数据结构第三次作业

学号：2022212408 姓名：胡宇杭

2023 年 10 月 19 日

1 选择题

1. 向一个栈顶指针为 **top** 的链栈中插入一个 **p** 所指结点时，其操作步骤为（**C**）

   A. top->next = p;

   B. p->next = top->next; top->next = p;

   C. p->next = top; top = p;

   D. p->next = top; top = top->next;

2. 一个栈的入栈序列是 **a, b, c, d, e**，则栈的不可能的输出序列是（**C**）

   A. e, d, c, b, a

   B. d, e, c, b, a

   C. d, c, e, a, b

   D. a, b, c, d, e

3. 设栈 **S** 和队列 **Q** 的初始状态均为空，元素 **a、b、c、d、e、f、g** 依次进入栈 **S**。若每个元素出栈后立即进入队列 **Q**，且 **7** 个元素出队的顺序是 **b、d、c、f、e、a、g**，则栈 **S** 的容量至少是（**C**）

   A. 1

B. 2

C. 3

D. 4

4. 若用大小为 **6** 的数组来实现循环队列，且当前 **front** 和 **rear** 的值分别为 **0** 和 **4**。当从队列中删除两个元素，再加入两个元素后，**front** 和 **rear** 的值分别为多少（ **A** ）

   A. 2 和 0

   B. 2 和 2

   C. 2 和 4

   D. 2 和 6

5. 在一个链队列中，若 **f, r** 分别为队首、队尾指针，则插入 **p** 所指结点的操作为（ **C** ）

   A. f->next = p; f = p;

   B. r->next = p; r = p;

   C. p->next = r; r = p;

   D. p->next = f; f = p;

6. 用不带头结点的单链表存储队列时，在进行删除运算时（ **D** ）

   A. 仅修改头指针

   B. 仅修改尾指针

   C. 头尾指针都要修改

   D. 头、尾指针可能都要修改

7. 关于递归概念描述不正确的是（ **B** ）

   A. 递归算法是直接或间接调用自身的算法

B. 所有递归函数都能用非递归的方式定义。

C. **Fibonacci** 数列可用递归定义出来。

D. 递归算法容易定义，结构清晰，但运行效率较低，一般地，其所耗费的计算时间和存储空间都要比非递归算法多

8. 设串□为 **n**，模式串□为 **m**，则 **KMP** 算法所需的附加空间为（ **C** ）

A. **O(n)**

B. **O(m + n)**

C. **O(m)**

D. **O(m * n)**

2 简答题

1. 简述以下算法的功能

```
1  Status a1go1(Stack S)
2  {
3      int i, n, A[255];
4      n = 0;
5      while (!StackEmpty(S)) {n++; Pop(S, A[n]);};
6      for (i = 1; i <= n; i ++ ) Push(S, A[i]);
7  }
```

第 **5** 行执行的代码是将栈中所有的元素取出并按取出的顺序存放在顺序表中

第 **6** 行执行的代码是将存放在顺序表中的元素从表头元素开始入栈

结合前两点，该算法实现的功能是将栈中元素反转

2. 简述以下算法的功能

```
1  Status algo2(Stack S, int e)
2  {
3      Stack T; int d;
4      InitStack(T);
5
6      while (!StackEmpty(S))
7      {
8          Pop(S, d);
9          if (d != e) Push(T, d);
10     }
11
12     while (!StackEmpty(T))
13     {
14         Pop(T, d);
15         Push(S, d);
16     }
17 }
```

在第一个 **while** 循环中，从栈 **S** 中取出值不为元素为 **e** 的元素并存放到另一个栈，直至栈空

在第二个 **while** 循环中，将栈 **T** 中的元素取出并放回栈 **S** 中

比较栈 **S** 前后的状态，其他元素相对顺序不变，但值为 **e** 的元素被取出了，所以算法的功能是删除栈中所有值为 **e** 的元素

3. 简述以下算法的功能

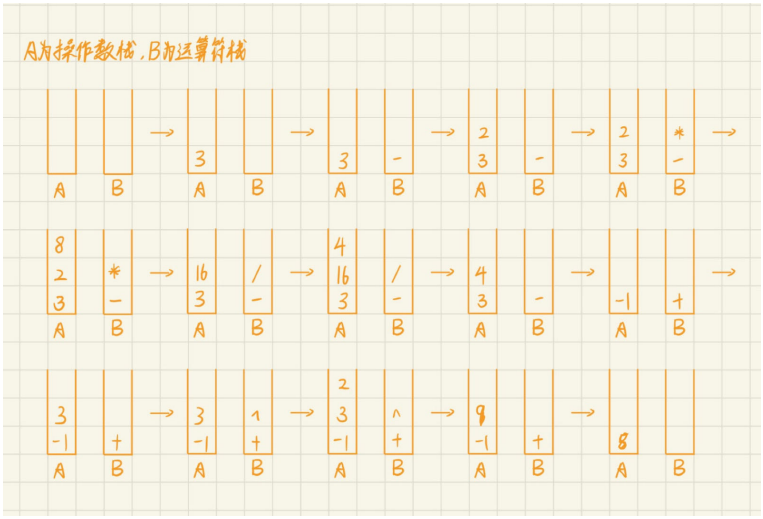
```
1  void algo3(Queue &Q)
2  {
3      Stack S; int d;
4      InitStack(S);
5
6      while (!QeueuEmpty(Q))
7      {
8          DeQueue(Q, d);
9          Push(S, d);
10     }
11
12     while (!StackEmpty(S))
13     {
14         Pop(S, d);
15         EnQueue(Q, d);
16     }
17 }
```

第一个 **while** 循环中，将队列 **Q** 的所有元素取出，并压入栈中

第二个 **while** 循环中，将栈中元素全部取出并入队

结合前两点，该算法实现的功能是将队列中的元素反转

4. 画出对表达式：$3-2*8/4+3^2$ 求值时，操作数栈和运算符栈的变化过程

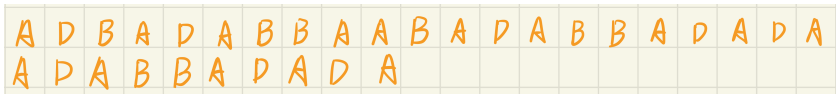


5. 已知主串 **s = 'ADBADABBAABADABBADADA'**，模式串 **pat = 'ADABBADADA'**，写出模式串的 **nextval** 函数值，并由此画出 **KMP** 算法匹配的过程

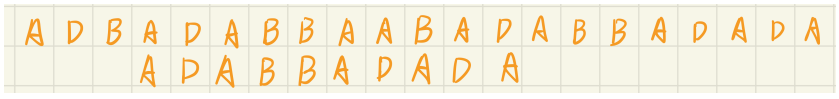
   nextval = {0, 1, 0, 2, 1, 0, 1, 0, 4, 0}

   默认字符串下标从 **1** 开始

   - 第一次比较发现在 **j = 3** 时不匹配，令 **j = nextval[j] = 0**;

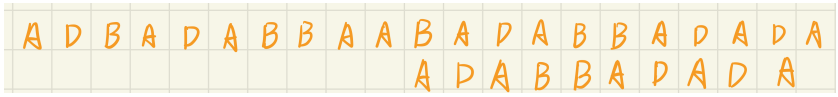


   - 第二次比较发现在 **j = 7** 时不匹配，令 **j = nextval[j] = 1**;

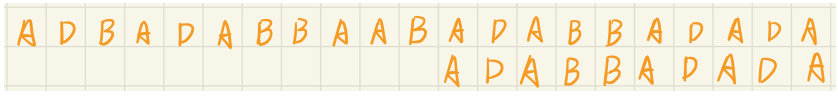


- 第三次比较发现在 $\mathbf{j=2}$ 时不匹配，令 $\mathbf{j=nextval[j]=1}$;

- 第四次比较发现在 $\mathbf{j=1}$ 时不匹配，令 $\mathbf{j=nextval[j]=0}$;



- 第五次比较模式串与主串匹配，程序结束



3 算法设计题

1. 假设以顺序存储结构实现一个双向栈，即在一维数组的存储空间中存在着两个栈，它们的栈底分别设在数组的两个端点。试编写实现这个双向栈 **tws** 的三个操作：

   A. 初始化栈 **InitStack(&tws)**;

```
Status InitStack(SqStack &tws)
{
    tws.base = (ElemType *)malloc(M * sizeof(ElemType));
    if (!tws.base) exit("OVERFLOW");

    tws.top1 = tws.base;
    tws.top2 = tws.base + M - 1;
    tws.stacksize = M;

    return OK;
}
```

   B. 入栈 **push(&tws, i, e)**;

```
Status push(SqStack &tws, int i, ElemType e)
{
    if (tws.top1 > tws.top2)
    {
        printf("StackFull\n");
        return ERROR;
    }

    if (i == 0) *tws.top1 ++ = e;
    else if (i == 1) *tws.top2 -- = e;
    else
    {
        printf("Unkown Opeator\n");
        return ERROR;
    }

    return OK;
}
```

C. 出栈 **pop(&tws, i, &e)**，其中 **i** 为 **0** 或 **1**，分别指示设在数组两端的两个栈

```
Status pop(SqStack &tws, int i, ElemType &e)
{
    if (i == 0)
    {
        if (tws.top1 == tws.base)
        {
            printf("Stack1 Underflow\n");
            return ERROR;
        }

        e = * -- wts.top1;
    }
    else if (i == 1)
    {
        if (tws.top2 == tws.base + tws.stacksize - 1)
        {
            printf("Stack2 Underflow\n");
            return ERROR;
        }

        e = * ++ wts.top2;
    }

    return OK;
}
```

2. 假设以带头结点的循环链表表示队列，并且只设一个指针指向队尾元素结点 (注意不设头指针)，试编写相应的队列初始化、入队列和出队列的算法

A. 循环链队初始化

初始时队列里没有元素，所以尾指针直接指向头结点

```
Status InitQueue(LinkQueue &Q)
{
    Q.rear = (QueuePtr)malloc(sizeof(Qnode));
```

```
    if (!Q.rear) exit("OVERFLOW");

    Q.rear->next = Q.rear;

    return OK;
}
```

B. 入队

入队时将新的元素的 **next** 指向头结点，然后将尾指针 (原队尾元素) 的 **next** 指向新元素，最后将尾指针指向新元素

```
Status EnQueue(LinkQueue &Q, ElemType e)
{
    QueuePtr p = (QueuePtr)malloc(sizeof(Qnode));
    if (!p) exit("OVERFLOW");

    p->data = e;
    p->next = Q.rear->next;
    Q.rear->next = p;
    Q.rear = p;

    return OK;
}
```

C. 出队

出队时先判断队列是否为空。如果为空直接返回；如果不为空，则将头结点后的结点 (队头元素) 删除。同时如果队列里只有一个结点，删除后队列为空，此时应将尾指针指向头指针

```
Status DeQueue(LinkQueue &Q, ElemType &e)
{
    if (Q.rear->next = Q.rear)
    {
        printf("Queue Underflow\n");
        return ERROR;
```

```
    }

    QueuePtr headNode = Q.rear->next;
    QueuePtr headElem = headNode->next;
    e = headElem->data;

    if (headElem == Q.rear)
    {
        Q.rear = headNode;
    }

    headNode->next = headElem->next;
    free(headElem);

    return OK;
}
```

3. 回文是前后两个方向拼写完全相同的字符串。很显然，空字符串是回文，任何一个只有 1 个字符的字符串是回文。编写一个函数 **testPalindrome**，判断一个字符串是否是回文

> 因为这两节课学的栈,所以这里用栈实现回文串的判断 (~~感觉不如双指针~~)
>
> 具体实现方法如下：
>
> 将字符串的前半段压入栈中，然后依次判断字符串的后半段和栈顶元素是否一致。如果不一致，则说明该字符串不是回文串；如果一致，则出栈，继续下一次判断
>
> 如果字符串的长度为奇数，不难发现正中间元素不会影响其是否为一个回文串，因此可以省略

```
bool testPalindrome(char *s)
{
    int len = strlen(s);
    if (len == 0 || len == 1) return true;

    int stack[N], top = -1;
    for (int i = 0; i < len / 2; i ++ ) stack[ ++ top] = s[i];
```

```
8
9       for (int i = len / 2 + 1; i < len; i ++ )
10          if (s[top -- ] != s[i])
11              return false;
12
13      return true;
14  }
```