**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №7

«Виртуальные функции и классы»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-24б |  | преподаватель каф. ИУ5 |
| Байдаков В.М. |  |  |
| Подпись и дата: |  | Подпись и дата: |
|  |  |  |

Москва, 2024 г.

**Постановка задачи:**

Целью лабораторной работы изучение механизмов динамического связывания. Студенты изучают понятия абстрактного класса, виртуальных функций и виртуальных классов. Они осваивают на практике по индивидуальным заданиям использование этих прием при программировании в среде С++.

**Разработка алгоритма:**

Библиотека Person

Класс Person - базовый класс, представляющий человека

unsigned age - возраст человека

char\* name - имя человека

bool gender - пол человека (true если мужчина, false если женщина)

static Person\*\* persons - статический массив указателей на объекты Person

static size\_t size - текущее количество объектов Person

static size\_t capacity - текущая вместимость массива persons

static bool autoAdd – добавляет объект в массив при значении 1

static void Print() – выводит массив объектов

Person() - конструктор, инициализирует объект Person

virtual ~Person() - виртуальный деструктор, освобождает память, занятую name

virtual void show() - виртуальная функция для вывода информации о человеке

virtual void add() - виртуальная функция для добавления объекта в массив persons

Person& operator=(const Person& per) - оператор присваивания, копирует данные из другого объекта Person

Person(const Person &per) – конструктор копирования

Класс Employee - производный класс от Person, представляющий сотрудника

char\* workplace - место работы сотрудника

Employee() - конструктор, инициализирует объект Employee

void show() - выводит информацию о сотруднике

void add() - добавляет объект Employee в массив persons, расширяет массив при необходимости

~Employee() - деструктор, освобождает память, занятую workplace

Employee& operator=(const Employee& emp) - оператор присваивания, копирует данные из другого объекта Employee

Employee(const Employee &per) – конструктор копирования

Класс Worker - производный класс от Employee, представляющий рабочего

unsigned salary - зарплата рабочего

Worker() - конструктор, инициализирует объект Worker

void show() - выводит информацию о рабочем

Worker& operator=(const Worker& per) - оператор присваивания, копирует данные из другого объекта Worker

Класс Engineer

Engineer - производный класс от Worker, представляющий инженера

char\* specialization - специализация инженера

Engineer() - конструктор, инициализирует объект Engineer

void show() - выводит информацию об инженере

~Engineer() - деструктор, освобождает память, занятую specialization

Engineer& operator=(const Engineer& per) - оператор присваивания, копирует данные из другого объекта Engineer

Engineer(const Engineer &per) – конструктор копирования

**Текст программы**

**Person.h**

#pragma once

#include <cstring>

#include <iostream>

class Person

{

protected:

  unsigned age;

  char \*name;

  bool gender;

public:

  static bool autoAdd;

  static Person \*\*persons;

  static size\_t size;

  static size\_t capacity;

  static void Print();

  Person(unsigned a, char \*na, bool ge);

  virtual ~Person()

  {

    delete[] name;

  }

  virtual void show() = 0;

  void add();

  Person &operator=(const Person &per);

  Person(const Person &per);

};

class Employee : public Person

{

protected:

  char \*workplace;

public:

  Employee(unsigned a, char \*na, bool ge, char \*workp);

  void show();

  ~Employee();

  Employee &operator=(const Employee &per);

  Employee(const Employee &per);

};

class Worker : public Employee

{

protected:

  unsigned salary;

public:

  Worker(unsigned a, char \*na, bool ge, char \*workp, unsigned sal);

  void show();

  Worker &operator=(const Worker &per);

  Worker(const Worker &per);

};

class Engineer : public Worker

{

protected:

  char \*specialization;

public:

  Engineer(unsigned a, char \*na, bool ge, char \*workp, unsigned sal, char \*spec);

  void show();

  ~Engineer();

  Engineer &operator=(const Engineer &per);

  Engineer(const Engineer &per);

};

**Person.cpp**

#include "person.h"

void Person::Print(){

  for (size\_t i = 0; i < Person::size; i++) {

    Person::persons[i]->show();

  }

}

Person::Person(unsigned a = 0, char \*na = "\0", bool ge = 0) : age(a), gender(ge)

{

  name = new char[strlen(na) + 1];

  strcpy(name, na);

  if (autoAdd)

  {

    add();

  }

}

Person &Person::operator=(const Person &per)

{

  age = per.age;

  if (name)

  {

    delete[] name;

  }

  name = new char[strlen(per.name) + 1];

  strcpy(name, per.name);

  gender = per.gender;

  return \*this;

}

Person::Person(const Person &per)

{

  age = per.age;

  name = new char[strlen(per.name) + 1];

  strcpy(name, per.name);

  gender = per.gender;

}

Employee::Employee(unsigned a, char \*na = "\0", bool ge = 0, char \*workp = "\0") : Person(a, na, ge)

{

  workplace = new char[strlen(workp) + 1];

  strcpy(workplace, workp);

}

void Employee::show()

{

  std::cout << "employee " << age << ' ' << name << ' ' << gender << ' ' << workplace << '\n';

}

void Person::add()

{

  if (size >= capacity)

  {

    capacity \*= 2;

    Person \*\*tmp = new Person \*[capacity];

    for (size\_t i = 0; i < size; i++)

    {

      tmp[i] = persons[i];

    }

    delete[] persons;

    persons = tmp;

  }

  persons[size++] = this;

}

Employee::~Employee()

{

  delete[] workplace;

}

Employee &Employee::operator=(const Employee &per)

{

  age = per.age;

  if (name)

  {

    delete[] name;

  }

  name = new char[strlen(per.name) + 1];

  strcpy(name, per.name);

  gender = per.gender;

  if (workplace)

  {

    delete[] workplace;

  }

  workplace = new char[strlen(per.workplace) + 1];

  strcpy(workplace, per.workplace);

  return \*this;

}

Employee::Employee(const Employee &per)

{

  age = per.age;

  name = new char[strlen(per.name) + 1];

  strcpy(name, per.name);

  gender = per.gender;

  workplace = new char[strlen(per.workplace) + 1];

  strcpy(workplace, per.workplace);

}

Worker::Worker(const Worker& per):Employee(per),salary(per.salary){}//тут11111111111!!!

Worker::Worker(unsigned a = 0, char \*na = "\0", bool ge = 0, char \*workp = "\0", unsigned sal = 0) : Employee(a, na, ge, workp)

{

  salary = sal;

}

void Worker::show()

{

  std::cout << "worker " << age << ' ' << name << ' ' << gender << ' ' << workplace << ' ' << salary << '\n';

}

Worker &Worker::operator=(const Worker &per)

{

  age = per.age;

  if (name)

  {

    delete[] name;

  }

  name = new char[strlen(per.name) + 1];

  strcpy(name, per.name);

  gender = per.gender;

  if (workplace)

  {

    delete[] workplace;

  }

  workplace = new char[strlen(per.workplace) + 1];

  strcpy(workplace, per.workplace);

  salary = per.salary;

  return \*this;

}

Engineer::Engineer(unsigned a = 0, char \*na = "\0", bool ge = 0, char \*workp = "\0", unsigned sal = 0, char \*spec = "\0") : Worker(a, na, ge, workp, sal)

{

  specialization = new char[strlen(spec) + 1];

  strcpy(specialization, spec);

}

void Engineer::show()

{

  std::cout << "engineer " << age << ' ' << name << ' ' << gender << ' ' << workplace << ' ' << salary << ' ' << specialization << '\n';

}

Engineer::~Engineer()

{

  delete[] specialization;

}

Engineer &Engineer::operator=(const Engineer &per)

{

  age = per.age;

  if (name)

  {

    delete[] name;

  }

  name = new char[strlen(per.name) + 1];

  strcpy(name, per.name);

  gender = per.gender;

  if (workplace)

  {

    delete[] workplace;

  }

  workplace = new char[strlen(per.workplace) + 1];

  strcpy(workplace, per.workplace);

  salary = per.salary;

  if (specialization)

  {

    delete[] specialization;

  }

  specialization = new char[strlen(per.specialization) + 1];

  strcpy(specialization, per.specialization);

  return \*this;

}

Engineer::Engineer(const Engineer &per)

{

  age = per.age;

  name = new char[strlen(per.name) + 1];

  strcpy(name, per.name);

  gender = per.gender;

  workplace = new char[strlen(per.workplace) + 1];

  strcpy(workplace, per.workplace);

  salary = per.salary;

  specialization = new char[strlen(per.specialization) + 1];

  strcpy(specialization, per.specialization);

}

**Main.cpp**

#include "person.h"

Person \*\*Person::persons = new Person\*[2];

size\_t Person::size = 0;

size\_t Person::capacity = 2;

bool Person::autoAdd = true;

int main() {

  Employee emp(1337, "cheliks", 1, "bmstu");

  Worker wrk(18, "vanek", 0, "zavod", 15);

  Engineer eng(12, "vlados", 1, "mai", 15000, "space engineer");

  Engineer eng1(123, "slavik", 1, "fabrica", 15000, "airplane specialization");

  Worker wrk1(wrk);

  // emp.add();

  // wrk.add();

  // eng.add();

  // eng1.add();

  // wrk1.add();

  Person::Print();

  delete[] Person::persons;

  return 0;

}

**Cmakelists.txt для всей лабы и библиотеки соответственно**

cmake\_minimum\_required(VERSION 3.23)

set(project "lab7")

project(${project})

set(CMAKE\_CXX\_STANDARD 17)

set(SOURCES

  main.cpp

)

add\_subdirectory(Person)

add\_executable(${project}

        ${SOURCES})

target\_link\_libraries(${project} Person)

target\_include\_directories(${project} PUBLIC

  ${CMAKE\_SOURCE\_DIR}/Person

)

cmake\_minimum\_required(VERSION 3.23)

set(project "Person")

project(${project})

set(CMAKE\_CXX\_STANDARD 17)

set(SOURCES

  person.cpp

)

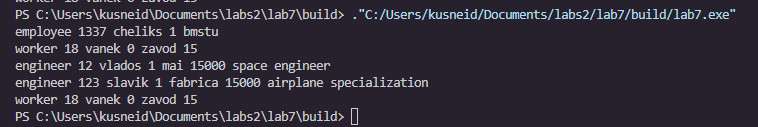
set(HEADERS

  person.h

)

add\_library(${project} STATIC ${SOURCES} ${HEADERS})

**Анализ результатов**

****