编译器构造实验3

19335206 韦媛馨

2022-05-19

本实验GitHub链接：https://github.com/Atopos-309/Compilation-principle

1 实验描述

表达式语义分析器的设计与实现

2 实验要求

使用递归下降翻译法或LL(1)翻译法实现高级编程语言的语义分析，将其翻译为四元式格式的中间语言，至少支持算术表达式的语义分析。

算数表达式至少支持加减乘除以及括号操作，即（+，-，*，/，（））

3 总体设计描述

四元式的基本形式为$q:(\omega,o_1,o_2,t)$，即（算符，对象1，对象2，结果）。总体的设计思路为：按照顺序将任意一个正确的算术表达式拆分成操作符和操作数部分并入栈，而后比较优先级，按照优先级高低出栈，执行操作——将算术表达式转换成四元式输出。

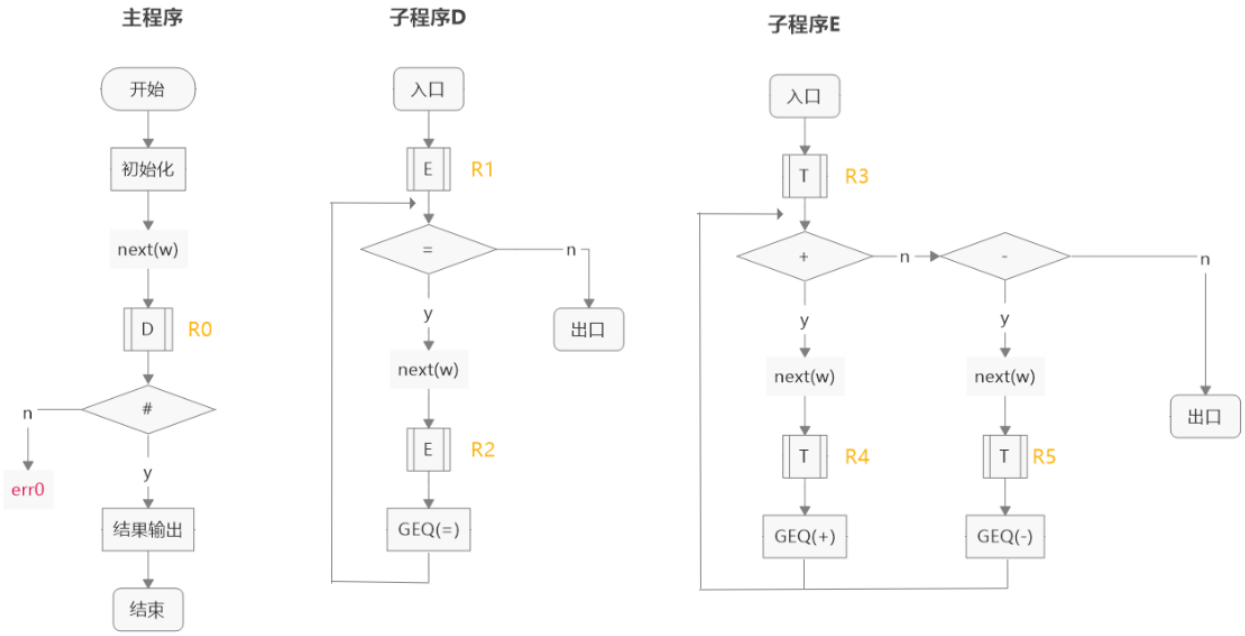
要先设计翻译文法，再根据文法设计子程序流程图，在每一个子程序入口，把返回地址压入返回地址栈并进入；在子程序出口，把返回地址弹出返回地址栈并返回。

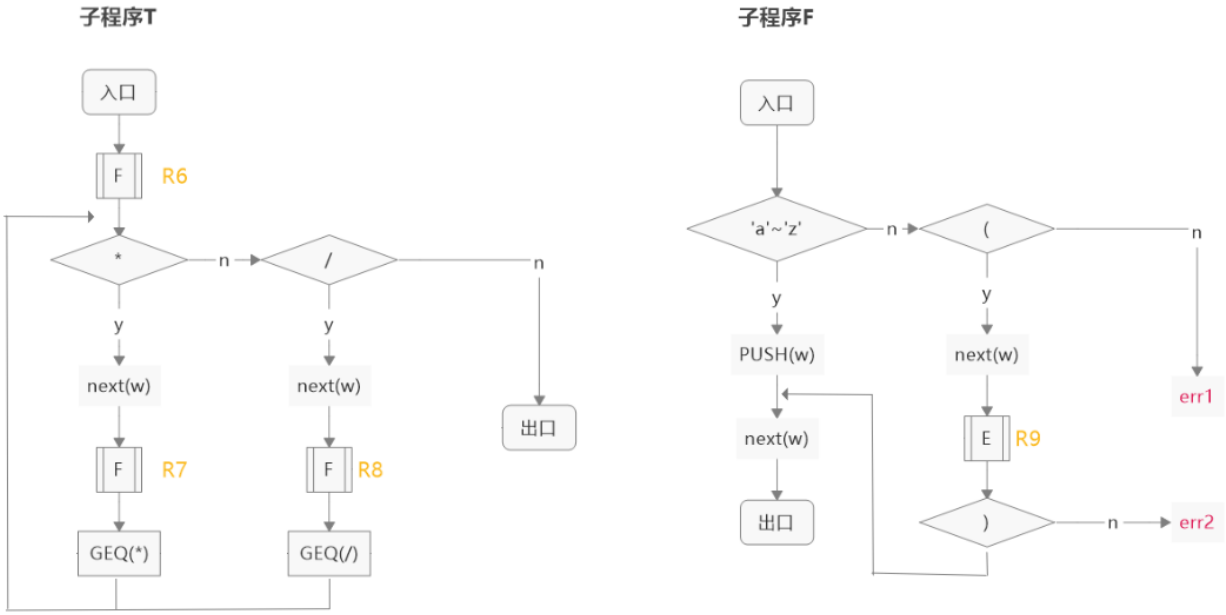
4 翻译文法设计

```
D -> E = E{GEQ(=)}
E -> T {+T{GEQ(+)} | -T{GEQ(-)}}
T -> F {*F{GEQ(*)} | /F{GEQ(/)}}
F -> i{PUSH(i)} | (E)
其中i: 'a'~'z'
PUSH(i):压栈函数
GEQ(w)：表达式四元式产生函数
```

该文法支持基本的加减乘除以及括号操作，即（+，-，*，/，（）），此外还支持赋值语句=。

5 子程序流程图





其中，上面R_i表示返回地址，err0，err1，err2是三种不同的错误类型：

> err0：输入没有以'#'结尾
>
> err1：无法识别输入符号
>
> err2：括号不匹配

6 算术表达式实例的分析表

以`y=a+(b*(c-f/g)+k)#`为例，构造该算术表达式的分析表如下：

待翻译的表达式：y=a+(b*(c-f/g)+k)#

递归子程序栈	返回地址栈	w	SEM[m]	qt[q]
ZDETF	R0R1R3R6	y	y	
ZD	R0	=	y	
ZDE	R0R2	a	y	
ZDETF	R0R2R3R6	+	ya	
ZDE	R0R2	(	ya	
ZDETF	R0R2R4R6	b	ya	
ZDETFETF	R0R2R4R6R9R3R6	*	yab	
ZDETFET	R0R2R4R6R9R3	(	yab	
ZDETFETF	R0R2R4R6R9R3R7	c	yab	
ZDETFETFE	R0R2R4R6R9R3R7R9	-	yabc	
ZDETFETFETF	R0R2R4R6R9R3R7R9R5R6	f	yabc	
ZDETFETFET	R0R2R4R6R9R3R7R9R5	/	yabcf	
ZDETFETFEF	R0R2R4R6R9R3R7R9R8	g	yabcf	
ZDETFETFEFTF	R0R2R4R6R9R3R7R9R8R3R6	)	yabcfg	(1)/, f, g, t1
ZDETFEFTF	R0R2R4R6R9R3R7R9	)	yabct1	(2)-, c, t1, t2
ZDETFEFT	R0R2R4R6R9R3R7	)	yabt2	(3)*, b, t2, t3
ZDETFEF	R0R2R4R6R9R3	+	yat3	
ZDETFEFTF	R0R2R4R6R9R3R6	k	yat3	
ZDETFEF	R0R2R4R6R9	)	yat3k	(4)+, t3, k, t4
ZDE	R0R2R4	#	yat4	(5)+, a, t4, t5
ZD	R0	#	yt5	(6)=, t5, _, y
Z		#	y	
OK!				

7 实验过程

7.1 数据结构设计

```
struct TOKEN{
    char t;
    int i;
};

struct QT{
    char op;
    struct TOKEN obj1;
    struct TOKEN obj2;
    struct TOKEN res;
};

struct TOKEN word;
struct TOKEN sem[10];  //语义栈
int sem_num;

struct QT qt[30];  //四元式区
int qt_num=0;
int t_idx=1;   //四元式下标

char exp[50]; //表达式
char cur;   //当前表达式输入
```

```
int idx=0;  //当前表达式输入下标
```

7.2 关键代码

7.2.1 主程序Z

```
int main(){
    printf("please input your expression:  ");
    scanf("%s",exp);
    int len=strlen(exp);
    next();
    D();
    if(cur=='#'&&idx==len){
        for(int i=0;i<qt_num;++i){
            printf("(%d)",i+1);
            if(qt[i].op!='='){
                printf("(%c",qt[i].op);
                if(qt[i].obj1.t!='t')
                    printf(",%c",qt[i].obj1.t);
                else
                    printf(",%c%d",qt[i].obj1.t,qt[i].obj1.i);
                if(qt[i].obj2.t!='t')
                    printf(",%c",qt[i].obj2.t);
                else
                    printf(",%c%d",qt[i].obj2.t,qt[i].obj2.i);
                printf(",%c%d)",qt[i].res.t,qt[i].res.i);
            }
            else{
                printf("(%c",qt[i].op);
                printf(",%c%d",qt[i-1].res.t,qt[i-1].res.i);
                printf(",_");
                printf(",%c)",qt[i].obj1.t);
            }
            printf("\n");
        }
        printf("OK!");
    }
    else if(idx==len+1)
        printf("Error: The input does not end with '#'\n");
}
```

7.2.2 子程序D

```
int D(){
    char w;
    E();
    while(cur=='='){
        w=cur;
        next();
        E();
        GEQ(w);
    }
    return 1;
}
```

7.2.3 子程序E

```
int E(){
    char w;
    T();
    while(cur=='+'||cur=='-'){
        w=cur;
        next();
        T();
        GEQ(w);
    }
    return 1;
}
```

7.2.4 子程序T

```
int T(){
    char w;
    F();
    while(cur=='*'||cur=='/'){
        w=cur;
        next();
        F();
        GEQ(w);
    }
    return 1;
}
```

7.2.5 子程序F

```
int F(){
    if(cur=='('){
        next();
        E();
```

```
        if(cur!=')'){
            printf("Error: bracket mismatch\n");
            return 0;
        }
    }
    else if(cur>='a'&&cur<='z'){  //PUSH(cur)
        word.t=cur;
        word.i=0;
        sem[++sem_num]=word;
    }
    else{
        printf("Error: unrecognized input symbol %c\n",cur);
        return 0;
    }
    next();
    return 1;
}
```

7.2.6 生成一个四元式送qt

```
void GEQ(char w){
    newt();
    qt[qt_num].op=w;
    qt[qt_num].obj1=sem[sem_num-1];
    qt[qt_num].obj2=sem[sem_num];
    qt[qt_num].res=word;
    sem_num--;
    sem_num--;
    sem[++sem_num]=word;
    qt_num++;
}
```

8 运行结果

8.1 运行方式说明

cd到当前文件夹，编译生成可执行文件 a.exe

```
gcc source.c
```

运行：

```
./a
```

根据提示输入算术表达式即可进行算术表达式到四元式的翻译，注意要以#结尾。

8.2 运行结果展示

对于表达式 `y=a+(b*(c-f/g)+k)#`，

```
PS D:\Study\大三下\编译原理\HW\19335206-韦媛馨-实验3> ./a
please input your expression:  y=a+(b*(c-f/g)+k)#
(1)(/,f,g,t1)
(2)(-,c,t1,t2)
(3)(*,b,t2,t3)
(4)(+,t3,k,t4)
(5)(+,a,t4,t5)
(6)(=,t5,_,y)
OK!
```

可见，该程序可以正确进行从算术表达式到四元组的输出，并完成赋值语句。该表达式输出的四元组和上面构造的分析表的四元组输出是一致的，进一步检验了输出结果正确。

对于算术表达式的语义分析，实验中涉及到三种错误类型，下面对它们分别测试：

1. 输入没有以 `#` 结尾： `a+(b*(c-f/g)+k)`

```
PS D:\Study\大三下\编译原理\HW\19335206-韦媛馨-实验3> ./a
please input your expression:  a+(b*(c-f/g)+k)
Error: The input does not end with '#'
```

2. 括号不匹配： `a+(b*(c-f/g+k)#`

```
PS D:\Study\大三下\编译原理\HW\19335206-韦媛馨-实验3> ./a
please input your expression:  a+(b*(c-f/g+k)#
Error: bracket mismatch
```

3. 无法识别的输入字符： `a+(b*(%-f/g)+k)#`

```
PS D:\Study\大三下\编译原理\HW\19335206-韦媛馨-实验3> ./a
please input your expression:  a+(b*(%-f/g)+k)#
Error: unrecognized input symbol %
```

可见，该程序可以正确识别三种不同的错误类型并进行相关提示输出。

以下是对于其他算术表达式的翻译结果展示：

1. `a*((b-c)+f/(g+h))#`

```
PS D:\Study\大三下\编译原理\HW\19335206-韦媛馨-实验3> ./a
please input your expression:  a*((b-c)+f/(g+h))#
(1)(-,b,c,t1)
(2)(+,g,h,t2)
(3)(/,f,t2,t3)
(4)(+,t1,t3,t4)
(5)(*,a,t4,t5)
OK!
```

2. `x=v*u+((i+p)/(k-z)-b*u)/g#`

```
PS D:\Study\大三下\编译原理\HW\19335206-韦媛馨-实验3> ./a
please input your expression:  x=v*u+((i+p)/(k-z)-b*u)/g#
(1)(*,v,u,t1)
(2)(+,i,p,t2)
(3)(-,k,z,t3)
(4)(/,t2,t3,t4)
(5)(*,b,u,t5)
(6)(-,t4,t5,t6)
(7)(/,t6,g,t7)
(8)(+,t1,t7,t8)
(9)(=,t8,_,x)
OK!
```

3. `a*(b/c)#`

```
PS D:\Study\大三下\编译原理\HW\19335206-韦媛馨-实验3> ./a
please input your expression:  a*(b/c)#
(1)(/,b,c,t1)
(2)(*,a,t1,t2)
OK!
```

由上述运行结果展示可知，该程序可以支持算术表达式的语义分析，可以正确分析运算符号的优先级并将其翻译为四元式格式的中间语言，对于表达式的错误也可以正确识别出错误类型并输出错误提示。

9 实验感想

通过本次实验，我通过递归下降子程序的方法，完成了将一个算术表达式翻译成四元式的语义分析程序，进一步掌握了语义分析实现翻译方案的思路和方法，对理论知识的学习和掌握更加深入。这次实验比起之前的实验有了一些改进，即对于错误处理的方法更加具体，对于不同类型的错误，不只是单纯的提示`err`，而是根据错误类型进行报错输入，对于用户来说这样可以更有针对性地知道输入表达式出错在何处。

之后可以进一步实验的方向主要是两个：第一个是将此次语义分析和之前的词法分析、语法分析结合起来，实现变量、整型数、浮点型熟的基本运算；第二个是扩展语义分析的功能，此次实验实现了基本的+,-,*,/,(),=，在以后可以加上循环语句、跳转语句等的语义分析功能，不断完善编译器功能。