源代码见下长图，更改部分为标红部分（图中也以标出）：

1.

void trans(char\* exp, char postexp[]) //将算术表达式exp转换成后缀表达式postexp

{

char e;

SqStack\* Optr; //定义运算符栈

InitStack(Optr); //初始化运算符栈

int i = 0; //i作为postexp的下标

while (\*exp != '\0') //exp表达式未扫描完时循环

{

switch (\*exp)

{

case '(': //判定为左括号

Push(Optr, '('); //左括号进栈

exp++; //继续扫描其他字符

break;

case ')': //判定为右括号

Pop(Optr, e); //出栈元素e

while (e != '(') //不为'('时循环

{

postexp[i++] = e; //将e存放到postexp中

Pop(Optr, e); //继续出栈元素e

}

exp++; //继续扫描其他字符

break;

case '+': //判定为加或减号

case '-':

while (!StackEmpty(Optr)) //栈不空循环

{

GetTop(Optr, e); //取栈顶元素e

if (e != '(') //e不是'('

{

postexp[i++] = e; //将e存放到postexp中

Pop(Optr, e); //出栈元素e

}

else //e是'(时退出循环

break;

}

Push(Optr, \*exp); //将'+'或'-'进栈

exp++; //继续扫描其他字符

break;

case '\*': //判定为'\*'或'/'号

case '/':

while (!StackEmpty(Optr)) //栈不空循环

{

GetTop(Optr, e); //取栈顶元素e

if (e == '\*' || e == '/') //将栈顶'\*'或'/'运算符出栈并存放到postexp中

{

postexp[i++] = e; //将e存放到postexp中

Pop(Optr, e); //出栈元素e

}

else //e为非'\*'或'/'运算符时退出循环

break;

}

Push(Optr, \*exp); //将'\*'或'/'进栈

exp++; //继续扫描其他字符

break;

case '^':

while (!StackEmpty(Optr)) //栈不空循环

{

GetTop(Optr, e); //取栈顶元素e

if (e == '^') //将栈顶'^'运算符出栈并存放到postexp中

{

postexp[i++] = e; //将e存放到postexp中

Pop(Optr, e); //出栈元素e 对乘方符号的判定

}

else //e为非'^'运算符时退出循环

break;

}

Push(Optr, \*exp); //将'^'进栈

exp++; //继续扫描其他字符

break;

default: //处理数字字符

while (\*exp >= '0' && \*exp <= '9') //判定为数字

{

postexp[i++] = \*exp;

exp++;

}

postexp[i++] = '#'; //用#标识一个数值串结束

}

}

while (!StackEmpty(Optr)) //此时exp扫描完毕,栈不空时循环

{

Pop(Optr, e); //出栈元素e

postexp[i++] = e; //将e存放到postexp中

}

postexp[i] = '\0'; //给postexp表达式添加结束标识

DestroyStack(Optr); //销毁栈

}

2.

double compvalue(char\* postexp) //计算后缀表达式的值

{

double d, a, b, c, e;

SqStack1\* Opnd; //定义操作数栈

InitStack1(Opnd); //初始化操作数栈

while (\*postexp != '\0') //postexp字符串未扫描完时循环

{

switch (\*postexp)

{

case '+': //判定为'+'号

Pop1(Opnd, a); //出栈元素a

Pop1(Opnd, b); //出栈元素b

c = b + a; //计算c

Push1(Opnd, c); //将计算结果c进栈

break;

case '-': //判定为'-'号

Pop1(Opnd, a); //出栈元素a

Pop1(Opnd, b); //出栈元素b

c = b - a; //计算c

Push1(Opnd, c); //将计算结果c进栈

break;

case '\*': //判定为'\*'号

Pop1(Opnd, a); //出栈元素a

Pop1(Opnd, b); //出栈元素b

c = b \* a; //计算c

Push1(Opnd, c); //将计算结果c进栈

break;

case '/': //判定为'/'号

Pop1(Opnd, a); //出栈元素a

Pop1(Opnd, b); //出栈元素b

if (a != 0)

{

c = b / a; //计算c

Push1(Opnd, c); //将计算结果c进栈

break;

}

else

{

printf("\n\t除零错误!\n");

exit(0); //异常退出

}

break;

case '^': //判定为'\*'号

Pop1(Opnd, a); //出栈元素a

Pop1(Opnd, b); //出栈元素b 对乘方运算的支持

c = (int)pow(double(b), double(a)); //计算c

Push1(Opnd, c); //将计算结果c进栈

break;

default: //处理数字字符

d = 0; //将连续的数字字符转换成对应的数值存放到d中

while (\*postexp >= '0' && \*postexp <= '9') //判定为数字字符

{

d = 10 \* d + \*postexp - '0';

postexp++;

}

Push1(Opnd, d); //将数值d进栈

break;

}

postexp++; //继续处理其他字符

}

GetTop1(Opnd, e); //取栈顶元素e

DestroyStack1(Opnd); //销毁栈

return e; //返回e

}

