Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра Информационные технологии и автоматизированные системы

Лабораторная работа № 2

Тема: «Классы и объекты. Использование конструкторов»

Выполнил: студент группы РИС-22-2б

Мизёв В.А. ф

Проверил: доцент кафедры ИТАС

Полякова О.А.

г. Пермь – 2024

*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.

*Задание вариант 1:*

Пользовательский класс СТУДЕНТ

ФИО – string

Группа – string

Средний балл – float

*Анализ задачи:*

Основываясь на задании необходимо реализовать один класс “Student”, с полями m\_fcs, m\_group, m\_averageScore, а также методы для получения и установки значений для этих полей. Три конструктора: конструктор по умолчанию Student(), копирования Student(const Student& student) и с параметрами Student(std::string fcs, std::string group, float averageScore).

*Код программы:*

Прикреплён в приложении 1.

*UML-диаграмма класса:*

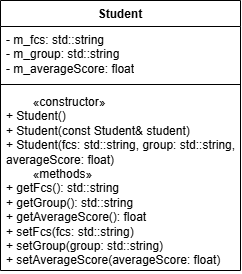


Рисунок 1 - UML-диаграмма класса Student

*Скриншот работы программы:*

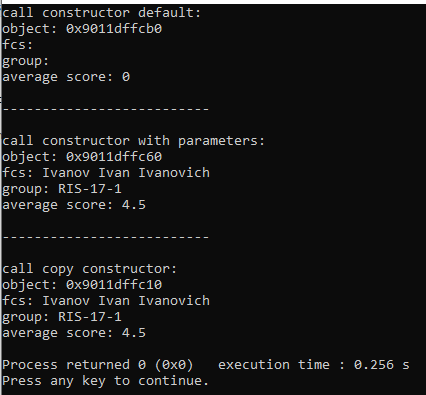


Рисунок 2 – Результат работы программы

*Контрольные вопросы:*

1. Для чего нужен конструктор?

Для инициализации класса. Конструктор вызывается один раз при создании экземпляра класса. Student(“test”, “fcs”, 2);

1. Сколько типов конструкторов существует в С++?

Три конструктора, конструктор по умолчанию, копирования и с параметрами.

Student(), Student(const Student& student), Student(std::string fcs)

1. Для чего используется деструктор? В каких случаях деструктор описывается явно?

Для освобождения ресурсов, выделенных конструктором объекту. Деструктор описывается явно в случаи если есть указатель на память, выделяемую динамически.

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

Конструктор без параметров используется для создания «пустого» объекта.

Конструктор с параметрами используется для инициализации объекта требуемыми значениями

Конструктор копирования вызывается в тех случаях, когда новый объект создаётся путём копирования существующего.

1. В каких случаях вызывается конструктор копирования?

Конструктор копирования вызывается в тех случаях, когда новый объект создаётся путём копирования существующего.

1. 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(например, при передаче объекта из функции).
* Конструктор вызывается, если в программе встретилась какая-либо из синтаксических конструкций.

1. Перечислить свойства деструкторов.

Имя метода называется также, как и типа данных, с тильдой, деструктор ничего не возвращает, деструктор не имеет параметров.

1. К каким атрибутам имеют доступ методы класса?

Всем атрибутам внутри класса, исключение если класс наследуется от другого класса, в котором поля классы объявлены в модификаторе доступа private.

1. Что представляет собой указатель this?

Указатель на текущий объект данного класса.

1. Какая разница между методами, определенными внутри класса и вне класса?

Методы определёнными внутри класса будут inline методами, вне класса необходимо определить пространство имён и название метода, в самом классе же останется прототип функции, в целом разницы в определениях нет, единственное могут быть проблемы с повторным подключением, но для этого можно использовать include guard

#ifndef H\_FILE

#define H\_FILE

#endif

class Student { void test() { std::cout << “test\n”; } void test2(); };

Student::test2(){ std::cout << “test2\n”; }

1. Какое значение возвращает конструктор?

Конструктор ничего не возвращает

1. Какие методы создаются по умолчанию?

Конструктор и деструктор по умолчанию.

1. Какое значение возвращает деструктор?

Деструктор ничего не возвращает.

1. 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 по умолчанию

1. Какие методы будут вызваны при выполнении следующих операторов:

Student\* s;

s=new student;

Конструктор по умолчанию

1. Какие методы будут вызваны при выполнении следующих операторов:

student s(“Ivanov”,. 20);

Конструктор с параметрами.

1. Какие методы будут вызваны при выполнении следующих операторов:

student s1(“Ivanov”, 20);

student s2=s1;

У s1 конструктор с параметрами, s2 конструктор копирования

1. Какие методы будут вызваны при выполнении следующих операторов:

student s1(“Ivanov”, 20);

student s2;

s2 = s1;

Вызывается конструктор с параметрами при инициализации s1.

При копировании данных через присвоение по умолчанию все данные одного объекта присваиваются другому путём побитового копирования.

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

void print(student a){

a.show();

}

Конструктор копирования

1. Класс описан следующим образом:

class Student {

string name;

int age;

public:

void set\_name(string);

void set\_age(int);

……

};

Student p;

Каким образом можно присвоить новое значение атрибуту name объекта p?

Вызвать метод set\_name(), передав имя в качестве аргумента.

**ПРИЛОЖЕНИЕ 1**

lab2.cpp

//-------------------------------------

#include <iostream>

#include "lab2\_student.hpp"

//-------------------------------------

#define IDENT\_PRINT printf("\n--------------------------\n\n")

//-------------------------------------

int main() {

//call constructor default

std::cout << "call constructor default:\n";

Student student;

student.debugShow();

IDENT\_PRINT;

//call constructor with parameters

std::cout << "call constructor with parameters:\n";

Student student1("Ivanov Ivan Ivanovich", "RIS-17-1", 4.5f);

student1.debugShow();

IDENT\_PRINT;

//call copy constructor

std::cout << "call copy constructor:\n";

Student student2(student1);

student2.debugShow();

return 0;

}

//-------------------------------------

lab2\_student.hpp

//-------------------------------------

#ifndef LAB2\_STUDENT\_HPP\_INCLUDED

#define LAB2\_STUDENT\_HPP\_INCLUDED

//-------------------------------------

#include <iostream>

//-------------------------------------

class Student {

public:

Student();

Student(const Student& student);

Student(std::string fcs, std::string group, float averageScore);

~Student();

std::string getFcs() { return m\_fcs; }

std::string getGroup() { return m\_group; }

float getAverageScore() { return m\_averageScore; }

void setFcs(std::string fcs) { m\_fcs = fcs; }

void setGroup(std::string group) { m\_group = group; }

void setAverageScore(float averageScore) { m\_averageScore = averageScore; }

#ifdef debug\_v

void debugShow() {

std::cout << "object: " << this

<< "\nfcs: " << m\_fcs

<< "\ngroup: " << m\_group

<< "\naverage score: " << m\_averageScore << std::endl;

}

#endif // debug\_v

private:

std::string m\_fcs,

m\_group;

float m\_averageScore;

};

//-------------------------------------

#endif // LAB2\_STUDENT\_HPP\_INCLUDED

//-------------------------------------

lab2\_student.cpp

//-------------------------------------

#include "lab2\_student.hpp"

//-------------------------------------

Student::Student() {

m\_averageScore = 0.0f;

}

//-------------------------------------

Student::Student(const Student& student) {

m\_fcs = student.m\_fcs;

m\_group = student.m\_group;

m\_averageScore = student.m\_averageScore;

}

//-------------------------------------

Student::Student(std::string fcs, std::string group, float averageScore) {

m\_fcs = fcs;

m\_group = group;

m\_averageScore = averageScore;

}

//-------------------------------------

Student::~Student() {

//

}

//-------------------------------------