**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МОЭВМ**

отчет

**по лабораторной работе №1**

**по дисциплине «Объектно-ориентированное программирование»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 6382 |  | Вайгачёв А.О. |
| Преподаватель |  | Филатов А.Ю. |

Санкт-Петербург

2018

**Цель работы.**

Создание шаблонных контейнеров, создание собственных внешних итераторов. Научиться перегружать операторы.

**Постановка задачи.**

Необходимо создать свой контейнер (двусвязный список) и написать для него внешний итератор. В итераторе перегрузить основные операторы. Собрать проект с помощью Makefile.

**Описание программы.**

В программе описан шаблонный двусвязный список, который представлен классом List, в котором находятся указатели на голову (\*Head) и на хвост (\*Tail) списка, а также поле info, которое содержит информацию об элементе списка и может быть любого типа.

**Выводы.**

В ходе работы был создан собственный контейнер (шаблонный двусвязный список), также для этого контейнера создан внешний итератор, в котором перегружены основные операторы. Также был создан Makefile, с помощью которого собирался проект.

**Приложение А (Код программы)**

**List.h**

template <typename T>

class List{

public:

Node<T> \*Head;

Node<T> \*Tail;

class Iterator{

friend List<T>;

Node<T> \*P;

public:

Iterator(Node<T>\* p = NULL): P(p){};

const T& operator \*() const{

return P -> info;

};

T& operator \*(){

return P -> info;

};

Iterator& operator ++(){

P = P -> Next;

return \*this;

};

Iterator& operator ++(int){

Iterator t(\*this);

P = P -> Next;

return t;

};

Iterator& operator --(){

P = P -> Prev;

return \*this;

};

Iterator& operator --(int){

Iterator t(\*this);

P = P -> Prev;

return t;

};

friend bool operator ==(const Iterator& x, const Iterator& y){

return x.P == y.P;

};

friend bool operator !=(const Iterator& x, const Iterator& y){

return x.P != y.P;

};

friend bool operator ==(int y, const Iterator& x){

return x.P->info == y;

};

friend bool operator !=(int y, const Iterator& x){

return x.P->info != y;

};

friend bool operator ==(const Iterator& x,int y){

return x.P->info == y;

};

friend bool operator !=(const Iterator& x,int y){

return x.P->info != y;

};

Node<T> operator->() {

return P;

}

};

Iterator it;

List():Head(NULL),Tail(NULL){}

~List();

Iterator begin()const{

return Iterator(Head);

};

Iterator end()const{

return NULL;

};

void Show();

void Add(T info);

void AddStart(T info);

void AddEnd(T info);

void AddIndex(T info,int index);

void DelStart();

void DelEnd();

void DelIndex(int index);

};

#endif // LIST\_H

**List.cpp**

#include <iostream>

//#include <cstlib>

#include "list.h"

template <typename T>

List<T>::~List(){

while(Head){

Tail= Head -> Next;

delete Head;

Head=Tail;

}

}

template <typename T>

void List<T>::AddStart(T info){

Node<T>\*temp = new Node<T>;

if(Head == NULL){

temp->info = info;

temp->Next = Tail;

temp->Prev = NULL;

Head = temp;

Tail = Head;

return;

}

temp->info = info;

temp->Next = Head;

temp->Prev = NULL;

Head->Prev = temp;

Head = temp;

}

template <typename T>

void List<T>::AddEnd(T info){

Node<T>\*temp = new Node<T>;

temp->info = info;

if(Tail == NULL){

temp->Next = NULL;

temp->Prev = Tail;

Tail = temp;

Head = Tail;

return;

}

temp->Next = NULL;

temp->Prev = Tail;

Tail->Next = temp;

Tail = temp;

}

template <typename T>

void List<T>::AddIndex(T info,int index){

if(index == 0) {

AddStart(info);

return;

};

int size = 0;

for(it = this->begin(); it != this->end(); it++) size++;

size--;

if(index >= size){

AddEnd(info);

return;

}

Node<T> \*tmp1 = new Node<T>;

Node<T> \*tmp2 = new Node<T>;

tmp2 = Head;

size = 0;

while(size != index){

tmp2 = tmp2->Next;

size++;

}

tmp1->info = info;

tmp1->Next = tmp2->Next;

tmp1->Next->Prev = tmp1;

tmp1->Prev = tmp2;

tmp1->Prev->Next = tmp1;

}

template <typename T>

void List<T>::DelStart(){

if(Head == NULL){

std::cerr << "Nothing to delete" << std::endl;

exit(EXIT\_FAILURE);

return;

}

if(Head->Next == NULL) Head = NULL;

else {

Head = Head->Next;

Head->Prev->Next = NULL;

Head->Prev = NULL;

}

}

template <typename T>

void List<T>::DelEnd(){

if(Tail == NULL){

std::cerr << "Nothing to delete" << std::endl;

exit(EXIT\_FAILURE);

return;

}

if(Tail->Prev == NULL) Tail = NULL;

else {

Tail = Tail->Prev;

Tail->Next->Prev = NULL;

Tail->Next= NULL;

}

}

template <typename T>

void List<T>::DelIndex(int index){

if(index == 0) {

DelStart();

return;

};

int size = 0;

for(it = this->begin(); it != this->end(); it++) size++;

size--;

if(index >= size){

DelEnd();

return;

}

Node<T> \*tmp2 = new Node<T>;

Node<T> \*tr = new Node<T>;

tmp2 = Head;

size = 0;

while(size - 1 != index){

tmp2 = tmp2->Next;

size++;

}

tr = tmp2->Prev;

tmp2->Prev = tmp2->Prev->Prev;

tmp2->Prev->Next= tmp2;

delete tr;

return;

}

template <typename T>

void List<T>:: Show(){

Node<T>\* temp=Tail;

temp=Head;

while(temp!=NULL){

std::cout << temp->info << " ";

temp=temp->Next;

}

std::cout<<"\n";

}

**Main.cpp**

#include <iostream>

#include <cstdlib>

#include "list.cpp"

using namespace std;

int main()

{

List <int> lst;

//List <int> :: Iterator it;

for(auto &Elem: lst){

Elem--;

}

List <int> l;

//List <int> :: Iterator it;

for(int i = 0; i < 10; i++){

l.AddEnd(i\*i);

}

l.AddStart(1234);

l.AddEnd(4321);

for(auto &Elem: l){

cout << Elem << endl;

}

l.AddIndex(777,5);

l.AddIndex(666,12);

for(auto &Elem: l){

cout << Elem << endl;

}

cout << "Deletting ..." << endl;

l.DelStart();

l.DelEnd();

l.DelEnd();

for(auto &Elem: l){

cout << Elem << endl;

}

l.DelIndex(5);

for(auto &Elem: l){

cout << Elem << endl;

}

cout << "Deletting all ..." << endl;

delete &l;

cout << "Sucsess!" << endl;

}