Звіт

Автор: Чугунов В.Ю

КІТ-119а

Лабораторна робота №12

**STL. ІТЕРАТОРИ. ПОСЛІДОВНІ КОНТЕЙНЕРИ. ЦИКЛ** *RANGE-FOR***. АСОЦІАТИВНІ КОНТЕЙНЕРИ**

**Мета:** отримати базові знання про *STL* контейнери. Освоїти основні механізми роботи з *STL* контейнерами.

1 ЗАВДАННЯ ДО РОБОТИ

**Загальне завдання.** Створити діалогове меню, що дозволяє продемонструвати продемонструвати роботу *STL* контейнерів та показати їх принципову різницю.

2 РОЗРОБКА АЛГОРИТМІВ ФУНКЦІЙ

**Клас ConsoleUI<typename T>**

**private поля:**

vector<T> Vector – контейнер std::vector<>, для використання: #include <vector>

set<T> Set – контейнер std::set<>, для використання: #include <set>

list<T> List – контейнер std::list<>, для використання: #include <list>

map<int, T> Map – контейнер std::map<>, для використання #include <map>

**Функції:  
 int main ( )** Призначення: головна функція.  
 Опис роботи:   
 1) створення об’єкта класу ConsoleUI;   
 2) виклик метода ConsoleUI::RunExe.  
 Схема алгоритму подана на рис. 1.

**void RunExe ( )** Призначення: головне меню програми.  
 Опис роботи: користувачу пропонується вибрати один з контейнерів. Будь-який з варіантів призведе до виклику вибору функцій, що відповідають за обраний контейнер. Функція буде працювати до тих пір, поки користувач не введе «0» (умова виходу з програми).  
 Схема алгоритму подана на рис. 2.

**int ContainerSelect ( )** Призначення: виведення на екран списку контейнерів на вибір.  
 Опис роботи: функція пропонує користувачу обрати один з контейнерів («0» для виходу), після чого вибір повертається як результат роботи функції.

**int OperationSelect ( )** Призначення: виведення на екран списку операцій над контейнерами.  
 Опис роботи: функція пропонує користувачу обрати одну із функцій («0» для виходу), після чого вибір повертається як результат роботи функції.

**void DoVectorOperation ( int choice )** Призначення: вибрати операцію над вектором.  
 Опис роботи: користувачу пропонується список операцій на вибір. Після вибору викликається відповідна функція. Функція працює доти, доки користувач не введе «0».

**void DoSetOperation ( int choice )** Призначення: вибрати операцію над сетом.  
 Опис роботи: користувачу пропонується список операцій на вибір. Після вибору викликається відповідна функція. Функція працює доти, доки користувач не введе «0».

**void DoListOperation ( int choice )** Призначення: вибрати операцію над списком.  
 Опис роботи: користувачу пропонується список операцій на вибір. Після вибору викликається відповідна функція. Функція працює доти, доки користувач не введе «0».

**void DoMapOperation ( int choice )** Призначення: вибрати операцію над мапом.  
 Опис роботи: користувачу пропонується список операцій на вибір. Після вибору викликається відповідна функція. Функція працює доти, доки користувач не введе «0».

**void AddToVector ( int choice )** Призначення: додавання елементу у вектор.  
 Опис роботи: користувачу пропонується обрати один із варіантів додавання елементу у вектор («0» - скасування операції). Додавати можна у кінець, або задавши індекс. В обох випадках спочатку пропонується ввести значення елемента, що буде додано у вектор. Але у другому випадку пропонується також вибрати й позицію елемента. Після чого відбувається процес додавання елементу.

**void AddToList ( int choice )** Призначення: додавання елементу у список.  
 Опис роботи: користувачу пропонується обрати один із варіантів додавання елементу у вектор («0» - скасування операції). Додавати можна у початок, кінець, або задавши індекс. В усіх випадках спочатку пропонується ввести значення елемента, що буде додано у вектор. Але у третьому випадку пропонується також вибрати й позицію елемента. Після чого відбувається процес додавання елементу.

**void AddToSet ( )** Призначення: додавання елементу у сет.  
 Опис роботи: користувачу пропонується ввести значення елемента, що буде додано у сет. Після чого відбувається процес додавання елементу.

**void AddToMap ( )** Призначення: додавання елементу у мап.  
 Опис роботи: користувачу пропонується ввести ключ, а потім значення елемента, що буде додано у мап. Після чого відбувається процес додавання елементу.

**void DelFromVector ( int choice )** Призначення: видалення елементу із вектору.  
 Опис роботи: перевіряється наявність елементів у контейнері. Якщо елементів не має, то функція виводить на екран відповідне повідомлення. В іншому випадку користувачу пропонується обрати один із варіантів видалення елементу із вектору («0» - скасування операції). Видаляти можна з кінця, або задавши індекс. У другому випадку пропонується також вибрати й позицію елемента. Після чого відбувається процес видалення елементу.

**void DelFromList ( int choice )** Призначення: видалення елементу зі списку.  
 Опис роботи: перевіряється наявність елементів у контейнері. Якщо елементів не має, то функція виводить на екран відповідне повідомлення. В іншому випадку користувачу пропонується обрати один із варіантів видалення елементу зі списку («0» - скасування операції). Видаляти можна з початку, з кінця, або задавши індекс. У третьому випадку пропонується також вибрати й позицію елемента. Після чого відбувається процес видалення елементу.

**void DelFromSet ( )** Призначення: видалення елементу із сету.  
 Опис роботи: перевіряється наявність елементів у контейнері. Якщо елементів не має, то функція виводить на екран відповідне повідомлення. В іншому випадку користувачу пропонується ввести значення елемента, що буде видалено. Якщо елемента з таким значенням в колекції не існує – виводиться відповідне повідомлення. Інакше відбувається процес видалення елементу.

**void DelFromMap ( )** Призначення: видалення елементу із мапу.  
 Опис роботи: перевіряється наявність елементів у контейнері. Якщо елементів не має, то функція виводить на екран відповідне повідомлення. В іншому випадку користувачу пропонується ввести ключ елемента, що буде видалено. Якщо елемента з таким ключем не існує в колекції – виводиться відповідне повідомлення. Інакше відбувається процес видалення елементу.

**T VectorGet ( int choice )**  
 Призначення: отримання даних з вектору.  
 Опис роботи: перевіряється наявність елементів у контейнері. Якщо елементів не має, то функція виводить на екран відповідне повідомлення. В іншому випадку користувачу пропонується обрати один із варіантів повернення елементів елементу із вектору («0» - скасування операції). Повернути можна перший, останній елемент, або задавши індекс. У третьому випадку пропонується також вибрати й позицію елемента. Після чого функція повертає елемент як результат своєї роботи.

**T ListGet ( int choice )**  
 Призначення: отримання даних зі списку.  
 Опис роботи: перевіряється наявність елементів у контейнері. Якщо елементів не має, то функція виводить на екран відповідне повідомлення. В іншому випадку користувачу пропонується обрати один із варіантів повернення елементів елементу зі списку («0» - скасування операції). Повернути можна перший, останній елемент, або задавши індекс. У третьому випадку пропонується також вибрати й позицію елемента. Після чого функція повертає елемент як результат своєї роботи.

**T SetGet ( int choice )**  
 Призначення: отримання даних із сету.  
 Опис роботи: перевіряється наявність елементів у контейнері. Якщо елементів не має, то функція виводить на екран відповідне повідомлення. В іншому випадку користувачу пропонується обрати один із варіантів повернення елементів елементу із сету («0» - скасування операції). Користувачу пропонується ввести значення. Якщо елементів з таким значенням в контейнері не має, виводиться повідомлення про відсутність елементу. Інакше, функція повертає необхідний елемент.

**T MapGet ( int choice )**  
 Призначення: отримання даних із мапу.  
 Опис роботи: перевіряється наявність елементів у контейнері. Якщо елементів не має, то функція виводить на екран відповідне повідомлення. В іншому випадку користувачу пропонується обрати один із варіантів повернення елементів елементу із мапу («0» - скасування операції). Користувачу пропонується ввести ключ. Якщо елементів з таким ключем в контейнері не має, виводиться повідомлення про відсутність елементу. Інакше, функція повертає необхідний елемент.

**void VectorShow ( )** Призначення: виведення на екран даних вектору.  
 Опис роботи: перевіряється наявність елементів у контейнері. Якщо елементів не має, то функція виводить на екран відповідне повідомлення. В іншому випадку використовуючи range-based цикл функція виводить усі елементи контейнера на екран.

**void ListShow ( )** Призначення: виведення на екран даних списку.  
 Опис роботи: перевіряється наявність елементів у контейнері. Якщо елементів не має, то функція виводить на екран відповідне повідомлення. В іншому випадку використовуючи range-based цикл функція виводить усі елементи контейнера на екран.

**void SetShow ( )** Призначення: виведення на екран даних сету.  
 Опис роботи: перевіряється наявність елементів у контейнері. Якщо елементів не має, то функція виводить на екран відповідне повідомлення. В іншому випадку використовуючи range-based цикл функція виводить усі елементи контейнера на екран.

**void MapShow ( )** Призначення: виведення на екран даних мапу.  
 Опис роботи: перевіряється наявність елементів у контейнері. Якщо елементів не має, то функція виводить на екран відповідне повідомлення. В іншому випадку використовуючи range-based цикл функція виводить усі елементи контейнера на екран.

3 ТЕКСТ ПРОГРАМИ

**Текст файлу main.cpp**

#include "Dialog.h"

int main()

{

setlocale(LC\_ALL, "rus");

ConsoleUI<Point> UI;

UI.RunExe();

system("PAUSE");

return 0;

}

**Текст файлу Dialog.h**

#pragma once

#include <vector>

#include <set>

#include <list>

#include <map>

#include <Windows.h>

#include"Point.h"

void SetColor(int text, int background)

{

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, (WORD)((background << 4) | text));

}

using std::vector;

using std::set;

using std::list;

using std::map;

using std::cout;

using std::cin;

using std::endl;

template<typename T>

class ConsoleUI

{

public:

/\* Меню - отвечает за работу диалогового окна \*/

void RunExe();

private:

// Используемые контейнеры

vector<T> Vector;

set<T> Set;

list<T> List;

map<int, T> Map;

// Выбор контейнера

int ContainerSelect();

// Выбор операции

int OperationSelect();

// Выполнение операции над Вектором

void DoVectorOperation(int ch);

// Добавление элемента в Вектор

void AddToVector(int ch);

// Удаление элемента из вектора

void DelFromVector(int ch);

// Получение данных из вектора

T VectorGet(int ch);

// Вывод содержимого Вектора на экран

void VectorShow();

// Уведомление об отсутствии данных в Векторе

bool NoVectorData();

// Уведомление о выборе Вектора

void ChoseVector();

// Выполнение операции над Сетом

void DoSetOperation(int ch);

// Добавление элемента в Сет

void AddToSet();

// Удаление элемента из Сета

void DelFromSet();

// Получение данных из Сета

T SetGet();

// Вывод содержимого Сета на экран

void SetShow();

// Уведомление об отсутствии данных в Сете

bool NoSetData();

// Уведомление о выборе Сета

void ChoseSet();

// Выполнение операции над Списком

void DoListOperation(int ch);

// Добавление элемента в Список

void AddToList(int ch);

// Удаление элемента из Списка

void DelFromList(int ch);

// Получение данных из Списка

T ListGet(int ch);

// Вывод содержимого Списка на экран

void ListShow();

// Уведомление об отсутствии данных в Списке

bool NoListData();

// Уведомление о выборе Списка

void ChoseList();

// Выпонение операции над Мапом

void DoMapOperation(int ch);

// Добавление элемента в Мап

void AddToMap();

// Удаление элемента из Мапа

void DelFromMap();

// Получение данных из Мапа

T MapGet();

// Вывод содержимого Мапа на экран

void MapShow();

// Уведомление об отсутствии данных в Мапе

bool NoMapData();

// Уведомление о выборе Мапа

void ChoseMap();

// Уведомление об успешной операции

void OperationPass();

// Уведомление об отмене операции

void OperationCancel();

// Информация о лабораторной работе

void LabInfo();

// Проверка выбора варианта

bool CheckSelect(int b, int e, int i);

};

// ########################## ГЛАВНОЕ МЕНЮ ##########################

template<typename T>

inline void ConsoleUI<T>::RunExe()

{

int choice;

do {

choice = ContainerSelect();

switch (choice)

{

case 1:

DoVectorOperation(choice);

system("cls");

break;

case 2:

DoListOperation(choice);

system("cls");

break;

case 3:

DoSetOperation(choice);

system("cls");

break;

case 4:

DoMapOperation(choice);

system("cls");

break;

case 0:

SetColor(13, 0);

cout << "\nСпасибо за работу!\n";

SetColor(7, 0);

}

} while (choice);

}

// ######################## ВЫБОР КОНТЕЙНЕРА ########################

template<typename T>

inline int ConsoleUI<T>::ContainerSelect()

{

int choice;

LabInfo();

cout << "Выберите контейнер:\n";

cout << "1. Vector.\n";

cout << "2. List.\n";

cout << "3. Set.\n";

cout << "4. Map.\n";

cout << "0. Выход.\n";

cout << "Введите ваш ответ сюда: ";

cin >> choice;

CheckSelect(0, 4, choice);

return choice;

}

// ######################### ВЫБОР ОПЕРАЦИЙ #########################

template<typename T>

inline int ConsoleUI<T>::OperationSelect()

{

int choice;

cout << "Список доступных операций:\n";

cout << "1. Добавление элемента.\n";

cout << "2. Удаление элемента.\n";

cout << "3. Получение данных.\n";

cout << "4. Вывод содержимого на экран.\n";

cout << "0. Вернуться к выбору контейнера.\n";

cout << "Введите ваш ответ сюда: ";

cin >> choice;

CheckSelect(0, 4, choice);

return choice;

}

// ######################## ОПЕРАЦИЯ УСПЕШНА ########################

template<typename T>

inline void ConsoleUI<T>::OperationPass()

{

SetColor(10, 0);

cout << "Операция прошла успешно!\n";

SetColor(7, 0);

system("PAUSE");

system("cls");

}

// ######################### ОТМЕНА ОПЕРАЦИИ ########################

template<typename T>

inline void ConsoleUI<T>::OperationCancel()

{

SetColor(12, 0);

cout << "Операция отменена.\n";

SetColor(7, 0);

system("PAUSE");

system("cls");

}

// ########################### ИНФО О ЛАБЕ ##########################

template<typename T>

inline void ConsoleUI<T>::LabInfo()

{

SetColor(9, 0);

cout << " Лабораторная работа №12: \"STL контейнеры\" \*Выполнил: Пумня Александр\n";

SetColor(7, 0);

}

// ################## ПРОВЕРКА ПРАВИЛЬНОСТИ ВЫБОРА ##################

template<typename T>

inline bool ConsoleUI<T>::CheckSelect(int b, int e, int i)

{

if (i < b || i > e)

{

SetColor(12, 0);

cout << "\nВы ошиблись. Введите число от " << b << " до " << e << endl;

SetColor(7, 0);

system("PAUSE");

system("cls");

return false;

}

return true;

}

#include "Vector.hpp" // Реализация функций над Вектором

#include "Set.hpp" // Реализация функций над Сетом

#include "List.hpp" // Реализация функций над Списком

#include "Map.hpp" // Реализация функций над Мапом

**Текст файлу Vector.hpp**

#pragma once

#include "Dialog.h"

// ##################### ОПЕРАЦИИ НАД ВЕКТОРОМ ######################

template<typename T>

inline void ConsoleUI<T>::DoVectorOperation(int ch)

{

do {

ChoseVector();

ch = OperationSelect();

switch (ch)

{

case 1:

AddToVector(ch);

break;

case 2:

DelFromVector(ch);

break;

case 3:

VectorGet(ch);

break;

case 4:

VectorShow();

break;

case 0:

return;

}

} while (ch);

}

// ###################### ДАБАВЛЕНИЕ В ВЕКТОР #######################

template<typename T>

inline void ConsoleUI<T>::AddToVector(int ch)

{

T value;

int index;

auto iterator = Vector.begin();

do

{

ChoseVector();

cout << "Выберите способ добавления:\n";

cout << "1. Добавление в конец.\n";

cout << "2. Вставка.\n";

cout << "0. Отмена.\n";

cout << "Введите ваш ответ сюда: ";

cin >> ch;

} while (!CheckSelect(0, 2, ch));

switch (ch)

{

case 1:

cout << "\nВведите значение для добавления: ";

cin >> value;

Vector.push\_back(value);

OperationPass();

break;

case 2:

cout << "\nВведите значение для добавления: ";

cin >> value;

do

{

cout << "Введите индекс: ";

cin >> index;

if (index > Vector.size() || index < 0)

{

SetColor(12, 0);

cout << "Вы ошиблись. Введите число от 0 до " << Vector.size() << endl;

SetColor(7, 0);

}

} while (index > Vector.size() || index < 0);

std::advance(iterator, index);

Vector.insert(iterator, value);

OperationPass();

break;

case 0:

cout << endl;

OperationCancel();

}

}

// ###################### УДАЛЕНИЕ ИЗ ВЕКТОРА #######################

template<typename T>

inline void ConsoleUI<T>::DelFromVector(int ch)

{

if (NoVectorData())

return;

int index;

auto iterator = Vector.begin();

do

{

ChoseVector();

cout << "Выберите способ удаления:\n";

cout << "1. Удаление с конца.\n";

cout << "2. Удаление по индексу.\n";

cout << "0. Отмена.\n";

cout << "Введите ваш ответ сюда: ";

cin >> ch;

} while (!CheckSelect(0, 2, ch));

switch (ch)

{

case 1:

Vector.pop\_back();

cout << endl;

OperationPass();

break;

case 2:

do

{

cout << "Введите индекс: ";

cin >> index;

if (index >= Vector.size() || index < 0)

{

SetColor(12, 0);

cout << "Вы ошиблись. Введите число от 0 до " << Vector.size() - 1 << endl;

SetColor(7, 0);

}

} while (index > Vector.size() || index < 0);

std::advance(iterator, index);

Vector.erase(iterator);

OperationPass();

break;

case 0:

cout << endl;

OperationCancel();

}

}

// ################## ПОЛУЧЕНИЕ ДАННЫХ ИЗ ВЕКТОРА ###################

template<typename T>

inline T ConsoleUI<T>::VectorGet(int ch)

{

if (NoVectorData())

return T();

int index;

auto iterator = Vector.begin();

do

{

ChoseVector();

cout << "Выберите вариант получения:\n";

cout << "1. Вывести первый элемент.\n";

cout << "2. Вывести элемент по индексу.\n";

cout << "3. Вывести последний элемент.\n";

cout << "0. Отмена.\n";

cout << "Введите ваш ответ сюда: ";

cin >> ch;

} while (!CheckSelect(0, 3, ch));

switch (ch)

{

case 1:

cout << endl << Vector.front() << endl;

OperationPass();

return Vector.front();

break;

case 2:

do

{

cout << "Введите индекс: ";

cin >> index;

if (index >= Vector.size() || index < 0)

{

SetColor(12, 0);

cout << "Вы ошиблись. Введите число от 0 до " << Vector.size() - 1 << endl;

SetColor(7, 0);

}

} while (index > Vector.size() || index < 0);

cout << endl << Vector[index] << endl;

OperationPass();

return Vector[index];

break;

case 3:

cout << endl << Vector.back() << endl;

OperationPass();

return Vector.back();

break;

case 0:

cout << endl;

OperationCancel();

}

return T();

}

// ######################### ВЫВОД ВЕКТОРА ##########################

template<typename T>

inline void ConsoleUI<T>::VectorShow()

{

if (NoVectorData())

return;

cout << endl;

for (auto &i : Vector)

cout << i << " ";

cout << endl;

OperationPass();

}

// ####################### НЕТ ДАННЫХ ВЕКТОРА #######################

template<typename T>

inline bool ConsoleUI<T>::NoVectorData()

{

if (!Vector.size())

{

SetColor(12, 0);

cout << "\nДанные отсутствуют!\n";

SetColor(7, 0);

system("PAUSE");

system("cls");

return true;

}

return false;

}

// ######################### ВЫБОР ВЕКТОРА ##########################

template<typename T>

inline void ConsoleUI<T>::ChoseVector()

{

system("cls");

LabInfo();

cout << "Выбран контейнер: ";

SetColor(11, 0);

cout << "vector\n";

SetColor(7, 0);

}

**Текст файлу Set.hpp**

#pragma once

#include "Dialog.h"

// ####################### ОПЕРАЦИИ НАД СЕТОМ #######################

template<typename T>

inline void ConsoleUI<T>::DoSetOperation(int ch)

{

do {

ChoseSet();

ch = OperationSelect();

switch (ch)

{

case 1:

AddToSet();

break;

case 2:

DelFromSet();

break;

case 3:

SetGet();

break;

case 4:

SetShow();

break;

case 0:

return;

}

} while (ch);

}

// ######################## ДОБАВЛЕНИЕ В СЕТ ########################

template<typename T>

inline void ConsoleUI<T>::AddToSet()

{

T value;

cout << "\nВведите данные для добавления: ";

cin >> value;

Set.insert(value);

OperationPass();

}

// ######################## УДАЛЕНИЕ ИЗ СЕТА ########################

template<typename T>

inline void ConsoleUI<T>::DelFromSet()

{

if (NoSetData())

return;

T value;

cout << "\nВведите удаляемые данные: ";

cin >> value;

auto iterator = Set.find(value);

if (iterator == Set.end())

{

SetColor(12, 0);

cout << "Данных с таким значением нет в контейнере.\n";

SetColor(7, 0);

system("PAUSE");

system("cls");

return;

}

Set.erase(iterator);

OperationPass();

}

// #################### ПОЛУЧЕНИЕ ДАННЫХ ИЗ СЕТА ####################

template<typename T>

inline T ConsoleUI<T>::SetGet()

{

if (NoSetData())

return T();

T value;

cout << "\nВведите искомые данные: ";

cin >> value;

auto iterator = Set.find(value);

if (iterator == Set.end())

{

SetColor(12, 0);

cout << "Данных с таким значением нет в контейнере.\n";

SetColor(7, 0);

system("PAUSE");

system("cls");

return T();

}

cout << "Данные с таким значением есть в контейнере и успешно возвращены!\n";

OperationPass();

return \*iterator;

}

// ########################### ВЫВОД СЕТА ###########################

template<typename T>

inline void ConsoleUI<T>::SetShow()

{

if (NoSetData())

return;

cout << endl;

for (auto &i : Set)

cout << i << " ";

cout << endl;

OperationPass();

}

// ######################## НЕТ ДАННЫХ СЕТА #########################

template<typename T>

inline bool ConsoleUI<T>::NoSetData()

{

if (!Set.size())

{

SetColor(12, 0);

cout << "\nДанные отсутствуют!\n";

SetColor(7, 0);

system("PAUSE");

system("cls");

return true;

}

return false;

}

// ########################### ВЫБОР СЕТА ###########################

template<typename T>

inline void ConsoleUI<T>::ChoseSet()

{

system("cls");

LabInfo();

cout << "Выбран контейнер: ";

SetColor(11, 0);

cout << "set\n";

SetColor(7, 0);

}

**Текст файлу List.hpp**

#pragma once

#include "Dialog.h"

// ###################### ОПЕРАЦИИ НАД СПИСКОМ ######################

template<typename T>

inline void ConsoleUI<T>::DoListOperation(int ch)

{

do {

ChoseList();

ch = OperationSelect();

switch (ch)

{

case 1:

AddToList(ch);

break;

case 2:

DelFromList(ch);

break;

case 3:

ListGet(ch);

break;

case 4:

ListShow();

break;

case 0:

return;

}

} while (ch);

}

// ###################### ДОБАВЛЕНИЕ В СПИСОЕ #######################

template<typename T>

inline void ConsoleUI<T>::AddToList(int ch)

{

T value;

int index;

auto iterator = List.begin();

do

{

ChoseList();

cout << "Выберите способ добавления:\n";

cout << "1. Добавление в начало.\n";

cout << "2. Вставка.\n";

cout << "3. Добавление в конец.\n";

cout << "0. Отмена.\n";

cout << "Введите ваш ответ сюда: ";

cin >> ch;

} while (!CheckSelect(0, 3, ch));

switch (ch)

{

case 1:

cout << "\nВведите значение для добавления: ";

cin >> value;

List.push\_front(value);

OperationPass();

break;

case 2:

cout << "\nВведите значение для добавления: ";

cin >> value;

do

{

cout << "Введите индекс: ";

cin >> index;

if (index > List.size() || index < 0)

{

SetColor(12, 0);

cout << "Вы ошиблись. Введите число от 0 до " << List.size() << endl;

SetColor(7, 0);

}

} while (index > List.size() || index < 0);

std::advance(iterator, index);

List.insert(iterator, value);

OperationPass();

break;

case 3:

cout << "\nВведите значение для добавления: ";

cin >> value;

List.push\_back(value);

OperationPass();

break;

case 0:

cout << endl;

OperationCancel();

}

}

// ####################### УДАЛЕНИЕ ИЗ СПИСКА #######################

template<typename T>

inline void ConsoleUI<T>::DelFromList(int ch)

{

if (NoListData())

return;

int index;

auto iterator = List.begin();

do

{

ChoseList();

cout << "Выберите способ удаления:\n";

cout << "1. Удаление с начала.\n";

cout << "2. Удаление по индексу.\n";

cout << "3. Удаление с конца.\n";

cout << "0. Отмена.\n";

cout << "Введите ваш ответ сюда: ";

cin >> ch;

} while (!CheckSelect(0, 3, ch));

switch (ch)

{

case 1:

List.pop\_front();

cout << endl;

OperationPass();

break;

case 2:

do

{

cout << "Введите индекс: ";

cin >> index;

if (index >= List.size() || index < 0)

{

SetColor(12, 0);

cout << "Вы ошиблись. Введите число от 0 до " << List.size() - 1 << endl;

SetColor(7, 0);

}

} while (index > List.size() || index < 0);

std::advance(iterator, index);

List.erase(iterator);

OperationPass();

break;

case 3:

List.pop\_back();

cout << endl;

OperationPass();

break;

case 0:

cout << endl;

OperationCancel();

}

}

// ################### ПОЛУЧЕНИЕ ДАННЫХ ИЗ СПИСКА ###################

template<typename T>

inline T ConsoleUI<T>::ListGet(int ch)

{

if (NoListData())

return T();

int index;

auto iterator = List.begin();

do

{

ChoseList();

cout << "Выберите вариант получения:\n";

cout << "1. Вывести первый элемент.\n";

cout << "2. Вывести элемент по индексу.\n";

cout << "3. Вывести последний элемент.\n";

cout << "0. Отмена.\n";

cout << "Введите ваш ответ сюда: ";

cin >> ch;

} while (!CheckSelect(0, 3, ch));

switch (ch)

{

case 1:

cout << endl << List.front() << endl;

OperationPass();

return List.front();

break;

case 2:

do

{

cout << "Введите индекс: ";

cin >> index;

if (index >= List.size() || index < 0)

{

SetColor(12, 0);

cout << "Вы ошиблись. Введите число от 0 до " << List.size() - 1 << endl;

SetColor(7, 0);

}

} while (index > List.size() || index < 0);

advance(iterator, index);

cout << endl << \*iterator << endl;

OperationPass();

return \*iterator;

break;

case 3:

cout << endl << List.back() << endl;

OperationPass();

return List.back();

break;

case 0:

cout << endl;

OperationCancel();

}

return T();

}

// ########################## ВЫВОД СПИСКА ##########################

template<typename T>

inline void ConsoleUI<T>::ListShow()

{

if (NoListData())

return;

cout << endl;

for (auto &i : List)

cout << i << " ";

cout << endl;

OperationPass();

}

// ####################### НЕТ ДАННЫХ СПИСКА ########################

template<typename T>

inline bool ConsoleUI<T>::NoListData()

{

if (!List.size())

{

SetColor(12, 0);

cout << "\nДанные отсутствуют!\n";

SetColor(7, 0);

system("PAUSE");

system("cls");

return true;

}

return false;

}

// ########################## ВЫБОР СПИСКА ##########################

template<typename T>

inline void ConsoleUI<T>::ChoseList()

{

system("cls");

LabInfo();

cout << "Выбран контейнер: ";

SetColor(11, 0);

cout << "list\n";

SetColor(7, 0);

}

**Текст файлу Map.hpp**

#pragma once

#include "Dialog.h"

// ####################### ОПЕРАЦИИ НАД МАПОМ #######################

template<typename T>

inline void ConsoleUI<T>::DoMapOperation(int ch)

{

do {

ChoseMap();

ch = OperationSelect();

switch (ch)

{

case 1:

AddToMap();

break;

case 2:

DelFromMap();

break;

case 3:

MapGet();

break;

case 4:

MapShow();

break;

case 0:

return;

}

} while (ch);

}

// ######################## ДОБАВЛЕНИЕ В МАП ########################

template<typename T>

inline void ConsoleUI<T>::AddToMap()

{

T value;

int key;

cout << "\nВведите ключ: ";

cin >> key;

cout << "Введите значение для добавления: ";

cin >> value;

Map.emplace(key, value);

OperationPass();

}

// ######################## УДАЛЕНИЕ ИЗ МАПА ########################

template<typename T>

inline void ConsoleUI<T>::DelFromMap()

{

if (NoMapData())

return;

int key;

cout << "\nВведите ключ для удаления: ";

cin >> key;

auto iterator = Map.find(key);

if (iterator == Map.end())

{

SetColor(12, 0);

cout << "Данных с таким ключом нет в контейнере.\n";

SetColor(7, 0);

system("PAUSE");

system("cls");

return;

}

Map.erase(iterator);

OperationPass();

}

// #################### ПОЛУЧЕНИЕ ДАННЫХ ИЗ МАПА ####################

template<typename T>

inline T ConsoleUI<T>::MapGet()

{

if (NoMapData())

return T();

int key;

cout << "\nВведите ключ для поиска: ";

cin >> key;

auto iterator = Map.find(key);

if (iterator == Map.end())

{

SetColor(12, 0);

cout << "Данных с таким ключом нет в контейнере.\n";

SetColor(7, 0);

system("PAUSE");

system("cls");

return T();

}

cout << "Данные по ключу: " << Map[key] << endl;

OperationPass();

return Map[key];

}

// ########################### ВЫВОД МАПА ###########################

template<typename T>

inline void ConsoleUI<T>::MapShow()

{

if (NoMapData())

return;

cout << endl;

for (auto &i : Map)

cout << '[' << i.first << "] : " << i.second << endl;

OperationPass();

}

// ######################## НЕТ ДАННЫХ МАПА #########################

template<typename T>

inline bool ConsoleUI<T>::NoMapData()

{

if (!Map.size())

{

SetColor(12, 0);

cout << "\nДанные отсутствуют!\n";

SetColor(7, 0);

system("PAUSE");

system("cls");

return true;

}

return false;

}

// ########################### ВЫБОР МАПА ###########################

template<typename T>

inline void ConsoleUI<T>::ChoseMap()

{

system("cls");

LabInfo();

cout << "Выбран контейнер: ";

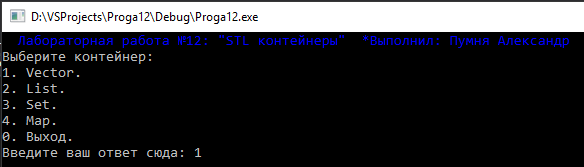
SetColor(11, 0);

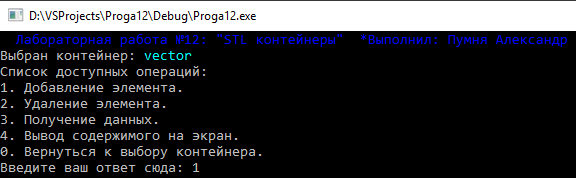
cout << "map\n";

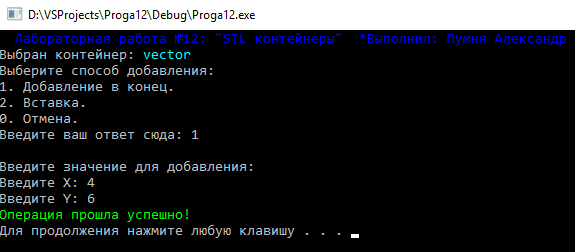
SetColor(7, 0);

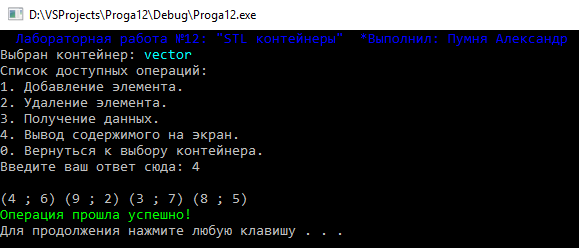
}

4 РЕЗУЛЬТАТ РОБОТИ ПРОГРАМИ

  
Рисунок 1 – Головне меню «Вибір контейнера»

  
Рисунок 2 – Меню «Вибір операцій»

  
Рисунок 3 – Пункт «Додавання елементів»

  
Рисунок 4 – Пункт «Виведення даних»

ВИСНОВКИ

При виконанні лабораторної роботи написано програму, яка демонструє роботу з контейнерами STL, використання їх основних функцій, головні відмінності структур контейнерів.