**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И.Раззакова**

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра: **Программное обеспечение компьютерных систем**

Курс «Объектно-ориентированное программирование»

**ОТЧЕТ**

**Лабораторная работа №6**

Выполнил: студент группы ПИ-3-21

Копжашаров Азамат

Проверил: Мусабаев Э.Б.

**Бишкек 2024**

# **Задание №1**

**Постановка задачи:**

1. Проблему экономии времени, доходов, а также увеличения надежности создаваемой объектно-ориентированной программы можно решить с помощью одного из главных принципов ООП - наследования, используя для создания класса-потомка готовый протестированный и отлаженный код класса-родителя. Для решения этой проблемы используйте преимущества **простого наследования**. Для этого создайте родительский класс работник (**Employee)**, протестируйте и отладьте его. В описании этого класса есть поля**:** имя работника - занимаемая должность - position[64], заработная плата за месяц- salary;

методы: конструктор по умолчанию, конструктор с аргументами, деструктор, расчет заработной платы за год, размещение информации о классе-родителе на консоли.

На основе отлаженного класса-родителя **Employee** создайте с помощью **механизма наследования** **public** класс-потомок менеджер (**Manager)**.

Класс - потомок Manager состоит из**:**

полей**:** ежегодный бонус **(annual\_bonus)**, машина компании (**company\_car[64]**)**,** опционы на акции (**stock\_options**) типа (**int**);

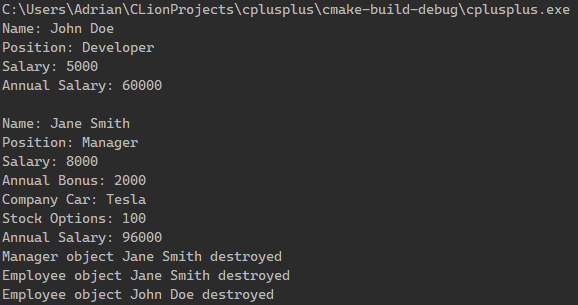
методов: конструктор по умолчанию, конструктор с аргументами, деструктор, размещение информации о классе-потомке на консоли.

*Определитесь с идентификаторами доступа к членам класса, не нарушая основного принципа ООП – инкапсуляции в наследовании***Исходные данные:**

**Исходный код программы на C++:**

// 6.1  
#include <iostream>  
#include <string>  
using namespace std;  
  
class Employee {  
protected:  
 string name;  
 string position;  
 double salary;  
  
public:  
 Employee() {  
 name = "";  
 position = "";  
 salary = 0.0;  
 }  
  
 Employee(const string& \_name, const string& \_position, double \_salary) {  
 name = \_name;  
 position = \_position;  
 salary = \_salary;  
 }  
  
 ~Employee() {  
 cout << "Employee object " << name << " destroyed" << endl;  
 }  
  
 double calculateAnnualSalary() {  
 return salary \* 12;  
 }  
  
 void displayInfo() {  
 cout << "Name: " << name << endl;  
 cout << "Position: " << position << endl;  
 cout << "Salary: " << salary << endl;  
 }  
};  
  
class Manager : public Employee {  
private:  
 double annual\_bonus;  
 string company\_car;  
 int stock\_options;  
  
public:  
 Manager() : Employee() {  
 annual\_bonus = 0.0;  
 company\_car = "";  
 stock\_options = 0;  
 }  
  
 Manager(const string& \_name, const string& \_position, double \_salary, double \_annual\_bonus, const string& \_company\_car, int \_stock\_options)  
 : Employee(\_name, \_position, \_salary) {  
 annual\_bonus = \_annual\_bonus;  
 company\_car = \_company\_car;  
 stock\_options = \_stock\_options;  
 }  
  
 ~Manager() {  
 cout << "Manager object " << name << " destroyed" << endl;  
 }  
  
 void displayInfo() {  
 Employee::displayInfo();  
 cout << "Annual Bonus: " << annual\_bonus << endl;  
 cout << "Company Car: " << company\_car << endl;  
 cout << "Stock Options: " << stock\_options << endl;  
 }  
};  
  
int main() {  
 Employee emp1("John Doe", "Developer", 5000.0);  
 emp1.displayInfo();  
 cout << "Annual Salary: " << emp1.calculateAnnualSalary() << endl << endl;  
  
 Manager manager1("Jane Smith", "Manager", 8000.0, 2000.0, "Tesla", 100);  
 manager1.displayInfo();  
 cout << "Annual Salary: " << manager1.calculateAnnualSalary() << endl;  
  
 return 0;  
}

**Тесты:**

****

# **Задание №2**

**Постановка задачи:**

Используйте преимущества **множественного наследования,** а именно: разработайте, протестируйте и отладьте родительские классы Дисплей **(*Display*)** и Материнская плата (M***otherBoard),*** и на их основе создайте класс-потомокКомпьютер **(*Сomputer*)**:

В описании *класса-родителя* ***Display*** есть

поля: тип монитора char type[32], количество цветов long colors, разрешение по оси х int x\_resolution, разрешение по оси у int y\_resolution.

методы: конструктор с аргументами; размещение информации о классе-родителе на консоли.

В описании *класса-родителя* ***MotherBoard*** *имеет*

поля: тип процессора int processor, скорость процессора int speed, объем оперативной памяти int RAM;

методы: конструктор с аргументами; размещение информации о классе-родителе на консоли.

- *Класс-потомок* ***Сomputer*** создать на основе родительских классовс помощью механизма наследования **public** и отобразить его на экране**,** добавив:

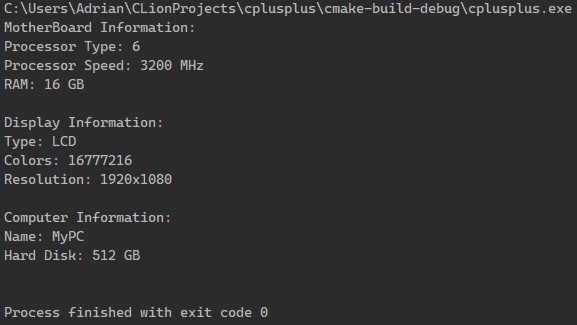
поля: марка компьютера (name [64]), объем жесткого диска (hard\_disk**)**.

методы: конструктор с аргументами, метод Show().**Исходные данные:**

**Исходный код программы на C++:**

// 6.2  
#include <iostream>  
#include <cstring>  
using namespace std;  
  
class Display {  
protected:  
 char type[32];  
 long colors;  
 int x\_resolution;  
 int y\_resolution;  
  
public:  
 Display(char \_type[], long \_colors, int \_x\_resolution, int \_y\_resolution) {  
 strcpy(type, \_type);  
 colors = \_colors;  
 x\_resolution = \_x\_resolution;  
 y\_resolution = \_y\_resolution;  
 }  
  
 void displayInfo() {  
 cout << "Display Information:" << endl;  
 cout << "Type: " << type << endl;  
 cout << "Colors: " << colors << endl;  
 cout << "Resolution: " << x\_resolution << "x" << y\_resolution << endl << endl;  
 }  
};  
  
class MotherBoard {  
protected:  
 int processor;  
 int speed;  
 int RAM;  
  
public:  
 MotherBoard(int \_processor, int \_speed, int \_RAM) : processor(\_processor), speed(\_speed), RAM(\_RAM) {}  
  
 void displayInfo() {  
 cout << "MotherBoard Information:" << endl;  
 cout << "Processor Type: " << processor << endl;  
 cout << "Processor Speed: " << speed << " MHz" << endl;  
 cout << "RAM: " << RAM << " GB" << endl << endl;  
 }  
};  
  
class Computer : public Display, public MotherBoard {  
private:  
 char name[64];  
 int hard\_disk;  
  
public:  
 Computer(char \_type[], long \_colors, int \_x\_resolution, int \_y\_resolution,  
 int \_processor, int \_speed, int \_RAM,  
 char \_name[], int \_hard\_disk)  
 : Display(\_type, \_colors, \_x\_resolution, \_y\_resolution),  
 MotherBoard(\_processor, \_speed, \_RAM) {  
 strcpy(name, \_name);  
 hard\_disk = \_hard\_disk;  
 }  
  
 void Show() {  
 MotherBoard::displayInfo();  
 Display::displayInfo();  
 cout << "Computer Information:" << endl;  
 cout << "Name: " << name << endl;  
 cout << "Hard Disk: " << hard\_disk << " GB" << endl << endl;  
 }  
};  
  
int main() {  
 Computer pc("LCD", 16777216, 1920, 1080, 6, 3200, 16, "MyPC", 512);  
 pc.Show();  
 return 0;  
}

**Тесты:**



# **Задание №3**

**Постановка задачи:**

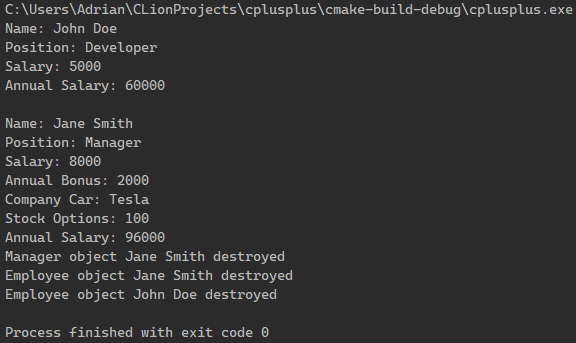
В задании № 1 изменить механизм наследования так, чтобы закрыть доступ к членам базового класса из производного класса. А затем восстановить этот доступ всеми известным Вам способами.

**Исходные данные:**

**Исходный код программы на C++:**

// 6.3  
#include <iostream>  
#include <string>  
using namespace std;  
  
class Employee {  
protected:  
 string name;  
 string position;  
 double salary;  
  
public:  
 Employee() {  
 name = "";  
 position = "";  
 salary = 0.0;  
 }  
  
 Employee(const string& \_name, const string& \_position, double \_salary) {  
 name = \_name;  
 position = \_position;  
 salary = \_salary;  
 }  
  
 ~Employee() {  
 cout << "Employee object " << name << " destroyed" << endl;  
 }  
  
 double calculateAnnualSalary() {  
 return salary \* 12;  
 }  
  
 void displayInfo() {  
 cout << "Name: " << name << endl;  
 cout << "Position: " << position << endl;  
 cout << "Salary: " << salary << endl;  
 }  
  
 friend class Manager;  
};  
  
class Manager : private Employee {  
private:  
 double annual\_bonus;  
 string company\_car;  
 int stock\_options;  
  
public:  
 Manager() : Employee() {  
 annual\_bonus = 0.0;  
 company\_car = "";  
 stock\_options = 0;  
 }  
  
 Manager(const string& \_name, const string& \_position, double \_salary, double \_annual\_bonus, const string& \_company\_car, int \_stock\_options)  
 : Employee(\_name, \_position, \_salary) {  
 annual\_bonus = \_annual\_bonus;  
 company\_car = \_company\_car;  
 stock\_options = \_stock\_options;  
 }  
  
 ~Manager() {  
 cout << "Manager object " << name << " destroyed" << endl;  
 }  
  
 double getAnnualSalary() {  
 return calculateAnnualSalary();  
 }  
  
 void displayInfo() {  
 Employee::displayInfo();  
 cout << "Annual Bonus: " << annual\_bonus << endl;  
 cout << "Company Car: " << company\_car << endl;  
 cout << "Stock Options: " << stock\_options << endl;  
 }  
};  
  
int main() {  
 Employee emp1("John Doe", "Developer", 5000.0);  
 emp1.displayInfo();  
 cout << "Annual Salary: " << emp1.calculateAnnualSalary() << endl << endl;  
  
 Manager manager1("Jane Smith", "Manager", 8000.0, 2000.0, "Tesla", 100);  
 manager1.displayInfo();  
 cout << "Annual Salary: " << manager1.getAnnualSalary() << endl;  
  
 return 0;  
}

**Тесты:**



# **Ответы на вопросы:**

1. Что такое в ООП? Приведите примеры.

ООП (объектно-ориентированное программирование) — это методология программирования, которая основана на использовании объектов, которые являются экземплярами классов. Классы определяют структуру и поведение объектов, а объекты, в свою очередь, могут взаимодействовать друг с другом, передавая сообщения и вызывая методы.

В корпорации есть несколько отделов. Каждый отдел имеет свой собственный персонал, с четко установленными обязанностями (функциями). Отделы имеют свои собственные данные: финансовый отдел – данные по заработной плате, отдел продаж – данные по продажам, отдел кадров – данные по каждому сотруднику и т. д. Эта схема обеспечивает правильную обработку данных и защищает их от повреждений неумелыми посторонними лицами.

С помощью объектов можно реализовать такой подход к организации работы отделов при автоматизации данной корпорации. Кроме того, при объектно-ориентированном подходе обеспечивается целостность данных.

1. В чем заключается выигрыш для программиста при использовании им наследования?

Выигрыш программиста при использовании наследования в объектно-ориентированном программировании заключается в повышении уровня повторного использования кода, улучшении организации кода и облегчении его поддержки и расширения за счет использования основных принципов ООП.

1. Какие члены базового класса не наследуются?

Private-члены базового класса не наследуются.

1. От чего зависит доступ к членам базового класса из произвольного класса?

Доступ к членам базового класса зависит от модификаторов доступа к этим членам, public и protected члены могут быть доступны из производного класса, тогда как private-члены не доступны.

1. Чем отличается идентификатор доступа от механизма доступа?

Идентификатор доступа определяет уровень доступа к членам класса, таким как переменные, методы и конструкторы. Он указывает на то, какие части программы имеют право обращаться к этим членам и каким образом. Общие идентификаторы доступа включают в себя:

public: Члены класса доступны из любого места в программе.

private: Члены класса доступны только внутри этого класса.

protected: Члены класса доступны внутри этого класса и его подклассов.

Ключевые слова public, private, protected используются не только для определения доступа к членам класса, но и являются индикаторами механизма наследования.

Механизм наследования определяет какой идентификатор доступа будет у полей и методов у унаследованного класса.

Таким образом, в зависимости от типа наследования (public, private, protected) и доступности членов (public, private, protected) получаем разный результат.

1. Назовите основные правила наследования (используйте таблицу наследования).

|  |  |  |  |
| --- | --- | --- | --- |
| **Механизм** | public- наследование | private- наследование | protected - наследование |
| **Доступность** |  |  |  |
| рublic - член | public | private | protected |
| рrivate- член | нет доступа | нет доступа | нет доступа |
| рrotected - член | public | private | protected |

Информацию о наследовании, представленную в виде таблицы 1, можно сформулировать в виде нескольких правил, которые приведены в порядке убывания важности:

1. private-члены базового класса не наследуются;
2. при public-наследовании уровень доступа члена класса не меняется;
3. при private-наследовании наследуемые члены становятся private-членом производного класса;
4. при protected-наследовании наследуемые члены становятся protected -членами производного класса.
5. Синтаксис объявления производного класса, как наследника базового класса?

**Синтаксис** объявления производного класса при наследовании базового класса имеет вид:

Class *производный\_класс*: тип\_наследования *базовый\_класс* {

программный код производного класса

};

1. Что такое простое, множественное и многократное наследование?

**Простое наследование** — это когда у производного класса имеется один базовый класс.

**Множественное наследование** - когда у производного класса имеется несколько базовых классов.

**Многократное** **наследование** – это когда производный класс может иметь и базовый и производный классы.

1. Особенности использования идентификатор доступа protected при наследовании классов?

Protected идентификатор позволяет получить производным классам доступ к protected-членам базового класса, но в то же время protected идентификатор не позволяет получит доступ к членам класса извне.

1. Какие существуют виды конструкторов при наследовании?

**Простой конструктор** при наследовании

Важным и принципиальным является вопрос использования конструкторов и деструкторов при наследовании. Действительно, если в базовом классе имеется конструктор с аргументами, то в производном классе как минимум должен быть предусмотрен механизм передачи аргументов этому конструктору. Такой механизм существует. С точки зрения синтаксиса языка С++ сводится он к тому, что при определении конструктора производного класса после имени конструктора указывается имя конструктора базового класса с указанием в круглых скобках аргументов, передаваемых этому конструктору. Если базовых классов несколько, для каждого из них через запятую перечисляются конструкции вида имя\_конструктора (аргументы) для передачи аргументов конструктору соответствующего базового класса.

**Конструктор при множественном наследовании**

**Синтаксис конструктора**

**Имя\_конструктора производного класса (аргументы конструктора базового класса, аргументы конструктора производного класса): имя\_ конструктора базового класса 1 (аргументы базового класс 1), имя\_ конструктора базового класса 2(аргументы базового класс 2),... {имя\_атрибута1 производного класса = значение; …}**

1. Каков синтаксис объявления конструктора производного класса?

Простой конструктор

**имя\_конструктора производного класса (аргументы конструктора базового класса, аргументы конструктора производного класса): имя\_ конструктора базового класса (аргументы базового класс), {имя\_атрибута1 производного класса = значение; …}**

Конструктор при множественном наследовании

**Имя\_конструктора производного класса (аргументы конструктора базового класса, аргументы конструктора производного класса): имя\_ конструктора базового класса 1 (аргументы базового класс 1), имя\_ конструктора базового класса 2(аргументы базового класс 2),... {имя\_атрибута1 производного класса = значение; …}**

1. Каковы особенности конструктора при простом наследовании?

При простом наследовании нужно сначала объявить аргументы базового класса, затем производного, позже объявить имя конструктора базового класса и передать ему значения в скобках.

1. Каковы особенности конструктора при многократном наследовании?

Схоже с простым наследованием, только в очередь добавляются классы в порядке наследования: все аргументы (базового класса, производного класса), затем конструктор\_1 (аргументы базового класса 1), конструкора\_2 (аргументы базового класса 2), аргументы производного класса.

1. Какова последовательность вызова конструкторов и деструкторов при наследовании?

Конструкторы вызываются в такой последовательности: конструктор класса Base1, конструктор класса Base2 и затем конструктор класса Derivative. Деструкторы вызываются в обратной последовательности: деструктор класса Derivative, деструктор класса Base2, и, наконец, деструктор класса Base1.