**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. И.РАЗЗАКОВА**

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

**КАФЕДРА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХ СИСТЕМ**

**ОТЧЕТ**

**ПО ДИСЦИПЛИНЕ: ООП**

**ПО ТЕМЕ: Наследование**

**Лабораторная работа №5**

**ВЫПОЛНИЛ:** студент гр. ПИ(б)-3-18

Гадайбаев М.А.

**ПРОВЕРИЛА:** Каткова С.Н.

**Бишкек 2020**

**ВОПРОСЫ К ЗАЩИТЕ ЛАБОРАТОРНОЙ РАБОТЫ № 5**

1. Что такое наследование в ООП? Приведите примеры.

Наследование – механизм языка, позволяющий описать новый класс на основе уже существующего (родительского, базового) класса.

Класс потомок может добавить собственные методы и свойства, а также пользоваться родительскими методами и свойствами.

Например, есть родительский класс Parents, и производный класс Child. Производный класс наследует все свойства и методы родительского класса.

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

Выигрыш от него состоит в том, что наследование позволяет использовать существующий код несколько раз. Имея написанный и отлаженный базовый класс, мы можем его больше не модифицировать, при этом механизм наследования позволит нам приспособить его для работы в различных ситуациях. Используя уже написанный код, мы экономим время и деньги, а также увеличиваем надёжность программы.

1. Какие члены базового класса не наследуются?

Какие поля и методы наследуются, а какие нет, определяется доступностью соответствующих членов базового класса и механизмом наследования, то есть если члены класса приватные или закрытые, то они не наследуются.

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

Доступ к членам базового класса зависит от типа наследования.

1. Чем отличается идентификатор доступа от механизма доступа?

Идентификатор доступа отличается от механизма доступа тем, что идентификатор доступа контролирует доступ к членам произвольного класса, а механизм доступа нужен для того, чтобы определить какие методы произвольный класс может наследовать от родительского.

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 |
|  |  |  |  |

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?

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?

При использовании идентификатор доступа protected, ты не сможешь получить доступ из вне класса, только в том классе, где объявлен класс protected.

1. Какие существуют виды конструкторов при наследовании?

Существуют два вида конструкторов при наследовании:

1. Простой конструктор
2. Конструктор при множественном наследовании
3. Каков синтаксис объявления конструктора производного класса?

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

1. Каковы особенности конструктора при простом наследовании?

Дело в том, что при создании объектов производного класса сначала вызывается конструктор базового класса, а уже затем конструктор производного класса.

1. Каковы особенности конструктора при многократном наследовании?

Особенность в том, что при многократном вызове конструктор вызывает сперва у родительских классов, затем у производных.

1. Какова последовательность вызова конструкторов и деструкторов при наследовании?

Конструкторы сперва вызываются у родительских классов, затем у производных, а в деструкторе наоборот, сперва производные классы, затем родительские.

**Задания 1.**

**Постановка задачи:**

создать родительский класс работник (**Employee)**, c полями**:** имя работника – name, занимаемая должность - position, 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** создать с помощью **механизма наследования** **public** класс-потомок менеджер (**Manager)**, с полями**:** ежегодный бонус **(annual\_bonus)**, машина компании (**company\_car[64]**)**,** опционы на акции (**stock\_options**) типа (**int**);методы: конструктор по умолчанию, конструктор с аргументами, деструктор, размещение информации о классе-потомке на консоли.

**Описание входных данных:**

**class Employee** – базовый класс, c полями: name - имя работника, position – должность, salary – зарплата; и с методами: конструктор инициализации (по умолчанию), конструктор инициализации (с аргументами), Деструктор, SalaryYear() – расчет зарплаты за год, output() – вывод инфо на консоль.

**сlass Manager** – производный класс c механизмом наследования public, с полями: annual\_bonus – годовой бонус, company\_car – машина компании, stock\_option – опционы на акции; и с методами: конструктор инициализации (по умолчанию), конструктор инициализации (с аргументами), Деструктор, output2() – вывод инфо на консоль.

**Код:**

#include<iostream>

using namespace std;

//родительский класс работник(Employee)

class Employee {

//поля класса

string name;

string position;

int salary;

//методы класса

public:

Employee() :name(""), position(""), salary(0) {} //конструктор по умолчанию

Employee(string addname, string addposition, int addsalary) : name(addname), position(addposition), salary(addsalary) {} //конструктор с аргументами

~Employee() { cout << "Данные у базоаого класса очищены\n"; } //деструктор

int SalaryYear() { //расчет заработной платы за год

return salary \* 12;

}

void output() { //размещение информации о классе-родителе на консоли

cout << "Имя работника: " << name << "\n" << "Занимаемая должность: " << position << "\n" << "Зарплата: " << salary << "\n" << "Зарплата за год: " << SalaryYear() << "\n";

}

};

//класс - потомок менеджер (Manager).

//механизм наследования public

class Manager : public Employee {

//поля класса

int annual\_bonus;

string company\_car;

int stock\_option;

//методы класса

public:

Manager() :Employee() { annual\_bonus = 0; company\_car = ""; stock\_option = 0; } //конструктор по умолчанию

//конструктор с аргументами

Manager(int addannual, string addcompany, int addstock, string addname, string addposition, int addsalary) : Employee(addname, addposition, addsalary) {

this->annual\_bonus = addannual;

this->company\_car = addcompany;

this->stock\_option = addstock;

}

~Manager() { cout << "Данные у производного класса очищены\n"; } //деструктор

void output2() { //размещение информации о классе-потомке на консоли.

cout << "Ежегодный бонус: " << annual\_bonus << "\n" << "Машина компании: " << company\_car << "\n" << "Опционы на акции: " << stock\_option << "\n";

}

};

int main() {

setlocale(LC\_ALL, "RUS");

string name, position, company\_car;

int salary, annual\_bonus, stock\_options;

cout << "Имя: "; cin >> name;

cout << "Должность: "; cin >> position;

cout << "Зарплата: "; cin >> salary;

cout << "Ежегодный бонус: "; cin >> annual\_bonus;

cout << "Машина компании: "; cin >> company\_car;

cout << "Опционы на акции: "; cin >> stock\_options;

cout << endl;

Manager rab(annual\_bonus, company\_car, stock\_options, name, position, salary); // Создаём объект работник

cout << "Базовый класс Employee" << endl;

cout << "++++++++++++++++++++++++++++++++++++++" << endl;

rab.output();

cout << "++++++++++++++++++++++++++++++++++++++" << endl << endl;

cout << "Производный класс Manager (свои свойства)" << endl;

cout << "++++++++++++++++++++++++++++++++++++++" << endl;

rab.output2();

cout << "++++++++++++++++++++++++++++++++++++++" << endl << endl;

cout << "Производный класс Manager (с свойствами базового класса)" << endl;

cout << "++++++++++++++++++++++++++++++++++++++" << endl;

rab.output();

rab.output2();

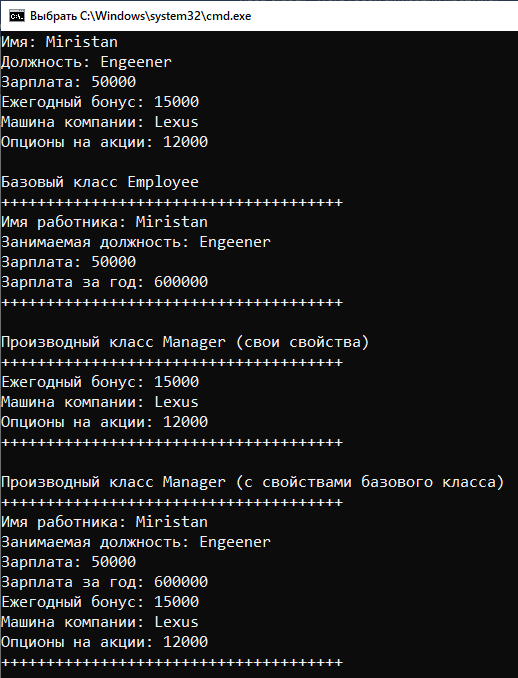
cout << "++++++++++++++++++++++++++++++++++++++" << endl << endl;

system("pause");

return 0;

}

**Результат:**

****

**Задания 2.**

Создать родительские классы Дисплей **(*Display*)** и Материнская плата (M***otherBoard),*** и на их основе создать класс-потомокКомпьютер **(*Сomputer*)**:

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

поля: тип монитора char type, 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 (namе), объем жесткого диска (hard\_disk**)**.

методы: конструктор с аргументами, метод Show().

**Описание входных данных:**

**class Display** – базовый класс, c полями: monitorType – тип монитора, longColor – количество цветов, x\_resolution – разрешение по оси х, y\_resolution – разрешение по оси y; и с методами: конструктор инициализации (с аргументами), output1() – вывод инфо на консоль.

**сlass MotherBoard** – базовый класс, с полями: processor; – тип процессора, speed – скорость процессора, RAM – объем оперативной памяти; и с методами: конструктор инициализации (с аргументами), output2() – вывод инфо на консоль.

**сlass Computer** – производный класс c механизмом наследования public, с полями: CompMark – марка компьютера, hardDisk – объем жесткого диска; и с методами: конструктор инициализации (с аргументами), show() – вывод инфо на консоль.

**Код:**

#include <iostream>

using namespace std;

//родительский класс Дисплей(Display)

class Display {

public:

//поля класса

string monitorType; //тип монитора

int longColor; //количество цветов

int x\_resolution; //разрешение по оси х

int y\_resolution; //разрешение по оси у

//методы класса

Display(string addMonitor, int addLong, int addX, int addY) { //конструктор с аргументами

monitorType = addMonitor;

longColor = addLong;

x\_resolution = addX;

y\_resolution = addY;

}

void output1() { //размещение информации о классе-родителе на консоли

cout << "Тип монитора: " << monitorType << "\n";

cout << "Количество цветов: " << longColor << "\n";

cout << "Разрешение по оси x: " << x\_resolution << "\n";

cout << "Разрешение по оси y: " << y\_resolution << "\n";

}

};

//родительский класс Материнская плата(MotherBoard)

class MotherBoard {

public:

//поля класса

string processor; //тип процессора

int speed; //скорость процессора

int RAM; //объем оперативной памяти

//методы класса

MotherBoard(string addPro, int addSpe, int addRAM) { //конструктор с агрументами

processor = addPro;

speed = addSpe;

RAM = addRAM;

}

void output2() { //размещение информации о классе-родителе на консоли

cout << "Тип процессора: " << processor << "\n";

cout << "Скорость процессора: " << speed << "\n";

cout << "Объём оперативной памяти: " << RAM << "\n";

}};

//класс - потомок компьютер (Computer).

//механизм наследования public

class Computer : public Display, public MotherBoard {

//поля класса

string CompMark; //марка компьютера

int hardDisk; //объем жесткого диска

//методы класса

public:

Computer(string addMonitor, int addLong, int addX, int addY, string addPro, int addSpe, int addRAM, string addMark, int addHard) //rконструктор с агрументами

:Display(addMonitor, addLong, addX, addY), MotherBoard(addPro, addSpe, addRAM) {

this->CompMark = addMark;

this->hardDisk = addHard;

}

void show() { //размещение информации о компьютере

cout << "Марка компьютера: " << CompMark << "\n";

cout << "Объём жесткого диска: " << hardDisk << "\n";

cout << "Тип процессора: " << processor << "\n";

cout << "Скорость процессора: " << speed << "\n";

cout << "Объём оперативной памяти: " << RAM << "\n";

cout << "Тип монитора: " << monitorType << "\n";

cout << "Количество цветов: " << longColor << "\n";

cout << "Разрешение по оси x: " << x\_resolution << "\n";

cout << "Разрешение по оси y: " << y\_resolution << "\n";

}

};

int main() {

setlocale(LC\_ALL, "rus");

string monitorType, CompMark, processor;

int longColor, x\_resolution, y\_resolution, speed, RAM, hardDisk;

cout << "Марка компьютера: "; cin >> CompMark;

cout << "Объём жесткого диска: "; cin >> hardDisk;

cout << "Тип процессора: "; cin >> processor;

cout << "Скорость процессора: "; cin >> speed;

cout << "Объём оперативной памяти: "; cin >> RAM;

cout << "Тип монитора: "; cin >> monitorType;

cout << "Количество цветов: "; cin >> longColor;

cout << "Разрешение по оси x: "; cin >> x\_resolution;

cout << "Разрешение по оси y: "; cin >> y\_resolution;

cout << "\n\n";

Computer comp(monitorType, longColor, x\_resolution, y\_resolution, processor, speed, RAM, CompMark, hardDisk); //Создаём объект компьютер

cout << "Класс родитель - Display" << endl;

cout << "++++++++++++++++++++++++++++++++++++++++" << endl;

comp.output1();

cout << "++++++++++++++++++++++++++++++++++++++++" << endl << endl;

cout << "Класс родитель - MotherBoard" << endl;

cout << "++++++++++++++++++++++++++++++++++++++++" << endl;

comp.output2();

cout << "++++++++++++++++++++++++++++++++++++++++" << endl << endl;

cout << "Класс потомок - Computer" << endl;

cout << "++++++++++++++++++++++++++++++++++++++++" << endl;

comp.show();

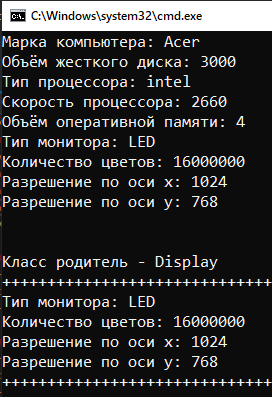
cout << "++++++++++++++++++++++++++++++++++++++++" << endl << endl;

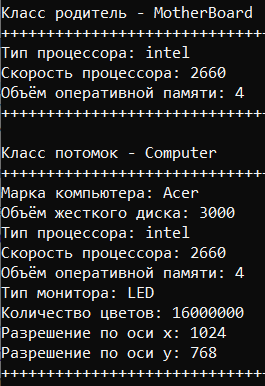
system("pause>NULL");

return 0;

}

**Результат:**

****

****

**Задания 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 способами.

**Описание входных данных:**

**class Employee** – базовый класс, c полями: name - имя работника, position – должность, salary – зарплата; и с protected методами: конструктор инициализации (по умолчанию), конструктор инициализации (с аргументами), Деструктор, set() – получить данные из полей, SalaryYear() – расчет зарплаты за год, output() – вывод инфо на консоль.

**сlass Manager** – производный класс c механизмом наследования protected, с полями: annual\_bonus – годовой бонус, company\_car – машина компании, stock\_option – опционы на акции; и с методами: конструктор инициализации (по умолчанию), конструктор инициализации (с аргументами), Деструктор, output2() – вывод инфо на консоль.

**Код:**

#include<iostream>

using namespace std;

//родительский класс работник(Employee)

class Employee {

//поля класса

string name;

string position;

int salary;

//методы класса

protected:

Employee() :name(""), position(""), salary(0) {} //конструктор по умолчанию

Employee(string addname, string addposition, int addsalary) : name(addname), position(addposition), salary(addsalary) {} //конструктор с аргументами

~Employee() { cout << "Данные у базоаого класса очищены\n"; } //деструктор

void set(string addname, string addposition, int addsalary) { //метод set(), используем для получения данных из поля класса

name = addname;

position = addposition;

salary = addsalary;

}

int SalaryYear() { //расчет заработной платы за год

return salary \* 12;

}

void output() { //размещение информации о классе-родителе на консоли

cout << "Имя работника: " << name << "\n" << "Занимаемая должность: " << position << "\n" << "Зарплата: " << salary << "\n" << "Зарплата за год: " << SalaryYear() << "\n";

}

};

//класс - потомок менеджер (Manager).

//механизм наследования protected

class Manager : protected Employee {

//поля класса

int annual\_bonus;

string company\_car;

int stock\_option;

//методы класса

public:

Manager() :Employee() { annual\_bonus = 0; company\_car = ""; stock\_option = 0; } //конструктор по умолчанию

Manager(int addannual, string addcompany, int addstock) : Employee() { //конструктор с аргументами

this->annual\_bonus = addannual;

this->company\_car = addcompany;

this->stock\_option = addstock;

}

~Manager() { cout << "Данные у производного класса очищены\n"; } //деструктор

void output2() { //размещение информации о классе-потомке на консоли.

cout << "Ежегодный бонус: " << annual\_bonus << "\n" << "Машина компании: " << company\_car << "\n" << "Опционы на акции: " << stock\_option << "\n";

}

Employee::set; //восстановление уровня доступа к методу

Employee::output; //восстановление уровня доступа к методу

};

int main() {

setlocale(LC\_ALL, "RUS");

string name, position, company\_car;

int salary, annual\_bonus, stock\_options;

cout << "Имя: "; cin >> name;

cout << "Должность: "; cin >> position;

cout << "Зарплата: "; cin >> salary;

cout << "Ежегодный бонус: "; cin >> annual\_bonus;

cout << "Машина компании: "; cin >> company\_car;

cout << "Опционы на акции: "; cin >> stock\_options;

cout << endl;

Manager rab(annual\_bonus, company\_car, stock\_options); // Создаём объект работник

cout << "Базовый класс Employee" << endl;

cout << "++++++++++++++++++++++++++++++++++++++" << endl;

rab.set(name, position, salary);

rab.output();

cout << "++++++++++++++++++++++++++++++++++++++" << endl << endl;

cout << "Производный класс Manager (свои свойства)" << endl;

cout << "++++++++++++++++++++++++++++++++++++++" << endl;

rab.output2();

cout << "++++++++++++++++++++++++++++++++++++++" << endl << endl;

cout << "Производный класс Manager (с свойствами базового класса)" << endl;

cout << "++++++++++++++++++++++++++++++++++++++" << endl;

rab.set(name, position, salary);

rab.output();

rab.output2();

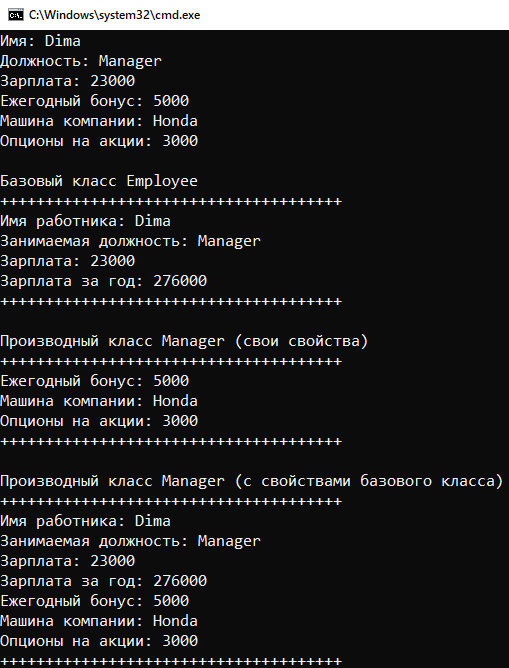
cout << "++++++++++++++++++++++++++++++++++++++" << endl << endl;

system("pause");

return 0;

}

**Результат:**

****