List2.h

#pragma once

#include <iostream>

using namespace std;

template <class T>

class List2

{

private:

template <class T>

class InsideList

{

T information;

InsideList<T>\* next;

InsideList<T>\* prev;

public:

InsideList(T information = T(), InsideList<T>\* next = NULL, InsideList<T>\* prev = NULL) :

information(information),

next(next),

prev(prev)

{}

T& get\_information()

{

return information;

}

T get\_information() const

{

return information;

}

void set\_informaion(T element)

{

information = element;

}

InsideList<T>\* get\_next()

{

return next;

}

void set\_next(InsideList<T>\* newNext)

{

if (newNext != NULL)

newNext->prev = this;

next = newNext;

}

InsideList<T>\* get\_prev()

{

return prev;

}

void set\_prev(InsideList<T>\* newPrev)

{

if (newPrev != NULL)

newPrev->next = this;

prev = newPrev;

}

};

InsideList<T>\* begin;

InsideList<T>\* end;

public:

class IteratorList2

{

private:

InsideList<T>\* ptr;

public:

IteratorList2(InsideList<T>\* ptr) :

ptr(ptr){}

virtual bool operator!=(const IteratorList2& it) const

{

return ptr != it.ptr;

}

virtual IteratorList2& operator++()

{

ptr = ptr->get\_next();

return \*this;

}

virtual IteratorList2& operator--()

{

ptr = ptr->get\_prev();

return \*this;

}

virtual IteratorList2 operator++(int)

{

IteratorList2 temp = \*this;

ptr = ptr->get\_next();

return temp;

}

virtual IteratorList2 operator--(int)

{

IteratorList2 temp = \*this;

ptr = ptr->get\_prev();

return temp;

}

virtual T& operator\*() const

{

return ptr->get\_information();

}

};

List2();

~List2();

bool empty() const;

void push\_back(T newInformation);

void push\_front(T newInformation);

void pop\_back();

void pop\_front();

void delate();

friend ostream& operator<< (ostream& is, const List2<T>& begin)

{

InsideList<T>\* newBegin = begin.begin;

while (newBegin)

{

is << newBegin->get\_information() << ' ';

newBegin = newBegin->get\_next();

}

return is;

}

void random(int sizeOfList);

IteratorList2 get\_begin()

{

return begin;

}

IteratorList2 get\_end()

{

return end;

}

};

template<class T>

istream& operator>> (istream& is, List2<T>& begin)

{

int sizeOfList;

cout << "Введите размер листа" << endl;

do {

is >> sizeOfList;

if (sizeOfList <= 0)

cout << "Неправильный ввод данных" << endl;

} while (sizeOfList <= 0);

T chr;

begin.delate();

while (sizeOfList > 0)

{

is >> chr;

begin.push\_back(chr);

sizeOfList--;

}

return is;

}

template<class T>

List2<T>::List2() :

begin(NULL),

end(NULL){}

template<class T>

List2<T>::~List2()

{

this->delate();

}

template<class T>

bool List2<T>::empty() const

{

return begin == NULL;

}

template<class T>

void List2<T>::push\_back(T newInformation)

{

if (begin)

{

end->set\_next(new InsideList<T>(newInformation));

end = end->get\_next();

}

else

{

begin = new InsideList<T>(newInformation);

end = begin;

}

}

template<class T>

void List2<T>::push\_front(T newInformation)

{

if (begin)

{

begin->set\_prev(new InsideList(newInformation));

begin = begin->get\_prev();

}

else

{

begin = new InsideList<T>(newInformation);

end = begin;

}

}

template<class T>

void List2<T>::pop\_back()

{

if (begin->get\_next())

{

auto temp = end;

end = end->get\_prev();

delete temp;

end->set\_next(NULL);

}

else

{

delete begin;

end = begin = NULL;

}

}

template<class T>

void List2<T>::pop\_front()

{

if (end->get\_prev())

{

auto temp = begin;

begin = begin->get\_next();

delete temp;

begin->set\_prev(NULL);

}

else

{

delete begin;

end = begin = NULL;

}

}

template<class T>

void List2<T>::delate()

{

while (begin)

{

this->pop\_back();

}

}

template<class T>

void List2<T>::random(int sizeOfList)

{

while (sizeOfList > 0)

{

this->push\_back((char)(rand() % 26 + 97));

sizeOfList--;

}

}

13.cpp

#include <iostream>

#include <ctime>

#include "List2.h"

int main()

{

srand((unsigned)time(NULL));

List2<int> l;

l.random(5);

std::cout << l << std::endl;

for (auto ite = l.get\_end(), it = l.get\_begin(); it != ite; ++it, --ite)

{

std::swap(\*it, \*ite);

}

std::cout << l << std::endl;

system("pause");

return 0;

}