\*\*\*\*\*\*\*\*\*\*基于数组的队列（AQueue）实现\*\*\*\*\*\*\*\*\*\*

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. **AQueue物理实现(AQueue.h)：**

// This is the declaration for Aqueue.

// Array-based queue implementation. 基于数组的顺序队列AQueue

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

private:

int maxSize; // Maximum size of queue 队列最大容量

int front; // Index of front element 头部的索引

int rear; // Index of rear element 尾部索引

E \*listArray; // Array holding queue elements 数组存储队列元素

public:

AQueue(int size =100) { // Constructor 构造函数

// Make list array one position larger for empty slot

//使数组比队列容量多一个位置做为空槽

maxSize = size+1;

rear = 0; front = 1;

listArray = new E[maxSize];

}

~AQueue() { delete [] listArray; } // Destructor 析构函数

void clear() { rear = 0; front = 1; } // Reinitialize 重新预置

void enqueue(const E& it) { // Put "it" in queue. it入队列

assert(((rear+2) % maxSize) != front); // "Queue is full" 队满则断言终止

rear = (rear+1) % maxSize; // Circular increment 头部索引在队列范围内循环加1

listArray[rear] = it;

}

E dequeue() { // Take element out. 一个元素出队列

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

E it = listArray[front];

front = (front+1) % maxSize; // Circular increment 尾部索引在队列范围内循环加1 return it;

}

const E& frontValue() const { // Get front value 获得头部元素

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

return listArray[front];

}

virtual int length() const // Return length 返回队列长度

{ return ((rear+maxSize) - front + 1) % maxSize; }

};

1. **主函数测试部分(AQueueTest.cpp)：**

#include "queue.h"

#include "AQueue.h"

#include<iomanip>

#include<iostream>

using namespace std;

// Main routine for array-based queue driver class

int main(){

int n;

AQueue<int> Q1(100);

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

cout<<"\*顺序队列中各个基本操作演示\*"<<endl;// Declare some sample lists

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

cout<<"向顺序队列中依次添加元素12、24、5、90、7、33"<<endl;

//调用顺序队列中的enqueue操作

Q1.enqueue(12);

Q1.enqueue(24);

Q1.enqueue(5);

Q1.enqueue(90);

Q1.enqueue(7);

Q1.enqueue(33);

cout<<"添加元素后顺序队列长度为：";

cout<<Q1.length()<<endl<<endl;//获取顺序队列长度

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

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

//调用顺序队列中的dequeue操作

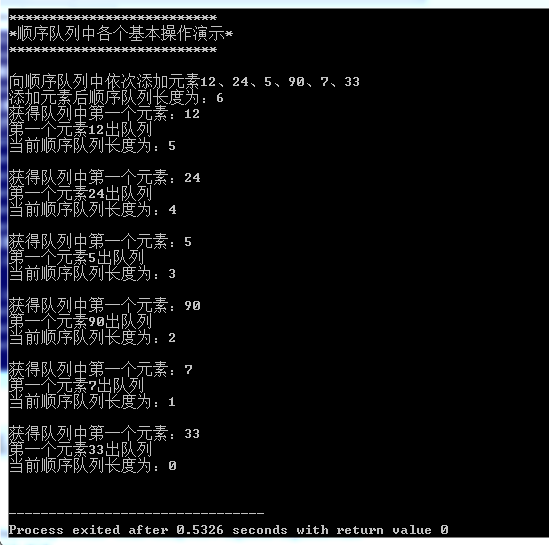
cout<<"第一个元素"<<Q1.dequeue()<<"出队列"<<endl;

cout<<"当前顺序队列长度为："<<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