Государственное ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ   
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«Донецкий национальный технический университет»

Д09.03.04-ИИ.20.21/5771.ЛР

***Кафедра*** искусственного интеллекта  
 и системного анализа

Лабораторная работа №3

по дисциплине "Объектно ориентированное программирование"

на тему: "Одиночное и множественное наследование. Полиморфизм.”

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_ асс. А.П. Семёнова

(дата, подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ асс. И.В. Савицкая

(дата, подпись)

Выполнил:

\_\_\_\_\_\_\_\_\_\_\_\_\_ ст.гр. ПИ-20г И.Д.Любченко

(дата, подпись)

Донецк — 2021

Лабораторная работа №3

Тема: «Использование контейнеров, итераторов, алгоритмов библиотеки STL. Реализация функциональных объектов и предикатов.»

Цель: приобрести практические навыки работы с шаблонными функциями, шаблонным классом, стандартной библиотекой шаблонов, изучить определение последовательного и ассоциативного контейнеров, адаптера контейнеров, итератора,адаптера итераторов, функционального объекта, адаптера функций, алгоритмы.

Задание:

1. Контейнеры. Создать контейнер (вектор), добавить в него объекты следующей природы (Список акционеров и их взносов). Протестировать методы контейнера, скопировать его элементы в другой контейнер.

2. Итераторы. Протестировать работу итераторов различного типа, нпример, при вводе и просмотре всех элементов контейнера, добавлении, поиске, удалении и пр. (begin(), end())

3. Алгоритмы. Протестировать по 2 алгоритма каждой из 4 групп стандартной библиотеки алгоритмов. Указать, где используются стандартные объекты-функции. Использовать функциональный объект из пункта 1.

**Листинг программы**

#include <iostream>  
#include <vector>  
#include <algorithm>  
#include <numeric>  
  
  
using namespace std;  
  
int GetNumFromChar(char a){  
 return a - 48;  
}  
  
struct ShopWorkSchedule{  
 string Open;  
 string Closed;  
public:  
 friend ostream & operator<<(ostream & out, const ShopWorkSchedule & item){  
 out << item.Open << " - " << item.Closed;  
 return out;  
 }  
  
 int GetTime(const string & item ) const{  
 int time\_hours = GetNumFromChar(item[0]) \* 10 + GetNumFromChar(item[1]);  
 int time\_minutes = GetNumFromChar(item[3]) \* 10 + GetNumFromChar(item[4]) \* 10;  
 return time\_hours \* 60 + time\_minutes;  
 }  
  
 int GetWorkTime() const{  
 return GetTime(Closed) - GetTime(Open);  
 }  
  
 bool operator<(const ShopWorkSchedule & second)const {  
 return GetWorkTime() < second.GetWorkTime();  
 }  
  
 bool operator!=(const ShopWorkSchedule & second)const{  
 return GetWorkTime() != second.GetWorkTime();  
 }  
  
 bool operator==(const ShopWorkSchedule & second) const{  
 return GetWorkTime() == second.GetWorkTime();  
 }  
  
 friend int operator+(int first, const ShopWorkSchedule & item ){  
 return item.GetWorkTime() + first;  
 }  
  
 int operator+(int second){  
 ++second;  
 return second;  
 }  
 bool operator>(const ShopWorkSchedule & second) const {  
 return GetWorkTime() > second.GetWorkTime();  
 }  
};  
  
  
string GetCorrectTime(int num){  
 return to\_string(num / 10) + to\_string(num % 10);  
}  
  
const ShopWorkSchedule RandomWorkSchedule(){  
 string opened = GetCorrectTime(rand() % 12) + ':' + GetCorrectTime(rand() % 60);  
 int closed\_hours = rand() % 24;  
 string closed = GetCorrectTime((closed\_hours % 24 > 12)? closed\_hours: closed\_hours + 12)  
 + ':' + GetCorrectTime(rand() % 60);  
 return {opened, closed};  
}  
  
  
int main(){  
 vector<ShopWorkSchedule> Shops;  
  
 for(int i = 0; i < 5; ++i){  
 Shops.push\_back(RandomWorkSchedule());  
 }  
  
 {  
 cout << "Initialize Shops:\n";  
 for (const auto item: Shops){  
 cout << item << endl;  
 }  
 }  
  
 cout << endl;  
  
 {  
 vector<ShopWorkSchedule> ShopsCopy = Shops;  
 cout << "Copy to the ShopsCopy: ";  
 for (const auto & item: ShopsCopy){  
 cout << item << endl;  
 }  
 }  
  
 cout << endl;  
  
 {  
 less<ShopWorkSchedule> a;  
 cout << "less: " << a(Shops[0], Shops[2]) << endl;  
 }  
  
 cout << endl;  
  
 {  
 equal\_to<ShopWorkSchedule> a;  
 cout << "equal\_to: " << a(Shops[0], Shops[1]) << endl;  
 }  
 cout << endl;  
  
 {  
 cout << "count: " << count(Shops.begin(), Shops.end(), RandomWorkSchedule()) << endl;  
 }  
  
 cout << endl;  
  
 {  
 cout << "for\_each: \n";  
 for\_each(Shops.begin(), Shops.end(), [](const auto & item){  
 cout << item << endl;  
 });  
 }  
  
 cout << endl;  
  
 {  
 rotate(Shops.begin(), Shops.begin() + 3,Shops.end());  
 cout << "rotate: \n";  
 for (const auto & item: Shops){  
 cout << item << endl;  
 }  
 }  
  
 cout << endl;  
  
 {  
 reverse(Shops.begin(), Shops.end());  
 cout << "reverse: \n";  
 for (const auto & item: Shops){  
 cout << item << endl;  
 }  
 }  
  
 cout << endl;  
  
 {  
 sort(Shops.begin(), Shops.end());  
 cout << "sort: \n";  
 for (const auto & item: Shops){  
 cout << item<< endl;  
 }  
 }  
  
 cout << endl;  
  
 {  
 auto copy = Shops;  
 vector<ShopWorkSchedule> result;  
 cout << "binary\_search: " << binary\_search(Shops.begin(), Shops.end(), Shops[0]) << endl;  
 cout << "binary\_search: " << binary\_search(Shops.begin(), Shops.end(), RandomWorkSchedule()) << endl;  
 }  
  
 cout << endl;  
  
 {  
 auto copy = Shops;  
 cout << "accumulate: " << accumulate(Shops.begin(), Shops.end(), 0) << " minutes";  
 }  
 cout << endl;  
  
  
 return 0;  
}

**Экранные формы**

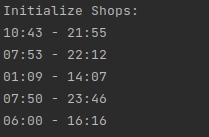


рис.1 Инициализация ветора

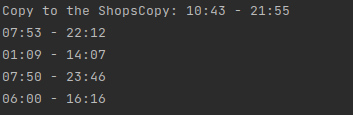


рис.2 Копирование в вектор b

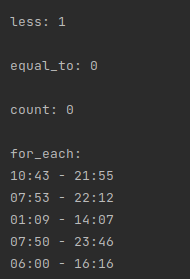


рис 3. Объекты-функции, итераторы, алгоритмы

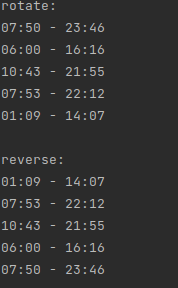


рис 4. reverse() и rotate()

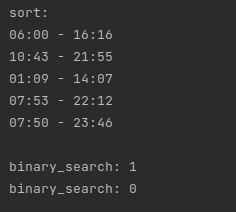


рис 5. sort() и binary\_search()



рис 6. accumulate()

**Вывод:** приобрел практические навыки работы с шаблонными функциями, шаблонным классом, стандартной библиотекой шаблонов, изучил определение последовательного и ассоциативного контейнеров, адаптера контейнеров, итератора,адаптера итераторов, функционального объекта, адаптера функций, алгоритмы.