Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

# Лабораторная работа №6. "Наследование и виртуальные функции”

Выполнил:

Ст. 2 курса гр. АС-53

Альциванович Н.В.

Проверила:

Давидюк Ю. И.

Брест, 2020

**1. Цель.** Получить практические навыки создания иерархии классов и использования статических компонентов класса.

**2. Постановка задачи (Вариант 1)**

Написать программу, в которой создается иерархия классов. Включить полиморфные объекты в связанный список, используя статические компоненты класса. Показать использование виртуальных функций.

**студент, преподаватель, персона, завкафедрой;**

Классы:

* Персона
  + Завкафедрой
  + Преподаватель
  + Студент

Классы выстроены в соответствии с их иерархией.

Конструкторы:

* Без параметров
* Копирования
* С параметрами

Деструктор виртуальный.

Функции:

* Добавления в лист(Вызывается при создании объекта класса)
* Вывода информации
* Возвращения и изменения параметров

3. Иерархия классов в виде графа:

* Person
  + HeadofDepartment
  + Teacher
  + Student

4. Определение пользовательских классов с комментариями.

//Базовый класс «Персона»

#pragma once

#include <iostream>

#include <string>

class Person {

protected:

std::string firstName;

std::string secondName;

std::string lastName;

public:

Person();

Person(

std::string,

std::string,

std::string

);

Person(const Person&);

virtual void Show() = 0;

virtual void Add() = 0;

~Person();

};

Person::Person() {

firstName = "UndefinedFirstName";

secondName = "UndefinedSecondName";

lastName = "UndefinedLastName";

}

Person::Person(std::string \_firstName, std::string \_secondName, std::string \_lastName) {

firstName = \_firstName;

secondName = \_secondName;

lastName = \_lastName;

};

Person::Person(const Person& \_Person) { };

Person::~Person() { };

// Класс «Студент»

#pragma once

#include "Person.h"

#include "List.h"

class Student :public Person {

protected:

int numberOfGroup;

public:

Student();

Student(

std::string,

std::string,

std::string,

int

);

void Show();

void Add();

Student(const Student&);

~Student();

};

Student::Student() {

numberOfGroup = 0;

};

Student::Student(std::string \_firstName, std::string \_secondName, std::string \_lastName, int \_numberOfGroup)

:Person(\_firstName, \_secondName, \_lastName) {

numberOfGroup = \_numberOfGroup;

};

void Student::Show() {

std::cout << "Student:\n\t" << Person::secondName << " " << Person::firstName << " " << Person::lastName << " | [Group number: " << numberOfGroup << "]\n" << std::endl;

};

Student::Student(const Student& \_Student) { };

Student::~Student() {};

void Student::Add() {

list\* Temp = new list;

Temp->Data = new Student(

firstName,

secondName,

lastName,

numberOfGroup

);

Temp->Next = nullptr;

if (List::\_List.GetHead() != nullptr) {

List::\_List.GetTail()->Next = Temp;

List::\_List.SetTail(Temp);

}

else {

List::\_List.SetHead(Temp);

List::\_List.SetTail(Temp);

}

} ;

//Класс «Преподаватель»

#pragma once

#include "Person.h"

#include "List.h"

class Teacher :public Person {

protected:

std::string subject;

public:

Teacher();

Teacher(

std::string,

std::string,

std::string,

std::string

);

void Add();

void Show();

Teacher(const Teacher&);

~Teacher();

};

Teacher::Teacher() {

subject = "UndefinedSubject";

};

Teacher::Teacher(std::string \_firstName, std::string \_secondName, std::string \_lastName, std::string \_subject)

:Person(\_firstName, \_secondName, \_lastName) {

subject = \_subject;

};

void Teacher::Show() {

std::cout << "Teacher:\n\t" << Person::secondName << " " << Person::firstName << " " << Person::lastName << " | Subject: " << subject << std::endl << std::endl;

};

Teacher::Teacher(const Teacher& \_Teacher) { };

Teacher::~Teacher() {};

void Teacher::Add() {

list\* Temp = new list;

Temp->Data = new Teacher(

firstName,

secondName,

lastName,

subject

);

Temp->Next = nullptr;

if (List::\_List.GetHead() != nullptr) {

List::\_List.GetTail()->Next = Temp;

List::\_List.SetTail(Temp);

}

else {

List::\_List.SetHead(Temp);

List::\_List.SetTail(Temp);

}

}

//Класс «Завкафедрой»

#pragma once

#include "Teacher.h"

#include "List.h"

class HeadOfDepartment :public Teacher {

protected:

std::string department;

public:

HeadOfDepartment();

HeadOfDepartment(

std::string,

std::string,

std::string,

std::string,

std::string

);

void Add();

void Show();

HeadOfDepartment(const HeadOfDepartment&);

~HeadOfDepartment();

};

HeadOfDepartment::HeadOfDepartment() {

department = "UndefinedDepartment";

};

HeadOfDepartment::HeadOfDepartment(std::string \_firstName, std::string \_secondName, std::string \_lastName, std::string \_subject, std::string \_department)

:Teacher(\_firstName, \_secondName, \_lastName, \_subject) {

department = \_department;

};

void HeadOfDepartment::Show() {

std::cout << "HeadOfDepartment:\n\t" << Person::secondName << " " << Person::firstName << " " << Person::lastName << " | Subject: " << Teacher::subject << " | Department: " << department << std::endl << std::endl;

};

HeadOfDepartment::HeadOfDepartment(const HeadOfDepartment& \_HeadOfDepartment) { };

HeadOfDepartment::~HeadOfDepartment() {};

void HeadOfDepartment::Add() {

list\* Temp = new list;

Temp->Data = new HeadOfDepartment(

firstName,

secondName,

lastName,

subject,

department

);

Temp->Next = nullptr;

if (List::\_List.GetHead() != nullptr) {

List::\_List.GetTail()->Next = Temp;

List::\_List.SetTail(Temp);

}

else {

List::\_List.SetHead(Temp);

List::\_List.SetTail(Temp);

}

}

// Реализация функций классов

#pragma once

#include <iostream>

struct list {

Person\* Data;

list\* Next;

};

class List {

private:

list\* \_Tail;

static list\* \_Head;

public:

static List \_List;

List();

List(const List&);

~List();

list\* GetHead() const;

list\* GetTail() const;

void SetHead(list\*);

void SetTail(list\*);

static void CheckList();

};

List List::\_List = { };

list\* List::\_Head = nullptr;

List::List() {

\_Head = nullptr;

\_Tail = nullptr;

}

List::List(const List& tList) { }

List::~List() {

while (\_Head != nullptr) {

list\* Temp = \_Head->Next;

delete \_Head;

\_Head = Temp;

}

}

list\* List::GetHead() const { return \_Head; }

list\* List::GetTail() const { return \_Tail; }

void List::SetHead(list\* Head) { this->\_Head = Head; }

void List::SetTail(list\* Tail) { this->\_Tail = Tail; }

void List::CheckList() {

if (nullptr == List::\_Head) {

std::cout << "Items not found" << std::endl;

}

else {

list\* Temp = new list;

Temp = List::\_Head;

while (Temp != nullptr) {

Temp->Data->Show();

Temp = Temp->Next;

}

}

}

5. Реализация конструкторов с параметрами и деструктора.

* Для класса Person:

class Person {

protected:

std::string firstName;

std::string secondName;

std::string lastName;

public:

Person();

Person(

std::string,

std::string,

std::string

);

Person(const Person&);

virtual void Show() = 0;

virtual void Add() = 0;

~Person();

};

* Для класса HeadofDepartment:

class HeadOfDepartment :public Teacher {

protected:

std::string department;

public:

HeadOfDepartment();

HeadOfDepartment(

std::string,

std::string,

std::string,

std::string,

std::string

);

void Add();

void Show();

HeadOfDepartment(const HeadOfDepartment&);

~HeadOfDepartment();

};

* Для класса Teacher:

class Teacher :public Person {

protected:

std::string subject;

public:

Teacher();

Teacher(

std::string,

std::string,

std::string,

std::string

);

void Add();

void Show();

Teacher(const Teacher&);

~Teacher();

};

* Для класса Student:

class Student :public Person {

protected:

int numberOfGroup;

public:

Student();

Student(

std::string,

std::string,

std::string,

int

);

void Show();

void Add();

Student(const Student&);

~Student();

};

**6. Реализация методов для добавления объектов в список.**

void CreateAndAddToList() {

Student student1;

Teacher teacher1;

HeadOfDepartment hOfD1;

Student student2("Nikolay", "Altsivanovich", "Vitalievich", 53);

Teacher teacher2("Ivan", "Ivanov", "Ivanovich", "Math");

HeadOfDepartment hOfD2("Olga", "Zodchenko", "Vasilievna", "OOP", "IIT");

student1.Add();

student2.Add();

teacher1.Add();

teacher2.Add();

hOfD1.Add();

hOfD2.Add();

}

**7. Реализация метода для просмотра списка.**

void List::CheckList() {

if (nullptr == List::\_Head) {

std::cout << "Items not found" << std::endl;

}

else {

list\* Temp = new list;

Temp = List::\_Head;

while (Temp != nullptr) {

Temp->Data->Show();

Temp = Temp->Next;

}

}

}

**8. Листинг демонстрационной программы.**

#include <iostream>

#include "Student.h"

#include "Teacher.h"

#include "HeadOfDepartment.h"

#include "List.h"

void CreateAndAddToList();

int main() {

CreateAndAddToList();

List::CheckList();

return 0;

}

void CreateAndAddToList() {

Student student1;

Teacher teacher1;

HeadOfDepartment hOfD1;

Student student2("Nikolay", "Altsivanovich", "Vitalievich", 53);

Teacher teacher2("Ivan", "Ivanov", "Ivanovich", "Math");

HeadOfDepartment hOfD2("Olga", "Zodchenko", "Vasilievna", "OOP", "IIT");

student1.Add();

student2.Add();

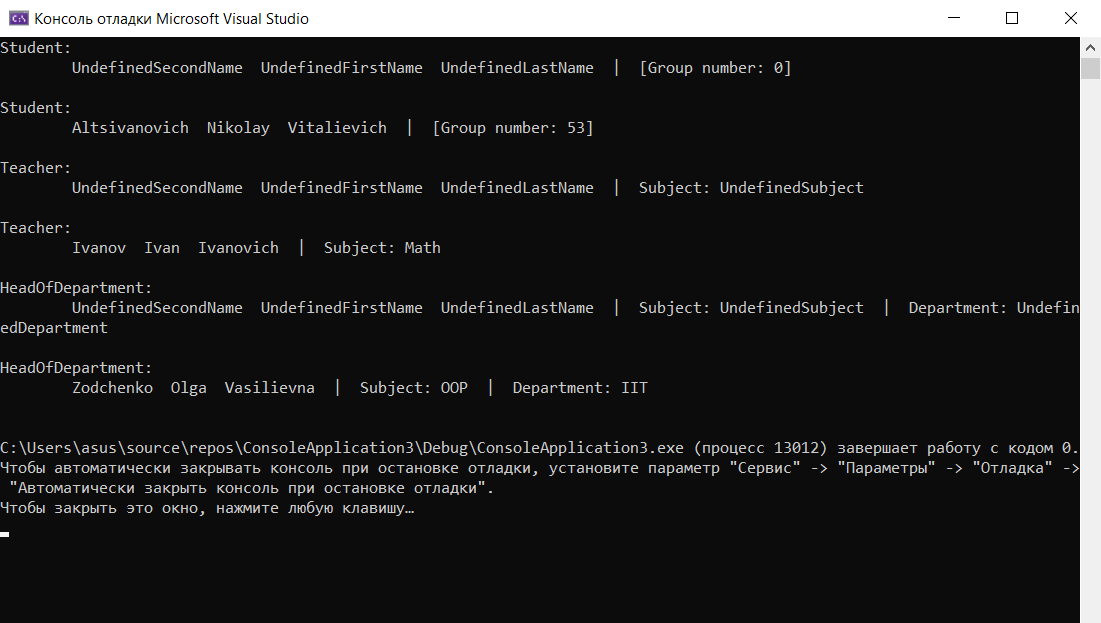
teacher1.Add();

teacher2.Add();

hOfD1.Add();

hOfD2.Add();

}**Результат выполнения:**



**9. Вывод:**

Получил практические навыки реализации классов на С++.