编译器专题实验报告

实验六:语义分析

实验内容：

目的：

1. 设计和实现一个符号表。
2. 使用散列或线性结构实现符号表数据结构。
3. 设计符号表可存储程序语言中的各种标识符（变量、常量、数组、结构、指针、函数和过程）及其属性和作用域信息。

功能：

- 填表:当分析到程序中的说明或定义语句时，将说明或定义的名字，以及与之有关的信息填入符号表中。
- 查表:填表前查表，检查在程序的同一作用域内名字是否重复定义；检查名字的种类是否与说明一致；对于强类型语言，要检查表达式中各变量的类型是否一致；生成目标指令时，要取得所需要的地址。

实验结果：

具体实验要求：（必做部分）

- 输入：一段代码。
- 输出：执行符号表程序，如该标识符是变量，输出变量的类型和作用域，如该标识符是数组，输出数组的类型和作用域，如该标识符是函数，则需要输出更多内容。
- 最简单的输出就是输入多个变量，有不同类型，然后符号表查表后，得到该变量的值和范围。当出现强制类型转换的时候，会报错。
- 考虑，结合词法分析的scaner的输出内容，可以将scaner的输出作为符号表的信息部分。

后两次实验使用flex与bison工具辅助，通过C语言实现Sample语言的编译器。

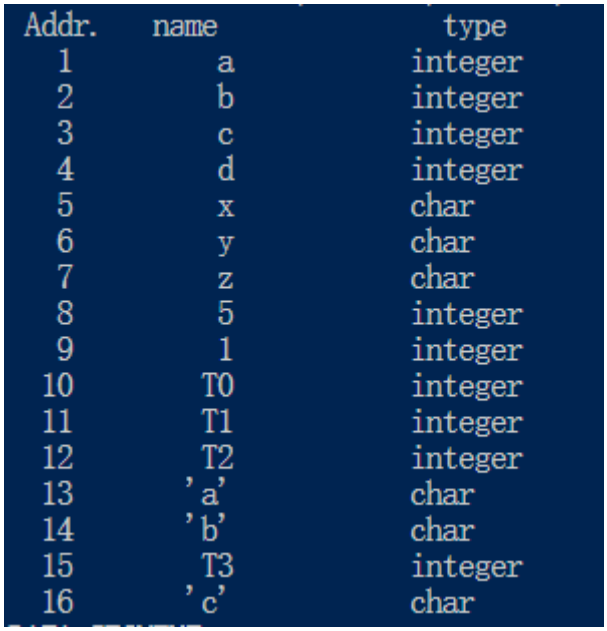
> *Sample*语言是*PASCAL*语言的简化版本。具有一般高级语言的共同特征：它的字符集包括所有的大小写字母、数字和一些界符；有多种数据类型：整型、实型、字符型等；有变量说明和常量说明；包括顺序、条件和循环三种语句结构。

Sample语言的具体定义格式不在此赘述，整体来说与C语言类似。PASCAL语言在编译原理课程中有所涉及，因此Sample语言对我们来说是不陌生的。

输入代码为：

```
program whiletest;
var
    a,b,c,d:integer;
    x,y,z:char;
begin
    a:=5;
    b:=a+1;
    c:=b+1;
    d:=c+1;
    x:='a';
    y:='b';
    while  (a<b)
    do
    begin
            if (c<d)
            then
               a:=a+1;
            z:='c';
    end
end.
```

输出符号表为：



Addr.	name	type
1	a	integer
2	b	integer
3	c	integer
4	d	integer
5	x	char
6	y	char
7	z	char
8	5	integer
9	1	integer
10	T0	integer
11	T1	integer
12	T2	integer
13	'a'	char
14	'b'	char
15	T3	integer
16	'c'	char

符号表中说明了变量的名称和类型。在填入符号表之前回去检查符号表中是否已经有了该符号，防止重复定义

对于每个在翻译过程中出现的符号(包括中间变量)，本程序会记录其名称、类型和地址。Entry()方法定义了在符号表中找到变量并返回其地址。NewTemp()方法返回一个新的中间变量Ti的地址。FillType()方法是将某一地址之后的所有元素类型都设置为一指定类型。

另外遇到的问题和解决思路(可选)：

Pascal语言与C语言有一些小的差别，在词法分析和语法分析阶段借助了flex和bison来完成。

代码：

产生符号表部分关键代码

```
int Entry(char *name)           //找到名位name的变量在符号表中的下标
{
      int i;
      for(i=1;i<=VarCount;i++)
             if(!strcmp(name,VarList[i].name))
                        return i;
      if(++VarCount>MAXMEMBER)
      {
              printf("Too many Variables!\n");exit(-1);
      }
```

```
     strcpy(VarList[VarCount].name,name);
     return VarCount;
}

int FillType(int first,int type)    //在符号表中first往后的元素类型都设为type
{
    int i;
    for(i=first;i<=VarCount;i++)
          VarList[i].type=type;
    return i-1;
}

int NewTemp()   //在符号表中插入Ti
{
     static int no=0;
     char Tname[10];
     sprintf(Tname,"T%o",no++);
     return Entry(Tname);
}

void OutputIList(void)  //输出符号表
{
     int i;
     printf(" Addr.\t name \t\t   type\n");
     for(i=1;i<=VarCount;i++)
     {
           printf("%4d\t%6s\t\t",i,VarList[i].name);
            if(VarList[i].type==INTEGER)
                printf(" integer  \n");
            else if(VarList[i].type==BOOL)
                printf(" bool     \n");
            else if(VarList[i].type==CHAR)
                printf(" char     \n");
            else if(VarList[i].type==REAL)
                printf(" real     \n");
            else
                printf("          \n");
      }
      return;
}
```

实验七:可执行代码构建

实验内容：

目的：将语义分析输出的符号表映射为内存映像，并生成依赖于栈帧的目标代码，将结果输出到文件中。

功能：

- 栈帧设计（含D表）；
- 序言、尾声、调用序列、返回序列构建；
- 名引用的代码变换（引用序列构建）；
- 目标语言指令模板（MIPS）

实验结果：

具体实验要求：（必做部分）

- 实现控制语句的拉链返填，形式不限

具体实验要求：（选做部分）

- 将语义分析程序生成的四元式作为输入，并通过代码实现输出相应的汇编语言。

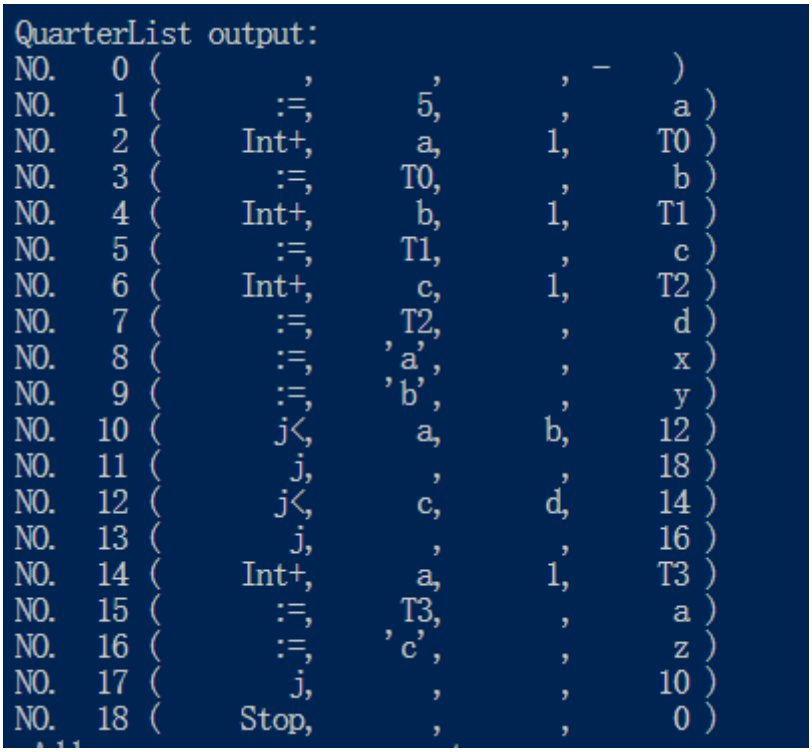
输入代码为：

```
program whiletest;
var
    a,b,c,d:integer;
    x,y,z:char;
begin
    a:=5;
    b:=a+1;
    c:=b+1;
    d:=c+1;
    x:='a';
    y:='b';
```

```
12      while  (a<b)
13      do
14      begin
15              if (c<d)
16              then
17                 a:=a+1;
18              z:='c';
19      end
20  end.
```

输出结果为：

四元式的拉链反填：



对应的汇编语言：

```
DATA SEGMENT
    TAB DW 16 DUP(?)
    Temp db '0000H','$'
    DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA

    START:
    MOV AX,DATA
    MOV DS,AX
NO1:
    MOV AX, 5
    MOV TAB[0], AX
NO2:
    MOV AX, TAB[0]
    ADD AX, 1
    MOV TAB[18], AX
NO3:
    MOV AX, TAB[18]
    MOV TAB[2], AX
NO4:
    MOV AX, TAB[2]
    ADD AX, 1
    MOV TAB[20], AX
NO5:
    MOV AX, TAB[20]
    MOV TAB[4], AX
NO6:
    MOV AX, TAB[4]
    ADD AX, 1
    MOV TAB[22], AX
NO7:
    MOV AX, TAB[22]
    MOV TAB[6], AX
NO8:
    MOV AX, 'a'
    MOV TAB[8], AX
NO9:
    MOV AX, 'b'
    MOV TAB[10], AX
NO10:
    MOV AX, TAB[0]
    CMP AX, TAB[2]
    JL NO12
NO11:
    JMP NO18
```

```
NO12:
    MOV AX, TAB[4]
    CMP AX, TAB[6]
    JL NO14
NO13:
    JMP NO16
NO14:
    MOV AX, TAB[0]
    ADD AX, 1
    MOV TAB[28], AX
NO15:
    MOV AX, TAB[28]
    MOV TAB[0], AX
NO16:
    MOV AX, 'c'
    MOV TAB[12], AX
NO17:
    JMP NO10
NO18:
    MOV CX, 16
    MOV SI, 0

AGAIN:
    MOV AX,TAB[SI]
    CALL PrintAX
    CALL PRINTHC
    INC SI
    INC SI
    LOOP AGAIN
NOO:
    MOV AH,4CH
    INT 21H
PrintAX proc
    PUSH CX
    PUSH SI
    jmp Next


Next:
    mov si,offset Temp+3;保存存储结果的字符串的最后一个字符偏移地址
    xor cx,cx           ;对cX清零
    mov cl,4            ;设置循环次数为4次
```

```
MainPart:
    mov DH,AL           ;将A1的内容传送给DH

    shr AX,1
    shr AX,1
    shr AX,1
    shr AX,1
                ;上述4句为使AX逻辑右移4位 理论上可以写成 shr AX ,c1(c1设置为4)
                ;但这个地方c1要记录循环次数每次循环会是c1-1所以无法满足正常的移位需要
    and dh,0FH
    add dh,30H
    cmp dh,':'  ;':'的ASCII比9大1 而字母的ASCII码与数字的ASCII码中间隔了7个其它字符
    ja isLetter ;如果为字母则跳转
    jb No       ;如果不是

isLetter:
    add dh,7H   ;ASCII码加7变为字母
No:
    mov [si],dh ;将字符存入,待存放内容的字符串的对应位置

    dec si      ;待存放内容的内存地址自减1
loop MainPart

print:
    mov dx,offset Temp  ;将带打印的字符串的偏移地址存放进dx中
    mov ah,09           ;设置DOS 09号功能
    int 21H         ;功能调用

    POP SI
    POP CX

    ret
PrintAX endp


PRINTHC proc
    MOV DL, 10
    MOV AH, 2
    INT 21H
    RET
PRINTHC ENDP
    CODE ENDS
    END START
```

另外遇到的问题和解决思路(可选)：

控制语句的拉链返填，采用结构体来存储四元式。

布尔表达式的翻译中，并不需要计算表达式的值，使用跳转语句控制程序流向即可。对于每个布尔项和布尔表达式，定义TC为表达式的真出口，FC为表达式的假出口。BackPatch(p,t)称为回填，能够把以p为链首的每个待回填的四元式第四段都填t；Merge(p1,p2)乘坐并链，能够把p1和p2为链头的两条链并为一条链。使用NXQ记录下一条要建立的四元式的序号。基于此实现布尔表达式的翻译。

对于每个生成的四元式，QUATERLIST记录其操作名称，两个运算量和结果单元的地址。根据地址可以在符号表中找到变量的名称和类型。若某一运算量为0则该运算量无效。四元式的地址从1开始，若执行到跳转到0的语句，则程序结束。GEN()方法能够制造一个四元式。

总体思想是，将符号表中每一个变量都存放在一个TAB数组中，每次从TAB中取数到AX寄存器中，继续操作。完成操作后将AX中的内容赋值给TAB数组。

代码：

产生四元式及拉链回填部分关键代码

```
fuzhiyuju:
    bianliang FZ biaodashi
    {
        printf("<赋值语句> ::= <变量>:=<表达式>     \n");
        // int t=NewTemp();
        //$1.type=integer
        if(VarList[$1].type==INTEGER && $3.type==INTEGER)
        {
            GEN(":=",$3.place,0,$1);
        }
        else if(VarList[$1].type==INTEGER && $3.type==BOOL)
        {
            int u=NewTemp();
            VarList[u].type=INTEGER;
            GEN("B->I",$3.place,0,u);
            GEN(":=",u,0,$1);
        }
        else if(VarList[$1].type==INTEGER && $3.type==CHAR)
        {
            int u=NewTemp();
            VarList[u].type=INTEGER;
            GEN("C->I",$3.place,0,u);
            GEN(":=",u,0,$1);
        }
        else if(VarList[$1].type==INTEGER && $3.type==REAL)
        {
            int u=NewTemp();
            VarList[u].type=INTEGER;
            GEN("R->I",$3.place,0,u);
            GEN(":=",u,0,$1);
```

```
31          }
32
33          //$1.type=bool
34          else if(VarList[$1].type==BOOL && $3.type==BOOL)
35          {
36              GEN(":=",$3.place,0,$1);
37          }
38          else if(VarList[$1].type==BOOL && $3.type==INTEGER)
39          {
40              int u=NewTemp();
41              VarList[u].type=BOOL;
42              GEN("I->B",$3.place,0,u);
43              GEN(":=",u,0,$1);
44          }
45          else if(VarList[$1].type==BOOL && $3.type==CHAR)
46          {
47              int u=NewTemp();
48              VarList[u].type=BOOL;
49              GEN("C->B",$3.place,0,u);
50              GEN(":=",u,0,$1);
51          }
52          else if(VarList[$1].type==BOOL && $3.type==REAL)
53          {
54              int u=NewTemp();
55              VarList[u].type=BOOL;
56              GEN("R->B",$3.place,0,u);
57              GEN(":=",u,0,$1);
58          }
59
60          //$1.type=char
61          else if(VarList[$1].type==CHAR && $3.type==CHAR)
62          {
63              GEN(":=",$3.place,0,$1);
64          }
65          else if(VarList[$1].type==CHAR && $3.type==INTEGER)
66          {
67              int u=NewTemp();
68              VarList[u].type=CHAR;
69              GEN("I->C",$3.place,0,u);
70              GEN(":=",u,0,$1);
71          }
72          else if(VarList[$1].type==CHAR && $3.type==BOOL)
```

```
        {
            int u=NewTemp();
            VarList[u].type=CHAR;
            GEN("B->C",$3.place,0,u);
            GEN(":=",u,0,$1);
        }
        else if(VarList[$1].type==CHAR && $3.type==REAL)
        {
            int u=NewTemp();
            VarList[u].type=CHAR;
            GEN("R->C",$3.place,0,u);
            GEN(":=",u,0,$1);
        }

        //$1.type=real
        else if(VarList[$1].type==REAL && $3.type==REAL)
        {
            GEN(":=",$3.place,0,$1);
        }
        else if(VarList[$1].type==REAL && $3.type==INTEGER)
        {
            int u=NewTemp();
            VarList[u].type=REAL;
            GEN("I->R",$3.place,0,u);
            GEN(":=",$3.place,0,$1);
        }
        else if(VarList[$1].type==REAL && $3.type==BOOL)
        {
            int u=NewTemp();
            VarList[u].type=REAL;
            GEN("B->R",$3.place,0,u);
            GEN(":=",u,0,$1);
        }
        else if(VarList[$1].type==REAL && $3.type==CHAR)
        {
            int u=NewTemp();
            VarList[u].type=REAL;
            GEN("C->R",$3.place,0,u);
            GEN(":=",u,0,$1);
        }
    }
;
```

```
void OutputQ(void)  //输出四元式
{
      int i;
      printf("\nQuarterList output:\n");
      for(i=0;i<NXQ;i++)
      {
        printf("NO.%4d ( %8s, ",i,QuaterList[i].op);
        if(QuaterList[i].arg1)
            printf("%6s, ",VarList[QuaterList[i].arg1].name);
        else printf("      , ");
        if(QuaterList[i].arg2)
            printf("%6s, ",VarList[QuaterList[i].arg2].name);
        else printf("      , ");
        if((QuaterList[i].op[0]=='j')||
(QuaterList[i].op[0]=='S'))
            printf("%6d ) \n",QuaterList[i].result);
        else if(QuaterList[i].result)
            printf("%6s
)\n",VarList[QuaterList[i].result].name);
        else printf("-\t )\n");
      }
     return;
}
int Merge(int p,int q)
{
   int r;
   if(!q)
        return p;
   else
   {
       r=q;
       while(QuaterList[r].result)
        r=QuaterList[r].result;
       QuaterList[r].result=p;
   }
   return q;
}

void BackPatch(int p,int t)
{
    printf("*****************BackPatch %d,%d\n",p,t);
     int q=p;
```

```
    while(q)
    {
          int q1=QuaterList[q].result;
          if(q1==q)
          {
              printf("backpatch error %d \n",q);
              break;
          }
          if(q==t)
          {
              printf("backpatch error2 %d \n",q);
              break;
          }
          QuaterList[q].result=t;
          q=q1;
    }
    return;
}

int GEN(char* op,int a1,int a2,int re)
{
    strcpy(QuaterList[NXQ].op,op);
    QuaterList[NXQ].arg1=a1;
    QuaterList[NXQ].arg2=a2;
    QuaterList[NXQ].result=re;
    NXQ++;
    return NXQ;
}
```

产生汇编语言关键代码

```
void generatehead(int table_size)
{
    printf("DATA SEGMENT\n\
    TAB DW %d DUP(?)\n\
    Temp db '0000H','$'\n\
    DATA ENDS\n",table_size);

    printf("CODE SEGMENT\n\
    ASSUME CS:CODE,DS:DATA\n\
    \n\
```

```
    START:\n\
    MOV AX,DATA\n\
    MOV DS,AX\n");
}

void generateNO(int i,char *res)
{
    sprintf(res,"NO%d",i);
}

void toAX(int arg,const VARLIST *varlist)
{
    char str[20];
    strcpy(str,varlist[arg].name);
    if(str[0]>='0'&&str[0]<='9'||str[0]=='\'')//立即数
    {
        printf("    MOV AX, %s\n",str);
        return;
    }
    if(str[0]=='T')//Tx 中间变量
    {
        int i=arg*2-2;
        printf("    MOV AX, TAB[%d]\n",i);
        return;
    }

    //变量
    int i=arg*2-2;
    printf("    MOV AX, TAB[%d]\n",i);
    return;
}

void generateOP(const QUATERLIST q,const VARLIST *v)
{
    int res_add=q.result*2-2;
    int arg2_add=q.arg2*2-2;
    char arg2_heheda[20];
    if(v[q.arg2].name[0]>='0'&&v[q.arg2].name[0]<='9')//arg2是立
即数
        strcpy(arg2_heheda,v[q.arg2].name);
    else
        sprintf(arg2_heheda,"TAB[%d]",arg2_add);
```

```
52
53      char tore[20];
54      generateNO(q.result,tore);
55
56      if(strcmp(q.op,":=")==0)
57      {
58          toAX(q.arg1,v);
59          printf("    MOV TAB[%d], AX\n",res_add);
60          return;
61      }
62      if(strcmp(q.op,"jnz")==0)
63      {
64          toAX(q.arg1,v);
65          printf("    CMP AX,0\n\
66      JNZ %s\n",tore);
67          return;
68      }
69      if(strcmp(q.op,"j<")==0)
70      {
71          toAX(q.arg1,v);
72          printf("    CMP AX, %s\n\
73      JL %s\n",arg2_heheda ,tore);
74          return;
75      }
76      if(strcmp(q.op,"j>")==0)
77      {
78          toAX(q.arg1,v);
79          printf("    CMP AX, %s\n\
80      JG %s\n",arg2_heheda,tore);
81          return;
82      }
83      if(strcmp(q.op,"j=")==0)
84      {
85          toAX(q.arg1,v);
86          printf("    CMP AX, %s\n\
87      JE %s\n",arg2_heheda,tore);
88          return;
89      }
90      if(strcmp(q.op,"j>=")==0)
91      {
92          toAX(q.arg1,v);
93          printf("    CMP AX, %s\n\
```

```
94      JGE %s\n",arg2_heheda,tore);
95          return;
96      }
97      if(strcmp(q.op,"j<=")==0)
98      {
99          toAX(q.arg1,v);
100         printf("    CMP AX, %s\n\
101     JLE %s\n",arg2_heheda,tore);
102         return;
103     }
104     if(strcmp(q.op,"j<>")==0)
105     {
106         toAX(q.arg1,v);
107         printf("    CMP AX, %s\n\
108     JNE %s\n",arg2_heheda,tore);
109         return;
110     }
111     if(strcmp(q.op,"j")==0)
112     {
113         printf("    JMP %s\n",tore);
114         return;
115     }
116
117     //转换语句
118     if(q.op[1]=='-'&&q.op[2]=='>')
119     {
120     if(strcmp(q.op,"I->R")==0||strcmp(q.op,"B-
    >R")==0||strcmp(q.op,"C->R")==0)
121     {
122         toAX(q.arg1,v);
123         printf("    MOV BX, 100\n\
124     IMUL BX\n\
125     MOV TAB[%d], AX\n",res_add);
126         return;
127     }
128     if(strcmp(q.op,"R->I")==0||strcmp(q.op,"R->C")==0)
129     {
130         toAX(q.arg1,v);
131         printf("    MOV BX, 100\n\
132     IDIV BX\n\
133     MOV TAB[%d], AX\n",res_add);
134         return;
```

```
135     }
136     if(strcmp(q.op,"I->B")==0||strcmp(q.op,"C-
    >B")==0||strcmp(q.op,"I->B")==0)
137     {
138         toAX(q.arg1,v);
139         printf("    AND AX, 0001H\n\
140     MOV TAB[%d], AX\n",res_add);
141         return;
142     }
143
144     else
145     {
146         toAX(q.arg1,v);
147         printf("    MOV TAB[%d], AX\n",res_add);
148         return;
149     }
150     return;
151     }
152
153     //运算语句
154     if(strcmp(q.op,"Minus")==0)
155     {
156         printf("    MOV AX, 0\n\
157     SUB AX, TAB[%d]\n\
158     MOV TAB[%d], AX\n",q.arg1*2-2,res_add);
159         return;
160     }
161     if(q.op[0]=='B')
162     {
163         toAX(q.arg1,v);
164         if(strcmp(q.op,"BOOL+")==0||strcmp(q.op,"BOOL-")==0)
165         {
166             printf("    XOR AX, %s\n\
167     MOV TAB[%d], AX\n",arg2_heheda,res_add);
168             return;
169         }
170         if(strcmp(q.op,"BOOL*")==0||strcmp(q.op,"BOOL/")==0)
171         {
172             printf("    AND AX, %s\n\
173     MOV TAB[%d], AX\n",arg2_heheda,res_add);
174             return;
175         }
```

```
        return;
    }

    int op_len=strlen(q.op);
    if(q.op[op_len-1]=='+')
    {
        toAX(q.arg1,v);
        printf("    ADD AX, %s\n\
    MOV TAB[%d], AX\n",arg2_heheda,res_add);
        return;
    }
    if(q.op[op_len-1]=='-')
    {
        toAX(q.arg1,v);
        printf("    SUB AX, %s\n\
    MOV TAB[%d], AX\n",arg2_heheda,res_add);
        return;
    }
    if(q.op[op_len-1]=='*')
    {
        toAX(q.arg1,v);
        printf("     MOV BX, %s\n\
    IMUL BX\n\
    MOV TAB[%d], AX\n",arg2_heheda,res_add);
        return;
    }
    if(q.op[op_len-1]=='/')
    {
        toAX(q.arg1,v);
        printf("    MOV BX, %s\n\
    IDIV BX\n\
    MOV TAB[%d], AX\n",arg2_heheda,res_add);
        return;
    }
    if(strcmp(q.op,"Stop")==0)
    {
        return;
    }
    else
    {
        fprintf (stderr, "Table.h: 无法识别的操作符！\n");
    }
```

```
218     return;
219 }
220 
221 void generateoutA(int n)
222 {
223     printf("    MOV CX, %d\n\
224     MOV SI, 0\n\
225 \n\
226 AGAIN:\n\
227     MOV AX,TAB[SI]\n\
228     CALL PrintAX\n\
229     CALL PRINTHC\n\
230     INC SI\n\
231     INC SI\n\
232     LOOP AGAIN\n",n);
233 }
234 
235 void generatepro()
236 {
237     printf("PrintAX proc\n\
238     PUSH CX\n\
239     PUSH SI\n\
240     jmp Next\n\
241 \n\
242 Next:\n\
243     mov si,offset Temp+3;保存存储结果的字符串的最后一个字符偏移地址\n\
244     xor cx,cx           ;对CX清零\n\
245     mov cl,4            ;设置循环次数为4次\n\
246 \n\
247 MainPart:\n\
248     mov DH,AL           ;将Al的内容传送给DH\n\
249 \n\
250     shr AX,1\n\
251     shr AX,1\n\
252     shr AX,1\n\
253     shr AX,1\n\
254                 ;上述4句为使AX逻辑右移4位 理论上可以写成 shr AX
    ,cl(cl设置为4)\n\
255                 ;但这个地方cl要记录循环次数每次循环会是cl-1所以无法满足
    正常的移位需要\n\
256     and dh,0FH\n\
257     add dh,30H\n\
```

```
    cmp dh,':'  ;':'的ASCII比9大1 而字母的ASCII码与数字的ASCII码中间隔了7个其它字符\n\
    ja isLetter ;如果为字母则跳转\n\
    jb No       ;如果不是\n\
\n\
isLetter:\n\
    add dh,7H   ;ASCII码加7变为字母\n\
No:\n\
    mov [si],dh ;将字符存入,待存放内容的字符串的对应位置\n\
\n\
    dec si      ;待存放内容的内存地址自减1\n\
loop MainPart\n\
\n\
print:\n\
    mov dx,offset Temp  ;将带打印的字符串的偏移地址存放进dx中\n\
    mov ah,09           ;设置DOS 09号功能\n\
    int 21H         ;功能调用\n\
\n\
    POP SI\n\
    POP CX\n\
\n\
    ret\n\
PrintAX endp\n\
\n\
\n\
PRINTHC proc\n\
    MOV DL, 10\n\
    MOV AH, 2\n\
    INT 21H\n\
    RET\n\
PRINTHC ENDP\n");
}
```