Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

**Лабораторная работа №4**

За 4 семестр

По дисциплине: “ООТПиСП”

Тема: “Наследование и виртуальные функции”

**Вариант №8**

**Выполнил**:

студент 2 курса

группы ПО-7 (перв.)

Крупенков М.Д.

**Проверил:**

Хацкевич М.В.

Брест 2022

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

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

**Методические задания:**

1. Для определения иерархии классов связать отношением наследования классы, приведенные в приложении (для заданного варианта). Из перечисленных классов выбрать один, который будет стоять во главе иерархии. Это абстрактный класс.
2. Определить в классах все необходимые конструкторы и деструктор.
3. Компонентные данные класса специфицировать как protected.
4. Пример определения статических компонентов:

static person\* begin; // указатель на начало списка

static void print(void); // просмотр списка

1. Статическую компоненту-данное инициализировать вне определения класса, в глобальной области.

Для добавления объекта в список предусмотреть метод класса, т.е. объект сам добавляет себя в список. Например, a.Add() - объект a добавляет себя в список.

1. Включение объекта в список можно выполнять при создании объекта, т.е. поместить операторы включения в конструктор. В случае иерархии классов, включение объекта в список должен выполнять только конструктор базового класса. Вы должны продемонстрировать оба этих способа.
2. Список просматривать путем вызова виртуального метода Show каждого объекта.
3. Статический метод просмотра списка вызывать не через объект, а через класс.
4. Определение классов, их реализацию, демонстрационную программу поместить в отдельные файлы.

**Вариант задания:**

8) место, область, город, мегаполис

Код программы

**main.cpp**

#include **"classes.h"  
  
int** main() {  
 Region brest\_region(**"Brest region"**, **"Belarus"**, **false**);  
 brest\_region.add();  
 City brest(**"Brest"**, brest\_region, 340000);  
 Region mid\_atlantic(**"Mid-Atlantic"**, **"USA"**);  
 Metropolis new\_york(**"New York City"**, mid\_atlantic, 10700000, 15);  
 Place::showAll();  
}

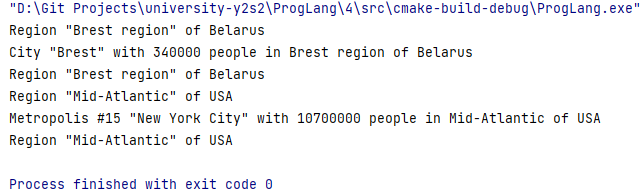
**classes.h**

#ifndef **CLASSES\_H**#define **CLASSES\_H**#include **<string>  
  
  
class** Place {  
**public**:  
 Place();  
 **explicit** Place(std::string name, **bool** doAdd = **true**);  
 Place(Place &place);  
 ~Place();  
  
 **static** Place \*\*places;  
 **static unsigned int** nPlaces;  
  
 **virtual void** print() **const** = 0;  
 **virtual void** add() **final**;  
 **static void** showAll();  
  
**protected**:  
 std::string name\_;  
};  
  
**class** Region : **public** Place {  
**public**:  
 Region();  
 Region(std::string name, std::string country, **bool** doAdd = **true**);  
 Region(Region &region);  
 ~Region();  
 **void** print() **const override**;  
  
**protected**:  
 std::string country\_;  
 **friend class** City;  
 **friend class** Metropolis;  
};  
  
**class** City : **public** Place {  
**public**:  
 City();  
 City(std::string name, Region &region, **unsigned int** nPeoples);  
 City(City &city);  
 ~City();  
 **void** print() **const override**;  
  
**protected**:  
  
 Region region\_;  
 **unsigned int** nPeoples\_;  
};  
  
**class** Metropolis : **public** City {  
**public**:  
 Metropolis();  
 Metropolis(std::string name, Region &region, **unsigned int** nPeoples, **unsigned short** top);  
 Metropolis(Metropolis &metropolis);  
 ~Metropolis();  
 **void** print() **const override**;  
  
**protected**:  
 **unsigned short** top\_;  
};  
  
  
#endif *//CLASSES\_H*

**classes.cpp**

#include **<string>**#include **<iostream>**#include **"classes.h"**Place \*\*Place::places = **nullptr**;  
**unsigned int** Place::nPlaces = 0;  
  
**void** Place::add() {  
 Place \*\*temp = places;  
 places = **new** Place \*[nPlaces + 1];  
 **for** (**int** i = 0; i < nPlaces; i++)  
 places[i] = temp[i];  
 **delete**[] temp;  
 places[nPlaces++] = **this**;  
  
}  
  
**void** Place::showAll() {  
 **for** (**int** i = 0; i < nPlaces; i++) {  
 places[i]->print();  
 }  
}  
  
Place::Place() : name\_(**"TOP SECRET"**) {  
 add();  
}  
  
Place::Place(std::string name, **bool** doAdd) : name\_(std::move(name)) {  
 **if** (doAdd)  
 add();  
}  
  
Place::Place(Place &place) : name\_(std::move(place.name\_)) {  
 add();  
}  
  
Place::~Place() = **default**;  
  
  
Region::Region() : Place(), country\_(**"Abobia"**) {}  
  
Region::Region(std::string name, std::string country, **bool** doAdd) : Place(std::move(name), doAdd),  
 country\_(std::move(country)) {}  
  
Region::Region(Region &region) : Place(region.name\_), country\_(region.country\_) {}  
  
Region::~Region() = **default**;  
  
**void** Region::print() **const** {  
 std::cout << **"Region \""** << name\_ << **"\" of "** << country\_ << **"\n"**;  
}  
  
  
City::City() : Place(), region\_(), nPeoples\_(0) {}  
  
City::City(std::string name, Region &region, **unsigned int** nPeoples) :  
 Place(std::move(name)), region\_(region), nPeoples\_(nPeoples) {}  
  
City::City(City &city) :  
 Place(std::move(city.name\_)), region\_(city.region\_), nPeoples\_(city.nPeoples\_) {}  
  
**void** City::print() **const** {  
 std::cout << **"City \""** << name\_ << **"\" with "** << nPeoples\_ << **" people in "** << region\_.name\_ << **" of "** << region\_.country\_ << **"\n"**;  
}  
  
City::~City() = **default**;  
  
  
Metropolis::Metropolis() : City(), top\_(0) {}  
  
Metropolis::Metropolis(std::string name, Region &region, **unsigned int** nPeoples, **unsigned short** top) :  
 City(std::move(name), region, nPeoples), top\_(top) {}  
  
Metropolis::Metropolis(Metropolis &metropolis) :  
 City(metropolis.name\_, metropolis.region\_, metropolis.nPeoples\_), top\_(metropolis.top\_) {}  
  
**void** Metropolis::print() **const** {  
 std::cout << **"Metropolis #"** << top\_ << **" \""** << name\_ << **"\" with "** << nPeoples\_ << **" people in "** << region\_.name\_  
 << **" of "** << region\_.country\_  
 << **"\n"**;  
}

Результаты программы



**Вывод:** было получено преисполнение при работе с классами в С++.