\*\*\*\*\*\*\*\*\*\*\*基于LStack的栈实现\*\*\*\*\*\*\*\*\*\*\*

1. **Link抽象类(link.h):**

#include <iostream>

template <typename E> class Link {

public:

E element; // Value for this node 结点值

Link \*next; // Pointer to next node in list 结点指针：在链表中指向下一结点

// Constructors 构造函数

Link(const E& elemval, Link\* nextval =NULL)

{ element = elemval; next = nextval; }

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

};

1. **Stack抽象类(stack.h):**

#ifndef STACK

#define STACK

template <typename E> class Stack {

private:

void operator =(const Stack&) {} // Protect assignment

Stack(const Stack&) {} // Protect copy constructor

public:

Stack() {} // Default constructor构造函数

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

// Reinitialize the stack. The user is responsible for

// reclaiming the storage used by the stack elements.初始化栈 ，用户负责回收被使用来存储栈元素的空间

virtual void clear() = 0;

// Push an element onto the top of the stack.

// it: The element being pushed onto the stack.在栈顶压入“it”

virtual void push(const E& it) = 0;

// Remove the element at the top of the stack.

// Return: The element at the top of the stack.删除并返回栈顶元素

virtual E pop() = 0;

// Return: A copy of the top element.返回栈顶元素

virtual const E& topValue() const = 0;

// Return: The number of elements in the stack.返回当前栈的长度

virtual int length() const = 0;

};

#endif

1. **LStack物理实现(lstack.h)：**

#include "link.h"

#include "stack.h"

#include <assert.h>

template <typename E> class LStack: public Stack<E> {

private:

Link<E>\* top; // Pointer to first element 指向第一个元素

int size; // Number of elements 元素数目

public:

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

{ top = NULL; size = 0; }

~LStack() { clear(); } // Destructor 析构函数

void clear() { // Reinitialize 初始化

while (top != NULL) { // Delete link nodes 删除节点元素

Link<E>\* temp = top;

top = top->next;

delete temp;

}

size = 0;

}

void push(const E& it) { // Put "it" on stack "it"入栈

top = new Link<E>(it, top);

size++;

}

E pop() { // Remove "it" from stack 删除栈顶元素

assert(top != NULL); // "Stack is empty" 若栈空则终止程序

E it = top->element;

Link<E>\* ltemp = top->next;

delete top;

top = ltemp;

size--;

return it;

}

const E& topValue() const { // Return top value 返回栈顶元素

assert(top != 0); // "Stack is empty" 若栈空则终止程序

return top->element;

}

int length() const { return size; } // Return length 返回栈长度

};

1. **主函数测试部分(main.cpp)：**

#include "stack.h"

#include "lstack.h"

#include<iomanip>

#include<iostream>

using namespace std;

void print(LStack<int> &A,int n);

int main(){

int n;

LStack<int> S1(100);

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

cout<<"\*链式栈中各个基本操作演示\*"<<endl;

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

cout<<"向链式栈中依次添加元素12、15、9、48、55"<<endl;

//调用链式栈中的push操作

S1.push(12);

S1.push(15);

S1.push(9);

S1.push(48);

S1.push(55);

cout<<"添加元素后链式栈中内容为："<<endl;

n=S1.length(); //获取链式栈长度

print(S1,n);

cout<<"当前链式栈长度为:"<<endl<<S1.length()<<endl<<endl;

cout<<"查看链式栈中栈顶元素:"<<endl<<S1.topValue()<<endl<<endl;

cout<<"删除当前位栈顶元素" <<endl<<endl;

//调用链式栈中的pop操作

S1.pop();

cout<<"删除元素后链式栈中的内容为："<<endl;

n=S1.length(); //获取链式栈长度

print(S1,n);

//调用clear操作

cout<<"清空顺序栈后栈中内容"<<endl;

S1.clear();

n=S1.length(); //获取链式栈长度

print(S1,n);

}

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

\*function of print

\*打印链式栈中元素

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

void print(LStack<int> &A,int n){

int i;

int t,a;

LStack<int> B(100);//栈的复制体，倒序

t=A.length();

for(i=0;i<t;i++){ //复制栈

a=A.pop();

B.push(a);

}

for(i=0;i<t;i++){ //还原栈

a=B.pop();

cout<<a<<" ";

A.push(a);

}

cout<<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

部分代码来源：

<http://people.cs.vt.edu/~shaffer/Book/>

资料整理人：  
信息科学与工程学院 物联1402班 201408080203 宁静仪

信息科学与工程学院 物联1402班 201408080222吴彦彦