## 2021.9.24 课后编程训练题（栈stack）：---陈鹏宇20204227

#### AStack.h

#ifndef ASTACK\_H

#define ASTACK\_H

#define defaultSize 100

#include "stack.h"

#include <iostream>

template <typename E>

class AStack : public Stack<E>

{

private:

int maxSize;

int top;

E \*listArray;

public:

void Assert(bool val, std::string s);

AStack(int size = defaultSize);

~AStack();

void clear();

void push(const E& it);

E pop();

const E& topValue() const;

int length() const;

};

#endif // ASTACK\_H

#### Stack.h

#ifndef STACK\_H

#define STACK\_H

template <typename E>

class Stack

{

private:

void operator = (const Stack&) {}

Stack(const Stack&) {}

public:

Stack() {};

virtual ~Stack() {};

virtual void clear() = 0;

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

virtual E pop() = 0;

virtual const E& topValue() const = 0;

virtual int length() const = 0;

};

#endif // STACK\_H

#### Link.h

#ifndef LINK\_H

#define LINK\_H

#include <cstddef>

/\*单向链表\*/

template <typename E>

class Link

{

private:

static Link<E>\* freelist;

public:

E element;

Link \*next;

Link(const E& elemval , Link\* nextval = nullptr){ element = elemval; next = nextval; }

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

void\* operator new(size\_t){

if(freelist == nullptr) return ::new Link;

Link<E>\* temp = freelist;

freelist =freelist->next;

return temp;

}

void operator delete(void\* ptr){

((Link<E>\*)ptr)->next = freelist;

freelist = (Link<E>\*)ptr;

}

};

template <typename E>

Link<E>\* Link<E>::freelist = nullptr;

#endif // LINK\_H

#### List.h

#ifndef LIST\_H

#define LIST\_H

#include <cassert>

#include <cstring>

#include <iostream>

template <typename E>

class List

{

private:

void operator = (const List&) {}

List(const List&) {}

public:

List() {};

virtual ~List() {};

virtual void clear() = 0;

virtual void insert(const E& item) = 0;

virtual E remove() = 0;

virtual void moveToStart() = 0;

virtual void moveToEnd() = 0;

virtual void prev() = 0;

virtual void next() = 0;

virtual int length() const = 0;

virtual int currPos() const = 0;

virtual void moveToPos(int pos) = 0;

virtual const E& getValue() = 0;

};

#endif // LIST\_H

#### Llist.h

#ifndef LLIST\_H

#define LLIST\_H

#define defaultsize 100

#include "link.h"

#include "list.h"

using namespace std;

template <typename E>

class LList : public List<E>

{

private:

Link<E>\* head;

Link<E>\* tail;

Link<E>\* curr;

int cnt;

void init()

{

curr = tail = head = new Link<E>;

cnt = 0;

}

void removeall()

{

while(head != nullptr)

{

curr = head;

head = head->next;

delete curr;

}

}

public:

void Assert(bool val, string s);

LList(int size =defaultsize);

~LList();

void print() const;

void clear();

void insert(const E& it);

void append(const E& it);

E remove();

void moveToStart();

void moveToEnd();

void prev();

void next();

int length() const;

int currPos() const ;

void moveToPos(int pos);

const E& getValue();

void setValue(const E& it); //新增setValue成员函数

};

#endif // LLIST\_H

#### AStack.cpp

#include "astack.h"

#include "stack.h"

template<typename E>

void AStack<E>::Assert(bool val, std::string s)

{

if (!val) {

std::cout << "Assertion Failed: " << s << std::endl;

exit(-1);

}

}

template<typename E>

AStack<E>::AStack(int size)

{

maxSize = size;

top = 0;

listArray = new E[size];

}

template<typename E>

AStack<E>::~AStack()

{

delete [] listArray;

}

template<typename E>

void AStack<E>::clear()

{

top = 0;

}

template<typename E>

void AStack<E>::push(const E& it)

{

Assert(top != maxSize , "Stack is full");

listArray[top++] = it;

}

template<typename E>

E AStack<E>::pop()

{

Assert(top != 0 , "Stack is empty");

return listArray[--top];

}

template<typename E>

const E& AStack<E>::topValue() const

{

//Assert(top >= 1 , "Stack is empty");

return listArray[top-1];

}

template<typename E>

int AStack<E>::length() const

{

return top;

}

#### Llist.cpp

#include "llist.h"

#include "list.h"

/\*单向链表版本\*/

template <typename E>

void LList<E>::Assert(bool val, string s)

{

if (!val) {

cout << "Assertion Failed: " << s << endl;

exit(-1);

}

}

template <typename E>

LList<E>::LList(int size){ init(); }

template <typename E>

LList<E>::~LList() { removeall(); }

template <typename E>

void LList<E>::print() const{

Link<E>\* temp;

temp = head->next;

for(int i = 1 ; i <= cnt ; i++)

{

cout<<temp->element<<" ";

temp = temp->next;

}

}

template <typename E>

void LList<E>::clear() {

removeall();

init();

}

template <typename E>

void LList<E>::insert(const E& it){

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

if(tail == curr) tail = curr->next;

cnt++;

}

template <typename E>

void LList<E>::append(const E& it){

tail = tail->next = new Link<E>(it , nullptr);

cnt++;

}

template <typename E>

E LList<E>::remove(){

Assert(curr->next != nullptr , "No element");

E it = curr->next->element;

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

if(tail == curr->next) tail = curr;

curr->next = curr->next->next;

delete ltemp;

cnt--;

return it;

}

template <typename E>

void LList<E>::moveToStart(){ curr = head; }

template <typename E>

void LList<E>::moveToEnd(){ curr = tail; }

template <typename E>

void LList<E>::prev(){

if(curr == head) return;

Link<E>\* temp = head;

while (temp->next != curr) temp = temp->next;

curr = temp;

}

template <typename E>

void LList<E>::next(){

if(curr->next != tail) curr = curr->next;

}

template <typename E>

int LList<E>::length() const { return cnt; }

template <typename E>

int LList<E>::currPos() const {

Link<E>\* temp = head;

int i;

for( i = 0 ; curr != temp ; i++)

temp = temp->next;

return i;

}

template <typename E>

void LList<E>::moveToPos(int pos){

Assert(pos >= 0 && pos <= cnt , "Position out of range");

curr = head;

for(int i = 0 ; i < pos ; i++) curr = curr->next;

}

template <typename E>

const E& LList<E>::getValue(){

Assert(curr->next != nullptr , "No value");

return curr->next->element;

}

template <typename E>

void LList<E>::setValue(const E& it){

curr->next->element = it;

}

#### 1. 利用栈实现单链表的逆置操作（写成外部函数，在单链表类型定义中如果需要，可以增加一个修改当前节点值的操作函数setvalue）

#include <iostream>

#include "astack.h"

#include "astack.cpp"

#include "llist.cpp"

using namespace std;

template <typename E>

void myReverse(LList<E> &a)

{

int n = a.length();

AStack<E> t(n);

a.moveToStart();

for(int i = 0 ; i < n ; i++)

{

t.push(a.getValue());

a.next();

}

a.moveToStart();

for(int i = 0 ; i < n ; i++)

{

E temp = t.topValue();

a.setValue(temp) ; a.next() ;

t.pop();

}

}

int main()

{

LList<int> a(5);

for(int i = 0 ; i < 5 ; i++)

{

int t ; cin >> t;

a.append(t);

}

a.print();

cout<<endl;

myReverse(a);

cout<<endl;

a.print();

}

}文本

描述已自动生成

#### 2. 编程实现将一个十进制数转换成d进制数

#include <iostream>

#include "astack.h"

#include "astack.cpp"

using namespace std;

void myTransfer(int a,int b)

{

AStack<int> t;

while(true)

{

int temp = a % b ; t.push(temp);

a /= b ; if(!a) break;

}

int time = t.length();

for(int i = 0 ; i < time ; i++)

{

cout<<t.topValue();

t.pop();

}

}

int main()

{

int tar ; int n ; cin >> tar >> n ;//输入十进制目标数和进制数

myTransfer(tar,n);

}

}文本

描述已自动生成

文本

描述已自动生成

文本

描述已自动生成

#### 3. 给定一个字符串，里边可能包含“()”、"{}"、“[]”三种括号，请编写程序检查该字符串的括号是否成对出现。true：代表括号成对出现并且嵌套正确，或字符串无括号字符。false：未正确使用括号字符。

#include <iostream>

#include "astack.h"

#include "astack.cpp"

using namespace std;

int main()

{

for(int j = 0 ; j < 4 ; j++)

{

AStack<char> a;

string s ; getline(cin,s);

int flag = 1;

for(unsigned long long i = 0 ; i < s.length() ; i++)

{

if(s[i] == '(' || s[i] == '[' || s[i] == '{'){

a.push(s[i]);

continue;

}

if(s[i] == ')' && a.topValue() == '(' && a.length() != 0){

a.pop();

continue;

}

else if(s[i] == ')' && (a.topValue() != '(' && a.length() == 0))

{

flag = 0;

break;

}

if(s[i] == ']' && a.topValue() == '[' && a.length() != 0){

a.pop();

continue;

}

else if(s[i] == ']' && (a.topValue() != '(' && a.length() == 0))

{

flag = 0;

break;

}

if(s[i] == '}' && a.topValue() == '{' && a.length() != 0){

a.pop();

continue;

}

else if(s[i] == '}' && (a.topValue() != '(' && a.length() == 0))

{

flag = 0;

break;

}

}

if(a.length() != 0)

flag = 0;

if(flag)

cout<<"true"<<endl;

else

cout<<"false"<<endl;

}

return 0;

}

文本

描述已自动生成