**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ**

**ТЕХНОЛОГІЙ І ЗВ’ЯЗКУ**

**Звіт**

**з дисципліни Проектний Практикум**

**Практична робота №8**

**на тему: «Проектування і опрацювання програми з родовими функціями і родовими класами»**

Виконав: студент 3 курсу, групи ІПЗ-3.04 спеціальності

121 Інженерія програмного забезпечення

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Бухта М.М.

Перевірив\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Багачук Д.Г.

**Одеса  2023**

**МЕТА РОБОТИ**

Засвоєння поняття родової функції та засобу її реалізації — шаблону функції; засвоєння поняття родового класу та засобу його реалізації — шаблону класу;

**ЗАВДАННЯ**

Розробити і реалізувати програмно шаблонний клас для подання розріджених одновимірних масивів. Розмір логічного масиву передавати через аргумент конструктора. Для йього виконати таки завдання:

1. Розробити і реалізувати програмно структуру класу розрідженого одновимірного масиву.

2. Реалізувати зберігання даних будь-якого типу Т у створеному класі.

3. Передбачити у класі родовий конструктор за замовчуванням, родовий конструктор копіювання і операцію присвоєння.

4. Реалізувати у класі операцію індексування, що повертає посилання на знайдений елемент у масиві. Якщо елемент із заданим індексом не знайдений, то операція повинна створити новий елемент з цих індексом і розмістити його у масив.

5. В основній частині програми продемонструвани використання ствоерного класу.

**КОД ПРОГРАМИ**

**main.cpp**

#include "List"

#include <iostream>

using namespace lab\_8;

template <typename T>

void print(const List<T> &list) {

for (const auto &item : list) {

std::cout << \*item << ", ";

}

std::cout << std::endl;

}

int main(int argc, char \*\*argv) {

List<std::int32\_t> list;

list[3] = 8;

list[5] = 12;

std::cout << "-------- LIST --------" << std::endl;

print(list);

list[2] = 2;

List<std::int32\_t> list\_2{list};

List<std::int32\_t> list\_3;

list\_3 = list\_2;

std::cout << "-------- LIST 2 --------" << std::endl;

print(list\_2);

std::cout << "-------- LIST 3 --------" << std::endl;

print(list\_3);

std::cout << "size of list 1/2/3:\t" << list.size() << " / " << list\_2.size() << " / " << list\_3.size() << std::endl;

return 0;

}

**List.hpp**

#ifndef LAB\_8\_PROJECTPRACTICUM\_BUKHTAMYKYTA\_SE304\_LIST\_HPP

#define LAB\_8\_PROJECTPRACTICUM\_BUKHTAMYKYTA\_SE304\_LIST\_HPP

#include <initializer\_list>

#include <cinttypes>

namespace lab\_8 {

template <typename T>

class ListIterator;

template <typename T>

struct Node {

~Node(void);

T val;

Node \*left;

Node \*right;

};

template <typename T>

class List final {

friend class ListIterator<T>;

public:

List(void);

List(const List<T> &other);

List(const std::initializer\_list<T> &init\_list);

~List(void);

void operator= (const List<T> &other);

void operator= (const std::initializer\_list<T> &init\_list);

T& operator[](std::uint64\_t index);

ListIterator<T> begin(void) const;

ListIterator<T> end(void) const;

std::size\_t size(void) const noexcept;

private:

void destroy(void);

private:

Node<T> \*m\_head;

std::size\_t m\_size;

};

template <typename T>

class ListIterator {

public:

ListIterator(Node<T> \*cur = nullptr);

*/\* Description:*

*\* Prefix operator;*

*\*/*

ListIterator<T>& operator++(void);

ListIterator<T>& operator--(void);

*/\* Description:*

*\* Postfix operator;*

*\*/*

ListIterator<T> operator++(int);

ListIterator<T> operator--(int);

bool operator==(const ListIterator<T> &other);

bool operator!=(const ListIterator<T> &other);

T\* operator\*(void) const;

private:

Node<T> \*m\_cur;

};

} *// !lab\_8;*

#endif *// !LAB\_8\_PROJECTPRACTICUM\_BUKHTAMYKYTA\_SE304\_LIST\_HPP;*

**List.cpp**

#include "List.hpp"

#include <memory>

namespace lab\_8 {

*//---------------------------- Node ----------------------------//*

template <typename T>

Node<T>::~Node(void) {

if (right) {

delete right;

}

}

*//---------------------------- List ---------------------------//*

template <typename T>

List<T>::List(void) : m\_size{0}, m\_head{nullptr} {

}

template <typename T>

List<T>::List(const List<T> &other) {

operator=(other);

}

template <typename T>

List<T>::List(const std::initializer\_list<T> &init\_list) : List() {

operator=(init\_list);

}

template <typename T>

List<T>::~List(void) {

destroy();

}

template <typename T>

void List<T>::operator= (const List<T> &other) {

destroy();

this->m\_size = other.m\_size;

if (!other.m\_head) {

return;

}

m\_head = new Node<T>();

m\_head->val = other.m\_head->val;

Node<T> \*current\_this\_node = this->m\_head;

Node<T> \*current\_other\_node = other.m\_head;

while (current\_other\_node->right) {

current\_this\_node->right = new Node<T>();

current\_this\_node->right->val = current\_other\_node->right->val;

current\_this\_node->right->left = current\_this\_node;

current\_this\_node = current\_this\_node->right;

current\_other\_node = current\_other\_node->right;

}

}

template <typename T>

void List<T>::operator= (const std::initializer\_list<T> &init\_list) {

destroy();

m\_size = init\_list.size();

auto init\_list\_begin = init\_list.begin();

auto init\_list\_end = init\_list.end();

if (!m\_head) {

m\_head = new Node<T>(\*init\_list\_begin);

++init\_list\_begin;

}

Node<T> \*current\_node = m\_head->right;

while (init\_list\_begin != init\_list\_end) {

current\_node = new Node<T>(\*init\_list\_begin);

current\_node = current\_node->right;

++init\_list\_begin;

}

}

template <typename T>

T& List<T>::operator[](std::uint64\_t index) {

if (!m\_head) {

m\_head = new Node<T>();

++m\_size;

}

Node<T> \*current\_node = m\_head;

for (; index != 0; --index) {

if (!current\_node->right) {

current\_node->right = new Node<T>();

++m\_size;

}

current\_node->right->left = current\_node;

current\_node = current\_node->right;

}

return current\_node->val;

}

template <typename T>

void List<T>::destroy(void) {

if (m\_head) {

delete m\_head;

}

}

template <typename T>

ListIterator<T> List<T>::begin(void) const {

return {this->m\_head};

}

template <typename T>

ListIterator<T> List<T>::end(void) const {

return {nullptr};

}

template <typename T>

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

return m\_size;

}

*//------------------------ ListIterator ------------------------//*

template <typename T>

ListIterator<T>::ListIterator(Node<T> \*cur) : m\_cur{cur} {

}

template <typename T>

ListIterator<T>& ListIterator<T>::operator++(void) {

m\_cur = m\_cur->right;

return \*this;

}

template <typename T>

ListIterator<T>& ListIterator<T>::operator--(void) {

m\_cur = m\_cur->left;

return \*this;

}

template <typename T>

ListIterator<T> ListIterator<T>::operator++(int) {

ListIterator<T> ret = \*this;

this->operator++();

return ret;

}

template <typename T>

ListIterator<T> ListIterator<T>::operator--(int) {

ListIterator<T> ret = \*this;

this->operator--();

return ret;

}

template <typename T>

T\* ListIterator<T>::operator\*(void) const {

return &m\_cur->val;

}

template <typename T>

bool ListIterator<T>::operator==(const ListIterator<T> &other)

{

return this->m\_cur == other.m\_cur;

}

template <typename T>

bool ListIterator<T>::operator!=(const ListIterator<T> &other)

{

return !(this->operator==(other));

}

} *// !lab\_8;*

**List**

#ifndef LAB\_8\_PROJECTPRACTICUM\_BUKHTAMYKYTA\_SE304\_LIST

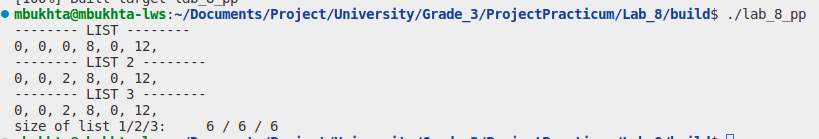
#define LAB\_8\_PROJECTPRACTICUM\_BUKHTAMYKYTA\_SE304\_LIST

#include "List.hpp"

#include "List.cpp"

#endif *// !LAB\_8\_PROJECTPRACTICUM\_BUKHTAMYKYTA\_SE304\_LIST;*

**РЕЗУЛЬТАТ ВИКОНАННЯ**



**ВИСНОВОК**

В ході виконання лабораторної роботи ми успішно засвоїли основні поняття родової функції та родового класу, а також засоби їх реалізації у вигляді шаблонів функцій та класів відповідно. Розглянувши концепції шаблонів функцій та класів, ми вивчили їхні переваги та можливості в контексті програмування.