Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

**ОТЧЕТ**

**о работе по информатике**

**Тема:** Лабораторная работа № 2 по ООП

Семестр: 2

Выполнил студент ИВТ-22-2б:

Солодов Александр Андреевич

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

Пермь 2023

**Постановка задачи**

1. Определить пользовательский класс.

2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

3. Определить в классе деструктор.

4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).

5. Написать демонстрационную программу, в которой продемонстрировать все три случая вызова конструктора-копирования, вызов конструктора с параметрами и конструктора без параметров.

*Вариант 15:*

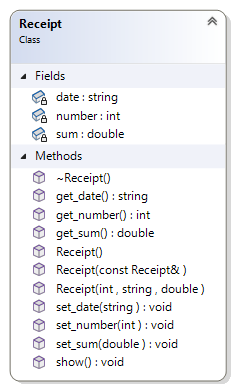
Пользовательский класс КВИТАНЦИЯ:

Номер – int;

Дата – string;

Сумма – double.

**UML-диаграмма**



**Программное решение**

Receipt.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Receipt {

private:

int number; //номер квитанции

string date; //дата

double sum; //сумма

public:

//конструкторы

Receipt();

Receipt(int, string, double);

Receipt(const Receipt&);

~Receipt();

//геттеры и сеттеры

int get\_number();

void set\_number(int);

string get\_date();

void set\_date(string);

double get\_sum();

void set\_sum(double);

//функция для вывода объекта на экран

void show();

};

Receipt.cpp

#include "Receipt.h"

Receipt::Receipt()

{

number = 0;

date = "";

sum = 0.0;

}

Receipt::Receipt(int n, string d, double s)

{

number = n;

date = d;

sum = s;

}

Receipt::Receipt(const Receipt& r)

{

number = r.number;

date = r.date;

sum = r.sum;

}

Receipt::~Receipt()

{

cout << "Вызов деструктора для " << this << endl;

}

int Receipt::get\_number()

{

return number;

}

void Receipt::set\_number(int n)

{

number = n;

}

string Receipt::get\_date()

{

return date;

}

void Receipt::set\_date(string d)

{

date = d;

}

double Receipt::get\_sum()

{

return sum;

}

void Receipt::set\_sum(double s)

{

sum = s;

}

void Receipt::show()

{

cout << "Номер квитанции: " << number << endl;

cout << "Дата: " << date << endl;

cout << "Сумма: " << sum << endl;

}

Main.cpp

/\* Вариант 12

Пользовательский класс КВИТАНЦИЯ Номер – int

Дата – string Сумма – double

\*/

#include "Receipt.h"

//функция для возврата объекта как результата

Receipt makeReceipt() {

int num;

string date;

double sum;

cout << "Введите номер квитанции: " << endl;

cin >> num;

cout << "Введите дату: " << endl;

cin >> date;

cout << "Введите сумму: " << endl;

cin >> sum;

Receipt temp(num, date, sum);

return temp;

}

//функция для передачи объекта, как параметра

void printReceipt(Receipt r) {

r.show();

}

int main() {

setlocale(LC\_ALL, "RU");

//конструктор без параметров

Receipt r1;

cout << "Квитанция № 1: " << endl;

r1.show();

//конструктор с параметрами

Receipt r2(2000222, "02/02/2022", 378.55);

cout << "\nКвитанция № 2: " << endl;

r2.show();

//конструктор копирования

Receipt r3 = r2;

//изменение полей сеттерами

r3.set\_number(1000055);

r3.set\_date("05/05/2005");

r3.set\_sum(2777.44);

cout << "\nКвитанция № 3: " << endl;

r3.show();

//функция для создания

r1 = makeReceipt();

cout << "\nКвитанция № 4" << endl;

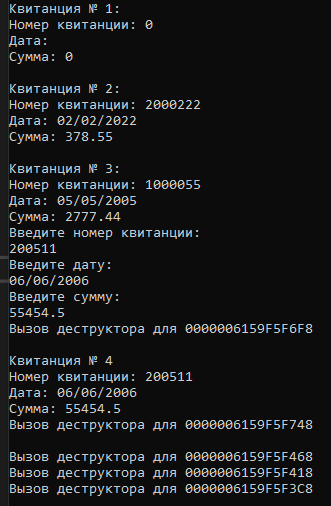
printReceipt(r1);

cout << endl;

return 0;

}

**Результат работы программы**



**Контрольные вопросы**

*1. Для чего нужен конструктор?*

**Конструктор** – это метод, который предназначен для инициализации объекта. С помощью конструктора задаются значения полей для экземпляра класса. Имя конструктора совпадает с именем класса, и он вызывается автоматически при инициализации объекта. Автоматический вызов конструктора позволяет избежать ошибок, связанных с использованием неинициализированных переменных.

*2. Сколько типов конструкторов существует в С++?*

Существует **три типа конструкторов**:

- Конструктор **без параметров**, используется для создания «пустого» объекта (либо поля задаются по умолчанию).

- Конструктор **с параметрами**, используется для инициализации объекта требуемыми значениями (тип передаваемых параметров совпадает с типом соответствующих полей).

- Конструктор **копирования** используется для создания объекта, аналогичного тому, который уже существует (в качестве параметра передается ссылка на объект того же класса и новому объекту присваиваются значения полей переданного объекта). Конструктор без параметров и конструктор копирования создаются по умолчанию.

*3. Для чего используется деструктор? В каких случаях деструктор описывается явно?*

**Деструктор** – это особый вид метода, применяющийся для освобождения ресурсов, выделенных конструктором объекту. Деструктор вызывается автоматически, когда объект удаляется из памяти. Имя деструктора начинается с тильды (~), непосредственно за которой следует имя класса. Если деструктор явным образом не определен, компилятор автоматически создает пустой деструктор. Описывать в классе деструктор явным образом требуется в случае, когда объект содержит указатели на память, выделяемую динамически — иначе при уничтожении объекта память, на которую ссылались его поля-указатели, не будет помечена как свободная.

Пример:

class Person

{

private:

char\* name;

int age;

public:

~Person() { delete[] name; } //Деструктор

};

*4. Для чего используется конструктор без параметров? Конструктор с параметрами? Конструктор копирования?*

*Конструктор без параметров:*

Конструктор без параметров используется для создания «пустого» объекта, либо значения полей задаются по умолчанию.

Пример:

class Person

{

private:

string name;

int age;

public:

Person() { //Конструктор без параметров

//Пустой объект

name = "";

age = 0;

}

Person() {

//Объект с полями

name = "Ivan";

age = 18;

}

};

Конструктор без параметров вызывается следующим образом:

int main() {

Person p;

return 0;

}

*Конструктор с параметрами:*

Конструктор с параметрами, используется для инициализации объекта требуемыми значениями.

Пример:

class Person

{

private:

string name;

int age;

public:

Person(string s, int i) { //Конструктор с параметрами

name = s;

age = i;

}

};

Конструктор с параметрами вызывается следующим образом:

int main() {

Person p("Ivan", 18);

return 0;

}

*Конструктор копирования:*

**Конструктор копирования** — это специальный вид конструктора, получающий в качестве единственного параметра указатель на объект этого же класса. Если программист не указал ни одного конструктора копирования, компилятор создает его автоматически. Такой конструктор выполняет поэлементное копирование полей. Если класс содержит указатели или ссылки, это, скорее всего, будет неправильным, поскольку и копия, и оригинал будут указывать на одну и ту же область памяти.

Пример:

class Person

{

private:

string name;

int age;

public:

Person(const Person& p) { //Конструктор копирования

name = p.name;

age = p.age;

}

};

Конструктор копирования вызывается следующим образом:

int main() {

Person p;

Person a(p); //Копирование через ()

Person h = p; //Копирование через =

return 0;

}

*5. В каких случаях вызывается конструктор копирования?*

Конструктор копирования вызывается в следующих случаях:

- Описан конструктор копирования, и использован оператор присваивания (вопрос 4);

- Описание нового объекта с инициализацией другим объектом (вопрос 4);

- При передаче объекта в функцию по значению. Компилятор создает временный (локальный) объект, и использует конструктор копирования по умолчанию.

void print(Person a) //Объект передается по значению

{

a.print\_person();

}

В функции создается временный объект, и метод print\_person() применяется ко временному объекту, а не к тому, что передается в функцию.

- При возврате объекта из функции.

Person set\_inf()

{

Person s;//создается временный объект

s.set\_person("Sidorov", 5, 2);

return s;//объект возвращается как значение функции

}

int main() {

Person h = set\_inf();

return 0;

}

Объект Person h является копией объекта Person s, который создаётся в функции.

*6. Перечислить свойства конструкторов.*

- Конструктор не возвращает значение, даже типа void. Нельзя получить указатель на конструктор.

- Класс может иметь несколько конструкторов с разными параметрами для разных видов инициализации (при этом используется механизм перегрузки). - Конструктор, вызываемый без параметров, называется конструктором по умолчанию.

- Параметры конструктора могут иметь любой тип, кроме этого же класса. Можно задавать значения параметров по умолчанию, но их может содержать только один из конструкторов.

- Если программист не указал ни одного конструктора, компилятор создает его автоматически. Такой конструктор вызывает конструкторы по умолчанию для полей класса. В случае, когда класс содержит константы или ссылки, при попытке создания объекта класса будет выдана ошибка, поскольку их необходимо инициализировать конкретными значениями, а конструктор по умолчанию этого делать не умеет.

- Конструкторы не наследуются.

- Конструкторы нельзя описывать с модификаторами const, virtual и static.

- Конструкторы глобальных объектов вызываются до вызова функции main. Локальные объекты создаются, как только становится активной область их действия. Конструктор запускается и при создании временного объекта (на пример, при передаче объекта из функции).

-Конструктор вызывается, если в программе встретилась какая-либо из синтаксических конструкций:

* имя\_класса имя\_объекта;

Person p1; //Конструктор без параметров

* имя\_класса имя\_объекта [(список параметров)]; // Список параметров не должен быть пустым

Person p2("Ivan",18); //Конструктор с параметрами

* имя\_класса (список параметров); // Создается объект без имени (список может быть пустым)

Person();

* имя\_класса имя\_объекта = выражение; Person p3(p2); //Копирование

имя\_класса имя\_объекта (выражение); Person p4 = p1; //Копирование

*7. Перечислить свойства деструкторов.*

- Не имеет аргументов и возвращаемого значения;

- Не наследуется;

- Не может быть объявлен как const или static;

- Может быть виртуальным;

*8. К каким атрибутам имеют доступ методы класса?*

Методы класса имеют неограниченный доступ ко **всем элементам класса**, независимо от спецификаторов доступа (поля и методы имеют одну область видимости – внутри класса) и порядка объявления методов в классе.

*9. Что представляет собой указатель this?*

Метод (кроме статических методов) неявно получает в качестве аргумента **указатель на тот объект, для которого он вызван**. Этот указатель обозначается ключевым словом this и может быть использован в теле метода. В явном виде этот указатель применяется в основном для возвращения из метода указателя (return this;) или ссылки (return \*this;) на вызвавший объект. Также this используется в методах, параметры которых совпадают с именами полей.

class Person

{

private:

string name;

int age;

public:

void set\_age(int age) {

this->age = age;

}

};

*10. Какая разница между методами определенными внутри класса и вне класса?*

Методы могут определяться как в классе, так и вне его. Определение метода **внутри класса** ничем не отличается от определения обычной функции. По умолчанию такой метод считается встроенной функцией (inline).

class Person

{

private:

string name;

int age;

public:

void set\_age(int age) {

this->age = age;

}

};

Если метод определяется **вне класса**, то принадлежность метода классу указывается с помощью имени класса:

Тип\_функции Имя\_класса:: Имя\_метода (параметры){тело метода;}.

В классе присутствует только прототип.

void Person::set\_age(int a) {

age = a;

}

*11. Какое значение возвращает конструктор?*

Конструктор **не возвращает значение**, даже типа void. Конструктор только инициализирует поля объекта.

*12. Какие методы создаются по умолчанию?*

По умолчанию создаются **конструктор без параметров**, **конструктор копирования** и **деструктор**.

*13. Какое значение возвращает деструктор?*

Деструктор **не имеет возвращаемого значения**.

*14. Дано описание класса*

class Student

{

string name;

int group;

public:

Student(string, int);

Student(const Student&);

~Student();

};

*Какой метод отсутствует в описании класса?*

Отсутствует **конструктор без параметров** (по умолчанию)

class Student

{

string name;

int group;

public:

Student(); //Конструктор без параметров

Student(string, int);

Student(const Student&);

~Student();

};

*15. Какой метод будет вызван при выполнении следующих операторов:*

int main() {

Student\* s;

s = new Student;

return 0;

}

Будет вызван **конструктор без параметров** (по умолчанию).

*16. Какой метод будет вызван при выполнении следующих операторов:*

int main() {

Student s("Ivanov", 20);

return 0;

}

Будет вызван **конструктор с параметрами**.

*17. Какие методы будут вызваны при выполнении следующих операторов:*

int main() {

Student s1("Ivanov", 20);

Student s2 = s1;

return 0;

} Будет вызван **конструктор с параметрами** и **конструктор копирования**.

*18. Какие методы будут вызваны при выполнении следующих операторов:*

int main() {

Student s1("Ivanov", 20);

Student s2;

s2 = s1;

return 0;

}

Будут вызваны **конструктор с параметрами**, **конструктор без параметров** и **конструктор копирования**.

*19. Какой конструктор будет использоваться при передаче параметра в функцию print():*

void print(Student a)

a.show();

}

Будет использоваться **конструктор копирования**.

*20. Класс описан следующим образом:*

class Student

{

string name;

int age;

public:

void set\_name(string);

void set\_age(int);

};

*Каким образом можно присвоить новое значение атрибуту name объекта р?*

Так как поле name имеет доступ private, то обратиться к нему напрямую (например, по имени) нельзя. Для того, чтобы присвоить полю новое значение, необходимо воспользоваться **public-методом set\_name** (модификатор).

int main() {

Student s1;

s1.set\_name("Ivan");

return 0;

}