（附录）

《编译技术》课程设计文 档

学号：\_\_\_\_\_\_76066002\_\_\_\_\_\_

姓名：\_\_\_\_\_\_周美廷\_\_\_\_\_\_\_\_

2019年 1 月 4 日

## 一．需求说明

### 1．文法说明

**扩充PL/0文法**

<程序> ::= <分程序>.

<分程序> ::= [<常量说明部分>][<变量说明部分>]{[<过程说明部分>]| [<函数说明部分>]}<复合语句>

<常量说明部分> ::= const<常量定义>{,<常量定义>};

<常量定义> ::= <标识符>＝ <常量>

<常量> ::= [+| -] (<无符号整数>| <无符号实数>)|<字符>

<字符> ::= '<字母>' | '<数字>'

<字符串> ::= "{十进制编码为32,33,35-126的ASCII字符}"

<无符号整数> ::= <数字>{<数字>}

<无符号实数> ::= <无符号整数>.<无符号整数>

<标识符> ::= <字母>{<字母>|<数字>}

<变量说明部分> ::= var <变量说明> ; {<变量说明>;}

<变量说明> ::= <标识符>{, <标识符>} : <类型>

<类型> ::= <基本类型>|array'['<无符号整数>']' of <基本类型> //此处<无符号整数>表示数组元素个数，其值需大于0

<基本类型> ::= integer | real | char

<过程说明部分> ::= <过程首部><分程序>;

<函数说明部分> ::= <函数首部><分程序>;

<过程首部> ::= procedure<标识符>'('[<形式参数表>]')'; //<形式参数表>是否出现分别对应了有无参数的情况

<函数首部> ::= function <标识符>'('[<形式参数表>]')': <基本类型>;

<形式参数表> ::= <形式参数>{; <形式参数>}

<形式参数> ::= [var] <标识符>{, <标识符>}: <基本类型>

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

<赋值语句> ::= <标识符> := <表达式>| <函数标识符> := <表达式>|<标识符>'['<表达式>']':= <表达式>

<函数标识符> ::= <标识符>

<表达式> ::= [+|-]<项>{<加法运算符><项>} // [+|-]只作用于第一个<项>

<项> ::= <因子>{<乘法运算符><因子>}

<因子> ::= <标识符>|<标识符>'['<表达式>']'|<无符号整数>| <无符号实数>|'('<表达式>')' | <函数调用语句>

<函数调用语句> ::= <标识符>'('[<实在参数表>]')'

<实在参数表> ::= <实在参数> {, <实在参数>}

<实在参数> ::= <表达式>

<加法运算符> ::= +|-

<乘法运算符> ::= \*|/

<条件> ::= <表达式><关系运算符><表达式>

<关系运算符> ::= <|<=|>|>= |=|<>

<条件语句> ::= if<条件>then<语句> | if<条件>then<语句>else<语句>

<repeat循环语句> ::= repeat <语句>until<条件>

<for循环语句> ::= for <标识符> := <表达式> (to|downto) <表达式> do <语句>

<过程调用语句> ::= <标识符>'('[<实在参数表>]')'

<复合语句> ::= begin<语句>{; <语句>}end

<读语句> ::= read'('<标识符>{,<标识符>}')'

<写语句> ::= write '(' <字符串>,<表达式>')'|write '(' <字符串>')'|write'('<表达式>')'

<字母> ::= a|b|c|d...|x|y|z |A|B...|Z

<数字> ::= 0|1|2|3...|8|9

附加说明：

（1）char类型的变量或常量，用字符的ASCII码对应的整数参加运算

（2）标识符区分大小写字母

（3）赋值语句中<函数标识符> := <表达式> 作为函数的返回值，其类型应与返回类型一致，此语句后面的语句可继续执行

（4）写语句中，字符串原样输出，表达式只有单个字符类型的变量或常量时按字符输出，其他表达式均按整型或实型输出

（5）数组的下标从0开始

（6）for循环语句步长为1

（7）带var的形式参数表示变量形参，参数传递方式是传地址，对应的实参只能是变量

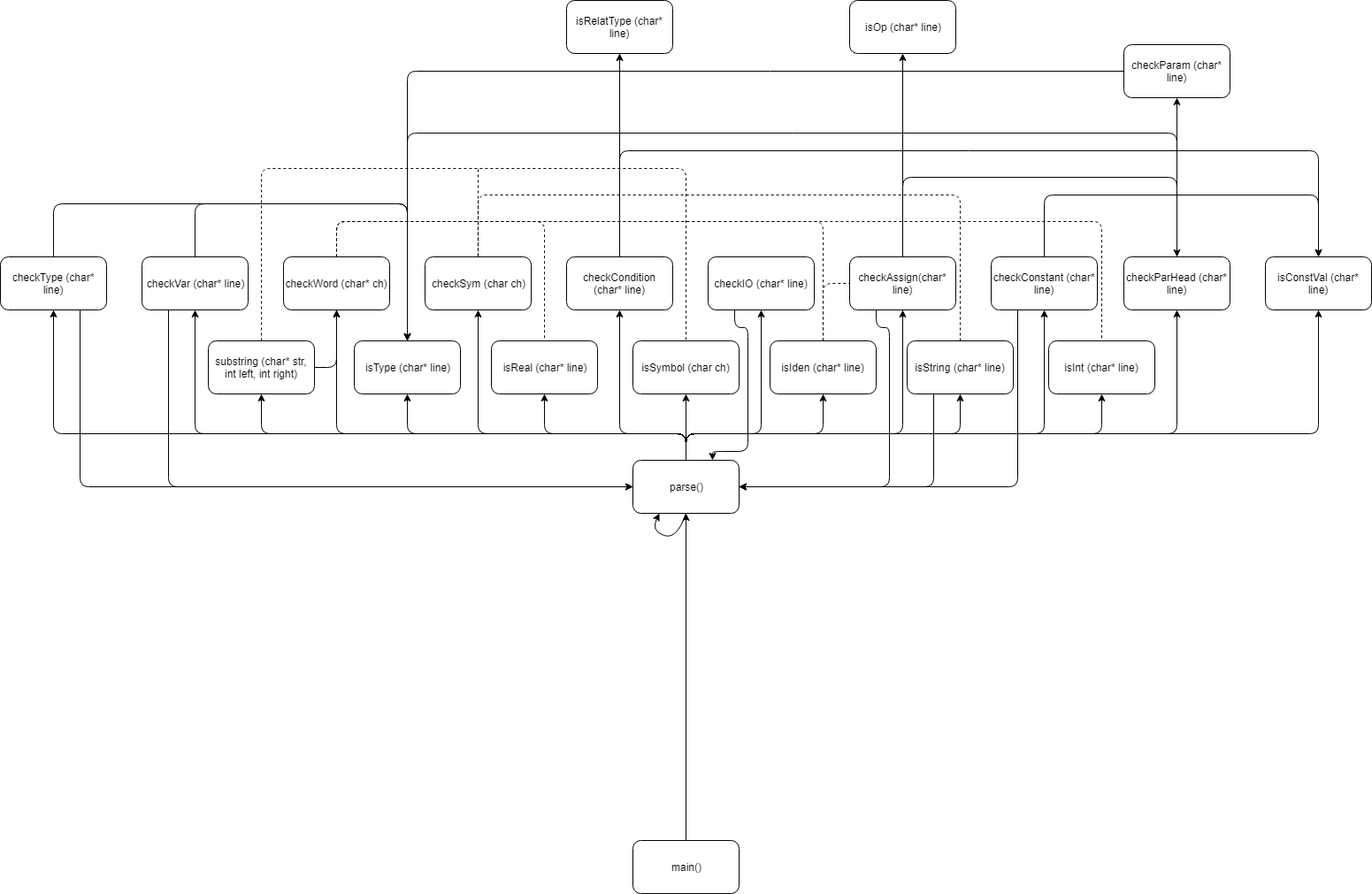
### 2．目标代码说明

目标代码：PCODE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 助记符 | f | l | a | 功能 |
| LDA |  | l | a | 把变量地址装入栈顶 |
| LOD |  | l | a | 装入值 |
| LDI |  | l | a | 间接装入 |
| DIS |  | l | a | 更新DISPLAY |
| INT |  |  | a | 将栈顶元素加上a |
| JMP |  |  | a | 无条件转移到a |
| JPC |  |  | a | 如栈顶内容为假，转移到a |
| F1U |  |  | a | 增量步长型for循环体的入口测试 |
| F2U |  |  | a | 增量步长型for循环体的再入测试 |
| F1D |  |  | a | 减量步长型for循环体的入口测试 |
| F2D |  |  | a | 减量步长型for循环体的再入测试 |
| MKS |  |  | a | 标记栈 |
| CAL |  |  | a | 调用用户过程或函数 |
| IDX |  |  | a | 取下标变量地址（元素长度=1） |
| IXX |  |  | a | 取下标变量地址 |
| LDB |  |  | a | 装入块 |
| CPB |  |  | a | 复制块 |
| LDC |  |  | a | 装入字面常量 |
| LDR |  |  | a | 装入实数 |
| FLT |  |  | a | 转换浮点数 |
| RED |  |  | a | 读（a表示1类型，1：整型，2：实型，3：字符型） |
| WRS |  |  | a | 写字符 |
| WRW |  |  | a | 写—隐含域宽 |
| HLT |  |  |  | 停止 |
| EXP |  |  |  | 退出过程 |
| EXF |  |  |  | 退出函数 |
| LDT |  |  |  | 取栈顶单元内容为地址的单元内容 |
| MUS |  |  |  | 求负 |
| STO |  |  |  | 将栈顶内容存入以栈顶次高远为地址的单元 |
| EQR |  |  |  | 实型等于比较 |
| NER |  |  |  | 实型不等比较 |
| LSR |  |  |  | 实型小于比较 |
| LER |  |  |  | 实型小于等于比较 |
| GTR |  |  |  | 实型大于比较 |
| GER |  |  |  | 实型大于等于比较 |
| EQL |  |  |  | 整型相等比较 |
| NEQ |  |  |  | 整型不等比较 |
| LSS |  |  |  | 整型小于比较 |
| LER |  |  |  | 整型小于等于比较 |
| GRT |  |  |  | 整型大于比较 |
| GEQ |  |  |  | 整型大于等于比较 |
| ADD |  |  |  | 整型加 |
| SUB |  |  |  | 整型减 |
| ADR |  |  |  | 实型加 |
| SUR |  |  |  | 实型减 |
| MUL |  |  |  | 整型乘 |
| DIV |  |  |  | 整型除 |
| MUR |  |  |  | 实型乘 |
| DIR |  |  |  | 实型除 |

## 二．详细设计

### 程序结构



### 类/方法/函数功能

这些是我目前语法分析程序的函数与功能：

* errorOut(int errnum) & fatal(int n)

错误处理函数

* isSymbol(char ch)

有效符号表的检查函数。若ch是有效符号，那就返回true，之后才用其他函数检查那是哪个符号。

* isOp(char\* line)

在被查到为有效符号的情况下（先调用isSymbol查看是否是有效符号），再查看那个符号是否是运算符（加减乘除），若是返回true，不然返回false。因为isSymbol返回的格式是字符串，所以查看也用字符串line。

* isRelatType(char\* line)

在被查到为有效符号的情况下（先调用isSymbol查看是否是有效符号），再查看那个符号是否是关系运算符（<>，=，<，>，<=，>=），若是返回true，不然返回false。

* isConstVal(char\* line)

用来查看line是否是是有效的常量/变量值。这有效的变量/常量值是整数，实数，字符串，自定义变量，字符。

* isArray(char\* line)

查看当前定义是否是正确的数组定义

* isType(char\* line)

用来查看line是否是变量类型。比如说在变量定义，var ch: char，那char是变量类型。这有效的变量类型有integer，char，string，boolean，array of 类型，real，或者自定义类型变量（type）。

* isInt(char\* line)

查看line是否是整数，若line是无符号整数那就返回对，若line为空或其他情况就返回false。

* isReal(char\* line)

查看line是否是实数。若line全为数字及其中有小数点，就返回true，不然返回false。

* isString(char\* line)

查看line是否是字符串。字符串中有可能会出现标识符，保留字等等，所以需要检查。比如说在write时，或者在常量定义时。

* isIden(char\* line)

查看line是否是标识符。若line是保留字，自然不是标识符，若不是以字母开头也不是有效的标识符。

* checkSym(char ch)

用来查看ch属于哪个符号（从符号表查看）。被查到的话，会按照那个符号返回对应的符号值。比如，ch是‘=’，则会返回“EQL”。

* checkWord(char\* ch)

从保留字表中查看ch是否是有效保留字。被查到的话会返回对应保留字的值（字符串形式）。比如说const会返回CONSTSYM。

* checkType(char\* line)

查看变量值类型号（int = 1，real = 2，char|str = 4，array = 5）

* checkKind(char\* line)

查看变量类型号（variable = 1，constant = 0，procedure = 3，function = 4）

* checkParHead(char\* line)

查看line是否是有效形式参数的大体文法（有括号，有分号，若是函数声明会有冒号和基本类型）。

* checkParam(char\* line)

查看line是否是有效的形式参数值。有效的形式参数可以有标识符，逗号，冒号，和类型。可以是一个标识符一个类型，也可以是多个标识符一个类型。比如x:integer，x,y:char。

* checkRead(char\* line)

检查读语句

* checkWrite(char\* line2)

检查写语句

* checkCondition()

检查条件语句

* resultType()

检查计算结果的类型（int或real）

* factor()

表达式中的因子

* term()

表达式中的项

* expression()

检查表达式

* checkAssign(char\* line2)

检查赋值语句或过程/函数调用语句

* checkConstant(char\* line)

检查常量定义

* checkVar(char\* line)

检查变量定义

* storeCode(int f, int l, int a)

存新指令

* storeTable(char kind[])

存新符号

* storeBlock()

存新分程序（块）

* storeArray(int l, int h)

存数组

* storeReal(float x)

存实数

* storeError(int n)

存错误信息

* findId(char \*iden)

在符号表中查找标识符

* substring(char\* str, int left, int right)

从文件中的一行拿一个单词，把那个单词当成返回值。

* parse()

主要解释函数，具体功能会在第6标题中描述。

写代码生成程序时，需再定义一个类似与checkWord的函数来判断PCODE的指令。返回值自然是对应指令的字符串。

### 调用依赖关系

以下是独立函数的依赖关系。因为是递归程序，全部都会调用解释程序本身。反的，parse也会调用全部函数。

* checkRead(char\* line)：findId，checkRead，parse
* checkWrite(char\* line2)：parse，expression，checkType
* checkCondition()：expression，isRelatType，parse
* factor()：parse，findId，expression
* term()：factor，parse，resultType，
* expression()：term，parse，resultType
* checkAssign(char\* line2)：findId，expression
* checkConstant(char\* line)：parse，isConstVal，checkConstant
* checkVar(char\* line)：parse，isType，isArray，checkVar

### 符号表管理方案

struct table {

char name[30];

char kind[10];

char type[10];

bool normal;

int link, ref, vali;

float valf;

char val[15];

int level;

int adr;

}valtab[200];

void storeTable(char kind[])

{

if (ti == tmax) fatal(1);

else

{

int j, l;

strcpy(valtab[0].name, id);

j = btab[display[level]].last;

l = j;

while (strcmp(valtab[j].name, id))

j = valtab[j].link;

if (j != 0) storeError(2);

else {

ti++;

strcpy(valtab[ti].name, id);

strcpy(valtab[ti].kind, kind);

valtab[ti].link = ti - 1;

if ((ti == 29) || (valtab[ti - 1].level == level && (!strcmp(valtab[ti - 1].kind, "procedure") || !strcmp(valtab[ti - 1].kind, "function")))) {

if (strcmp(kind, "procedure") && strcmp(kind, "function"))

valtab[ti].link = 0;

}

valtab[ti].normal = true;

valtab[ti].level = level;

valtab[ti].ref = 0;

if (!strcmp(kind, "variable")) {

valtab[ti].adr = address;

address++;

}

else if (!strcmp(kind, "constant")) {

if (!strcmp(valtab[ti].type, "STRING")) {

valtab[ti].adr = valc2;

append(valtab[ti].val, valc2);

}

else if (!strcmp(valtab[ti].type, "REAL")) {

valtab[ti].valf = valf;

storeReal(valf);

valtab[ti].adr = c1;

}

else {

valtab[ti].vali = vali;

valtab[ti].adr = vali;

}

}

else if (!strcmp(kind, "procedure") || !strcmp(kind, "function"))

strcpy(valtab[ti].val, "(NULL)");

btab[display[level]].last = ti;

resetData();

}

}

return;

}

### 存储分配方案

struct errtab {

int line, col, n;

}errtab[stacksize];

struct stacktab {

int i; // in case cn is an integer

float r; // in case cn is a real

bool b; // in case cn is a boolean

char c; // in case cn is a char

}s[stacksize];

struct blocktab {

int last, lastpar, psize, vsize;

} btab[20];

struct arraytab {

char inxtyp[10], eltyp[10];

int elref, low, high, elsize, size;

}atab[30];

struct table {

char name[30];

char kind[10];

char type[10];

bool normal;

int link, ref, vali;

float valf;

char val[15];

int level;

int adr;

}valtab[200];

struct opcode {

int f; // {fct}

int l : 3;

int a : 16;

} code[cmax];

### 解释执行程序\*

void interpret()

{

int fld[4], pc, k;

enum ps { run, fin, divchk, inxchk, stkchk, lngchk };

puts("Start pl/0");

pc = run;

t = btab[2].vsize - 1; b = 0; p = valtab[s[4].i].adr;

s[1].i = s[2].i = 0; s[3].i = -1; s[4].i = btab[1].last;

display[0] = display[1] = 0;

lncount = chrcount = 0;

ocnt = 0;

do {

struct opcode ir = code[p]; p = p + 1;

ocnt = ocnt + 1;

switch (ir.f) {

default:

break; case lda: t = t + 1; // load address

{

if (t > stacksize) { pc = stkchk; }

else s[t].i = display[ir.l] + ir.a;

}

break; case lod: t = t + 1; // load value

{

if (t > stacksize) { pc = stkchk; }

else s[t] = s[display[ir.l] + ir.a];

}

break; case ldi: t = t + 1; // load indirect

{

if (t > stacksize) { pc = stkchk; }

else s[t] = s[s[display[ir.l] + ir.a].i];

}

break; case dis: h1 = ir.a; h2 = ir.l; h3 = b;// update display

{

while (h1 != h2) {

display[h1] = h3; h1 = h1 - 1; h3 = s[h3 + 2].i;

}

}

break; case Int: s[t].i = s[t].i + ir.a; // offset

break; case jmp: p = ir.a; // jump

break; case jpc: // conditional jump

{

if (!s[t].b) p = ir.a;

t = t - 1;

}

break; case f1u: h1 = s[t - 1].i; // for1up

{

if (h1 <= s[t].i) s[s[t - 2].i].i = h1;

else { t = t - 3; p = ir.a; }

}

break; case f2u: h2 = s[t - 2].i; h1 = s[h2].i + 1; // for2up

{

if (h1 <= s[t].i) { s[h2].i = h1; p = ir.a; }

else t = t - 3;

}

break; case f1d: h1 = s[t - 1].i; // for1down

{

if (h1 >= s[t].i) s[s[t - 2].i].i = h1;

else { p = ir.a; t = t - 3; }

}

break; case f2d: h2 = s[t - 2].i; h1 = s[h2].i - 1; // for2down

{

if (h1 >= s[t].i) { s[h2].i = h1; p = ir.a; }

else t = t - 3;

}

break; case mks: h1 = btab[valtab[ir.a].ref].vsize; // mark stack

{

if (t + h1 > stacksize) { pc = stkchk; }

else { t = t + 5; s[t - 1].i = h1 - 1; s[t].i = ir.a; }

}

break; case cal: h1 = t - ir.a; h2 = s[h1 + 4].i; h3 = valtab[h2].level; display[h3 + 1] = h1;// call

{

h4 = s[h1 + 3].i + h1; s[h1 + 1].i = p; s[h1 + 2].i = display[h3]; s[h1 + 3].i = b;

for (h3 = t + 1; h3 < h4; h3++) s[h3].i = 0;

b = h1; t = h4; p = valtab[h2].adr;

//if (stackdump) dump;

}

break; case idx: h1 = ir.a; h2 = atab[h1].low; h3 = s[t].i; // index1

{

if ((h3 < h2) || (h3 > atab[h1].high)) { pc = inxchk; }

else { t = t - 1; s[t].i = s[t].i + (h3 - h2); }

}

break; case ixx: h1 = ir.a; h2 = atab[h1].low; h3 = s[t].i; // index

{

if ((h3 < h2) || (h3 > atab[h1].high)) { pc = inxchk; }

else { t = t - 1; s[t].i = s[t].i + (h3 - h2)\*atab[h1].elsize; }

}

break; case ldb: h1 = s[t].i; t = t - 1; h2 = ir.a + t; // load block

{

if (h2 > stacksize) { pc = stkchk; }

else {

while (t < h2) { t = t + 1; s[t] = s[h1]; h1 = h1 + 1; }

}

}

break; case cpb: h1 = s[t - 1].i; h2 = s[t].i; h3 = h1 + ir.a; // copy block

{

while (h1 < h3) { s[h1] = s[h2]; h1 = h1 + 1; h2 = h2 + 1; }

t = t - 2;

}

break; case ldc: t = t + 1; // literal

{

if (t > stacksize) { pc = stkchk; }

else s[t].i = ir.a;

}

break; case ldr: t = t + 1; // load real

{

if (t > stacksize) { pc = stkchk; }

else s[t].r = rconst[ir.a];

}

break; case flt: h1 = t - ir.a; s[h1].r = s[h1].i; // float

break; case red: // read

{

switch (ir.a) {

default:

break; case 1: scanf("%d", &s[s[t].i].i);

break; case 2: scanf("%f", &s[s[t].i].r);

break; case 4: scanf("%c", &s[s[t].i].c);

}

t = t - 1;

}

break; case wrs: h1 = s[t].i; h2 = ir.a; t = t - 1; chrcount = chrcount + h1; // write string

{

if (chrcount > lineleng) { pc = lngchk; }

while (h1 != 0) { printf("%c", stab[h2]); h1 = h1 - 1; h2 = h2 + 1; }

chrcount = 0;

}

break; case wrw: chrcount = chrcount + fld[ir.a]; // write1

{

if (chrcount > lineleng) { pc = lngchk; }

else switch (ir.a) {

default:

break; case 1: printf("%d", s[t].i);

break; case 2: printf("%.2f", s[t].r);

break; case 3: if (s[t].b) printf("true"); else printf("false");

break; case 4: printf("%c", s[t].i);

}

t = t - 1;

chrcount = 0;

}

break; case hlt: pc = fin;

break; case Exp: t = b - 1; p = s[b + 1].i; b = s[b + 3].i; // exit procedure

break; case exf: t = b; p = s[b + 1].i; b = s[b + 3].i; // exit function

break; case ldt: s[t] = s[s[t].i];

break; case mus: s[t].i = -s[t].i;

break; case sto: s[s[t - 1].i] = s[t]; t = t - 2; // store

break; case eqr: t = t - 1; s[t].b = (s[t].r == s[t + 1].r);

break; case ner: t = t - 1; s[t].b = (s[t].r != s[t + 1].r);

break; case lsr: t = t - 1; s[t].b = (s[t].r < s[t + 1].r);

break; case ler: t = t - 1; s[t].b = (s[t].r <= s[t + 1].r);

break; case gtr: t = t - 1; s[t].b = (s[t].r > s[t + 1].r);

break; case ger: t = t - 1; s[t].b = (s[t].r >= s[t + 1].r);

break; case eql: t = t - 1; s[t].b = (s[t].i == s[t + 1].i);

break; case neq: t = t - 1; s[t].b = (s[t].i != s[t + 1].i);

break; case lss: t = t - 1; s[t].b = (s[t].i < s[t + 1].i);

break; case leq: t = t - 1; s[t].b = (s[t].i <= s[t + 1].i);

break; case grt: t = t - 1; s[t].b = (s[t].i > s[t + 1].i);

break; case geq: t = t - 1; s[t].b = (s[t].i >= s[t + 1].i);

break; case add: t = t - 1; s[t].i = s[t].i + s[t + 1].i;

break; case sub: t = t - 1; s[t].i = s[t].i - s[t + 1].i;

break; case adr: t = t - 1; s[t].r = s[t].r + s[t + 1].r;

break; case sur: t = t - 1; s[t].r = s[t].r - s[t + 1].r;

break; case mul: t = t - 1; s[t].i = s[t].i\*s[t + 1].i;

break; case Div: t = t - 1;

{

if (s[t + 1].i == 0) { pc = divchk; }

else s[t].i = s[t].i / s[t + 1].i;

}

break; case mur: t = t - 1; s[t].r = s[t].r\*s[t + 1].r;

break; case dir: t = t - 1; s[t].r = s[t].r / s[t + 1].r;

}

} while (pc == run); //< (sizeof(code)/sizeof(struct opcode)) );

puts("\n end pl/0");

if (pc != fin) {

printf("halt at %d because of ", p);

switch (pc) {

default:

break; case divchk: printf("division by 0\n");

break; case inxchk: printf("invalid index\n");

break; case stkchk: printf("storage overflow\n");

break; case lngchk: printf("line too long\n");

}

}

for (k = 0; k < errcnt; k++) {

printf("At line: %d column: %d ", errtab[k].line, errtab[k].col);

errorOut(errtab[k].n);

}

}

### 出错处理

可以看出在解释执行程序有errorOut的循环（输出全部错误）。这些错误是用storeError存储，存错误号，错误列和行数。

void storeError(int n)

{

errtab[errcnt].line = lc;

errtab[errcnt].col = r;

errtab[errcnt].n = n;

errcnt++;

}

void errorOut(int errnum)

{

switch (errnum) {

case 0: printf("Identifier not defined"); break;

case 1: printf("Identifier redefinition"); break;

case 2: printf("Identifier expected"); break;

case 3: printf("Program should started with 'program'"); break;

case 4: printf("')' expected"); break;

case 5: printf("':' expected when defining a variable type"); break;

case 6: printf("Invalid symbol, some symbols will be skipped"); break;

case 7: printf("Parameters should start with an identifier or 'var'"); break;

case 8: printf("'of' expected"); break;

case 9: printf("'(' expected"); break;

case 10: printf("Type should start with identifier, array, or record"); break;

case 11: printf("'[' expected"); break;

case 12: printf("']' expected"); break;

case 13: printf("'..' expected"); break;

case 14: printf("';' expected"); break;

case 15: printf("Function return type should be integer, real, boolean, or char"); break;

case 16: printf("'=' expected, ':=' should only be used in assignment statements not declarations"); break;

case 17: printf("Expression following 'if', 'while', or 'until' should be boolean type"); break;

case 18: printf("The loop variable after for can only be integer, char or boolean"); break;

case 19: printf("The initial value or result value of expression in 'for' statement should be the same as loop variable type"); break;

case 20: printf("Arguement 'output' or 'input' expected in the program head"); break;

case 21: printf("Number out of range (Maximum of 14 integers)"); break;

case 22: printf("The end of program is '.', check the corresponding 'begin' and 'end'"); break;

case 23: printf("Expression after 'case' should be integer, char or boolean type"); break;

case 24: printf("Invalid character"); break;

case 25: printf("Constant value or identifier expected after '=' in constant declaration"); break;

case 26: printf("The subscript expression type must be the same as the subscript type in the array description"); break;

case 27: printf("In the array description, the lower bound cannot exceed the upper bound, and their types must be the same"); break;

case 28: printf("No such array"); break;

case 29: printf("Type identifier expected"); break;

case 30: printf("Type undefined"); break;

case 31: printf("No such record"); break;

case 32: printf("The operand of 'not', 'and', or 'or' should be of boolean type"); break;

case 33: printf("The type of this arithmetic expression is invalid"); break;

case 34: printf("The operand of 'div' and 'mod' should be of integer type"); break;

case 35: printf("The object types to be compared must be the same, unless one is an integer and the other is a real"); break;

case 36: printf("The actual parameter and the corresponding formal parameter type should be the same, unless the formal parameter is a real value parameter and the actual parameter is an integer."); break;

case 37: printf("Variable expected"); break;

case 38: printf("String should at least have a character"); break;

case 39: printf("The number of actual parameters is different with the number of formal parameters"); break;

case 40: printf("No number found after '.'"); break;

case 41: printf("Wrong arguments in 'write' or 'read'"); break;

case 42: printf("The expression should be a real type"); break;

case 43: printf("The domain width of the write statement should be an integer"); break;

case 44: printf("Type or procedure identifier cannot appear in an expression"); break;

case 45: printf("Should be a variable or procedure/function identifier"); break;

case 46: printf("The assigned variable in the assignment statement should be the same as the expression type, unless the former is a real type and the latter is an integer type."); break;

case 47: printf("The label in the case statement must be the same type as the case clause expression"); break;

case 48: printf("The argument expression type of the standard function is incorrect"); break;

case 49: printf("Memory overflow"); break;

case 50: printf("Constant expected"); break;

case 51: printf("':=' expected"); break;

case 52: printf("'then' expected"); break;

case 53: printf("'until' expected"); break;

case 54: printf("'do' expected"); break;

case 55: printf("'to' or 'downto' expected"); break;

case 56: printf("'begin' expected"); break;

case 57: printf("'end' expected"); break;

case 58: printf("Factor should start with identifier, constant, not, or '('"); break;

}

printf("\n");

return;

}

void fatal(int n)

{

switch (n) {

case 1: printf("Compiler table for identifier is too small\n"); break;

case 2: printf("Compiler table for procedures is too small\n"); break;

case 3: printf("Compiler table for reals is too small\n"); break;

case 4: printf("Compiler table for arrays is too small\n"); break;

case 5: printf("Compiler table for levels is too small\n"); break;

case 6: printf("Compiler table for code is too small\n"); break;

case 7: printf("Compiler table for strings is too small\n"); break;

}

return (-1);

}

## 三．操作说明

### 1．运行环境

运行环境使用Visual Studio 2017 Community Edition。

### 2．操作步骤

* 把测试程序存到某个位置（比如跟源代码一样的一个文件夹里）
* 打开Visual Studio工程项目文件按F5
* Cmd出现后会提示用户输入测试文件名
* 输入后程序会自动编译（解释-执行）

## 四．测试报告

### 1．测试程序及测试结果

1. Test1.txt（正确）

测试点：

* + - 变量说明
    - 赋值语句
    - 写语句
    - 过程说明
    - 函数说明
    - 形式参数表
    - 复合语句
    - 函数调用
    - 过程调用

结果：正确

1. Test2.txt（正确）

测试点：

* + - 常量说明
    - 变量说明
    - 赋值语句
    - 写语句
    - 读语句
    - 表达式（加法，减法，除法，乘法）

结果：正确

1. Test3.txt（正确）

测试点：

* + - 变量说明
    - 写语句
    - For-to|downto-do循环语句

结果：正确

1. Test4.txt（正确）

测试点：

* + - 变量说明
    - 写语句
    - 读语句
    - If-then条件语句
    - If-then-else条件语句

结果：正确

1. Test5.txt（正确）

测试点：

* + - 变量说明
    - 赋值语句
    - 写语句
    - Repeat-until循环语句

结果：正确

1. Test6\_wrong.txt（错误）

测试点：

* + - At line: 1 column: 11 '=' expected, ':=' should only be used in assignment statements not declarations
    - At line: 1 column: 22 Identifier expected
    - At line: 2 column: 21 ':' expected when defining a variable type
    - At line: 4 column: 11 Identifier not defined
    - At line: 5 column: 9 Identifier not defined
    - At line: 5 column: 10 ')' expected
    - At line: 6 column: 12 Identifier not defined
    - At line: 7 column: 23 '(' expected

结果：正确

1. Test7\_wrong.txt（错误）

测试点：

* At line: 1 column: 27 Number out of range (Maximum of 14 integers)
* At line: 1 column: 27 Constant value or identifier expected after '=' in constant declaration
* At line: 1 column: 35 Constant value or identifier expected after '=' in constant declaration
* At line: 2 column: 52 ';' expected
* At line: 9 column: 34 Identifier not defined
  + - At line: 10 column: 3 The end of program is '.', check the corresponding 'begin' and 'end'

结果：正确

1. Test8\_wrong.txt（错误）

测试点：

* + - At line: 3 column: 10 String should at least have a character
    - At line: 4 column: 8 ':=' expected
    - At line: 7 column: 23 'do' expected
    - At line: 10 column: 13 'to' or 'downto' expected
    - At line: 12 column: 4 'begin' expected

结果：正确

1. Test9\_wrong.txt（错误）

测试点：

* halt at 7 because of division by 0
* At line: 5 column: 3 'begin' expected

结果：正确

1. Test10\_wrong.txt（错误）

测试点：

* + - At line: 1 column: 18 Identifier redefinition
    - At line: 3 column: 18 Parameters should start with an identifier or 'var'

结果：正确

### 2．测试结果分析

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 测试1 | 测试2 | 测试3 | 测试4 | 测试5 | 测试6 | 测试7 | 测试8 | 测试9 | 测试10 |
| 常量说明 |  | √ |  |  |  | √ | √ |  |  |  |
| 变量说明 | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| 过程说明 | √ |  |  |  |  |  |  |  |  | √ |
| 函数说明 | √ |  |  |  |  |  |  |  |  |  |
| 表达式计算 |  | √ |  |  |  | √ | √ |  |  |  |
| 赋值语句 | √ | √ |  |  | √ | √ | √ |  |  | √ |
| 写语句 | √ | √ | √ | √ | √ | √ | √ | √ | √ | √ |
| 读语句 |  | √ |  | √ |  | √ | √ |  |  |  |
| 形式参数表 | √ |  |  |  |  |  |  |  |  |  |
| 复合语句 | √ |  |  |  |  | √ | √ | √ | √ | √ |
| 函数调用 | √ |  |  |  |  |  |  |  |  |  |
| 过程调用 | √ |  |  |  |  |  |  |  |  |  |
| For-to|downto-do循环语句 |  |  | √ |  |  |  |  | √ |  |  |
| If-then条件语句 |  |  |  | √ |  |  |  |  | √ |  |
| If-then-else条件语句 |  |  |  | √ |  |  |  |  | √ |  |
| Repeat-until循环语句 |  |  |  |  | √ |  |  |  |  |  |
| Division by 0 |  |  |  |  |  | √ |  |  | √ |  |

## 五．总结感想

对于编译器的知识与了解是有明显的进步。之前只知道编译器是做什么但具体怎么做到并不了解。通过这次编译器课设不但可以了解到而且还亲手写一个编译器是巨大的收获。开发当中有各种各样的bug和困难。有时候反而不是为了大逻辑或操作而困惑而是为了鸡毛蒜皮的事，比如空格的事没有考虑到，换行的事这种。但最后能达到这种结果已经很满足了。