**PL0语言基础设计报告**

By 邵维科，李展鹏，于敬禹

**1.实验内容解说：**首先是（1）词法分析正规式：

*digit*  -> [0-9]

*number* ->

*letter* -> [A-Za-z]

*identifier* -> *letter*

*rel\_logicop*  -> < | > | <> | <= | >= | := | && | ‘||’| ! | [ | ]

*op*  -> + | - | \* | / | , | . | ; | ( | ) | = | &

*word*  -> begin | const | call | do | end | if |else | odd | procedure |while | for |var | then

上面是词法分析的正规式，覆盖了可能出现的合法的单词和符。

（2）下面描述语言的文法：

program ::=

body**.**

body ::=

“const” { identifier “=”number“,”}<identifier “=” number “;”>

| “var”{identifier[ {“[”number

“]”}],}<identifier[{“[”number“]”}]“;”>

| “procedure” “;” body “;”

| stmt\_mul

stmt\_mul ::=

{stmt “;”}

stmt ::=

identifier[{“[”number“]”}] “:=” expression

| “call” identifier

| “begin” stmt\_mul “end”

| “if” <condition> then stmt “else” stmt

| “while” <condition> “do” stmt

|“for”“(“ ((expression“;”)|“;”)[expression] “;”[expression] “;”

condition ::=

“odd” expression

| expression (“=”| “<>”| “<”| “>”|

“<=”| “>=”|“&&”|“||”)expression

| “！”expression

expression ::=

term {(“+”| “-“) term}

term ::=

factor {(“\*”| “/”) term}

factor ::=

identifier[{“[”number“]”}]

| number

| “-”expression

| “(”expression“)”

上面使用BNF来描述的PL0的文法。

**2.各小题设计：**

**第二题：**对逻辑算符的操作首先在getsym函数中增加识别这三个算符的代码，他们被设计成和关系运算符是在相同的层次和用法。对于和C语言风格相似的逻辑值设置，在使用条件跳转时，设置只有在表达式值为0的时候才进行跳转。短路计算需要为每个逻辑运算符两边的逻辑表达式设置跳转语句，若是满足条件就直接跳到整个逻辑表达式的后方。

**第三题：**

**第四题：**

1. 增加了当前作用域内冲突声明的检测。在当前作用域内已声明过的非过程名字再次声明时会报错（31），在当前作用域及其先祖作用域内已声明过的过程名再次声明时会报错（31）。扩大过程名字的检测范围是为了更好地支持递归调用，禁止声明父过程的名字，防止声明错误在编译阶段无法检测。为此，修改了函数position的实现，增加参数tx\_beg，记录当前查找的作用域的开始位置。该参数只有在检测冲突声明时才置非0，过去已存在的调用均为0。为了传递正确的tx\_beg，又添加了全局变量tx\_b，它在每次声明过程前修改，并且进入过程体后由block\_dx记录它的值，因为一个过程体内可能声明多个子过程，每个过程都要更改tx\_b的值。所以要提前记录当前域的tx\_b值，以使与过程声明间杂的变量声明可以拿到正确的tx\_b值。
2. 利用加入数组时增加的符号表项ptr，记录过程名字附属的参数的信息。由于现在参数只有整型，只需一个域记录个数即可。之后添加地址和过程参数时还需要记录各参数的类型。
3. 增加了声明带参函数的代码。参数在活动记录中正向储存（与C不同），并且在过程头声明时直接加入符号表，其地址直接绑定到调用函数存放实参的地方。因此，越早声明的参数相对偏移（为负）越小，其值要在扫描完全部参数后才能确定。
4. 在声明和调用含参过程的代码中提供了多种错误检测，包括超过参数个数上限，参数过多、过少，参数不能为数组、缺少分隔符等。
5. 删除了call符号，改为直接使用名字调用。
6. 对set.c中set集合的操作增加了查重功能，对于重复的符号在集合合并、扩大时只添加一个，减少查找特定符号的时间。
7. #debug 修正了在语句和程序块结束时，出现编译错误有可能会重复报错的bug。这个bug源于这两个函数末尾的错误检查test(fsys,ksys,*)，如果语句或块后的符号（比如；）漏写了（这是常有的事），在这里会通不过检测而报错，从而先于后面真正的错误而报错。这里test的参数选择照搬自原来的pl0程序，虽然整个错误检测系统已经重构了一遍，去除了一些bug，但是仍然有很多问题。在正确的位置检测出正确个数、内容正确的错误，并且尽可能保证后面的分析能继续下去，应当是本实验最难的地方。这里修改成test(ksys,ksys,*)，弱化了test函数的作用，也许会在某些错误下引发新的问题。

**第五题：**

（1）在原有的if语句的基础上增加else语句，这需要设计文法使else不会出现悬空现象，但试了书上和几种其他方法后，文法始终无法契合LL文法的要求，在这种情况下，只有退而求其次，将else设计成语句的必要成分，不影响程序将要实现的功能，只是是在书写时有时会有些繁琐，出现不必要的else，其后其实是空语句。

（2）exit语句要求遇到执行这个语句时直接退出程序的执行，为了实现这个功能，保存每次遇到exit时code数组的下标，生成一个跳转代码JMP，跳转到程序体的最后一个code值之后，因为所有exit都要跳到最后，不可能只用一个变量来标记exit语句，为了保存下标不致下标覆盖，将下标设计成整型数组，每个exit对应其中一项。目前的exit语句只能支持直接返回整型常量，以后还会继续增加他的功能。

（3）return语句和exit相似，但是只是退出当前的过程，同样保存code数组下标，生成JMP指令，在block函数的最后对每个JMP指令的跳转地址赋值后，再将下标数组初始化，保证每个return只作用于最靠近它的过程

（4）for语句的设计比较保守，括号中第一项只能是一个赋值语句，第二项是条件语句，最后跟上一个赋值语句。为了方便说明，以a,b,c,d分别表示括号内三个语句和for的循环部分。设计中需要在b的后面设置两个跳转指令，第一个是JPC，在条件为假是退出循环，跳到d的后面，第二个是JMP，无条件跳到d的第一条指令，然后在d的后面设置JMP跳到c的第一条指令,在c的后面设置JMP无条件跳到条件语句b的第一条指令。这样可以实现循环。

**3.程序设计特色：**

（1）对于错误处理的修改和理解，

test函数的基本左右是和各个函数的形参（即Follow和关键字集）相配合，判断当前sym是不是属于他第一个形参的符号，在不是的情况下报错并且跳过错误代码，查找属于它第二个形参的符号。所以要想使test函数发挥正确的作用，既能准确找到错误，又不会跳过过多代码，就需要第一个形参精确输入需要查找的符号的集合，第二个形参输入一个把所有后续可能的关键符号都包括在内的集合。而调用test函数的非终结符可能在不同的产生式里，每一个产生式中该非终结符的后续符号都可能是不同的，所以要通过传参的形式，由产生式左部的非终结符产生右部非终结符的后续符号集合。针对test函数两个形参一精确、一宽泛的要求，非终结符之间也传递两个形参。要求精确的Follow集合严格按照产生式中非终结符后面的符号来产生；而要求宽泛的关键字集合是三个集合的并：非终结符的Follow集、非终结符所在的产生式的左部非终结符的关键字集和一个因各个产生式具体的错误处理方法而异的集合。修改文法符号以后，需要考虑产生式的错误处理方法、Follow集合和关键字集合需不需要修改，用test函数做的错误检查需不需要修改。

为了更好实现错误恢复，基本所有非终结符号对应的函数的形参都从一个调整为两个。同时还增加了允许多参数合并关键字和FOLLOW集的unite\_mul函数和多参数的setinsert\_mul函数。

**实验分工说明**：实验中第一小题很容易完成，其余四题有我们三人分作，第2题由于敬禹同学完成，第3,4题由邵维科同学完成，第5题由李展鹏同学完成。在实验中的错误检测和恢复方法由邵维科同学做了修改。