PL0技术文档初始文稿-第4组

组员：周泉、曹艳、辛媛、张冒莉

本队完成了本次上机实践的要求(1)—(9)，以及增加了部分功能：赋值运算的扩充、自增自减表达式运算（++、--前置和后置运算），具体如下：

1. 新增了注释功能：

**思路**：通过修改词法分析部分的getsym函数，对注释部分内容进行循环处理，处理过程中要与/、/=等运算区分开，注意细节处理。

修改人：张冒莉

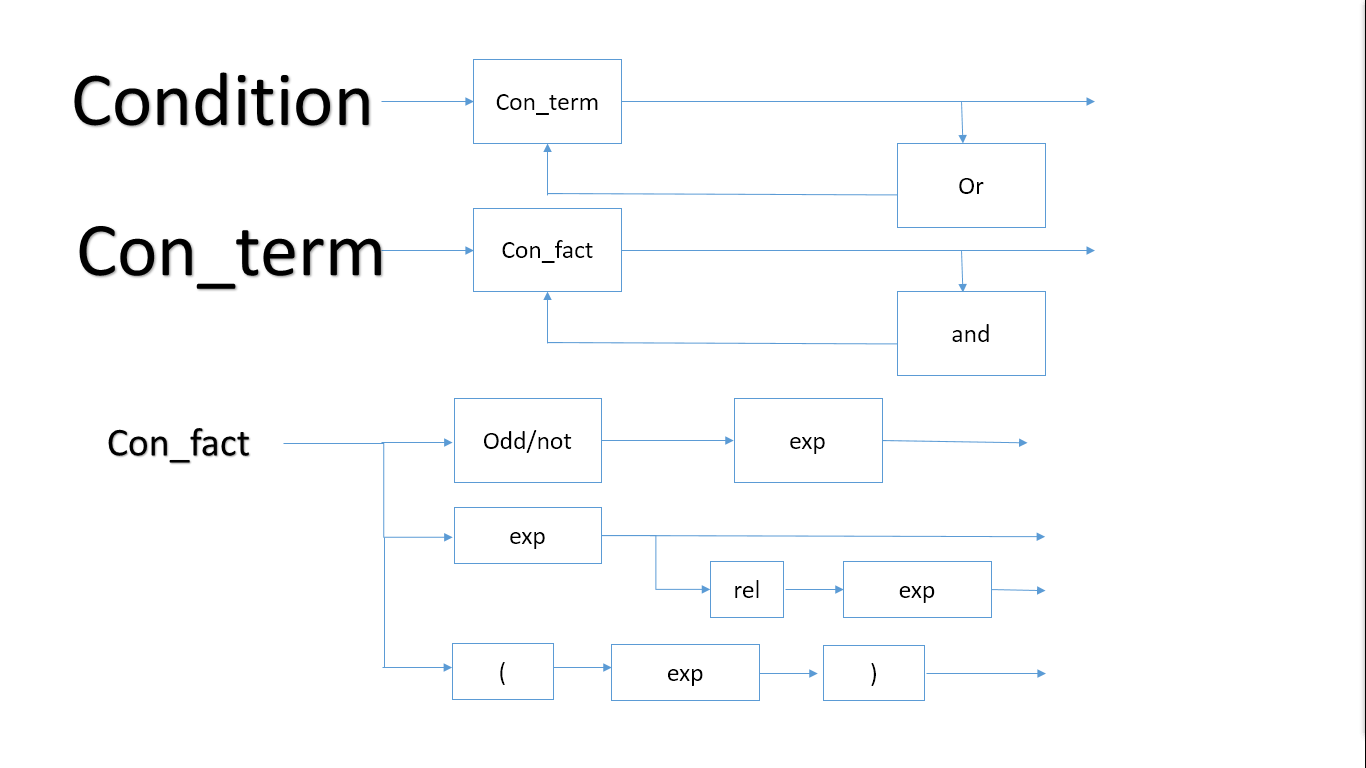
1. 扩展了PL/0中“条件”，包括：

(1)增加逻辑运算符and、or和not；

(2)把PL/0语言中的“条件”概念一般化为C语言那样(表达式值非零即为“真”);

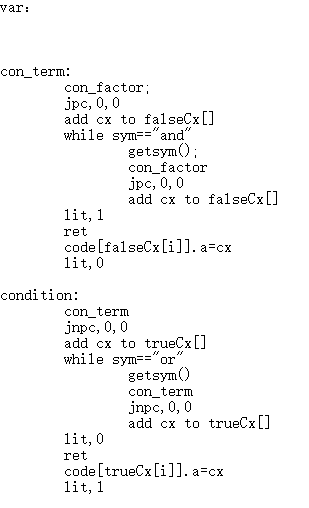
(3)“条件”的短路计算

**思路**：

流程图：

通过修改condition部分内容，来一般化pl0中的条件概念，增加了con\_term() 和con\_factor()；

短路计算：



修改人：辛媛（前两个）、周泉

1. 增加了数组类型：

允许有数组变量声明、对数组元素赋值、在表达式中引用数组元素等；可以有多维数组，数组的维度范围可以自行设定，目前设为10。

文法：dimDeclaration → **[**const**]** dimDeclaration

dimDeclaration →ε

const → **ident** | **number**

**思路**：关键问题是数组在table中的存放结构以及必要信息和数组下标的计算。首先需要修改table中的信息需要记录数组的name（ID），层次，相对数组首地址，类型（数组），维数，指向每维维度链表的指针（修改了table的数据结构）。

当前得到一个标识符的话，并不知道其是数组类型还是变量类型，所以需要看下一个sym，如果是左方括号则表明这是数组，若不是则为变量，这里需要修改vardeclaration以及新设一个函数来处理数组类型dimdeclaration。

计算得到的数组内相对偏移量放置在栈顶，因此需要利用此时的栈顶值来得到数组元素存储单元的地址，新增两个指令STA和LAD来实现：STA：将栈顶数据存放到“次栈顶的偏移量加上数组元素首地址”所指示的存储单元；

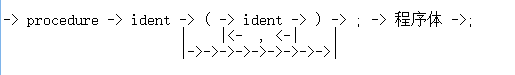
LAD：将“次栈顶的偏移量加上数组元素首地址”所指示的存储单元内的数据复制到栈顶。

修改人：曹艳

1. 过程参数实现，采用传值方式，并实现简单的语义检查，实参和形参个数、整数和数组类型检查。

**思路**：过程定义: procedure p(a,b)

调用： call p(i,j)



增加错误新信息（err\_msg[34]）： The number of segments doesn't match.；

增加全局变量： zx[level]： 每个层次的过程的参数个数；

增加的函数： enterPar: 参数填入table；

modifyTable： 修改填入table的参数的偏移地址；

关于如何传参：call p(i,j)把i和j的值压栈，再cal p，参数个数记录在表里,相对于p的偏移量分别是-2和-1。主要修改statement.call和block.procedure。

修改人：周泉

1. else子句实现。

**思路**：if <condition> then

statement(如果有else则不能接分号，无需要加上分号）

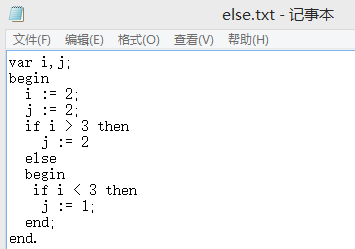
else

…

注：else前必须有if语句，但if语句之后不一定要加else;

根据pascal语法，当if后面不加else语句时源pl0代码以‘；’结尾，加else则不加‘；’。

eg:



增加符号类型(symtype)：SYM\_ELSE

增加保留字(word[],wsym[]):“else”，“SYM\_ELSE”，常数NRW++

主要在statement部分的SYM\_IF分支修改，注意地址回填，要将SYM\_ELSE加入到if后继集合中去。

修改人：张冒莉、辛媛（都写了else）

1. for语句的实现。

**思路**：for var := E1 to/downto E2 do

Statement;

注：(1)to 递增，downto 递减，步长相等，可以自行声明定义值的大小；E1、E2是表达式。

(2)注意地址回填，以及与break结合使用的地址处理。

增加声明：#define STEP 2 //或者其他值，步长

增加符号类型(symtype)：SYM\_FOR,SYM\_DOWNTO,SYM\_TO

增加错误信息(err\_msg[])：

/\* 30 \*/ "'to' or 'downto' expected."；

/\* 4 \*/ 错误信息内容扩充。

增加保留字(word[],wsym[])：“for”，“downto”，“to”

SYM\_FOR，SYM\_DOWNTO，SYM\_TO

主要修改statement部分，添加else if(sym == SYM\_FOR){ ... }。

修改人：张冒莉

1. exit语句（退出当前执行过程）。

**思路**：增加symtype: SYM\_EXIT

word[]: "exit"

wsym[]: SYM\_EXIT

比较简单，即执行一个OPR\_RET指令。

修改人：周泉

1. break语**句**（跳出包含它的最内层循环）。

**思路**：主要处理两个问题:多地址回填和嵌套地址回填。

（1）如果一个for里面有多个break，如下：

For ......

...

If ......

Break；

If......

Break；

...

可以用一个链表的形式把这两个break的地址链起来，然后for的结尾处处理这个回填链即可。

（2）嵌套的形式如下：

For ......

...

Break；

For ......

...

Break;

...

利用statement的嵌套调用来写break；把break当做一条语句，在读到for的时候需要调用statementA，在statementA里面会调用statementB来处理内层for，在statementB里接着调用statementC，假如内层for只有一个break；语句，则statmentC分析break，先会生成jmp指令，这条指令的代码保持在一个全局变量里，为了日后回填，然后break分析完毕，回到内层for的处理，这时候回到的是statementB，那么到了for的末尾，这个时候知道了回填什么地址（jmp应跳到这里），所以用刚刚的全局变量里面存放的地址回填回去。此时有一个问题，就是全局变量跟着全局变，可能会覆盖了statementA里面的break的地址，所以需要在调用statement之前保存全局变量之前的值，然后再调用statement，statement出来之后，处理完回填后再将全局变量的旧值恢复，然后退出此层的statement，这样就不会覆盖之前的break地址。

break的作用是跳出当前循环，如果没有在循环内调用，则会报错，实现原理就是设置一个信号sign，如果前面分析的时候有循环，那我就将这个sign置位，每读到break的时候就查看则个sign是否有效就可判断是否出错，当然，这个sign是和上述的全局变量一起的结构体中的一个域。

最终修改人: 曹艳

参与修改人：周泉（调试）、张冒莉（全体讨论另外一种实现方式是通过数组实现）

1. PL/0输入/输出语句。可参考C的相关函数，实现任意个数参数的输入或输出功能。

**思路**：（1）write(a): 输出a并且换行；

writeln(a,b)：输出a和b并换行，以空格分隔。

注：参数要大于0，不支持write(a<b)。

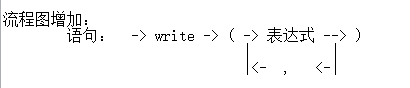
增加操作指令(oprcode):

OPR\_PRT(print) 输出栈顶的数字并去掉弹出栈顶的数， OPR\_WTL(writeln) 新的一行；

增加保留字（word[],wsym[]）："write" 相应常数NRW++；

增加符号类型（symtype）：SYM\_WRITE；

增加错误新信息（err\_msg[33]）： '(' expected。

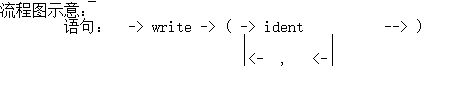


主要修改statement部分；

（2）读入：read(a) read(a,b)

OPR\_RED: 从标准输入设备输入一个整数，存放在栈顶。

增加word[],wsym[] "read" SYM\_READ ~NRW++



修改人：周泉

1. 补充功能：（1）增加了repeat … until循环语句。

**思路**：repeat

statement;

until condition;

在symtype里加SYM\_REPEAT,SYM\_UNTIL

word[]加“repeat”,"until",

wsym[]加SYM\_REPEAT,SYM\_UNTIL

增加错误类型：ERROR(30) "missing until"。

（2）扩充了赋值运算，增加+=，-=，/=，\*=，%=，--，++

**思路**：以‘+’，‘++’，‘+=’为例，在词法分析的getsym()中读到‘+’后再往后读一个字符，判断出运算符号保留字的类型，是SYM\_ADD,SYM\_ADDADD还是SYM\_ADDEQU；然后语法分析中修改statement部分，后置‘++’和‘--’都跟在标识符之后，前置‘++’和‘--’在标识符之前，单独为分支。其余‘-’‘\*’‘/’类似；

%=运算在.h文件的总类型里加上SYM\_MODEQU, 同时对应 ssym加SYM\_MODEQU，在csym[]里对应加"%" ，在oprcode里加取模运算， OPR\_MOD，并且在解释执行函数里加

case OPR\_MOD:

top--;

stack[top]%=stack[top+1];

break;

修改人：辛媛

1. 分离了解释和编译器与加强的语法错误恢复机制。

**思路**：将原pl0代码中的编译和翻译过程分离开来，编译过程生成中间代码文件hbin.txt，解释过程以这个文件为输入，调用interpret函数进行解释；即将代码分成compile和interpret两部分。

修改人：曹艳、张冒莉、辛媛、周泉

1. 原代码修正：老师所给的程序中的int有错误，主要错误在于block\_dx的不作为。

经全体讨论得出两种修改方案：

1. 在block处理procedure之前让block\_dx = dx;即用block\_dx存储当前的dx，在处理statement之前分配空间是

gen(INT, 0, block\_dx);

1. 定义全局数组变量：dx[MAXLEVEL] ，不需要使用block\_dx,在所有dx部分改为dx[level].
2. 实现了代码优化.

编译完后的代码删除掉了只声明未引用的多余变量和过程。

新增功能：

-代码优化

对于声明不用的变量、数组以及过程，将会被删除；

对于不用的变量和数组，其在执行时将不分配存储空间；

对于不用的过程，其代码段将被从最终代码里删除；

-检查重定义：

如果常量、变量、数组以及过程名一样，则报错（但不停止分析），以这个名字出现的第一处为定义点；

内部变量和外部变量重名是可以的。

-增加常量公式：

数组声明可用，常量声明时赋值可用，如8+1\*9,n+1-8（n为常量）

还有优化在进行中。

分工：以上修改人为该部分参与撰写者，但有些内容前期讨论和中期纠错商讨是全体四人一起进行的，比如break、优化等，还有些部分多人尝试，一起debug，最终得出正确的方法。