HW3 林宸昊 PB20000034

3.19

```
int check1(ElemType c)
{
    if (c == ')') return 0;
    else return 1;
}
int check2(ElemType c)
{
    if (c == ']') return 0;
    return 1;
}
int check3(ElemType c)
{
    if (c == '}') return 0;
    return 1;
}
void check(SqList L){
    int flag = 0;
    for(int i = L.Length - 1;i >= 0;i--){
        switch(L.elem[i]){
            case ')':
        case ']':
        case '}': push(L.elem[i]); //顺序表中后半部分的括号入栈
        break;
                //接下来对前半部分括号进行对应检验
                //每遇到一个对应括号则出栈，若与当前栈顶括号不对应则必出错
            case '(': if(check1(pop())) flag = 1;
        break;
            case '[': if(check2(pop())) flag = 1;
        break;
        case '{': if(check3(pop())) flag = 1;
        break;
        }
        if(flag) break;
    }
    if(flag) printf("false\n");
    else printf("right\n");
}
```

3.21

```
//原表达式已储存在顺序表中
void trans(SqList L, SqStack s)
{
    char *ans;//用于存放最终结果
    ans = (char *)malloc(100 * sizeof(char));
```

```
    int j = 0;
    for(int i = 0;i < L.Length;i++){
    if(L.elem[i] == '*'||L.elem[i]  == '/'){
            //如果当前栈为空或栈顶运算符优先级小于即将入栈的运算符，则直接入栈
        if(s.stackEmpty() || s.getTop() == '+' || s.getTop() == '-')
s.push(a[i]);
        else
        while(s.getTop() != '+' && s.getTop() != '-'){
                    //出栈，直至满足入栈条件
                ans[j++] = s.pop();
                if(s.stackEmpty()){
                s.push(a[i]);
                break;
            }
        }
    }
    else if(L.elem[i] == '+' || L.elem[i] == '-'){
            //加减优先级最低，直接全部出栈
        while(!s.stackEmpty()){
        b[j++] = pop();
        }
        push(a[i]);
    }
        //若为操作数直接输出
    else b[j++] = a[i];
    }
    while(!stackEmpty()){
    b[j++] = pop();
    }
    puts(b);
}
```

3.31

```
int check(SqList L,SqStack s){
    char c;
    char *ans = (char *)malloc(100 * sizeof(char));
    int i = 0;
    scanf("%c", &c);
    while(c != '@){
        //正序输入
        L.ListInsert(L, L.Length, c);
        s.push(c);
        scanf("%c", &c);
    }
    while(!s.StackEmpty()){
        //逆序输出
        ans[i++] = s.pop();
    }
    //返回比较结果，为0代表输入为回文
```

```
    return strcmp(ans, L.elem);
}
```

3.32

```
Status EnQueue(SqQueue Q, int e, int k){
    Q.base[Q.rear] = e;
    Q.rear = (Q.rear + 1) % k;
    return OK;
}
int sum(SqQueue Q, int k){
    int sum = 0;
    for(int i = 0;i < k;i++){
        sum += Q.base[i];
    }
    return sum;
}
void Fibonacci(SqQueue Q, int max, int k){
    InitQueue(Q, k);//创造一个容量为k的循环队列
    //初始化队列
    for(int i = 1;i <= k-1;i++){
        EnQueue(Q, 0, k);
    }
    EnQueue(Q, 1, k);
    while(sum(Q, k) <= max){
        //未满足条件时不断更新队列
        EnQueue(Q, sum(Q, k), k);
    }
}
```