Темнохуд Михаил, ПЗПИ-16-2

Лабораторная работа №2

# Задание:

Для обраного завдання реалізувати необхідні для коректної роботи програми (з різними варіантами ініціалізації об'єктів) конструктори, деструктори, функції та перевантажити оператори, необхідні в класах (відповідно до концепції та поведінки об'єктів реального світу).

15. Потік студентів (включає в себе дане класу Група);

# Код программы:

// lab3.cpp: Определяет точку входа для приложения.

//

#include "stdafx.h"

#include "Flow.h"

int main()

{

setlocale(LC\_ALL, "Russian");

Student people[] = {

{ 1, (char\*)"Понедельников Геннадий Михайлович" },

{ 2, (char\*)"Белоус Полина" },

{ 3, (char\*)"Шевяков Вячеслав Игоревич" },

{ 4, (char\*)"Цыбульник Богдан Валентинович" },

{ 5, (char\*)"Темнохуд Михаил Игоревич" },

{ 6, (char\*)"Валуйский Владислав Евгеньевич" },

{ 7, (char\*)"Рублёв Андроид" },

{ 8, (char\*)"Губарь Сергей" }

};

char groupname[100];

cin >> groupname;

Group group1((char\*)groupname);

group1.AddStudent(people[0]);

group1.AddStudent(people[1]);

group1.PrintStudents();

group1.RenameGroup((char\*)"ПЗПИ-16-1");

group1.PrintStudents();

Group group2((char\*)"ПЗПИ-16-2");

group2.AddStudent(people[2]);

group2.AddStudent(people[3]);

group2.AddStudent(people[4]);

group2.AddStudent(people[5]);

group2.PrintStudents();

group2.RemoveStudent(4);

group2.PrintStudents();

Group group3((char\*)"ПЗПИ-16-3");

group3.AddStudent(people[6]);

group3.AddStudent(people[7]);

group3.PrintStudents();

Flow flow((char\*)"ПЗПИ");

flow.AddGroup(group1);

flow.AddGroup(group2);

flow.AddGroup(group3);

flow.Get(0)->RenameGroup((char\*)"PZPI-16-1");

flow.PrintInfo();

flow.PrintGroups();

flow.PrintStudents();

Group\* firstGroup = flow.Get(1);

firstGroup->RemoveStudent(6);

flow.PrintInfo();

flow.PrintGroups();

flow.PrintStudents();

system("pause");

return 0;

}

// Group.h: Определяет точку входа для приложения.

//

#pragma once

#define STUDENT\_NULL {-1, "null"}

struct Student

{

ulong id;

char\* name;

};

class Group

{

char\* \_code;

ulong \_countStudents;

ulong \_maxCount;

Student\* \_students;

public:

Group();

Group(char\* code);

Group(char\* code, Student\* students, ulong count);

Group(const Group &obj);

Group& operator=(Group &group);

bool AddStudent(Student student);

bool HasStudent(ulong id);

bool RemoveStudent(ulong id);

bool RenameGroup(char\* newCode);

bool IsEmpty();

ulong StudentsCount();

Student\* Get(int index);

char\* GetCode();

void PrintInfo();

void PrintStudents();

~Group();

private:

void ExpandArray();

};

// Flow.h: Определяет точку входа для приложения.

//

#pragma once

#include "Group.h"

class Flow

{

char\* \_code;

ulong \_countGroups;

ulong \_maxCount;

Group\* \_groups;

public:

Flow();

Flow(char\* code);

Flow(const Flow &obj);

bool AddGroup(Group student);

bool RemoveGroup(char\* groupCode);

bool RenameFlow(char\* newCode);

bool HasGroup(char\* code);

bool HasStudent(ulong id);

ulong GroupsCount();

ulong StudentsCount();

bool IsEmpty();

bool HasStudents();

Group\* Get(int index);

char\* GetCode();

void PrintInfo();

void PrintGroups();

void PrintStudents();

~Flow();

private:

void ExpandArray();

};

// Group.cpp: Определяет точку входа для приложения.

//

#pragma once

#define STUDENT\_NULL {-1, "null"}

struct Student

{

ulong id;

char\* name;

};

class Group

{

char\* \_code;

ulong \_countStudents;

ulong \_maxCount;

Student\* \_students;

public:

Group();

Group(char\* code);

Group(char\* code, Student\* students, ulong count);

Group(const Group &obj);

Group& operator=(Group &group);

bool AddStudent(Student student);

bool HasStudent(ulong id);

bool RemoveStudent(ulong id);

bool RenameGroup(char\* newCode);

bool IsEmpty();

ulong StudentsCount();

Student\* Get(int index);

char\* GetCode();

void PrintInfo();

void PrintStudents();

~Group();

private:

void ExpandArray();

};

// Flow.cpp: Определяет точку входа для приложения.

//

#include "stdafx.h"

#include "Flow.h"

Flow::Flow()

{

\_code = (char\*)"";

\_countGroups = 0;

\_maxCount = 30;

\_groups = new Group[\_maxCount];

}

Flow::Flow(char\* code)

{

\_code = new char[strlen(code) + 1];

strcpy(\_code, code);

\_countGroups = 0;

\_maxCount = 30;

\_groups = new Group[\_maxCount];

}

Flow::Flow(const Flow &obj)

{

Flow newFlow(\_code);

for (int i = 0; i < \_countGroups; i++)

{

newFlow.AddGroup(\_groups[i]);

}

}

bool Flow::AddGroup(Group group)

{

if (\_countGroups + 1 >= \_maxCount)

ExpandArray();

\_groups[\_countGroups] = group;

\_countGroups++;

return true;

}

bool Flow::HasGroup(char\* groupCode)

{

for (int i = 0; i < \_countGroups; i++)

{

if (\_groups[i].GetCode() == groupCode)

{

return true;

}

}

return false;

}

bool Flow::HasStudent(ulong id)

{

for (int i = 0; i < \_countGroups; i++)

{

if (\_groups[i].HasStudent(id))

{

return true;

}

}

return false;

}

bool Flow::RemoveGroup(char\* groupCode)

{

if (IsEmpty())

return false;

int index = -1;

for (int i = 0; i < \_countGroups; i++)

{

if (\_groups[i].GetCode() == groupCode)

{

index = i;

break;

}

}

if (index != -1)

{

for (int i = index; i < \_countGroups - 1; i++)

{

\_groups[i] = \_groups[i + 1];

}

\_countGroups--;

return true;

}

else

{

return false;

}

}

bool Flow::RenameFlow(char\* newCode)

{

if (newCode == "")

return false;

\_code = new char[strlen(newCode) + 1];

strcpy(\_code, newCode);

return true;

}

ulong Flow::GroupsCount()

{

return \_countGroups;

}

ulong Flow::StudentsCount()

{

ulong count = 0;

for (int i = 0; i < \_countGroups; i++)

{

count += \_groups[i].StudentsCount();

}

return count;

}

bool Flow::IsEmpty()

{

return \_countGroups == 0;

}

bool Flow::HasStudents()

{

ulong groupsCount = GroupsCount();

for (int i = 0; groupsCount; i++)

{

if (\_groups[i].StudentsCount() > 0)

{

return true;

}

}

return false;

}

Group\* Flow::Get(int index)

{

return &\_groups[index];

}

char\* Flow::GetCode()

{

return \_code;

}

void Flow::PrintInfo()

{

cout << "== Flow Info " << endl;

cout << "Code : " << \_code << endl;

cout << "Groups : " << \_countGroups << endl;

cout << "Students: " << StudentsCount() << endl;

cout << endl;

}

void Flow::PrintGroups()

{

if (IsEmpty())

{

cout << "Flow has no groups";

return;

}

cout << "== Flow: " << \_code << endl;

cout << "num: count, name" << endl;

for (int i = 0; i < \_countGroups; i++)

{

Group\* currGroup = Get(i);

cout << setw(3) << i + 1 << ": "

<< setw(8) << currGroup->StudentsCount() << ", "

<< currGroup->GetCode() << endl;

}

cout << endl;

}

void Flow::PrintStudents()

{

if (IsEmpty())

{

cout << "Flow has no groups";

return;

}

if (!HasStudents())

{

cout << "Flow has no members";

return;

}

ulong totalCount = 1;

cout << "== Flow: " << \_code << endl;

cout << "num: id, name" << endl;

for (int i = 0; i < \_countGroups; i++)

{

Group\* currGroup = Get(i);

ulong studentCount = currGroup->StudentsCount();

for (int j = 0; j < studentCount; j++)

{

Student\* currStudent = currGroup->Get(j);

cout << setw(3) << totalCount << ": " << setw(8) << currStudent->id << ", " << currStudent->name << endl;

totalCount++;

}

}

cout << endl;

}

Flow::~Flow()

{

\_code = nullptr;

delete[] \_groups;

}

void Flow::ExpandArray()

{

int newSize = \_countGroups \* 2;

Group\* temp = new Group[newSize];

for (int i = 0; i < \_countGroups; i++)

{

temp[i] = \_groups[i];

}

\_groups = temp;

\_maxCount = newSize;

delete[] temp;

}

# Результат выполнение:

