Министерство науки и высшего образования

Российской Федерации

Федеральное государственное бюджетное образовательное   
учреждение высшего образования

«Новосибирский государственный технический университет»



Кафедра теоретической и прикладной информатики

### Практическая работа № 2 по дисциплине «Объектно-ориентированные технологии разработки программного обеспечения»

**Последовательные контейнеры STL**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Факультет: | ПМИ |  |  |
| Группа: | ПМИМ-01 |  |  |
| Студент: | Ничипиенко В.А. |  |  |
| Вариант: | 2 |  |  |
| Преподаватель: | Лисицин Д.В. |  |  |

Новосибирск

2021

1. **Цель работы:**

Изучить последовательные контейнеры и способы работы с ними.

1. **Задание:**
2. Изучить элементы стандартной библиотеки шаблонов С++: последовательные контейнеры, адаптеры контейнеров, функциональные объекты, функциональные адаптеры, лямбда-выражения.
3. Переработать программу из работы №1 так, чтобы объекты разработанного класса хранились в одном из следующих контейнеров STL: список (list), односвязный список (forward\_list). Использовать функции-члены контейнера для реализации следующих возможностей:
   1. Добавление объектов в контейнер.
   2. Поиск объектов в контейнере по различным критериям.
   3. Модификация объектов в контейнере (возможно, в соответствии с некоторыми критериями).
   4. Удаление объектов из контейнера по различным критериям.
   5. Вывод всех объектов контейнера, отсортированных по различным критериям, в файл и/или на экран.
4. При формировании критериев поиска, удаления, сортировки использовать функциональные объекты (пользовательские и/или стандартные) и/или лямбда-выражения. Атрибуты класса, хранящиеся как строки, но по смыслу являющиеся числами, должны участвовать в сравнениях как числа.
5. Оформить отчет. Отчет должен содержать постановку задачи, алгоритм, описание и текст разработанной программы, результаты тестирования (со скриншотами) и выводы.
6. Защитить работу, ответив на вопросы преподавателя.
7. **Вариант задания**

Класс с именем AIRPLANE, содержащий следующие атрибуты:

– название пункта назначения;

– номер рейса;

– время вылета;

– тип самолета.

Находить самолеты, отправляющиеся

– в заданный пункт назначения;

– в течение часа после заданного времени.

1. **Описание алгоритма**

string destination – Поле названия пункта назначения

string flightNumber – Поле номера рейса

string departureTime – Поле времени вылета

string airplaneType – Поле типа самолёта

AIRPLANE() – Конструктор класса по умолчанию

~AIRPLANE() – Деструктор класса по умолчанию

void setDestination(string destination) – Функция установки названия пункта назначения

void setFlightNumber(string flightNumber) – Функция установки номера рейса

void setDepartureTime(string departureTime) – Функция установки времени вылета

void setAirplaneType(string airplaneType) – Функция установки типа самолёта

string getDestination() – Функция получения названия пункта назначения

string getFlightNumber() – Функция получения номера рейса

string getDepartureTime()– Функция получения времени вылета

string getAirplaneType() – Функция получения типа самолёта

bool CheckTime(string time) – Функция проверки введённого времени на корректность

friend ostream& operator << (ostream& os, AIRPLANE& airplane) – Перегрузка оператора вывода для класса AIRPLANE

friend istream& operator >> (istream& in, AIRPLANE& airplane) – Перегрузка оператора ввода для класса AIRPLANE

void Input(list<AIRPLANE>& Airplane) – Функция ввода данных

void Sort(list<AIRPLANE>& Airplane) – Функция сортировки данных

void Output(list<AIRPLANE>& Airplane) – Функция вывода данных

void Add(list<AIRPLANE>& Airplane) – Функция добавления данных

void Delete(list<AIRPLANE>& Airplane) – Функция удаления данных

void Edit(list<AIRPLANE>& Airplane) – Функция редактирования данных

void Select(list<AIRPLANE>& Airplane) – Функция поиска данных

1. **Текст программы**

**AIRPLANE.h**

#pragma once

#include <sstream>

#include <iostream>

#include <string>

#include <vector>

using namespace std;

#pragma region Custom flags

const int specialIndex = ios\_base::xalloc();

template <typename charT, typename traits>

inline basic\_ostream <charT, traits>& rowOutput(basic\_ostream<charT, traits>& out)

{

out.iword(specialIndex) = true;

return out;

}

template <typename charT, typename traits>

inline basic\_ostream<charT, traits>& columnOutput(basic\_ostream<charT, traits>& out)

{

out.iword(specialIndex) = false;

return out;

}

#pragma endregion

class AIRPLANE

{

private:

#pragma region Fields

string destination;

string flightNumber;

string departureTime;

string airplaneType;

#pragma endregion

public:

#pragma region Ctor

AIRPLANE();

~AIRPLANE();

#pragma endregion

#pragma region Methods

void setDestination(string destination);

void setFlightNumber(string flightNumber);

void setDepartureTime(string departureTime);

void setAirplaneType(string airplaneType);

string getDestination();

string getFlightNumber();

string getDepartureTime();

string getAirplaneType();

bool CheckTime(string time);

friend istream& operator >> (istream& in, AIRPLANE& airplane);

template <typename charT, typename traits>

friend basic\_ostream<charT, traits>& operator << (basic\_ostream<charT, traits>& out, const AIRPLANE& airplane);

#pragma endregion

};

#pragma region Operators

template <typename charT, typename traits>

inline basic\_ostream<charT, traits>& operator << (basic\_ostream<charT, traits>& out, const AIRPLANE& airplane)

{

basic\_ostringstream<charT, traits> s;

s.copyfmt(out);

s.width(0);

if (s.iword(specialIndex))

{

s << "Destination - " << airplane.destination << endl;

s << "Flight number - " << airplane.flightNumber << endl;

s << "Departure time - " << airplane.departureTime << endl;

s << "Airplane type - " << airplane.airplaneType << endl;

}

else

{

s << airplane.destination << " " << airplane.flightNumber << " " << airplane.departureTime << " " << airplane.airplaneType;

}

out << s.str();

return out;

}

#pragma endregion

**AIRPLANE.cpp**

#include "AIRPLANE.h"

#pragma region Ctor

AIRPLANE::AIRPLANE(){}

AIRPLANE::~AIRPLANE(){}

#pragma endregion

#pragma region Methods

void AIRPLANE::setDestination(string destination)

{

this->destination = destination;

}

void AIRPLANE::setFlightNumber(string flightNumber)

{

this->flightNumber = flightNumber;

}

void AIRPLANE::setDepartureTime(string departureTime)

{

if (CheckTime(departureTime) == true)

{

this->departureTime = departureTime;

}

}

void AIRPLANE::setAirplaneType(string airplaneType)

{

this->airplaneType = airplaneType;

}

string AIRPLANE::getDestination()

{

return destination;

}

string AIRPLANE::getFlightNumber()

{

return flightNumber;

}

string AIRPLANE::getDepartureTime()

{

return departureTime;

}

string AIRPLANE::getAirplaneType()

{

return airplaneType;

}

bool AIRPLANE::CheckTime(string time)

{

vector<string> arr;

string inputStr = time;

string delim = ":";

size\_t prev = 0;

size\_t next;

size\_t delta = delim.length();

while ((next = inputStr.find(delim, prev)) != string::npos)

{

string tmp = inputStr.substr(prev, next - prev);

arr.push\_back(inputStr.substr(prev, next - prev));

prev = next + delta;

}

arr.push\_back(inputStr.substr(prev));

int watches = stoi(arr[0]);

int minutes = stoi(arr[1]);

if (watches >= 24 || watches < 0 || minutes >= 60 || minutes < 0 || arr.size() != 2)

{

cout << "Incorrect time entered" << endl;

return false;

}

return true;

}

#pragma endregion

#pragma region Operators

istream& operator >> (istream& in, AIRPLANE& airplane)

{

string destination, flightNumber, departureTime, airplaneType;

in >> destination >> flightNumber >> departureTime >> airplaneType;

if (airplane.CheckTime(departureTime) == true)

{

airplane.setDestination(destination);

airplane.setFlightNumber(flightNumber);

airplane.setDepartureTime(departureTime);

airplane.setAirplaneType(airplaneType);

}

else

{

in.clear(ios::failbit);

return in;

}

return in;

}

#pragma endregion

**OOTRPO.cpp**

#include <iostream>

#include <fstream>

#include <string>

#include "AIRPLANE.h"

#include <list>

#include <vector>

using namespace std;

void Input(list<AIRPLANE>& Airplane)

{

cout << "Input method:" << endl;

cout << "1 - manual input, 2 - input from file, 3 - generate array" << endl;

int command;

cin >> command;

if (command == 1)

{

cout << "Enter the count of plane tickets" << endl;

int size = 0;

cin >> size;

cout << "Enter: name of the destination, flight number, departure time, aircraft type" << endl;

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

{

AIRPLANE newAirplane;

cin >> newAirplane;

Airplane.push\_back(newAirplane);

cout << i + 1 << " of " << size << " tickets entered" << endl;

}

}

if (command == 2)

{

ifstream read;

read.open("Input.txt");

while (!read.eof())

{

AIRPLANE newAirplane;

read >> newAirplane;

cout << newAirplane << endl;

Airplane.push\_back(newAirplane);

}

read.close();

}

if (command == 3)

{

cout << "Enter the count of plane tickets" << endl;

int size = 0;

cin >> size;

string cityArray[3] = { "NSK", "SPB", "KRD" };

string timeArray[5] = { "10:25", "11:45", "12:35", "13:00", "14:00" };

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

{

cout << "--- Flight ticket number [" << i << "] ---" << endl;

AIRPLANE newAirplane;

newAirplane.setDestination(cityArray[rand() % 3]);

newAirplane.setFlightNumber(to\_string(rand() % 1000));

newAirplane.setDepartureTime(timeArray[rand() % 5]);

newAirplane.setAirplaneType("Airplane");

cout << rowOutput << newAirplane << endl;

Airplane.push\_back(newAirplane);

}

}

}

void Sort(list<AIRPLANE>& Airplane)

{

Airplane.sort([](AIRPLANE& airplane1, AIRPLANE& airplane2)

{

int number1 = stoi(airplane1.getFlightNumber());

int number2 = stoi(airplane2.getFlightNumber());

return number1 < number2;

});

}

void Output(list<AIRPLANE>& Airplane)

{

Sort(Airplane);

cout << "Output method" << endl;

cout << "1 - Console output, 2 - Output to file" << endl;

int command;

cin >> command;

cout << "Output format" << endl;

cout << "1 - Output in row, 2 - Output in column" << endl;

int format;

cin >> format;

if (command == 1)

{

if (format == 1)

{

for (auto i = Airplane.begin(); i != Airplane.end(); i++)

{

cout << rowOutput << \*i << endl;

}

}

if (format == 2)

{

for (auto i = Airplane.begin(); i != Airplane.end(); i++)

{

cout << columnOutput << \*i << endl;

}

}

cout << "Successfully outputed" << endl;

}

if (command == 2)

{

ofstream write;

write.open("Output.txt", ofstream::out | ofstream::trunc);

if (format == 1)

{

for (auto i = Airplane.begin(); i != Airplane.end(); i++)

{

write << rowOutput << \*i << endl;

}

}

if (format == 2)

{

for (auto i = Airplane.begin(); i != Airplane.end(); i++)

{

write << columnOutput << \*i << endl;

}

}

write.close();

cout << "Successfully outputed" << endl;

}

}

void Add(list<AIRPLANE>& Airplane)

{

cout << "1 - Adding from the file, 2 - Adding from the keyboard" << endl;

int command;

cin >> command;

if (command == 1)

{

string fileName = "Add.txt";

ifstream read;

read.open(fileName);

while (!read.eof())

{

AIRPLANE newAirplane;

read >> newAirplane;

cout << rowOutput << newAirplane << endl;

Airplane.push\_back(newAirplane);

}

read.close();

cout << "Successfully added" << endl;

}

if (command == 2)

{

AIRPLANE newAirplane;

cout << "Enter new elem: name of the destination, flight number, departure time, aircraft type" << endl;

cin >> newAirplane;

Airplane.push\_back(newAirplane);

cout << "Successfully added" << endl;

}

}

void Delete(list<AIRPLANE>& Airplane)

{

cout << "Sign for removal - Flight number" << endl;

string sign;

cout << "Enter Flight number for removal: " << endl;

cin >> sign;

list<AIRPLANE>::iterator iter;

iter = find\_if(Airplane.begin(), Airplane.end(), [&](AIRPLANE currentAirplane)

{

if (currentAirplane.getFlightNumber() == sign)

{

return true;

}

else

{

return false;

}

});

Airplane.erase(iter);

cout << "Successfully deleted" << endl;

}

void Edit(list<AIRPLANE>& Airplane)

{

cout << " Enter the ticket number to change " << endl;

string sign;

cin >> sign;

for (auto i = Airplane.begin(); i != Airplane.end(); i++)

{

AIRPLANE newAirplane = \*i;

if (newAirplane.getFlightNumber() == sign)

{

cout << "Enter new ticket parameters" << endl;

cout << "Name of the destination, flight number, departure time, aircraft type" << endl;

cin >> \*i;

}

}

cout << "Successfully edited" << endl;

}

void Select(list<AIRPLANE>& Airplane)

{

cout << "Enter destination for search" << endl;

string sign;

cin >> sign;

cout << "Search 1, Airplanes to destination = " << endl;

list<AIRPLANE>::iterator iter;

iter = Airplane.begin();

while (iter != Airplane.end())

{

iter = find\_if(iter, Airplane.end(), [&](AIRPLANE currentAirplane)

{

if (currentAirplane.getDestination() == sign)

{

cout << rowOutput << currentAirplane << endl;

return true;

}

else

{

return false;

}

});

iter++;

}

cout << "Enter time for search" << endl;

cin >> sign;

cout << "Search 2, Airplanes that departed within an hour after the specified time = " << endl;

time\_t signTime = (time\_t)atoi(sign.c\_str());

iter = Airplane.begin();

while (iter != Airplane.end())

{

iter = find\_if(iter, Airplane.end(), [&](AIRPLANE currentAirplane)

{

time\_t time = (time\_t)atoi(currentAirplane.getDepartureTime().c\_str()) - signTime;

if ((time <= 1) && (time >= 0))

{

cout << rowOutput << currentAirplane << endl;

return true;

}

else

{

return false;

}

});

iter++;

}

}

void main()

{

list<AIRPLANE> airplane;

Input(airplane);

cout << "Array is full, what's next?" << endl;

cout << "1 - Input " << endl;

cout << "2 - Output" << endl;

cout << "3 - Add" << endl;

cout << "4 - Delete" << endl;

cout << "5 - Edit" << endl;

cout << "6 - Select" << endl;

cout << "7 - Help" << endl;

int command;

while (true)

{

cin >> command;

if (command == 1)

{

cout << "Input" << endl;

Input(airplane);

}

if (command == 2)

{

cout << "Output" << endl;

Output(airplane);

}

if (command == 3)

{

cout << "Add" << endl;

Add(airplane);

}

if (command == 4)

{

cout << "Delete" << endl;

Delete(airplane);

}

if (command == 5)

{

cout << "Edit" << endl;

Edit(airplane);

}

if (command == 6)

{

cout << "Select" << endl;

Select(airplane);

}

if (command == 7)

{

cout << "Array is full, what's next?" << endl;

cout << "1 - Input " << endl;

cout << "2 - Output" << endl;

cout << "3 - Add" << endl;

cout << "4 - Delete" << endl;

cout << "5 - Edit" << endl;

cout << "6 - Select" << endl;

cout << "7 - Help" << endl;

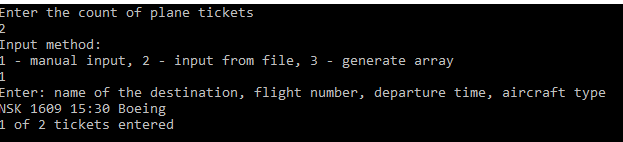
}

}

}

1. **Примеры работы программы**

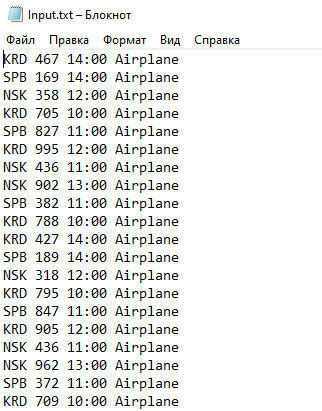
Ручной ввод данных

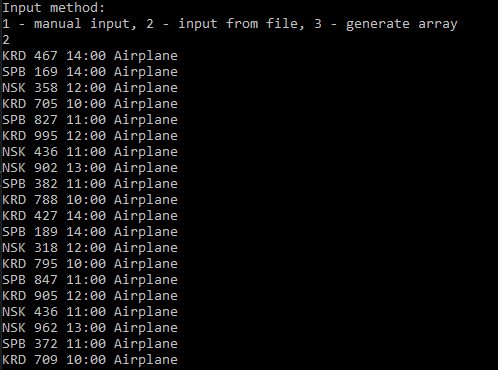
****

Попытка ввести некорректное время

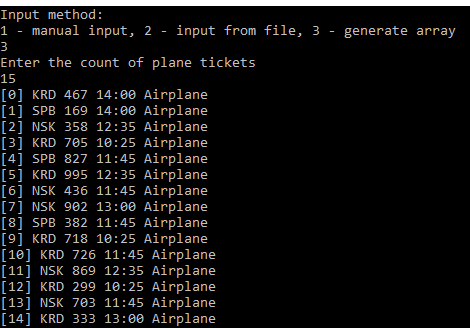
****

Ввод данных из файла

****

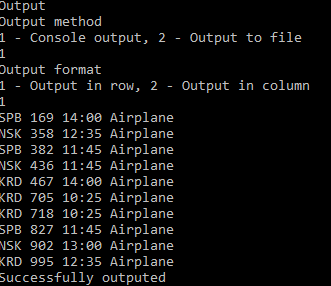
****

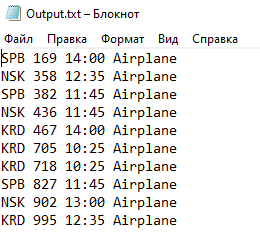
Ввод случайных сгенерированных данных

****

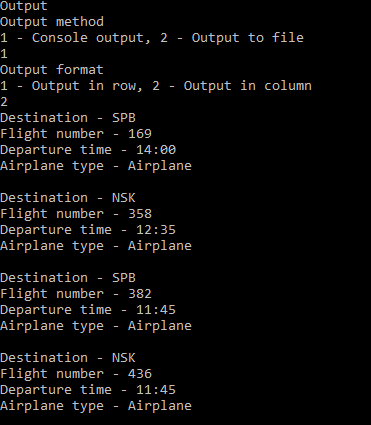
Вывод данных отсортированных по номеру рейса

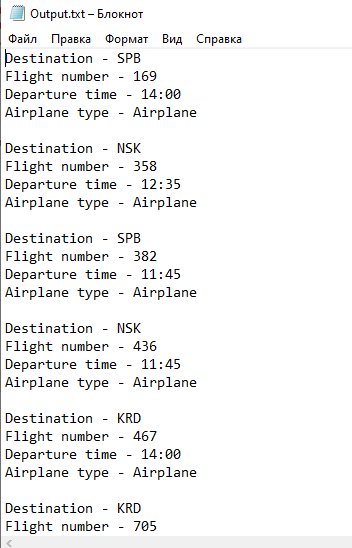
Первый формат вывода – вывод в строку



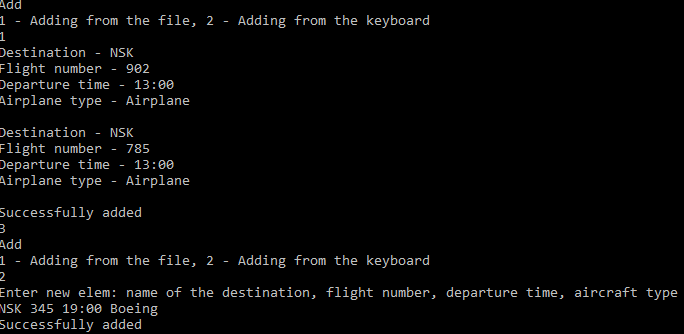


Второй формат вывода данных – в столбец

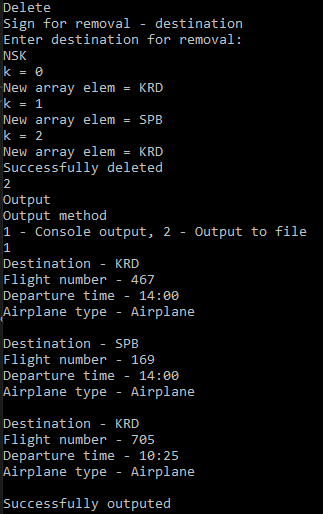
****

****

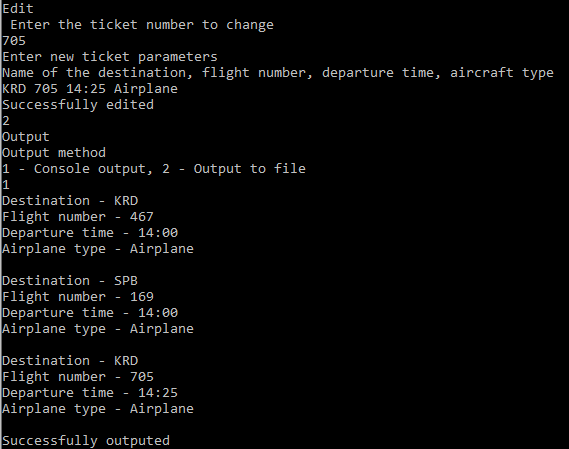
Добавление новых данных из файла и из ввода с клавиатуры

****

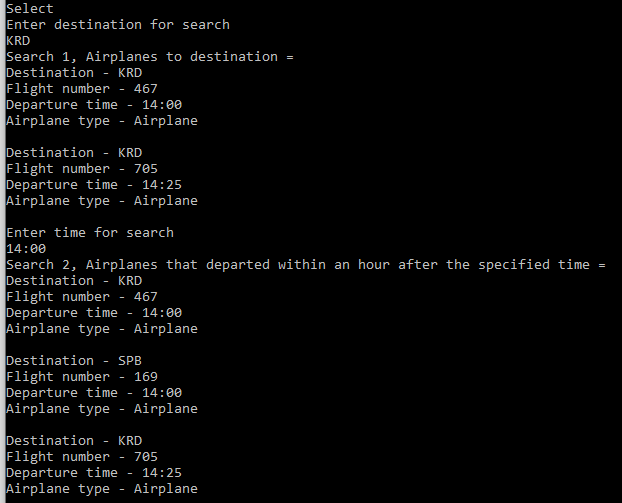
Удаление данных по названию места назначения

****

Редактирование данных

****

Поиск данных

****