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
Демидович А. Г.
Проверила:
Давидюк Ю. И.

Брест, 2020

(Вариант 8)

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

8) место, область, город, мегаполис;

Классы: place, state, city, metropolis
Конструкторы:

- Пустой
- С параметрами
- Копирования

Деструктор виртуальный.
Фукнции:

- Добавления в лист
- Вывода информации
- Возвращения и изменения параметров

3. Иерархия классов в виде графа:

- Place
  - City
    - Metropolis
  - State

4. Определение пользовательских классов с комментариями.

```
#pragma once

// Базовый класс
class Place {
private:
	std::string name; // Название
protected:
	virtual void printInformation(std::ostream& out) const; // Вывод информации
	virtual void AddToList(); // Добавление в список
public:
	Place() : name("Unnamed") { AddToList(); } // Контсруктор без параметров
	Place(std::string _name) : name(_name) { AddToList(); } // С 1 параметром
	Place(const Place& _place) : name(_place.name) { AddToList(); } // Копирования
	virtual ~Place() { }; // Виртуальный деструктор
	static Place* begin; // Начало списка
	static Place* back; // Конец списка
	Place* next; // Следующий элемент
	static unsigned int placeListSize; // Размер списка
	void ShowList(); // Показать список
	inline std::string getName() const { return name; } // Вернуть имя
	inline void setName(std::string _name) { name = _name; } // Задать имя
	friend std::ostream& operator<<(std::ostream& out, const Place& _place); // Вывод в поток
};
// Город
```

```
class City : public Place {
private:
	int population = 0.0f; // Население
protected:
	virtual void printInformation(std::ostream& out) const;
public:
	// Конструкторы
	City() : Place(), population(0) {}
	City(std::string _name, int _population)
		: Place(_name), population(_population) {}
	City(const City& _city) : Place(_city.getName()),
		population(_city.population) {}
	~City() { }; // Деструктор
	inline float getPopulation() const { return population; }
	inline void setPopulation(float _value) { population = _value; }
};
// Штат
class State : public Place {
private:
	int cityCount = 0; // Количество городов
protected:
	virtual void printInformation(std::ostream& out) const;
public:
	State() : Place(), cityCount(0) {}
	State(std::string _name, int _cityCount)
		: Place(_name), cityCount(_cityCount) {}
	State(const State& _state) : Place(_state.getName()), cityCount(_state.cityCount) {}
	~State() { };
	inline float getCityCount() const { return cityCount; }
	inline void setCityCount(int _cityCount) { cityCount = _cityCount; }
};

// Мегаполис
class Metropolis : public City {
private:
	int aglomerationSize = 0; // Алгломерация
protected:
	virtual void printInformation(std::ostream& out) const;
public:
	Metropolis() : City(), aglomerationSize(0.0f) {}
	Metropolis(std::string _name, float _population, float _aglomerationSize)
		: City(_name, _population), aglomerationSize(_aglomerationSize) {}
	Metropolis(const Metropolis& _metropolis)
		: City(_metropolis.getName(), _metropolis.getPopulation()), aglomerationSize(_metropolis.aglomerationSize)
{}
	~Metropolis() { };
	inline float getΦglomerationSize() const { return aglomerationSize; }
	inline void setΦglomerationSize(float _value) { aglomerationSize = _value; }
};
```

Методы класса

```
void Place::printInformation(std::ostream& out) const {
	out << "\nName: " << name << "\n";
}

void City::printInformation(std::ostream& out) const {
	Place::printInformation(out);
	out << "Population: " << population << " peoples\n";
}

void State::printInformation(std::ostream& out) const {
```

```
	Place::printInformation(out);
	out << "Cities: " << cityCount << "\n";
}

void Metropolis::printInformation(std::ostream& out) const {
	City::printInformation(out);
	out << "Aglomeration: " << aglomerationSize << " cities\n";
}

std::ostream& operator<<(std::ostream& out, const Place& _place) {
	_place.printInformation(out);
	return out;
}
```

Реализация методов для добавления объектов в список.

```
Place* Place::begin = nullptr;
Place* Place::back = nullptr;
unsigned int Place::placeListSize = 0;

void Place::AddToList() {
	if (begin == nullptr) {
		begin = this;
		back = this;
		placeListSize = 1;
	}
	else {
		back->next = this;
		back = this;
		placeListSize++;
	}
}
```

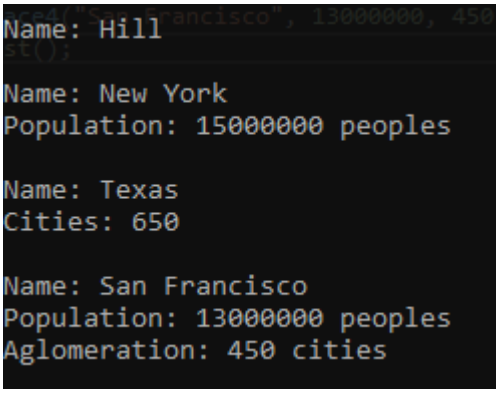
Реализация метода для просмотра списка.

```
void Place::ShowList() {
	Place* element = begin;
	for (int i = 0; i < placeListSize; i++) {
		std::cout << *element;
		element = element->next;
	}
	std::cout << std::endl;
}
```

Листинг демонстрационной программы.

```
#include <iostream>
#include "place.h"
#include "state.h"
#include "city.h"
#include "metropolis.h"

int main() {
	Place place1("Hill");
	City place2("New York", 15000000);
	State place3("Texas", 650);
	Metropolis place4("San Francisco", 13000000, 450);
	place1.ShowList();
	return 0;
}
```



Объяснение необходимости виртуальных функций. Следует показать, какие результаты будут в случае виртуальных и не виртуальных функций.

При наследовании бывает необходимо, чтобы поведение некоторых методов базового класса и классов-наследников различалось, именно для этого и требуется наличие виртуальных функций virtual void printInformation(std::ostream& out) const; В данном коде, в случае отсутствия виртуальной функции нельзя будет переопределить поведение, и, при отсутствии переопределений в коде, после компиляции будет вызываться функция базового класса.