/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

循环队列类CirQueue的构造函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

CirQueue<DataType> :: CirQueue()

{

rear = front = QueueSize - 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

循环队列类CirQueue的析构函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

CirQueue<DataType> :: ~CirQueue()

{

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

循环队列类CirQueue的入队操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

void CirQueue<DataType> :: EnQueue(DataType x)

{

if ((rear + 1) % QueueSize == front)

throw "上溢";

rear = (rear + 1) % QueueSize; //队尾指针在循环意义下加1

data[rear] = x; //在队尾处插入元素

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

循环队列类CirQueue的出队操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

DataType CirQueue<DataType> :: DeQueue( )

{

if (rear == front) throw "下溢";

front = (front + 1) % QueueSize; //队头指针在循环意义下加1

return data[front]; //读取并返回出队前的队头元素

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

循环队列类CirQueue的取队头操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

DataType CirQueue<DataType> :: GetQueue( )

{

if (rear == front) throw "下溢";

return data[(front + 1) % QueueSize]; //注意不修改队头指针

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

循环队列类CirQueue的判空操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

int CirQueue<DataType> :: Empty()

{

if(front == rear)

return 1;

else

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

对应教材3.3.2节,循环队列类CirQueue的使用范例

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

using namespace std;

const int QueueSize = 100; //100是示例性数据，根据需要重新定义

template <typename DataType> //定义模板类CirQueue

class CirQueue

{

public:

CirQueue( ); //构造函数，初始化空队列

~ CirQueue( ); //析构函数

void EnQueue(DataType x); //入队操作，将元素x入队

DataType DeQueue( ); //出队操作，将队头元素出队

DataType GetQueue( ); //取队头元素（并不删除）

int Empty( ); //判断队列是否为空

private:

DataType data[QueueSize]; //存放队列元素的数组

int front, rear; //游标，队头和队尾指针

};

template <typename DataType>

CirQueue<DataType> :: CirQueue()

{

rear = front = QueueSize - 1;

}

template <typename DataType>

CirQueue<DataType> :: ~CirQueue()

{

}

template <typename DataType>

void CirQueue<DataType> :: EnQueue(DataType x)

{

if ((rear + 1) % QueueSize == front)

throw "上溢";

rear = (rear + 1) % QueueSize; //队尾指针在循环意义下加1

data[rear] = x; //在队尾处插入元素

}

template <typename DataType>

DataType CirQueue<DataType> :: DeQueue( )

{

if (rear == front) throw "下溢";

front = (front + 1) % QueueSize; //队头指针在循环意义下加1

return data[front]; //读取并返回出队前的队头元素

}

template <typename DataType>

DataType CirQueue<DataType> :: GetQueue( )

{

if (rear == front) throw "下溢";

return data[(front + 1) % QueueSize]; //注意不修改队头指针

}

template <typename DataType>

int CirQueue<DataType> :: Empty()

{

if(front == rear)

return 1;

else

return 0;

}

int main( )

{

int x;

CirQueue<int> Q{ }; //定义对象变量Q

cout << "对5和8执行入队操作，";

Q.EnQueue(5); Q.EnQueue(8);

cout << "当前队头元素为：" << Q.GetQueue( ) << endl; //输出队头元素5

try

{

x = Q.DeQueue( );

cout << "执行一次出队操作，出队元素是：" << x << endl; //输出出队元素5

}catch(char\* str){

cout << str << endl;

}

try

{

cout <<"请输入入队元素：";

cin >> x;

Q.EnQueue(x);

}catch(char\* str){

cout << str << endl;

}

if (Q.Empty( ) == 1) cout << "队列为空" << endl;

else cout << "队列非空" << endl; //队列有2个元素，输出队列非空

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链队列类LinkQueue的类定义

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

struct Node

{

DataType data; //数据域

Node<DataType> \*next; //指针域

};

template <typename DataType>

class LinkQueue

{

public:

LinkQueue( ); //构造函数，初始化一个空的链队列

~LinkQueue( ); //析构函数，释放链队列各结点的存储空间

void EnQueue(DataType x); //入队操作，将元素x入队

DataType DeQueue( ); //出队操作，将队头元素出队

DataType GetQueue( ); //取链队列的队头元素

int Empty( ); //判断链队列是否为空

private:

Node<DataType> \*front, \*rear; //队头和队尾指针，分别指向头结点和终端结点

};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链队列类LinkQueue的构造函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

LinkQueue<DataType> :: LinkQueue( )

{

Node<DataType> \*s = nullptr;

s = new Node<DataType>; s->next = nullptr; //创建头结点s

front = rear = s; //将队头指针和队尾指针都指向头结点s

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链队列类LinkQueue的析构函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

LinkQueue<DataType> :: ~LinkQueue()

{

Node<DataType> \*q = nullptr;

while (front != nullptr) //释放单链表的每一个结点的存储空间

{

q = front; //暂存被释放结点

front = front->next; // front指向被释放结点的下一个结点

delete q;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链队列类LinkQueue的入队操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

void LinkQueue<DataType> :: EnQueue(DataType x)

{

Node<DataType> \*s = nullptr;

s = new Node<DataType>; //申请结点s

s->data = x; s->next = nullptr;

rear->next = s; rear = s; //将结点s插入到队尾

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链队列类LinkQueue的出队操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

DataType LinkQueue<DataType> :: DeQueue( )

{

DataType x;

Node<DataType> \*p = nullptr;

if (rear == front) throw "下溢";

p = front->next; x = p->data; //暂存队头元素

front->next = p->next; //将队头元素所在结点摘链

if (p->next == nullptr) rear = front; //判断出队前队列长度是否为1

delete p;

return x;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链队列类LinkQueue的取队头操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

DataType LinkQueue<DataType> :: GetQueue()

{

if(front == rear)

throw "下溢异常";

else

return front->next->data;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

链队列类LinkQueue的判空操作

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

template <typename DataType>

int LinkQueue<DataType> :: Empty()

{

if(front == rear)

return 1;

else

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

对应教材3.3.3节,链队列类LinkQueue的使用范例

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

using namespace std;

template <typename DataType>

struct Node

{

DataType data; //数据域

Node<DataType> \*next; //指针域

};

template <typename DataType>

class LinkQueue

{

public:

LinkQueue( ); //构造函数，初始化一个空的链队列

~LinkQueue( ); //析构函数，释放链队列各结点的存储空间

void EnQueue(DataType x); //入队操作，将元素x入队

DataType DeQueue( ); //出队操作，将队头元素出队

DataType GetQueue( ); //取链队列的队头元素

int Empty( ); //判断链队列是否为空

private:

Node<DataType> \*front, \*rear; //队头和队尾指针，分别指向头结点和终端结点

};

template <typename DataType>

LinkQueue<DataType> :: LinkQueue( )

{

Node<DataType> \*s = nullptr;

s = new Node<DataType>; s->next = nullptr; //创建头结点s

front = rear = s; //将队头指针和队尾指针都指向头结点s

}

template <typename DataType>

LinkQueue<DataType> :: ~LinkQueue()

{

Node<DataType> \*q = nullptr;

while (front != nullptr) //释放单链表的每一个结点的存储空间

{

q = front; //暂存被释放结点

front = front->next; // front指向被释放结点的下一个结点

delete q;

}

}

template <typename DataType>

void LinkQueue<DataType> :: EnQueue(DataType x)

{

Node<DataType> \*s = nullptr;

s = new Node<DataType>; //申请结点s

s->data = x; s->next = nullptr;

rear->next = s; rear = s; //将结点s插入到队尾

}

template <typename DataType>

DataType LinkQueue<DataType> :: DeQueue( )

{

DataType x;

Node<DataType> \*p = nullptr;

if (rear == front) throw "下溢";

p = front->next; x = p->data; //暂存队头元素

front->next = p->next; //将队头元素所在结点摘链

if (p->next == nullptr) rear = front; //判断出队前队列长度是否为1

delete p;

return x;

}

template <typename DataType>

DataType LinkQueue<DataType> :: GetQueue()

{

if(front == rear)

throw "下溢异常";

else

return front->next->data;

}

template <typename DataType>

int LinkQueue<DataType> :: Empty()

{

if(front == rear)

return 1;

else

return 0;

}

int main( )

{

int x;

LinkQueue<int> Q{ }; //定义对象变量Q

cout << "对5和8执行入队操作，";

Q.EnQueue(5); Q.EnQueue(8);

cout << "当前队头元素为：" << Q.GetQueue( ) << endl; //输出队头元素5

try

{

x = Q.DeQueue( );

cout << "执行一次出队操作，出队元素是：" << x << endl; //输出出队元素5

} catch(char\* str){

cout << str << endl;

}

try

{

cout << "请输入入队元素：";

cin >> x;

Q.EnQueue(x);

} catch(char\* str){

cout << str << endl;

}

if (Q.Empty( ) == 1) cout << "队列为空" << endl;

else cout << "队列非空" << endl; //队列有2个元素，输出队列非空

return 0;

}