# 实验-2-栈和队列-参考程序

王 新 宇
计算机科学与技术系

```
//实验题第1题
//SqStack.h
#include "Status.h"
const int DEFAULT_SIZE = 100;
template<class DataType>
class SqStack
{
    DataType *elems;                          //元素存储空间
    int maxSize;                              //栈的容量
    int top1, top2;                           //栈顶指针
public:
    SqStack(int size = DEFAULT_SIZE);         //构造函数
    virtual ~SqStack();                       //析构函数
    int GetLength(int no) const;              //求栈的长度
    bool IsEmpty(int no) const;               //判断栈是否为空
    void Clear(int no);                       //清空栈
    Status Push(int no, const DataType &e);   //入栈
    Status Pop(int no, DataType &e);          //出栈
    Status Top(int no, DataType &e) const;    //取栈顶元素
    void Traverse(int no, void(*visit)(const DataType &)) const;  //遍历栈
};
```

```
//构造一个容量为size的空栈
template<class DataType>
SqStack<DataType>::SqStack(int size) {
    elems = new DataType[size];
    maxSize = size;
    top1 = -1;
    top2 = maxSize;
}

//销毁栈
template<class DataType>
SqStack<DataType>::~SqStack() {
    delete[]elems;
}
```

```
//返回栈中元素个数
template <class DataType>
int SqStack<DataType>::GetLength(int no) const {
    switch (no)
    {
        case 1:   return top1 + 1;
        case 2:   return maxSize - top2;
    }
}

//栈为空返回true，否则返回false
template<class DataType>
bool SqStack<DataType>::IsEmpty(int no) const {
    switch (no)
    {
        case 1:   return top1 == -1 ? true : false;
        case 2:   return top2 == maxSize ? true : false;
    }
}
```

```
template<class DataType>
void SqStack<DataType>::Clear(int no)          //将栈置为空栈
{
      switch (no)
      {
            case 1:    top1 = -1;    break;
            case 2:    top2 = maxSize;
      }
}

template<class DataType>
Status SqStack<DataType>::Push(int no, const DataType &e)      //入栈
{
      if (top1 + 1 == top2)     // 栈已满
            return OVER_FLOW;
      else  {
            switch (no)
            {
                  case 1:    elems[++top1] = e;    break;
                  case 2:    elems[--top2] = e;    break;
            }
            return SUCCESS; // 操作成功
      }
}
```

```
template<class DataType>
Status SqStack<DataType>::Pop(int no, DataType &e)  //出栈
{
    if (IsEmpty(no))          // 栈空
        return UNDER_FLOW;
    else {
        switch (no)
        {
            case 1:   e = elems[top1--];   break;
            case 2:   e = elems[top2++];   break;
        }
        return SUCCESS;          // 操作成功
    }
}
```

```
//取栈顶元素
template<class DataType>
Status SqStack<DataType>::Top(int no, DataType &e) const
{
    if (IsEmpty(no))              // 栈空
        return UNDER_FLOW;
    else {
        switch (no)
        {
            case 1:   e = elems[top1];   break;
            case 2:   e = elems[top2];   break;
        }
        return SUCCESS; // 栈非空,操作成功
    }
}
```

```
//遍历栈中的每个元素
template <class DataType>
void SqStack<DataType>::Traverse(int no, void(*visit)(const DataType &)) const
{
    int i;
    switch (no)
    {
        case 1:
            for (i = top1; i >= 0; i--)
                (*visit)(elems[i]);
            break;
        case 2:
            for (i = top2; i <= maxSize - 1; i++)
                (*visit)(elems[i]);
            break;
    }
}
```

```
//实验题第2题
//Queue.h
template<class DataType>
class Queue
{
protected:
    SqStack<DataType> s1,s2;
public:
    Queue()  {}
    virtual ~Queue()  {}
    Status EnQueue(DataType x);
    Status DeQueue(DataType &x);
    bool IsEmpty();
};
```

```
template<class DataType>
Status Queue<DataType>::EnQueue(DataType x)
{
        DataType data;
        if(s1.IsFull()&& !s2.IsEmpty())
                return OVER_FLOW;
        if(s1.IsFull() && s2.IsEmpty())
                while(!s1.IsEmpty())
                {    s1.Pop(data);    s2.Push(data);    }
        s1.Push(x);
        return SUCCESS;
}
```

```
template<class DataType>
Status Queue<DataType>::DeQueue(DataType &x)
{
    if(!s2.IsEmpty())    s2.Pop(x);
    else  {
        if(s1.IsEmpty())     return UNDER_FLOW;
        else  {
            while(!s1.IsEmpty())
            {   s1.Pop(x);    s2.Push(x);    }
            s2.Pop(x);
        }
    }
    return SUCCESS;
}
```

```
template<class DataType>
bool Queue<DataType>::IsEmpty()
{
        if(s1.IsEmpty() && s2.IsEmpty())
                return true;
        else
                return false;
}
```

```
//实验题第3题
//SqQueue.h
#include "Status.h"
const int DEFAULT_SIZE = 100;
template<class DataType>
class SqQueue
{
    DataType *elems;                    //元素存储空间
    int maxSize;                        //队列的容量
    int front;                          //队头指针
    int length;                         //队列长度
public:
    SqQueue(int size = DEFAULT_SIZE);   //构造函数
    virtual ~SqQueue();                 //析构函数
    int GetLength() const;              //求队列的长度
    bool IsEmpty() const;               //判断队列是否为空
    void Clear();                       //清空队列
    Status EnQueue(const DataType &e);  //入队
    Status DelQueue(DataType &e);       //出队
    Status GetHead(DataType &e) const;  //取队头元素
    void Traverse(void(*visit)(const DataType &)) const;//遍历队列
};
```

```
template<class DataType>
SqQueue<DataType>::SqQueue(int size)          //构造一个容量为size的空队列
{
    elems = new DataType[size];
    maxSize = size;
    front = 0;                                //队头指针初始化为0
    length = 0;
}
template <class DataType>
SqQueue<DataType>::~SqQueue()                 //销毁一个队列
{
    delete []elems;
}
template<class DataType>
int SqQueue<DataType>::GetLength() const      //求队列的长度
{
    return length;
}
```

```
template<class DataType>
bool SqQueue<DataType>::IsEmpty() const     //队列空返回true，否则返回false
{
    return length == 0;
}
template<class DataType>
void SqQueue<DataType>::Clear()             //清空队列
{
    front = 0;    length = 0;
}
template<class DataType>
Status SqQueue<DataType>::EnQueue(const DataType &e)        //入队
{
    if (length == maxSize)    return OVER_FLOW;
    else {
        elems[(front + length) % maxSize] = e;
        length++;
        return SUCCESS;
    }
}
```

```
template<class DataType>
Status SqQueue<DataType>::DelQueue(DataType &e)  //出队
{
    if (!IsEmpty()) {
        e = elems[front];
        front = (front + 1) % maxSize;
        length--;
        return SUCCESS;
    }
    else    return UNDER_FLOW;
}
template<class DataType>
Status SqQueue<DataType>::GetHead(DataType &e) const        //取队头元素
{
    if(IsEmpty())    return UNDER_FLOW;
    else {
        e = elems[front];
        return SUCCESS;
    }
}
```

```
//遍历队列
template<class DataType>
void SqQueue<DataType>::Traverse(void (*visit)(const DataType &)) const {
    int rear = (front + length) % maxSize;
    for (int i = front; i != rear; i = (i + 1) % maxSize)
        (*visit)(elems[i]);
}
```

```
//实验题第4题
void YangHui(int n)
{
    LinkQueue<int> queue;
    int s,t;
    if(n <= 0)    return;
    queue.EnQueue(1);    queue.EnQueue(1);    //预先放入第一行的两个系数
    for(int i=1; i<=n; i++)  {
        for(int j=1; j<=n-i; j++)    cout << ' '; //输出每一行开始时的空格
        s = 0;
        for(int j=1; j<=i+1; j++)  {
            queue.DelQueue(t);              //取队头元素
            cout << t << ' ';               //输出队头
            queue.EnQueue(s+t);             //入队下一行的元素
            s = t;
        }
        queue.EnQueue(1);                   //每行放入最后的1
        cout << endl;
    }
}
```