\*\*\*\*\*\*\*\*\*\*基于链表的队列（LQueue）实现\*\*\*\*\*\*\*\*\*\*

1. **link类(link.h)：**

#include<iostream>

template <typename E> class Link {

public:

E element; // Value for this node 节点值

Link \*next; // Pointer to next node in list 指向下一节点的指针

Link(const E& elemval, Link\* nextval =NULL) // Constructors 构造函数

{ element = elemval; next = nextval; }

Link(Link\* nextval =NULL) { next = nextval; }

};

1. **queue抽象类(queue.h)：**

#ifndef QUEUE

#define QUEUE

// Abstract queue class 队列抽象类

template <typename E> class Queue {

private:

void operator =(const Queue&) {}

Queue(const Queue&) {}

public:

Queue() {} // Default 构造函数

virtual ~Queue() {} // Base destructor 析构函数

// Reinitialize the queue.

//The user is responsible for reclaiming the storage used by the queue elements.

// 重新启用队列。用户负责回收使用的队列元素。

virtual void clear() = 0;

// Place an element at the rear of the queue. 在队列的后面添加元素

// it: The element being enqueued. it:要被加入队列的元素

virtual void enqueue(const E&) = 0;

// Remove and return element at the front of the queue. 删除并返回在队列头部的元素

virtual E dequeue() = 0;

// Return: A copy of the front element. 返回队列头部元素（不删除）

virtual const E& frontValue() const = 0;

// Return: The number of elements in the queue. 返回队列中元素个数

virtual int length() const = 0;

};

#endif

1. **LQueue物理实现(LQueue.h)：**

#include "link.h"

#include "queue.h"

#include "assert.h"

// Implementations for linked queue function members链式队列成员函数的实现

// Linked queue implementation 链式队列的实现

template <typename E> class LQueue: public Queue<E> {

private:

Link<E>\* front; // Pointer to front queue node 队列头节点指针

Link<E>\* rear; // Pointer to rear queue node 队尾节点指针

int size; // Number of elements in queue 队列容量

public:

LQueue(int sz = 100) // Constructor 构造函数

{ front = rear = new Link<E>(); size = 0; }

~LQueue() { clear(); delete front; } // Destructor 析构函数

void clear() { // Clear queue 清空队列

while(front->next != NULL) { // Delete each link node 删除所有节点

rear = front;

delete rear;

}

rear = front;

size = 0;

}

void enqueue(const E& it) { // Put element on rear 队尾添加节点

rear->next = new Link<E>(it, NULL);

rear = rear->next;

size++;

}

E dequeue() { // Remove element from front 头部删除节点

assert(size != 0);// "Queue is empty"

E it = front->next->element; // Store dequeued value 存储被删除节点的数据

Link<E>\* ltemp = front->next; // Hold dequeued link 存储被删除的节点

front->next = ltemp->next; // Advance front 重新建立指向关系

if (rear == ltemp) rear = front; // Dequeue last element 若被删元素为最后一个节点

delete ltemp; // Delete link 删除节点

size --;

return it; // Return element value 返回被删节点数据

}

const E& frontValue() const { // Get front element 获取头部元素

assert(size != 0); // "Queue is empty" 队空则断言终止

return front->next->element;

}

virtual int length() const { return size; }

};

1. **主函数测试部分(LQueueTest.cpp)：**

#include "queue.h"

#include "LQueue.h"

#include<iomanip>

#include<iostream>

using namespace std;

//void print(AQueue<int>& AQ,int n);

// Main routine for array-based queue driver class

int main(){

int n;

LQueue<int> Q1(100);

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*链式队列中各个基本操作演示\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl<<endl;

cout<<"向链式队列中依次添加元素12、24、5、90、7、33"<<endl;

//调用链式队列中的enqueue操作

Q1.enqueue(12);

Q1.enqueue(24);

Q1.enqueue(5);

Q1.enqueue(90);

Q1.enqueue(7);

Q1.enqueue(33);

cout<<"添加元素后链式队列长度为：";

cout<<Q1.length()<<endl<<endl;//获取链式队列长度

while(Q1.length()!=0){

cout<<"获得队列中第一个元素："<<Q1.frontValue()<<endl;

//调用链式队列中的dequeue操作

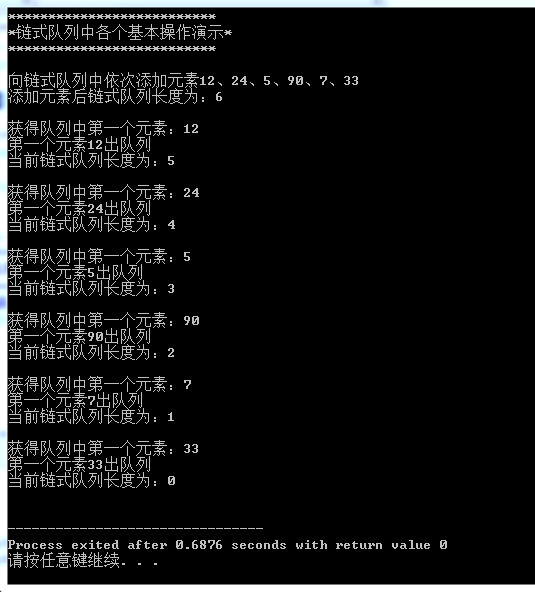
cout<<"第一个元素"<<Q1.dequeue()<<"出队列"<<endl;

cout<<"当前链式队列长度为："<<Q1.length()<<endl<<endl;

}

}

1. **测试结果示例：**



**【附录】**

版本信息声明：

Dev-C++ 5.11

TDM-GCC 4.9.2 64-bit Release

资料整理人：  
物联网工程1402班 宁静仪

物联网工程1402班 吴彦彦