Міністерство освіти і науки України КПІ ім. Ігоря Сікорського

Кафедра ІПІ

ЗВІТ

з виконання лабораторної роботи № 6 “Структури даних”

з кредитного модуля

“Основи програмування-2. Методології програмування” Варіант № 5

Виконав:

студент 1-го курсу гр. ІП-22 ФІОТ

Гринько О.М.

Київ 2023

## **Завдання варіанту**

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

**Вимоги до програми**

Изображение выглядит как текст, Шрифт, снимок экрана, алгебра

Автоматически созданное описание

## **ТЕКСТ ПРОГРАМИ**

**main.cpp**

#include "functions.cpp"

int main() {

int type=1;

cout << "Enter value type(string-1,double-2,char-3,int-4):";cin>>type;

switch (type) {

case 1:

start\_program<string>();

break;

case 2:

start\_program<double>();

break;

case 3:

start\_program<char>();

break;

case 4:

start\_program<int>();

break;

}

return 0;

}

**List.h**

#ifndef LIST\_H

#define LIST\_H

#include <iostream>

using namespace std;

template <typename T> class List;

template <typename T> class Iterator;

template <class T>

class Node{

Node<T>\* next;

Node<T>\* prev;

T data;

public:

friend class Iterator<T>;

friend class List<T>;

Node(T \_data);

};

template <typename T>

class Iterator{

List<T> \_list;

Node<T>\* \_curr;

public:

Iterator(List<T>&);

Iterator(const List<T>&,Node<T>\*);

//setting to a head of the list

Iterator<T> begin();

//traversing to the next element

void operator++(int);

//traversing to the prev element

void operator--();

//accessing a current element

T& operator\*() const;

//checking if a tail

bool is\_tail();

};

template <typename T>

class List {

Node<T>\* \_head;

Node<T>\* \_tail;

size\_t \_size;

public:

//checking for emptiness

bool is\_empty();

//clear the list

void clear();

//access by index

Iterator<T> operator[](size\_t index)const;

//delete by index

T delete\_i(size\_t);

size\_t size() const;

void insert(T obj);

List();

friend class Iterator<T>;

};

#endif

**List.cpp**

#include "List.h"

//NODE FUNCTIONS

template <typename T>

Node<T>::Node(T \_data):data(\_data),prev(nullptr),next(nullptr){};

//ITERATOR FUNCTIONS

template <typename T>

Iterator<T>::Iterator(List<T>& list){

\_list=list;

\_curr=list.\_head;

}

template <typename T>

Iterator<T> Iterator<T>::begin(){

\_curr=\_list.\_head;

return \*this;

}

template <typename T>

Iterator<T>::Iterator(const List<T>& list,Node<T>\* node):\_list(list),\_curr(node){};

template <typename T>

void Iterator<T>::operator++(int i){

\_curr=\_curr->next;

}

template <typename T>

void Iterator<T>::operator--(){

\_curr=\_curr->prev;

}

template <typename T>

T& Iterator<T>::operator\*() const{

return \_curr->data;

}

template <typename T>

bool Iterator<T>::is\_tail(){

return \_curr==\_list.\_tail;

}

//LIST FUNCTIONS

template <typename T>

bool List<T>::is\_empty(){

return \_size==0;

}

template <typename T>

void List<T>::clear() {

Node<T>\* curr = \_head;

while (\_size != 0) {

Node<T>\* nextNode = curr->next;

delete curr;

curr = nextNode;

--\_size;

}

}

template <typename T>

Iterator<T> List<T>::operator[](size\_t index)const{

if(index<0||index>\_size-1)

throw out\_of\_range("Only indices in range from 0 to "+to\_string(\_size-1)+" allowed");

int i=0;

Node<T>\* curr=\_head;

while(i!=index){

curr=curr->next;

++i;

}

return Iterator(\*this,curr);

}

template <typename T>

T List<T>::delete\_i(size\_t index){

if(index<0||index>\_size-1)

throw out\_of\_range("Only indices in range from 0 to "+to\_string(\_size-1)+" allowed");

int i = 0;

Node<T>\* curr = \_head;

while (i != index) {

curr = curr->next;

++i;

}

if (curr == \_head) {

\_head = curr->next;

\_head->prev=\_tail;

} else if (curr == \_tail){

\_tail = curr->prev;

\_tail->next=\_head;

}

else{

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

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

}

T value = curr->data;

delete curr;

--\_size;

return value;

}

template <typename T>

size\_t List<T>::size() const{

return \_size;

}

template <typename T>

void List<T>::insert(T obj){

if(\_size==0){

//insert as a first elem

\_head=new Node(obj);

\_tail=\_head;

\_head->next=\_head;

\_head->prev=\_head;

}else{

//insert to the end of the list

Node<T>\* newNode=new Node(obj);

newNode->prev=\_tail;

newNode->next=\_head;

\_tail->next=newNode;

\_tail=newNode;

\_head->prev=\_tail;

}

\_size++;

}

template <typename T>

List<T>::List():\_head(nullptr),\_tail(nullptr),\_size(0){};

**functions.h**

#ifndef LAB\_6\_FUNCTIONS\_H

#define LAB\_6\_FUNCTIONS\_H

using namespace std;

#include <iostream>

#include "List.cpp"

template<typename T>

void start\_program();

template<typename T>

void print\_list(List<T>& list);

#endif //LAB\_6\_FUNCTIONS\_H

**functions.cpp**

#include "functions.h"

template<typename T>

void start\_program(){

int choice;

List<T>list;

size\_t size;

cout<<"Enter a structure size:";cin>>size;

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

cout<<"Enter "+ to\_string(i+1)+" value:";

T value;

cin >> value;

list.insert(value);

}

do {

print\_list(list);

cout << "Choose operation" << endl;

cout << "->0. Quit"<<endl;

cout << "->1. Check if empty"<<endl;

cout << "->2. Clear a list"<<endl;

cout << "->3. Find element by index"<<endl;

cout << "->4. Delete by index"<<endl;

cout << "->5. Insert an element"<<endl;

cin>>choice;

size\_t index;

T value;

switch (choice) {

case 1:

cout << "Is Empty: "<<(list.is\_empty()==1?"true":"false")<<endl;

break;

case 2:

cout << "List cleared"<<endl;

list.clear();

break;

case 3:

cout << "Enter index:";cin>>index;

try {

cout << "Value: "<< \*list[index]<<endl;

}catch(out\_of\_range err) {

cerr << err.what()<<endl;

}

break;

case 4:

cout << "Select position: ";cin>>index;

try {

list.delete\_i(index);

}catch(out\_of\_range err){

cerr << err.what()<<endl;

}

break;

case 5:

cout << "Enter a value: ";cin>>value;

list.insert(value);

break;

}

} while (choice!=0);

}

template<typename T>

void print\_list(List<T>& list){

cout <<"Double-linked list"<<endl << "{ ";

auto it = Iterator<T>(list);

for(int i=0;i<list.size();++i){

cout << \*it << " ";

it++;

}

cout <<"}"<< endl;

}

## **РЕЗУЛТАТИ ТЕСТУВАННЯ**

1. Перевірка списку на пустоту (рис.1)

Зображення, що містить текст, знімок екрана, дизайн

Автоматично згенерований опис

Рисунок 1. Перевірка на пустоту

1. Очищення списку (рис.2)

Зображення, що містить текст, знімок екрана, дизайн

Автоматично згенерований опис

Рисунок 2. Очищення списку

1. Доступ до значення елемента із заданим номером у списку (рис.3)

Зображення, що містить текст, знімок екрана, меню, дизайн

Автоматично згенерований опис

Рисунок 3. Доступ за індексом

1. Видалення елемента з позиції з вказаним номером (рис.4)

Зображення, що містить текст, знімок екрана, Шрифт, дизайн

Автоматично згенерований опис

Рисунок 4. Видалення за індексом

**Github: https://github.com/esk4nz/OP\_labs**