P184 12

(1)

```
template <class T>
inline void linkedStack<T>::pushNode(chainNode<T> *theNode)
{
    theNode->next=stackTop;
    stackTop=theNode;
    stackSize++;
}

template <class T>
inline T linkedStack<T>::popNode()
{
    if(empty())
        throw stackEmpty();
    chainNode<T>* deleteNode=stackTop;
    T value=deleteNode->element;
    stackTop=stackTop->next;
    stackSize--;
    delete deleteNode;
    return value;
}
```

Linkedstack 类实现

```
#include"stack.h"
#include"chainNode.h"
#include"myExceptions.h"
template<class T>
class linkedStack:public stack<T>
{
public:
    linkedStack(int theCapacity=10)
    {
        stackTop=NULL;
        stackSize=0;
    }
    ~linkedStack();
    bool empty()const
    {
        return stackSize==0;
    }
    int size()const
    {
        return stackSize;
```

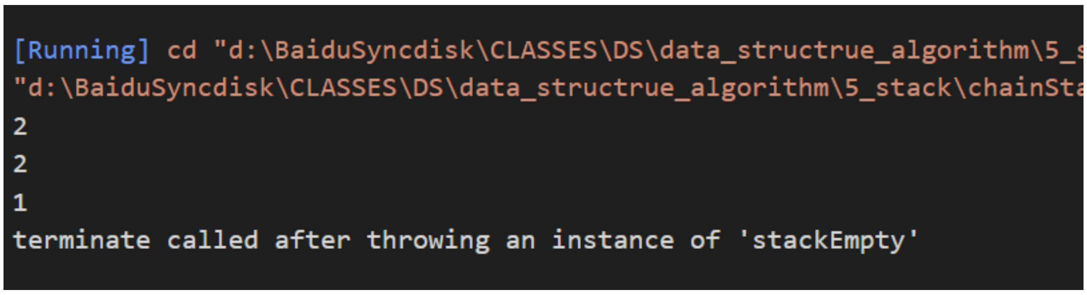
```cpp
    }
    T top()
    {
        if(stackSize==0)
            throw stackEmpty();
        return stackTop->element;
    }
    void pop();
    void push(const T& theElement)
    {
        stackTop=new chainNode<T>(theElement,stackTop);
        stackSize++;
    }

    void pushNode(chainNode<T>* theNode);
    T popNode();

private:
    int stackSize;
    chainNode<T>* stackTop;
};

template <class T>
inline linkedStack<T>::~linkedStack()
{
   while(stackTop!=NULL)
    {
        chainNode<T>* nextNode=stackTop->next;
        delete stackTop;
        stackTop=nextNode;
    }
}


template<class T>
inline void linkedStack<T>::pop()
{
    if(stackSize==0)
        throw stackEmpty();

    chainNode<T>* nextNode=stackTop->next;
    delete stackTop;
    stackTop=nextNode;
    stackSize--;
```

```
}

template <class T>
inline void linkedStack<T>::pushNode(chainNode<T> *theNode)
{
    theNode->next=stackTop;
    stackTop=theNode;
    stackSize++;
}

template <class T>
inline T linkedStack<T>::popNode()
{
    if(empty())
        throw stackEmpty();
    chainNode<T>* deleteNode=stackTop;
    T value=deleteNode->element;
    stackTop=stackTop->next;
    stackSize--;
    delete deleteNode;
    return value;
}
```

(2)测试

```
#include"linkedStack.hpp"
int main()
{
    linkedStack<int>s;
    chainNode<int>*a=new chainNode<int>(1);
    chainNode<int>*b=new chainNode<int>(2);
    s.pushNode(a);s.pushNode(b);
    cout<<s.top()<<endl;//2
    cout<<s.popNode()<<endl;//2
    cout<<s.popNode()<<endl;//1
    cout<<s.top();
    return 0;
}
```

输出



```
[Running] cd "d:\BaiduSyncdisk\CLASSES\DS\data_structrue_algorithm\5_s
"d:\BaiduSyncdisk\CLASSES\DS\data_structrue_algorithm\5_stack\chainSta
2
2
1
terminate called after throwing an instance of 'stackEmpty'
```

(3)大规模数据下,push()函数可能会进行内存扩展,从而产生 O(n)的时间复杂度,造成性能的开销,而 pop(),pushNode(),popNode()函数时间复杂度均为 O(1),无明显差异,

P211 练习 5
增加 ADT 函数实现

```
template <class T>
inline void ExtendedArrayQueue<T>::inputQueue()
{
    T element;
    cin>>element;
    push(element);
}

template <class T>
inline void ExtendedArrayQueue<T>::outputQueue()
{
    for(int
queueCurrent=(queueFront+1)%arrayLength;queueCurrent!=(queueBack+1)%arr
ayLength;queueCurrent=(queueCurrent+1)%arrayLength)
    {
        cout<<queue[queueCurrent]<<" ";
    }
    cout<<endl;
}

template <class T>
inline void ExtendedArrayQueue<T>::devideQueue()
{
    ExtendedArrayQueue<T>p1(10),p2(10);
    bool flag=true;
    for(int
queueCurrent=(queueFront+1)%arrayLength;queueCurrent!=(queueBack+1)%arr
ayLength;queueCurrent=(queueCurrent+1)%arrayLength)
    {
        int t=queue[queueCurrent];
        if (flag)
        {
            p1.push(t);
            flag=false;
        }
        else
        {
            p2.push(t);
            flag=true;
        }
    }
    cout<<"divide result"<<endl;
    cout<<"p1: "<<endl;
    p1.outputQueue();
    cout<<"p2: "<<endl;
    p2.outputQueue();
}

template <class T>
inline void ExtendedArrayQueue<T>::mergeQueue()
{
    clear();

    ExtendedArrayQueue<T>p1(10),p2(10);
    cout<<"enter numbers of values of p1 and p2"<<endl;
    int n,m;cin>>n>>m;
    cout<<"enter values of p1 and p2"<<endl;
    while(n--)p1.inputQueue();
    while(m--)p2.inputQueue();

    bool flag=true;
    while(!p1.empty()&&!p2.empty())
    {
        T value1=p1.front();
        T value2=p2.front();
        if(flag)
        {
            push(value1);
            p1.pop();
            flag=false;
        }
        else
        {
            push(value2);
            p2.pop();
            flag=true;
        }
    }
    while(!p1.empty())
    {
        push(p1.front());
        p1.pop();
    }
    while (!p2.empty())
    {
        push(p2.front());
        p2.pop();
    }
    cout<<"merge result"<<endl;
    outputQueue();
}

template <class T>
inline void ExtendedArrayQueue<T>::clear()
{
    queueFront=queueBack=0;
}
```

ExtendedArrayQueue 类实现

```
template <class T>
class ExtendedArrayQueue :public Queue<T>,public ExtendedQueue<T>
{
    public:
    ExtendedArrayQueue(int theCapacity=10);
    ExtendedArrayQueue(){delete[] queue;}
    bool empty() const{return queueBack==queueFront;}
    bool full() const {return queueFront==(queueBack+1)%arrayLength;}
    int size()const {return (queueBack-
queueFront+arrayLength)%arrayLength;}
    int capacity()const {return arrayLength;}
    T front()
    {
        if(empty())
            throw queueEmpty();
        return queue[(queueFront+1)%arrayLength];
    }
    T back()
    {
        if(empty())
            throw queueEmpty();
        return queue[queueBack];
    }
    void pop()
    {
        if(empty())
            throw queueEmpty();
        queueFront=(queueFront+1)%arrayLength;
        queue[queueFront].~T();
    }
    void push(const T& theElement);
    void doubleQueueLength();
    //新增 ADT
    void inputQueue();
    void outputQueue();
    void devideQueue();
    void mergeQueue();
    void clear();
private:
    int queueFront;
    int queueBack;
    int arrayLength;
    T *queue;
};

template <class T>
inline ExtendedArrayQueue<T>::ExtendedArrayQueue(int theCapacity)
{
    if(theCapacity<1)
        throw illegalParameterValue("Capacity must >0");
    arrayLength=theCapacity;
    queue=new T[theCapacity];
    queueFront=queueBack=0;
}

template <class T>
inline void ExtendedArrayQueue<T>::push(const T &theElement)
{
    //扩容
    if(full())
    {
        doubleQueueLength();
    }
    queueBack=(queueBack+1)%arrayLength;
    queue[queueBack]=theElement;
```

```
}

template <class T>
inline void ExtendedArrayQueue<T>::doubleQueueLength()
{
    T* newQueue=new T[2*arrayLength];

    int start=(queueFront+1)%arrayLength;
    //未形成环形
    if(start<2)
        copy(queue+start,queue+start+arrayLength-1,newQueue);
    //形成环形
    else
    {
        copy(queue+start,queue+arrayLength,newQueue);
        copy(queue,queue+queueBack+1,newQueue+arrayLength-start);
    }

    queueFront=2*arrayLength-1;
    queueBack=arrayLength-2;
    arrayLength*=2;
    delete[]queue;
    queue=newQueue;
}
```

抽象类 Queue

```
#include"queue.h"

template <class T>
class ExtendedQueue:public Queue<T>
{
public:
    virtual void inputQueue() =0;
    virtual void outputQueue()=0;
    virtual void devideQueue()=0;
    virtual void mergeQueue()=0;
};
```

抽象类 ExtendedQueue

```
#include"queue.h"

template <class T>
class ExtendedQueue:public Queue<T>
{
public:
    virtual void inputQueue() =0;
    virtual void outputQueue()=0;
    virtual void devideQueue()=0;
    virtual void mergeQueue()=0;
};
```

输入

5

1 2 3 4 5

7 4

1 3 4 6 9 1 1

2 1 4 8

输出

size: 5 capacity 8

1 2 3 4 5

divide result

p1:

1 3 5

p2:

2 4

enter numbers of values of p1 and p2

enter values of p1 and p2

merge result

1 2 3 1 4 4 6 8 9 1 1