理 |
| vardeclaration | 变量定义处理 |
| listcode | 列出p-code指令清单 |
| statement | 语句分析处理 |
| expression | 表达式分析处理 |
| term | 项分析处理 |
| factor | 因子分析处理 |
| condition | 条件分析处理 |
| interpret | P-code解释执行 |
| base | 通过静态链求出数据区的基地址 |

**2)关键算法**

////////////////////词法分析////////////////////////////////////////////（暂）

getsym(); 二分折半查找保留字

i = 0;

j = norw - 1;

do

{

k = (i + j) / 2;

if (strcmp(id, word[k])==0) break;

else if (strcmp(id,word[k])<0)

{

j = k - 1;

}

else if (strcmp(id,word[k])>0)

{

i = k + 1;

}

} while (i <= j);

if (i>j)

{

strcpy\_s(sym,al+1,ident);

}

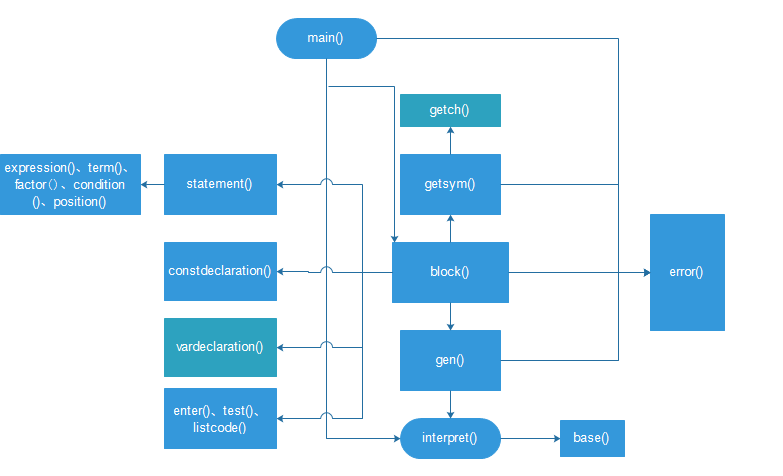
else

{

strcpy\_s(sym, al+1, wsym[k]);

}

### 3．调用依赖关系



### 4．符号表管理方案

**1)符号表table定义**

#define txmax       100

enum object {

constant, variable, proc

};

struct {

char name[al+1];

enum object kind;

int val;

int level;

int addr;

}table[txmax+1]

1. **符号表管理**

|  |  |
| --- | --- |
| 方法 | 作用 |
| enter() | 登记符号表 |
| position() | 查找标识符在符号表中的位置 |

### 5．存储分配方案

**1)存储组织**

运行栈以数组的形式存储、运行。

通过栈顶指针的移动，栈顶数据的加载和保存来运行程序。

**2)运行栈**

#define stacksize 500

int s[stacksize];

int p,b,t;

// p为程序指令指针，指向下一条要运行的代码

//b为基址指针，指向每个过程被调用时数据区中分配给它的局部变量数据段基址

//t记录这个计算机的当前栈顶位置

instructioni; // i变量中存放当前在运行的指令

### 6. 解释执行程序\*

**1)解释执行根据产生的p-code代码解释执行程序，用case语句确定指令并执行**

1. **关键算法：通过静态链求出数据区基地址的函数base()**

int base(int b, int l) {

int b1;

b1=b;

while (l>0)  {

b1=s[b1];

 l=l-1; }

return b1;}

1. **解释执行程序**

void interpret()

{

t=0;//程序开始运行时 栈顶寄存器置0

b=1;//数据段基址为1

p=0;//从0号代码开始执行程序

for(j=0;j<500;j++)

s[j]=0;

do

{

i=table[p];

p++;

switch (i.f)

{

case Lit: //取语句中的常量放到栈顶

t++;

s[t]=i.a;

break;

case Opr://运算操作

switch((int)(i.a))

{

case 0://返回上一个调用的位置

t=b-1;

p=(int)s[t+4];

b=(int)s[t+2];

break;

case 1://栈顶单元的值取反

s[t]=-s[t];

break;

case 2://加法（栈顶加次栈顶，保存至栈顶）

t--;

s[t]=s[t]+s[t+1];

break;

case 3:// 减法（次栈顶减栈顶，保存至栈顶）

t--;

s[t]=s[t]-s[t+1];

break;

case 4://乘法（次栈顶乘以栈顶，保存至栈顶）

t--;

s[t]=s[t]\*s[t+1];

break;

case 5://除法（次栈顶除以栈顶，保存至栈顶）

t--;

s[t]=s[t]/s[t+1];

break;

case 6://逻辑运算：相等比较，将结果（0或1）保存至栈顶

t--;

if(s[t]==s[t+1])

s[t]=1;

else

s[t]=0;

break;

case 7://不等

t--;

if(s[t]!=s[t+1])

s[t]=1;

else

s[t]=0;

break;

case 8://大于

t--;

if(s[t]>s[t+1])

s[t]=1;

else

s[t]=0;

break;

case 9://大于等于

t--;

if(s[t]>=s[t+1])

s[t]=1;

else

s[t]=0;

break;

case 10://小于

t--;

if(s[t]<s[t+1])

s[t]=1;

else

s[t]=0;

break;

case 11://小于等于

t--;

if(s[t]<=s[t+1])

s[t]=1;

else

s[t]=0;

break;

case 12://整数除法

t--;

s[t]=(int)((int)s[t]/(int)s[t+1]);

break;

default:

break;

}

break;

case Lod://找到变量的地址，并存入栈顶

t++;

s[t]=s[base(i.l)+(int)i.a];

break;

case Sto://将数据栈顶内容保存入变量

s[base(i.l)+(int)i.a]=s[t];

t--;

break;

case Cal://过程函数调用

s[t+1]=base(i.l);//动态链

s[t+2]=b;//静态链

s[t+4]=p;//下条语句地址

b=t+1;

p=(int)i.a;

break;

case Int://分配空间

t=t+(int)i.a;

break;

case Jmp://无条件跳转

p=(int)i.a;

break;

case Jpc://有条件跳转

if(((int)s[t])==0)

{

p=(int)i.a;

t--;

}

break;

case Red://读入数据

scanf("%d",&tempI);

s[base(i.l)+(int)i.a]=tempI;

break;

case Wrt:

printf("%d",(int)s[t]);

t++;

break;

default:

break;

}

}while(p!=0);//主程序的子程序返回指令，也就是整个程序运行的结束

}

### 7. 目标代码生成方案\*

**1)目标代码生成gen()**

enumfct {

lit, opr, lod, sto, cal, Int, jmp, jpc,red,wrt

};

void gen(enumfct x, int y, int z) {

if(cx >cxmax)     { printf("program too long\n"); exit(1); }

code[cx].f = x; code[cx].l = y; code[cx].a = z;

cx = cx + 1;

}

### 8. 出错处理

char ERROR\_MESSAGE[60][100]={

"错误0：",

"错误1：需要’=’而不是’:=’！",

"错误2：需要数字！",

"错误3：需要等号/赋值号！",

"错误4：需要标识符！",

"错误5：需要分号’;’！",

"错误6：",

"错误7：",

"错误8：",

"错误9：需要句号’.’",

"错误10：需要分号’;’！",

"错误11：未定义的标识符！",

"错误12：标识符不是变量！",

"错误13：需要赋值号’:=’！",

"错误14：call后跟的不是标识符！",

"错误15：call后跟的不是过程名！",

"错误16：if后没有then！",

"错误17：需要end",

"错误18：需要do",

"错误19：",

"错误20：缺少二元逻辑运算符！",

"错误21：错误使用标识符，标识符是过程名！",

"错误22：需要右括号’)’！",

"错误23：",

"错误24：",

"错误25：",

"错误26：",

"错误27：",

"错误28：",

"错误29：",

"错误30：数字太大！",

"错误31：",

"错误32：嵌套层数大于最大允许的套层数！",

"错误33：",

"错误34：",

"错误35：",

"错误36：",

"错误37：",

"错误38：",

"错误39：",

"错误40：需要左括号’(’",

};//暂

## 三．操作说明

### 1．运行环境

Windows /VS Express 2013

### 2．操作步骤

运行编译程序。

输入需要编译的源文件名称，回车。

若程序无误，将会产生正确的目标代码，并解释执行且在控制台输出运行结果；

否则，在控制台打印错误信息。

## 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】