PL0 编译器源代码（pl0.cpp）

```
/*编译和运行环境:
*1Visual C++6.0,VisualC++.NET and Visual C++.NET 2003
*WinNT, Win 200, WinXP and    Win2003
*2 gcc version 3.3.2    20031022(Red Hat Linux 3.3.2-1)
*Redhat Fedora core 1
*Intel 32 platform
*使用方法:
*运行后输入 PL/0  源程序文件名
*回答是否输出虚拟机代码
*回答是否输出名字表
*fa.tmp  输出虚拟机代码
*fa1.tmp   输出源文件及其各行对应的首地址
*fa2.tmp   输出结果
*fas.tmp   输出名字表
*/
#include <stdio.h>
#include "pl0.h"
#include "string.h"
/*解释执行时使用的栈*/
#define stacksize 500
int main()
{
    bool nxtlev[symnum];
    printf("Input pl/0 file ?");
    scanf("%s",fname);                                  /*输入文件名*/
    fin=fopen(fname,"r");
    if(fin)
    {
        printf("List object code ?(Y/N)");              /*是否输出虚拟机代码*/
        scanf("%s",fname);
        listswitch=(fname[0]=='y'||fname[0]=='Y');
        printf("List symbol table ? (Y/N)");            /*是否输出名字表*/
        scanf("%s",fname);
        tableswitch=(fname[0]=='y'||fname[0]=='Y');
        fa1=fopen("fa1.tmp","w");
        fprintf(fa1,"Iput pl/0 file ?");
        fprintf(fa1,"%s\n", fname);
        init();                                         /*初始化*/
        err=0;
        cc=cx=ll=0;
        ch=' ';
```

```
        if(-1!=getsym())
        {
            fa=fopen("fa.tmp","w");
            fas=fopen("fas.tmp","w");
            addset(nxtlev,declbegsys,statbegsys,symnum);
            nxtlev[period]=true;
            if(-1==block(0,0,nxtlev))          /*调用编译程序*/
            {
                fclose(fa);
                fclose(fa1);
                fclose(fas);
                fclose(fin);
                printf("\n");
                return 0;
            }
            fclose(fa);
            fclose(fa1);
            fclose(fas);
            if(sym!=period)
            {
                error(9);
            }
            if(err==0)
            {
                fa2=fopen("fa2.tmp", "w");
                interpret();
                fclose(fa2);
            }
            else
            {
                printf("Errors in pl/0 program");
            }
        }
        fclose(fin);
    }
    else
    {
        printf("Can't open file! \n");
    }
    printf("\n");
    return 0;
}
/*
*初始化
```

```
*/
void init()
{
	int i;
	for(i=0;i<=255;i++)
	{
		ssym[i]=nul;
	}
	ssym['+']=plus;
	ssym['-']=minus;
	ssym['*']=times;
	ssym['/']=slash;
	ssym['(']=lparen;
	ssym[')']=rparen;
	ssym['=']=eql;
	ssym[',']=comma;
	ssym['.']=period;
	ssym['#']=neq;
	ssym[';']=semicolon;
	/*设置保留字名字,按照字母顺序,便于折半查找*/
	strcpy(&(word[0][0]),"begin");
	strcpy(&(word[1][0]),"call");
	strcpy(&(word[2][0]),"const");
	strcpy(&(word[3][0]),"do");
	strcpy(&(word[4][0]),"end");
	strcpy(&(word[5][0]),"if");
	strcpy(&(word[6][0]),"odd");
	strcpy(&(word[7][0]),"procedure");
	strcpy(&(word[8][0]),"read");
	strcpy(&(word[9][0]),"then");
	strcpy(&(word[10][0]),"var");
	strcpy(&(word[11][0]),"while");
	strcpy(&(word[12][0]),"write");
	/*设置保留字符号*/
	wsym[0]=beginsym;
	wsym[1]=callsym;
	wsym[2]=constsym;
	wsym[3]=dosym;
	wsym[4]=endsym;
	wsym[5]=ifsym;
	wsym[6]=oddsym;
	wsym[7]=procsym;
	wsym[8]=readsym;
	wsym[9]=thensym;
```

```
        wsym[10]=varsym;
        wsym[11]=whilesym;
        wsym[12]=writesym;

        /*设置指令名称*/
        strcpy(&(mnemonic[lit][0]),"lit");
        strcpy(&(mnemonic[opr][0]),"opr");
        strcpy(&(mnemonic[lod][0]),"lod");
        strcpy(&(mnemonic[sto][0]),"sto");
        strcpy(&(mnemonic[cal][0]),"cal");
        strcpy(&(mnemonic[inte][0]),"int");
        strcpy(&(mnemonic[jmp][0]),"jmp");
        strcpy(&(mnemonic[jpc][0]),"jpc");

        /*设置符号集*/
        for(i=0;i<symnum;i++)
        {
            declbegsys[i]=false;
            statbegsys[i]=false;
            facbegsys[i]=false;
        }

        /*设置声明开始符号集*/
        declbegsys[constsym]=true;
        declbegsys[varsym]=true;
        declbegsys[procsym]=true;
        /*设置语句开始符号集*/
        statbegsys[beginsym]=true;
        statbegsys[callsym]=true;
        statbegsys[ifsym]=true;
        statbegsys[whilesym]=true;
        /*设置因子开始符号集*/
        facbegsys[ident]=true;
        facbegsys[number]=true;
        facbegsys[lparen]=true;
}
  /*
    *用数组实现集合的集合运算
    */
int inset(int e,bool* s)
{
        return s[e];
}
int addset(bool* sr,bool* s1,bool* s2,int n)
```

```
{
	int i;
	for(i=0;i<n;i++)
	{
		sr[i]=s1[i]||s2[i];
	}
	return 0;
}
int subset(bool* sr,bool* s1,bool* s2,int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		sr[i]=s1[i]&&(!s2[i]);
	}
	return 0;
}
int mulset(bool* sr,bool* s1,bool* s2,int n)
{
	int i;
	for(i=0;i<n;i++)
	{
		sr[i]=s1[i]&&s2[i];
	}
	return 0;
}
/*
 *出错处理，打印出错位置和错误编码
 */
void error(int n)

{
	char space[81];
	memset(space,32,81); printf("-------%c\n",ch);
	space[cc-1]=0;//出错时当前符号已经读完，所以 cc-1
	printf("****%s!%d\n",space,n);
	err++;
}
/*
 *	漏掉空格，读取一个字符
 *
 *	每次读一行，存入 line 缓冲区，line 被 getsym 取空后再读一行
 *
 *	被函数 getsym 调用
```

```
 */
int getch()
{
	if(cc==ll)
	{
		if(feof(fin))
		{
			printf("program incomplete");
			return -1;
		}
		ll=0;
		cc=0;
		printf("%d ",cx );
		fprintf(fa1,"%d ",cx);
		ch=' ';
		while(ch!=10)
		{
			//fscanf(fin,"%c",&ch)
			if(EOF==fscanf(fin,"%c",&ch))
			{
				line[ll]=0;
				break;
			}
			printf("%c",ch);
			fprintf(fa1,"%c",ch);
			line[ll]=ch;
			ll++;
		}
		printf("\n");
		fprintf(fa1,"\n");
	}
	ch=line[cc];
	cc++;
	return 0;
}
 /*词法分析，获取一个符号
 */

int getsym()
{
	int i,j,k;
	while( ch==' '||ch==10||ch==9)
	{
		getchdo;
```

```
}
if(ch>='a'&&ch<='z')
{
	k=0;
	do{
		if(k<al)
		{
			a[k]=ch;
			k++;
		}
		getchdo;
	}while(ch>='a'&&ch<='z'||ch>='0'&&ch<='9');
	a[k]=0;
	strcpy(id,a);
	i=0;
	j=norw-1;
	do{
		k=(i+j)/2;
		if(strcmp(id,word[k])<=0)
		{
			j=k-1;
		}
		if(strcmp(id,word[k])>=0)
		{
			i=k+1;
		}

	}while(i<=j);
	if(i-1>j)
	{
		sym=wsym[k];
	}
	else
	{
		sym=ident;
	}
}
else
{
	if(ch>='0'&&ch<='9')
	{
		k=0;
		num=0;
		sym=number;
```

```
        do{
            num=10*num+ch-'0';
            k++;
            getchdo;
        }while(ch>='0'&&ch<='9'); /*获取数字的值*/
        k--;
        if(k>nmax)
        {
            error(30);
        }
    }
    else
    {
        if(ch==':')                    /*检测赋值符号*/
        {
            getchdo;
            if(ch=='=')
            {
                sym=becomes;
                getchdo;
            }
            else
            {
                sym=nul;                /*不能识别的符号*/
            }
        }
        else
        {
            if(ch=='<')             /*检测小于或小于等于符号*/
            {
                getchdo;
                if(ch=='=')
                {
                    sym=leq;
                    getchdo;
                }
                else
                {
                    sym=lss;
                }
            }
            else
            {
                if(ch=='>')             /*检测大于或大于等于符号*/
```

```
                        {
                            getchdo;
                            if(ch=='=')
                            {
                                sym=geq;
                                getchdo;
                            }
                            else
                            {
                                sym=gtr;
                            }
                        }
                        else
                        {
                            sym=ssym[ch];/* 当符号不满足上述条件时，全部按照单字符号
处理*/
                            //getchdo;
                            //richard
                            if(sym!=period)
                            {
                                getchdo;
                            }
                            //end richard
                        }
                    }
                }
            }
        }
        return 0;
}
/*
*生成虚拟机代码
*
*x:instruction.f;
*y:instruction.l;
*z:instruction.a;
*/
int gen(enum fct x,int y,int z)
{
        if(cx>=cxmax)
        {
            printf("Program too long"); /*程序过长*/
            return -1;
        }
```

```
    code[cx].f=x;
    code[cx].l=y;
    code[cx].a=z;
    cx++;
    return 0;
}
/*
*测试当前符号是否合法
*
*在某一部分（如一条语句，一个表达式）将要结束时时我们希望下一个符号属于某集合
*（该部分的后跟符号） test 负责这项检测，并且负责当检测不通过时的补救措施
*程序在需要检测时指定当前需要的符号集合和补救用的集合（如之前未完成部分的后跟
*符号），以及不通过时的错误号
*
*S1：我们需要的符号
*s2:如果不是我们需要的，则需要一个补救用的集合
*n:错误号
*/


int test(bool* s1,bool* s2,int n)
{
    if(! inset(sym,s1))
    {
        error(n);
        /*当检测不通过时，不停获取符号，直到它属于需要的集合或补救的集合*/
        while((! inset(sym,s1))&&(! inset(sym,s2)))
        {
            getsymdo;
        }
    }
    return 0;
}
/*
 *编译程序主体
 *
 *lev:当前分程序所在层
 *tx:名字表当前尾指针
 *fsys:当前模块后跟符号集合
 */
int block(int lev,int tx,bool* fsys)
{
    int i;
    int dx;                                 /*名字分配到的相对地址*/
```

```
    int tx0;                        /*保留初始 tx*/
    int cx0;                        /*保留初始 cx*/
    bool nxtlev[symnum];            /*在下级函数的参数中，符号集合均为值参，但由于
使用数组
                                    实现，传递进来的是指针，为防止下级函数改变
上级函数的
                                    集合，开辟新的空间传递给下级函数*/
    dx=3;
    tx0=tx;                         /*记录本层名字的初始位置*/
    table[tx].adr=cx;
    gendo(jmp,0,0);
    if(lev > levmax)
    {
        error(32);
    }
    do{
        if(sym==constsym)           /*收到常量声明符号，开始处理常量声明*/
        {
            getsymdo;
            do{
                constdeclarationdo(&tx,lev,&dx);    /*dx 的值会被 constdeclaration 改变，
使用
                                                    指针*/
                while(sym==comma)
                {
                    getsymdo;
                    constdeclarationdo(&tx,lev,&dx);
                }
                if(sym==semicolon)
                {
                   getsymdo;
                }
                else
                {
                     error(5); /*漏掉了逗号或者分号*/
                }
            }while(sym==ident);
        }
        if(sym==varsym)/*收到变量声名符号，开始处理变量声名*/
        {
            getsymdo;
            do{
                vardeclarationdo(&tx,lev,&dx);
                while(sym==comma)
```

```
            {
                getsymdo;
                vardeclarationdo(&tx,lev,&dx);
            }
            if(sym==semicolon)
            {
                getsymdo;
            }
            else
            {
                error(5);
            }
        }while(sym==ident);
    }
    while(sym==procsym)/*收到过程声名符号，开始处理过程声名*/
    {
        getsymdo;
        if(sym==ident)
        {
            enter(procedur,&tx,lev,&dx);/*记录过程名字*/
            getsymdo;
        }
        else
        {
            error(4);/*procedure 后应为标识符*/
        }
        if(sym==semicolon)
        {
            getsymdo;
        }
        else
        {
            error(5);/*漏掉了分号*/
        }
        memcpy(nxtlev,fsys,sizeof(bool)*symnum);
        nxtlev[semicolon]=true;
        if(-1==block(lev+1,tx,nxtlev))
        {
            return -1;/*递归调用*/
        }
        if(sym==semicolon)
        {
            getsymdo;
            memcpy(nxtlev,statbegsys,sizeof(bool)*symnum);
```

```
                nxtlev[ident]=true;
                nxtlev[procsym]=true;
                testdo(nxtlev,fsys,6);
            }
            else
            {
                error(5);                           /*漏掉了分号*/
            }
        }
        memcpy(nxtlev,statbegsys,sizeof(bool)*symnum);
        nxtlev[ident]=true;
        nxtlev[period]=true;
        testdo(nxtlev,declbegsys,7);
    }while(inset(sym,declbegsys));                  /*直到没有声明符号*/
    code[table[tx0].adr].a=cx;                      /*开始生成当前过程代码*/
    table[tx0].adr=cx;                              /*当前过程代码地址*/
    table[tx0].size=dx;                             /*声明部分中每增加一条声明都会给
dx 增加 1,声明部分已经结束,dx 就是当前过程数据的 size*/

    cx0=cx;
    gendo(inte,0,dx);                               /*生成分配内存代码*/
    if(tableswitch)                                 /*输出名字表*/
    {
        printf("TABLE:\n");
        if(tx0+1>tx)
        {
            printf("NULL\n");
        }
        for(i=tx0+1;i<=tx;i++)
        {
            switch(table[i].kind)
            {
                case constant:
                    printf("%d const %s",i,table[i].name);
                    printf("val=%d\n",table[i].val);
                    fprintf(fas,"%d const %s",i,table[i].name);
                    fprintf(fas,"val=%d\n",table[i].val);
                    break;
                case variable:
                    printf("%d var%s",i,table[i].name);
                    printf("lev=%d addr=%d\n",table[i].level,table[i].adr);
                    fprintf(fas,"%d var %s",i,table[i].name);
                    fprintf(fas,"lev=%d addr=%d\n",table[i].level,table[i].adr);
                    break;
```

```
                    case procedur:
                        printf("%d proc%s",i,table[i].name);
                        printf("lev=%d                                    addr=%d
size=%d\n",table[i].level,table[i].adr,table[i].size);
                        fprintf(fas,"%d proc%s",i,table[i].name);
                        fprintf(fas,"lev=%d          adr=%d          size=%d
\n",table[i].level,table[i].adr,table[i].size);
                        break;
                }
            }
            printf("\n");
        }
        /*语句后跟符号为分号或 end*/
      memcpy(nxtlev,fsys,sizeof(bool)*symnum);/*每个后跟符号集和都包含上层后跟符号集和，
以便补救*/
        nxtlev[semicolon]=true;
        nxtlev[endsym]=true;
        statementdo(nxtlev,&tx,lev);
        gendo(opr,0,0); /*每个过程出口都要使用的释放数据段命令*/
        memset(nxtlev,0,sizeof(bool)*symnum); /*分程序没有补救集合*/
        test(fsys,nxtlev,8);                    /*检测后跟符号正确性*/
        listcode(cx0);                          /*输出代码*/
        return 0;
}
/*
*在名字表中加入一项
*
*k:名字种类 const,var or procedure
*ptx:名字表尾指针的指针，为了可以改变名字表尾指针的数值
*lev:名字所在的层次，以后所有的 lev 都是这样
*pdx:为当前应分配的变量的相对地址，分配后要增加 1
*/
void enter (enum object k,int *ptx,int lev, int *pdx)
{
        (*ptx)++;
        strcpy(table[(*ptx)].name,id);          /*全局变量 id 中已存有当前名字的名字*/
        table[(*ptx)].kind=k;
        switch(k)
        {
            case constant:                          /*常量名字*/
                if (num>amax)
                {
                    error(31);
                    num=0;
```

```
                }
                table[(*ptx)].val=num;
                break;
            case variable:                          /*变量名字*/
                table[(*ptx)].level=lev;
                table[(*ptx)].adr=(*pdx);
                (*pdx)++;
                break;                              /*过程名字*/
            case procedur:
                table[(*ptx)].level=lev;
                break;
        }

}
/*
 *查找名字的位置
 *找到则返回在名字表中的位置，否则返回 0
 *
 *idt: 要查找的名字
 *tx:：当前名字表尾指针
 */
int position(char *    idt,int    tx)
{
    int i;
    strcpy(table[0].name,idt);
    i=tx;
    while(strcmp(table[i].name,idt)!=0)
    {
        i--;
    }
    return i;
}
/*
 *常量声明处理
 */
int constdeclaration(int *    ptx,int lev,int *    pdx)
{
    if(sym==ident)
    {
        getsymdo;
        if(sym==eql ||sym==becomes)
        {
            if(sym==becomes)
            {
```

```
                error(1);                       /*把=写出成了：=*/
            }
            getsymdo;
            if(sym==number)
            {
                enter(constant,ptx,lev,pdx);
                getsymdo;
            }
            else
            {
                error(2);                       /*常量说明=后应是数字*/
            }
        }
        else
        {
            error(3);                           /*常量说明标识后应是=*/
        }
    }
    else
    {
        error(4);                               /*const 后应是标识*/
    }
    return 0;
}
/*
 *
 */
int vardeclaration(int * ptx,int lev,int * pdx)
{
    if(sym==ident)
    {
        enter(variable,ptx,lev,pdx);//填写名字表
        getsymdo;
    }
    else
    {
        error(4);
    }
    return 0;
}
 /*
  *输入目标代码清单
  */
void listcode(int cx0)
```

```
{
    int i;
    if (listswitch)
    {
        for(i=cx0;i<cx;i++)
        {
            printf("%d %s %d %d\n",i,mnemonic[code[i].f],code[i].l,code[i].a);
            fprintf(fa,"%d %s %d %d\n",i,mnemonic[code[i].f],code[i].l,code[i].a);
        }
     }
}
/*
*语句处理
*/
int statement(bool* fsys,int * ptx,int lev)
{
    int i,cx1,cx2;
    bool nxtlev[symnum];
    if(sym==ident)
    {
        i=position(id,*ptx);
        if(i==0)
        {
            error(11);
        }
        else
        {
            if(table[i].kind!=variable)
            {
                error(12);
                i=0;
            }
            else
            {
                getsymdo;
                if(sym==becomes)
                {
                    getsymdo;
                }
                else
                {
                    error(13);
                }
                memcpy(nxtlev,fsys,sizeof(bool)* symnum);
```

```
                expressiondo(nxtlev,ptx,lev);
                if(i!=0)
                {
                    gendo(sto,lev-table[i].level,table[i].adr);
                }
            }
        }
    }
    else
    {
        if(sym==readsym)
        {
            getsymdo;
            if(sym!=lparen)
            {
                error(34);
            }
            else
            {
                do{
                    getsymdo;
                    if(sym==ident)
                    {
                        i=position(id, *ptx);
                    }
                    else
                    {
                        i=0;
                    }
                    if(i==0)
                    {
                        error(35);
                    }
                    else
                    {
                        gendo(opr,0,16);
                        gendo(sto,lev-table[i].level,table[i].adr);      /* 储存到变量*/
                    }
                    getsymdo;
                }while (sym==comma);  /*一条 read 语句可读多个变量 */
            }
            if(sym!=rparen)
            {
                error(33);            /* 格式错误，应是右括号*/
```

```
                while(!inset(sym,fsys))/* 出错补救，直到收到上层函数的后跟符号*/
                {
                    getsymdo;
                }
            }
            else
            {
                getsymdo;
            }
        }
        else
        {
            if(sym==writesym)           /* 准备按照 write 语句处理，与 read 类似*/
            {
                getsymdo;
                if(sym==lparen)
                {
                    do{
                        getsymdo;
                        memcpy(nxtlev,fsys,sizeof(bool)*symnum);
                        nxtlev[rparen]=true;
                        nxtlev[comma]=true;         /* write 的后跟符号为）or，*/
                        expressiondo(nxtlev,ptx,lev);/* 调用表达式处理，此处与 read 不同，read 为给变量赋值*/
                        gendo(opr,0,14);/* 生成输出指令，输出栈顶的值*/
                    }while(sym==comma);
                    if(sym!=rparen)
                    {
                        error(33);/* write()应为完整表达式*/
                    }
                    else
                    {
                        getsymdo;
                    }
                }
                gendo(opr,0,15);        /* 输出换行*/
            }
            else
            {
                if(sym==callsym)        /* 准备按照 call 语句处理*/
                {
                    getsymdo;
                    if(sym!=ident)
                    {
```

```
                    error(14);                  /*call 后应为标识符*/
                }
                else
                {
                    i=position(id,*ptx);
                    if(i==0)
                    {
                        error(11);              /*过程未找到*/
                    }
                    else
                    {
                        if(table[i].kind==procedur)
                        {
                            gendo(cal,lev-table[i].level,table[i].adr);   /*生成 call 指令
*/
                        }
                        else
                        {
                            error(15);          /*call 后标识符应为过程*/
                        }
                    }
                    getsymdo;
                }
            }
            else
            {
                if(sym==ifsym)          /*准备按照 if 语句处理*/
                {
                    getsymdo;
                    memcpy(nxtlev,fsys,sizeof(bool)*symnum);
                    nxtlev[thensym]=true;
                    nxtlev[dosym]=true;       /*后跟符号为 then 或 do*/
                    conditiondo(nxtlev,ptx,lev);    /*调用条件处理（逻辑运算）函数
*/
                    if(sym==thensym)
                    {
                        getsymdo;
                    }
                    else
                    {
                        error(16);              /*缺少 then*/
                    }
                    cx1=cx;                     /*保存当前指令地址*/
                    gendo(jpc,0,0);             /*生成条件跳转指令，跳转地址暂写 0*/
```

```
                statementdo(fsys,ptx,lev);     /*处理 then 后的语句*/
                code[cx1].a=cx;          /*经 statement 处理后，cx 为 then 后语句
执行
                    完的位置，它正是前面未定的跳转地址*/
            }
            else
            {
                if(sym==beginsym)     /*准备按照复合语句处理*/
                {
                    getsymdo;
                    memcpy(nxtlev,fsys,sizeof(bool)*symnum);
                    nxtlev[semicolon]=true;
                    nxtlev[endsym]=true;/*后跟符号为分号或 end*/
                    /*循环调用语句处理函数，直到下一个符号不是语句开始符
号或收到 end*/
                    statementdo(nxtlev,ptx,lev);
                    while(inset(sym,statbegsys)||sym==semicolon)
                    {
                        if(sym==semicolon)
                        {
                            getsymdo;
                        }
                        else
                        {
                            error(10);/*缺少分号*/
                        }
                        statementdo(nxtlev,ptx,lev);
                    }
                    if(sym==endsym)
                    {
                        getsymdo;
                    }
                    else
                    {
                        error(17); /*缺少 end 或分号*/
                    }
                }
                else
                {
                    if(sym==whilesym)/*准备按照 while 语句处理*/
                    {
                        cx1=cx;             /*保存判断条件超作的位置*/
                        getsymdo;
                        memcpy(nxtlev,fsys,sizeof(bool)*symnum);
```

```
                                nxtlev[dosym]=true;/*后跟符号为 do*/
                                conditiondo(nxtlev,ptx,lev);  /*调用条件处理*/
                                cx2=cx;          /*保存循环体的结束的下一个位置*/
                                gendo(jpc,0,0);/*生成条件跳转，但跳出循环的地址未知
*/
                                if(sym==dosym)
                                {
                                    getsymdo;
                                }
                                else
                                {
                                    error(18);        /*缺少 do*/
                                }
                                statementdo(fsys,ptx,lev); /*循环体*/
                                gendo(jmp,0,cx1);/*回头重新判断条件*/
                                code[cx2].a=cx;    /*反填跳出循环的地址，与 if 类似*/
                            }
                            else
                            {
                                memset(nxtlev,0,sizeof(bool)*symnum);/*语句结束无补
救集合*/
                                testdo(fsys,nxtlev,19);/*检测语句结束的正确性*/
                            }
                        }
                    }
                }
            }
        }
    }
    return 0;
}
/*
*表达式处理
*/
int expression(bool*fsys,int*ptx,int lev)
{
    enum symbol addop;                    /*用于保存正负号*/
    bool nxtlev[symnum];
    if(sym==plus||sym==minus)             /*开头的正负号，此时当前表达式被看作一个
正的或负的项*/
    {
        addop=sym;                        /*保存开头的正负号*/
        getsymdo;
        memcpy(nxtlev,fsys,sizeof(bool)*symnum);
```

```
            nxtlev[plus]=true;
            nxtlev[minus]=true;
            termdo(nxtlev,ptx,lev);                    /*处理项*/
            if(addop==minus)
            {
                gendo(opr,0,1);                        /*如果开头为负号生成取负指令*/
            }
        }
        else                                           /*此时表达式被看作项的加减*/
        {
            memcpy(nxtlev,fsys,sizeof(bool)*symnum);
            nxtlev[plus]=true;
            nxtlev[minus]=true;
            termdo(nxtlev,ptx,lev);                    /*处理项*/
        }
        while(sym==plus||sym==minus)
        {
            addop=sym;
            getsymdo;
            memcpy(nxtlev,fsys,sizeof(bool)*symnum);
            nxtlev[plus]=true;
            nxtlev[minus]=true;
            termdo(nxtlev,ptx,lev);                    /*处理项*/
            if(addop==plus)
            {
                gendo(opr,0,2);                        /*生成加法指令*/
            }
            else
            {
                gendo(opr,0,3);                        /*生成减法指令*/
            }
        }
        return 0;
}
/*
*项处理
*/
int term(bool*fsys,int *ptx,int lev)
{
        enum symbol mulop;                             /*用于保存乘除法符号*/
        bool nxtlev[symnum];
        memcpy(nxtlev,fsys,sizeof(bool)*symnum) ;
        nxtlev[times]=true;
        nxtlev[slash]=true;
```

```
    factordo(nxtlev,ptx,lev);          /*处理因子*/
    while(sym==times||sym==slash)
    {
        mulop=sym;
        getsymdo;
        factordo(nxtlev,ptx,lev);
        if(mulop==times)
        {
            gendo(opr,0,4);            /*生成乘法指令*/
        }
        else
        {
            gendo(opr,0,5);            /*生成除法指令*/
        }
    }
     return 0;
}
/*
*因子处理
*/
int factor(bool*fsys,int *ptx,int lev)
 {
    int i;
    bool nxtlev[symnum];
    testdo(facbegsys,fsys,24);         /*检测因子的开始符好号*/
    while(inset(sym,facbegsys))        /*循环直到不是因子开始符号*/
    {
        if(sym==ident)                 /*因子为常量或者变量*/
        {
            i=position(id,*ptx);       /*查找名字*/
            if(i==0)
            {
                error(11);             /*标识符未声明*/
            }
            else
            {
                switch(table[i].kind)
                {
                    case  constant:                                  /*名字为
常量*/
                        gendo(lit,0,table[i].val);                   /*直接把常量
的值入栈*/
                        break;
                    case  variable:                                  /*名字为变
```

```
量*/
                    gendo(lod,lev-table[i].level,table[i].adr);      /*找到变量地址并
将其值入栈*/
                    break;
                case  procedur:                                     /*名字为
过程*/
                    error(21);                                      /*不能为
过程*/
                    break;
            }
        }
        getsymdo;
    }
    else
    {
        if(sym==number)                                             /*因子为
数*/
        {
            if(num>amax)
            {
                error(31);
                num=0;
            }
            gendo(lit,0,num);
            getsymdo;
        }
        else
        {
            if(sym==lparen)                                         /*因子
为表达式*/
            {
                getsymdo;
                memcpy(nxtlev,fsys,sizeof(bool)*symnum);
                nxtlev[rparen]=true;
                expressiondo(nxtlev,ptx,lev);
                if(sym==rparen)
                {
                    getsymdo;
                }
                else
                {
                    error(22);                                      /*缺少右
括号*/
                }
```

```
                    }
                    testdo(fsys,facbegsys,23);                    /*银子后有非法符号
*/
                }
            }
        }
        return 0;
}
/*
 条件处理*/
int condition(bool* fsys,int* ptx,int lev)
{
        enum symbol relop;
        bool nxtlev[symnum];
        if(sym==oddsym)                         /*准备按照 odd 运算处理*/
        {
            getsymdo;
            expressiondo(fsys,ptx,lev);
            gendo(opr,0,6);                          /*生成 odd 指令*/
        }
        else
        {
            memcpy(nxtlev,fsys,sizeof(bool)*symnum);
            nxtlev[eql]=true;
            nxtlev[neq]=true;
            nxtlev[lss]=true;
            nxtlev[leq]=true;
            nxtlev[gtr]=true;
            nxtlev[geq]=true;
            expressiondo(nxtlev,ptx,lev);
            if(sym!=eql&&sym!=neq&&sym!=lss&&sym!=leq&&sym!=gtr&&sym!=geq)
            {
                error(20);
            }
            else
            {
                relop=sym;
                getsymdo;
                expressiondo(fsys,ptx,lev);
                switch(relop)
                {
                    case eql:
                        gendo(opr,0,8);
                        break;
```

```
                    case neq:
                        gendo(opr,0,9);
                        break;
                    case lss:
                        gendo(opr,0,10);
                        break;
                    case geq:
                        gendo(opr,0,11);
                        break;
                    case gtr:
                        gendo(opr,0,12);
                        break;
                    case leq:
                        gendo(opr,0,13);
                        break;
                }

            }
        }
        return 0;
}                                                       /*解释程序*/

void interpret()
{
        int p,b,t;                  /*指令指针，指令基址，栈顶指针*/
        struct instruction i;   /*存放当前指令*/
        int s[stacksize];          /*栈*/
        printf("start pl0\n");
        t=0;
        b=0;
        p=0;
        s[0]=s[1]=s[2]=0;
        do{
            i=code[p];              /*读当前指令*/
            p++;
            switch(i.f)
            {
                case lit:             /*将 a 的值取到栈顶*/
                    s[t]=i.a;
                    t++;
                    break;
                case opr:             /*数字、逻辑运算*/
                    switch(i.a)
                    {
```

```
case 0:
    t=b;
    p=s[t+2];
    b=s[t+1];
    break;
case 1:
    s[t-1]=-s[t-1];
    break;
case 2:
    t--;
    s[t-1]=s[t-1]+s[t];
    break;
case 3:
    t--;
    s[t-1]=s[t-1]-s[t];
    break;
case 4:
    t--;
    s[t-1]=s[t-1]*s[t];
    break;
case 5:
    t--;
    s[t-1]=s[t-1]/s[t];
    break;
case 6:
    s[t-1]=s[t-1]%2;
    break;
case 8:
    t--;
    s[t-1]=(s[t-1]==s[t]);
    break;
case 9:
    t--;
    s[t-1]=(s[t-1]!=s[t]);
    break;
case 10:
    t--;
    s[t-1]=(s[t-1]<s[t]);
    break;
case 11:
    t--;
    s[t-1]=(s[t-1]>=s[t]);
    break;
case 12:
```

```
                t--;
                s[t-1]=(s[t-1]>s[t]);
                break;
            case 13:
                t--;
                s[t-1]=(s[t-1]<=s[t]);
                break;
            case 14:
                printf("%d",s[t-1]);
                fprintf(fa2,"%d",s[t-1]);
                t--;
                break;
            case 15:
                printf("\n");
                fprintf(fa2,"\n");
                break;
            case 16:
                printf("?");
                fprintf(fa2,"?");
                scanf("%d",&(s[t]));
                fprintf(fa2,"%d\n",s[t]);
                t++;
                break;
        }
        break;
    case lod:       /*取相对当前过程的数据基地址为 a 的内存的值到栈顶*/
        s[t]=s[base(i.l,s,b)+i.a];
        t++;
        break;
    case sto:       /*栈顶的值存到相对当前过程的数据基地址为 a 的内存*/
        t--;
        s[base(i.l,s,b)+i.a]=s[t];
        break;
    case cal:               /*调用子程序*/
        s[t]=base(i.l,s,b); /*将父过程基地址入栈*/
        s[t+1]=b;           /*将本过程基地址入栈，此两项用于 base 函数*/
        s[t+2]=p;           /*将当前指令指针入栈*/
        b=t;                /*改变基地址指针值为新过程的基地址*/
        p=i.a;              /*跳转*/
        break;
    case inte:              /*分配内存*/
        t+=i.a;
        break;
    case jmp:               /*直接跳转*/
```

```
                p=i.a;
                break;
            case jpc:                   /*条件跳转*/
                t--;
                if(s[t]==0)
                {
                    p=i.a;
                }
                break;
        }
    }while (p!=0);
}
/*通过过程基址求上 1 层过程的基址*/
int base(int l,int * s,int b)
{
    int b1;
    b1=b;
     while(l>0)
     {
         b1=s[b1];
         l--;
     }
     return b1;
}
```

PL0 编译器源代码（pl0.h）

/*PL/0 编译系统 C 版本头文件 pl0.h*/

```
# define norw 13                    /*关键字个数*/
# define txmax 100                  /*名字表容量*/
# define nmax  14                    /*number 的最大位数*/
# define al 10                      /*符号的最大长度*/
# define amax 2047                  /*地址上界*/
# define levmax 3                   /*最大允许过程嵌套声明层数[0，lexmax]*/
# define cxmax 200                  /*最多的虚拟机代码数*/
/*符号*/
enum symbol{
        nul,    ident,    number,    plus,    minus,
        times,  slash,    oddsym,    eql,     neq,
lss,    leq,    gtr,      geq,       lparen,
rparen, comma,  semicolon,period,    becomes,
beginsym, endsym, ifsym,  thensym,   whilesym,
writesym, readsym,  dosym,  callsym,  constsym,
varsym,  procsym,
};
#define symnum 32
/*-------------*/
enum object{
     constant,
     variable,
     procedur,
};
/*--------------*/
enum fct{
lit, opr,  lod,  sto,  cal,  inte,  jmp,  jpc,
};
#define fctnum 8
/*--------------*/
struct instruction
{
   enum fct f;
   int l;
   int a;
};

FILE * fas;
FILE * fa;
```

```
FILE * fa1;
FILE * fa2;

bool tableswitch;
bool listswitch;
char ch;
enum symbol sym;
char id[al+1];
int    num;
int cc,ll;
int cx;
char line[81];
char a[al+1];
struct instruction code[cxmax];
char word[norw][al];
enum symbol wsym[norw];
enum symbol ssym[256];
char mnemonic[fctnum][5];
bool declbegsys[symnum];
bool statbegsys[symnum];
bool facbegsys[symnum];
/*------------------------------*/



struct tablestruct
{
    char name[al];                          /*名字*/
    enum object kind;                       /*类型：const，var，array or procedure*/
    int val;                                /*数值，仅 const 使用*/
    int level;                              /*所处层，仅 const 不使用*/
    int adr;                                /*地址，仅 const 不使用*/
    int size;                               /*需要分配的数据区空间，仅 procedure 使
用*/
};
struct tablestruct table[txmax];            /*名字表*/
FILE * fin;
FILE* fout;
char fname[al];
int err;                                    /*错误计数器*/
/*当函数中会发生 fatal error 时，返回－1 告知调用它的函数，最终退出程序*/
#define getsymdo                            if(-1==getsym())return -1
#define getchdo                             if(-1==getch())return -1
```

```
#define testdo(a,b,c)                   if(-1==test(a,b,c))return -1
#define gendo(a,b,c)                    if(-1==gen(a,b,c))return -1
#define expressiondo(a,b,c)             if(-1==expression(a,b,c))return -1
#define factordo(a,b,c)                 if(-1==factor(a,b,c))return -1
#define termdo(a,b,c)                   if(-1==term(a,b,c))return -1
#define conditiondo(a,b,c)              if(-1==condition(a,b,c))return -1
#define statementdo(a,b,c)              if(-1==statement(a,b,c))return -1
#define constdeclarationdo(a,b,c)       if(-1==constdeclaration(a,b,c))return -1
#define vardeclarationdo(a,b,c)         if(-1==vardeclaration(a,b,c))return -1
void error(int n);
int getsym();
int getch();
void init();
int gen(enum fct x,int y,int z);
int test(bool*s1,bool*s2,int n);
int inset(int e,bool*s);
int addset(bool*sr,bool*s1,bool*s2,int n);
int subset(bool*sr,bool*s1,bool*s2,int n);
int mulset(bool*sr,bool*s1,bool*s2,int n);
int block(int lev,int tx,bool* fsys);
void interpret();
int factor(bool* fsys,int* ptx,int lev);
int term(bool*fsys,int*ptx,int lev);
int condition(bool*fsys,int*ptx,int lev);
int expression(bool*fsys,int*ptx,int lev);
int statement(bool*fsys,int*ptx,int lev);
void listcode(int cx0);
int vardeclaration(int* ptx,int lev, int* pdx);
int constdeclaration(int* ptx,int lev, int* pdx);
int position(char* idt,int tx);
void enter(enum object k,int* ptx,int lev,int* pdx);
int base(int l,int* s,int b);
```