

```

/* 词法分析源代码 */
# include< stdio.h>
# include< string.h>
char prog[80],token[8];
char ch;
int syn,p,m,n,sum;
char * rwtab[6]= {"function","if","then","while","do","end-
func"};
main()
{p= 0;
printf("\n please input string :\n");
do{
    scanf("%c",&ch);
    prog[+ + p]= ch;}
while(ch!= '# ');
p= 0;
do
{scanner();
switch(syn)
{ case 11: printf("\n(%d,%d)",syn,sum);break;
  case - 1: printf("\n errorn");break;
  default: printf("\n(%d,%s)",syn,token);
}
}while(syn!= 0);
}

```

```

scanner()

```

```

{for(n= 0;n< 8;n+ + ) token[n]= NULL;

```

```

  ch= prog[+ + p];

```

```

  while(ch!= '# ')ch= prog[+ + p];

```

```

  if(ch>= 'a'&&ch<= 'z')

```

```

  { m= 0;

```

```

    while(ch>= 'a'&&ch<= 'z' || ch>= '0'&&ch<= '9')

```

```
    { token[m+ + ]= ch;
```

```
      ch= prog[+ + p];
```

```
    }
```

```
token[m]= '\0';
```

```
ch= prog[- - p]; syn= 10;
```

```
for(n= 0;n< 6; n+ + )
```

```
if(strcmp (token,rwtab[n])= = 0)
```

```
{ syn= n+ 1;
```

```
  break;
```

```
}
```

```
}
```

```
else
```

```
if(ch> = '0'&&ch< = '9')
```

```
{ sum= 0;
```

```
  while (ch> = '0'&&ch< = '9')
```

```
{sum= sum* 10+ ch- '0';
```

```
  ch= prog[+ + p];
```

```
}
```

```
ch= prog[- - p]; syn= 11;
```

```
}
```

```
else
```

```
switch (ch)
```

```
{
```

```
  case '< ': m= 0; token[m+ + ]= ch;
```

```
    ch= prog[+ + p];
```

```
    if(ch= = '= ')
```

```
    {syn= 22;
```

```
      token[m+ 1]= ch;
```

```
    }
```

```
  else
```

```
    {syn= 20; ch= prog[- - p];}
```

```
  break;
```

```
case '> ': m= 0; token[m+ + ]= ch;
```

```
  ch= prog[+ + p];
```

```
  if(ch= = '= ')
```

```

        { syn= 24;
          token[m+ + ]= ch;
        }
      else
        {syn= 23;
          ch= prog[- - p];
        }
      break;
case '=' : m= 0; token[m+ + ]= ch;
ch= prog[+ + p];
if (ch== '=')
  { syn= 25;
    token[m+ + ]= ch;
  }
  else
    {syn= 18;
      ch= prog[- - p];
    }
    break;
case '!': m= 0; token[m+ + ]= ch;
ch= prog[+ + p];
if (ch== '=')
  { syn= 22;
    token[m+ + ]= ch;
  }
  else
    syn= - 1;
    break;
case '+': syn= 13; token[0]= ch; break;
case '-': syn= 14; token[0]= ch; break;
case '*': syn= 15; token[0]= ch; break;
case '/': syn= 16; token[0]= ch; break;
case ';': syn= 26; token[0]= ch; break;
case '(': syn= 27; token[0]= ch; break;
case ')': syn= 28; token[0]= ch; break;

```

```

    case '#': syn= 0; token[0]= ch;break;
default:syn= - 1;
}
}
/* 语法分析源代码 */
#include <stdio.h>
#include <string.h>
char prog[80],token[8];
char ch;
int syn,p,m= 0,n,sum,kk= 0;
char* rwtab[6]= {"function","if","then","while","do","endfunc"};
void scanner()
{
    for(n= 0;n< 8;n+ + )
        token[n]= NULL;
    while(ch= ' ')
        ch= prog[p+ + ];
    m= 0;
    if((ch<= 'z'&&ch>= 'a')||(ch<= 'Z'&&ch>= 'A'))
    {
        while((ch<= 'z'&&ch>= 'a')||(ch<= 'Z'&&ch>= 'A')
|| (ch<= '9'&&ch>= '0'))
        {
            token[m+ + ]= ch;
            ch= prog[p+ + ];
        }
        syn= 10;
        for(n= 0;n< 6;n+ + )
            if(strcmp(token,rwtab[n])== 0)
            {
                syn= n+ 1;
                break;
            }
        token[m+ + ]= '\0';
    }
}

```

else

if (ch <= '9' && ch >= '0')

{

sum = 0;

while (ch <= '9' && ch >= '0')

{

sum = sum * 10 + ch - '0';

ch = prog[p++];

}

syn = 11;

}

else

{

switch (ch)

{

case '<': m = 0; token[m++] = ch;

ch = prog[++p];

if (ch == '=')

{ syn = 22;

token[m+1] = ch;

}

else

{ syn = 20; ch = prog[--p]; }

break;

case '>': m = 0; token[m++] = ch;

ch = prog[++p];

if (ch == '=')

{ syn = 24;

token[m++] = ch;

}

else

{ syn = 23;

ch = prog[--p];

}

break;

```

case '=' : m = 0; token[m + ] = ch;
        ch = prog[ + + p];
        if (ch == '=')
        { syn = 25;
          token[m + ] = ch;
        }
        else
        { syn = 18;
          ch = prog[ - - p];
        }
        break;

```

```

case '!': m = 0; token[m + ] = ch;
        ch = prog[ + + p];
        if (ch == '=')
        { syn = 22;
          token[m + ] = ch;
        }
        else
        { syn = - 1;
        }
        break;

```

```

case '+': syn = 13; token[0] = ch; break;
case '-': syn = 14; token[0] = ch; break;
case '*': syn = 15; token[0] = ch; break;
case '/': syn = 16; token[0] = ch; break;
case ';': syn = 26; token[0] = ch; break;
case '(': syn = 27; token[0] = ch; break;
case ')': syn = 28; token[0] = ch; break;
case '#': syn = 0; token[0] = ch; break;
default: syn = - 1;

```

```

}
ch = prog[p + + ];

```

```

}

```

```

}

```

```

lrparser()

```

```

{
    if (syn == 1)
    {
        scanner();
        yucu(); /* 语句串分析 */
        if (syn == 6) /* 读到 endfunc */
        {
            scanner();
            if (syn == 0 && kk == 0) /* 程序分析识别完 */
                printf("success");
        }
        else
        {
            if (kk != 1) /* 没以 endfunc 结束 */
            {
                printf("error! need 'endfunc'");
                kk = 1;
            }
        }
    }
    else
    {
        printf("error! need 'function'");
        kk = 1;
    }
}

yucu() /* 语句串分析 */
{
    statement(); /* 调用语句分析函数 */
    while (syn == 26) /* 一个语句识别结束, 继续识别 */
    {
        scanner();
        statement();
    }
}

```

```

    return;
}

statement()
{
    if (syn == 10)
    {
        scanner();
        if (syn == 18)
        {
            scanner();
            expression();
        }
        else
        {
            printf("error! evaluate tag error");
            kk = 1;
        }
    }
    else
    {
        printf("error! the statement error!");
        kk = 1;
    }
}

```

expression() /* 表达式分析函数 */

```

{
    term();
    while (syn == 13 || syn == 14)
    {
        scanner();
        term();
    }
    return;
}

```



```
}
```

```
term() /* 项分析函数 */
```

```
{
```

```
    factor();
```

```
    while(syn== 15||syn== 16)
```

```
    {
```

```
        scanner();
```

```
        factor();
```

```
    }
```

```
    return;
```

```
}
```

```
factor() /* 因子分析函数 */
```

```
{
```

```
    if(syn== 10||syn== 11)
```

```
    {
```

```
        scanner();
```

```
    }
```

```
    else /* 看是否是表达式 */
```

```
    {
```

```
        if(syn== 27)
```

```
        {
```

```
            scanner();
```

```
            expression();
```

```
            if(syn== 28)
```

```
            {
```

```
                scanner();
```

```
            }
```

```
            else
```

```
            {
```

```
                printf("error! need another ')\n");
```

```
                kk= 1;
```

```
            }
```

```
        }
```

```

        else
        {
            printf("error! expression error!");
        }
    }
}

```

```
main()
```

```

{
    p= 0;
    printf("\nplease input the string:\n");
    do
    {
        ch= getchar();
        prog[p+ + ]= ch;
    }while(ch!= '#' );
    p= 0;
    ch= prog[p+ + ];
    scanner();
    lrparser();
}

```

```
/* 语义分析源代码 */
```

```

#include< stdio.h>
#include< string.h>
#include< conio.h>
#include< malloc.h>
#include< STDLIB.H>
struct quad
{
    char result[12];
    char ag1[12];
    char op[12];
    char ag2[12];
}

```

```

};
struct quad quad[30];
int count= 0;

char * expression(void);
char prog[200], token[8];
char ch;
int syn, p, m, n, sum= 0;
int kk= 0, k= 0;
char * rwtab[6]= {"function", "if", "then", "while", "do", "endfunc"};
scanner()
{
m= 0;
for (n= 0; n< 8; n++ )
    token[n]= '\0';
ch= prog[p++ ];
while (ch== ' ')
    ch= prog[p++ ];
if ((ch>= 'a' && ch<= 'z') || (ch>= 'A' && ch<= 'Z'))
{
    while ((ch>= 'a' && ch<= 'z') || (ch>= 'A' && ch<= 'Z') || (ch>=
'0' && ch<= '9'))
    {
        token[m++ ]= ch;
        ch= prog[p++ ];
    } /* end of while */
    token[m++ ]= '\0';
    p-- ;
    syn= 10;
    for (n= 0; n< 6; n++ )
    {
        if (strcmp(token, rwtab[n])== 0)
        {
            syn= n+1;
            break;

```

```

    }
    }/* end of for */
}
else if (ch >= '0' && ch <= '9')
{
    sum = 0;
    while (ch >= '0' && ch <= '9')
    {
        sum = sum * 10 + ch - '0';
        ch = prog[p++];
    }
    p--;
    syn = 11;
}
else
    switch (ch)
    {
        case '<': m = 0; token[m++] = ch;
        ch = prog[++p];
            if (ch == '=')
            {
                syn = 22;
                token[m+1] = ch;
            }
            else
            {
                syn = 20; ch = prog[--p];
            }
            break;
        case '>': m = 0; token[m++] = ch;
        ch = prog[++p];
            if (ch == '=')
            {
                syn = 24;
                token[m++] = ch;
            }
            else
            {
                syn = 23;
                ch = prog[--p];
            }
    }
}

```

```

        }
        break;
case '=' : m= 0; token[m+ + ]= ch;
        ch= prog[+ + p];
        if (ch== '=' )
        { syn= 25;
token[m+ + ]= ch;
        }
        else
        {syn= 18;
ch= prog[- - p];
        }
        break;
case '!': m= 0; token[m+ + ]= ch;
        ch= prog[+ + p];
        if (ch== '=' )
        { syn= 22;
token[m+ + ]= ch;
        }
        else
        syn= - 1;
        break;
case '+': syn= 13; token[0]= ch; break;
case '-': syn= 14; token[0]= ch; break;
case '*': syn= 15; token[0]= ch; break;
case '/': syn= 16; token[0]= ch; break;
case ';': syn= 26; token[0]= ch; break;
case '(': syn= 27; token[0]= ch; break;
case ')': syn= 28; token[0]= ch; break;
case '#': syn= 0; token[0]= ch; break;
default: syn= - 1;
}
}/* end of scanner */

```

```

void emit(char * result, char * ag1, char * op, char * ag2)

```

```

{
    strcpy(quad[count].result,result);
    strcpy(quad[count].ag1,ag1);
    strcpy(quad[count].op,op);
    strcpy(quad[count].ag2,ag2);
    count+ + ;
    return;
}

```

```

char * newtemp ()

```

```

{
    char * p;
    char m[8];
    p= (char *) malloc(8);
    k+ + ;
    itoa(k,m,10);
    strcpy(p+1,m);
    p[0]='t';
    return(p);
}

```

```

char * factor(void)

```

```

{
    char * fplace;
    fplace= (char *) malloc(12);
    strcpy(fplace," ");
    if(syn== 10)
    {
        strcpy(fplace,token);
        scanner();
    }
    else if(syn== 11)
    {
        itoa(sum,fplace,10);
        scanner();
    }
}

```

```

}
else if (syn == 27)
{
    scanner();
    fplace = expression();
    if (syn == 28)
        scanner();
    else
    {
        printf("\n') '错误");
        kk = 1;
    }
}
else
{
    printf("\n') ('错误");
    kk = 1;
}
return (fplace);
}

```

```

char * term (void)

```

```

{
    char * tp, * ep2, * eplace, * tt;
    tp = (char *) malloc(12);
    ep2 = (char *) malloc(12);
    eplace = (char *) malloc(12);
    tt = (char *) malloc(12);
    strcpy(eplace, factor());
    while (syn == 15 || syn == 16)
    {
        if (syn == 15)
        {
            tt[0] = '*';
            tt[1] = '\0';

```

```

    }
    else if (syn == 16)
    {
        tt[0] = '/';
        tt[1] = '\0';
    }
    scanner();
    strcpy(ep2, factor());
    strcpy(tp, newtemp());
    emit(tp, eplace, tt, ep2);
    strcpy(eplace, tp);
}
return(eplace);
}

```

```

char * expression(void)
{
    char * tp, * ep2, * eplace, * tt;
    tp = (char *) malloc(12);
    ep2 = (char *) malloc(12);
    eplace = (char *) malloc(12);
    tt = (char *) malloc(12);
    strcpy(eplace, term());
    while (syn == 13 || syn == 14)
    {
        if (syn == 13)
        {
            tt[0] = '+';
            tt[1] = '\0';
        }
        else if (syn == 14)
        {
            tt[0] = '-';
            tt[1] = '\0';
        }
    }
}

```



```

        scanner();
        strcpy(ep2, term());
        strcpy(tp, newtemp());
        emit(tp, eplace, tt, ep2);
        strcpy(eplace, tp);
    }
    return(eplace);
}

int statement()
{
    char tt[8], eplace[8];
    int schain= 0;
    switch(syn)
    {case 10:
        strcpy(tt, token);
        scanner();
        if(syn== 18)
        {
            scanner();
            strcpy(eplace, expression());
            emit(tt, eplace, " ", " ");
            schain= 0;
        }
        else
        {
            printf("\n 缺少赋值号\n");
            kk= 1;
        }
        break;
    }
    return(schain);
}

int yucu()

```

```

{
    int schain= 0;
    schain= statement();
    while(syn= = 26)
    {
        scanner();
        schain= statement();
    }
    return(schain);
}

```

```

int lrparser()
{
    int schain= 0;
    kk= 0;
    if(syn= = 1)
    {
        scanner();
        schain= yucu();
        if(syn= = 6)
        {
            scanner();
            if(syn= = 0&&kk= = 0)
                printf("\n 语法,语义分析成功");
        }
        else
        {
            if(kk! = 1)
            {
                printf("\n 缺 endfunc\n");
                kk= 1;
            }
        }
    }
    else

```

```

    {
        printf("\n 缺 function\n");
        kk= 1;
    }
    return(schain);
}

void main()
{
    int i;
    p= 0;
    printf("请输入语句 (以# 结束,不要换行):");
    do
        {ch= getchar();
         prog[p+ ]= ch;
         }while(ch!= '# ');
    p= 0;
    printf("种别码          单词符号\n");
    do
        {scanner();
         switch(syn)
         {case 11:printf("%- 3d      %d\n",syn,sum);break;
          case - 1:printf("词法分析失败,程序终止! \n");return;
          default:printf("%- 3d  %s\n",syn,token);
          }
         }while(syn!= 0);
    printf("词法分析成功,按任意键进行语法,语义分析");
    getch();
    p= 0;
    scanner();
    lrparser();
    if(kk!= 0)
    {
        printf("语法分析失败,程序终止!");
        return;
    }
}

```

```
}  
printf("\n 三地址指令如下:\n");  
for(i= 0;i< count;i+ + )  
{  
printf("%s= ",quad[i].result);  
printf("%s",quad[i].ag1);  
printf("%s",quad[i].op);  
printf("%s\n",quad[i].ag2);  
}  
getch();  
return;
```

```
}
```