МИНИСТЕРСТВО СВЯЗИ И ИНФОРМАТИЗАЦИИ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«БЕЛОРУССКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ СВЯЗИ»

ФАКУЛЬТЕТ ЭЛЕКТОСВЯЗИ

КАФЕДРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СЕТЕЙ ТЕЛЕКОММУНИКАЦИЙ

Написание программ с использованием пользовательских классов

Отчет по лабораторной работе №1

по учебному предмету

«Объектно-ориентированное программирование»

|  |  |
| --- | --- |
| Выполнил учащийся гр. ТП312 | Рудский Геннадий |
| Руководители | Олескевич А.А.  Малолеткин А.Б. |

Минск 2025

**Цель**: изучить основы программирования в среде Visual Studio, научиться создавать простейшие проекты, познакомиться со структурой объектно- ориентированной программы.

**Практические задания:**

*Изучение модификаторов доступа*

Для управления защитой элементов, входящих в состав класса, необходимо явно указать нужный атрибут, после которого добавить символ ":" (двоеточие).

Пример:

class X

{

int a, b; //Поля a и b будут иметь доступ private по умолчанию public:

double a1, a2;; //Поля a1 и a2 будут иметь доступ public protected:

char p, l; //Поля p и l будут иметь доступ protected

};

*Создание объектов и инициализация д-х*

Существует два способа создания объектов с таким составом данных:

1. Между символами "}" и ";" помещается список идентификаторов объектов, и уже на этапе компиляции будут созданы объекты с указанным типом данных. При таком способе задания атрибут ID\_класса можно опускать.

Пример:

#include "pch.h" #include <iostream> using namespace std; class Test

{ public: int a, b;

double c; };

m, \*t = &m; //Инициализируем объект m типа Test и указатель этого типа t, который указывает на наш объект m

int main()

{

//Поля объекта m в данном пример не инициализированы, так как в классе не объявлен конструктор для их инициализации

// Инициализировать поля можно вручную

t->a = 5; //Не забываем про то, что t – это указатель и сейчас мы даем значение 5 полю "a" для объекта m через косвенное обращение

m.b = 3; //Даем значение 3 полю "b" объекта m прямым обращением

cout<<t->a; //Косвенное обращение к полю "a" объекта m и вывод его значе-

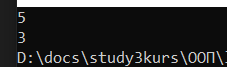
ния

cout << endl << m.b; //Прямое обращение к полю "b" объекта m и вывод его

значения

}

Работа примера:



1. Декларация объектов класса осуществлятся в любом месте программы по мере надобности: ID\_класса список ID\_объектов;

Пример:

#include "pch.h" #include <iostream> using namespace std; class Test

{

public:

int a, b; double c;

};

int main()

{

//Поля переменных t и m в данном пример не инициализированы, так как в классе не объявлен конструктор для их инициализации

Test x; //Инициализируем объект x

X.a = 4; //Даем значение полю a объекта x Test \*y; //Создаем указатель

y = &x; //Указатель теперь указывает на место расположения объекта x в па-

мяти

cout<<y->a; //Косвенное обращение к полю "a" объекта x, так как y хранит

указатель на адрес объекта x в памяти cout<<endl<<x.a; //Прямое обращение к полю "a"

}

Работа примера:



*Декларация методов класса*

Декларация методов класса возможна в двух формах:

1. в форме определения, при которой приводится полный текст метода;

Пример:

#include "pch.h" #include <iostream> using namespace std; class Test

{

public:

int a, b; double c;

void show() //Полный текст метода написан внутри класса

{ cout << a<< b;

}

};

int main()

{

Test x; //Инициализируем объект x

x.a = 5;

x.b = 9;

x.c = 0; x.show();

}

Работа программы



1. в форме описания (прототип метода): в данном случае вне шаблона должно быть полное определение метода, при этом необходимо указать, к какому классу принадлежит данный метод, для этого используется операция привязки "::", назна- чение этой операции – привязать функцию к конкретному классу.

Пример:

#include "pch.h"

#include <iostream> using namespace std; class Test

{

public: int a, b;

double c;

void show(); //Инициализируем метод в классе

};

void Test::show() //Пишем код метода, Test:: – говорит о том, что данный код должен выполниться при вызове метода show(), инициализированного в классе Test

{

cout << a << b;

}

void show() //Мы можем создать метод с таким же названием и такой же сиг- натурой, но для объекта код этого метода не выведется, так как нет привязки к классу

{

cout << "User metods.";

}

int main()

{

Test x; //Инициализируем объект x

x.a = 5;

x.b = 9;

x.c = 0;

x.show(); //Output: 59

show(); //Output: User metods.

}

Работа примера:



*Прямая и косвенная адресация*

1) Прямой вызов метода ID\_объекта.ID\_метода; – использована операция привязки "точка".

#include "pch.h" #include <iostream> using namespace std; class Test

{

public: int a, b; double c;

void show() //Полный текст метода написан внутри класса

{ cout << a << b; }}; int main()

{

Test x; //Инициализируем объект x

//прямой вызов полей

x.a = 5;

x.b = 9;

x.c = 0;

//Прямой вызов метода x.show();

}

Работа примера:



2)Косвенная адресация – объявлен указатель с типом ID\_класса \*ID\_указа- теля. Теперь указатель типа ID\_класса можно устанавливать на существующие объ- екты этого класса: ID\_указателя = & ID\_объект; – указатель идентифицирован ад- ресом объекта.

Косвенный вызов метода: ID\_указателя -> ID\_метода; – использована опера- ция привязки "стрелка".

Пример:

#include "pch.h" #include <iostream> using namespace std; class Test

{

public:

int a, b; double c;

void show() //Полный текст метода написан внутри класса

{ cout << a << b; }

};

int main()

{

Test x; //Инициализируем объект x

Test \*y; //Инициализируем указатель типа Test

y = &x; //Теперь указатель y указывает на адрес объекта x и теперь мы можем кос- венно к нему обращаться

//Косвенный вызов полей cout << y->a << y->b << y->c;

//Косвенный вызов метода y->show();

}

Работа примера:



Создание объектов (безымянных) в динамической области памяти ID\_класса \* ID\_указателя = new ID\_класса; Компилятор создает объект на этапе работы проекта. Косвенный вызов метода: ID\_указателя -> ID\_метода;

Пример:

#include "pch.h" #include <iostream> using namespace std; class Test

{

public:

int a, b; double c;

void show() //Полный текст метода написан внутри класса

{

cout << a << b;

}

};

int main()

{

Test \*y = new Test(); //Создаем динамический объект в памяти, на расположение которого указывает указатель y

//Косвенный вызов полей cout << y->a << y->b << y->c;

//Косвенный вызов метода y->show();

}

Работа примера:



**Индивидуальные задания**:

На лабораторной работе были выполнены индивидуальные задания варианта 3.

1 задание:

Сегодня вы заступили на должность главврача больницы "Healthy like a bull". И первое, что бы вы хотели узнать, – это процент выздоровевших, больных и симулянтов от общего числа пациентов.

Ваша задача: создать класс с 3 полями (типа int) – convalescents, sick, simula- tors – и метод, который вернет количество выздоровевших, больных и симулянтов в процентах от общего числа пациентов. Общее количество пациентов:

(*convalescents* + *sick* + *simulators*).

На листинге 1 размещён код файла task1.cpp с функцией main.

#include <iostream>

#include "pacient.h"

int main()

{

setlocale(LC\_ALL, "Russian");

pacients\* Hospital = new pacients(0, 0, 0);

Hospital->inputPacients();

Hospital->outputPacients();

delete Hospital;

}

Листинг 1 – task1.cpp

На листинге 2 размещён код файла pacient.h с созданным классом и созданными прототипами методов.

#pragma once

class pacients

{

private:

int Convalescents;

int Sick;

int Simulators;

public:

pacients(int Convalescents, int Sick, int Simulators);

~pacients();

int getConvalescents()const;

int getSick()const;

int getSimulators()const;

void setConvalescents(int Convalescents);

void setSick(int Sick);

void setSimulators(int Simulators);

void coutSetConvalescentsMessage()const;

void coutSetSickMessage()const;

void coutSetSimulatorsMessage()const;

void inputPacients();

void outputPacients()const;

};

Листинг 2 – pacient.h

На листинге 3 размещён код файла pacient.cpp с определением функций класса.

#include "pacient.h"

#include <iostream>

using namespace std;

pacients::pacients(int Convalescents, int Sick, int Simulators)

{

this->Convalescents = Convalescents;

this->Sick = Sick;

this->Simulators = Simulators;

cout << "Объект создан!" << endl;

}

pacients::~pacients()

{

cout << "Объект удалён!" << endl;

}

//Getters

int pacients::getConvalescents() const

{

return Convalescents;

}

int pacients::getSick() const

{

return Sick;

}

int pacients::getSimulators() const

{

return Simulators;

}

//Setters

void pacients::setConvalescents(int Convalescents)

{

this->Convalescents = Convalescents;

}

void pacients::setSick(int Sick)

{

this->Sick = Sick;

}

void pacients::setSimulators(int Simulators)

{

this->Simulators = Simulators;

}

void pacients::coutSetConvalescentsMessage() const

{

cout << "Введите количество выздоровевших: ";

}

void pacients::coutSetSickMessage() const

{

cout << "Введите количество больных: ";

}

void pacients::coutSetSimulatorsMessage() const

{

cout << "Введите количество симулянтов: ";

}

//Special methods

void pacients::inputPacients()

{

int Convalescents;

int Sick;

int Simulators;

coutSetConvalescentsMessage();

cin >> Convalescents;

setConvalescents(Convalescents);

coutSetSickMessage();

cin >> Sick;

setSick(Sick);

coutSetSimulatorsMessage();

cin >> Simulators;

setSimulators(Simulators);

}

void pacients::outputPacients() const

{

double total;

double totalPecentConvalescents;

double totalPecentSick;

double totalPecentSimulators;

total = getConvalescents() + getSick() + getSimulators();

totalPecentConvalescents = getConvalescents() \* 100 / total;

totalPecentSick = getSick() \* 100 / total;

totalPecentSimulators = getSimulators() \* 100 / total;

cout << "Выздоровевших в больнице " << totalPecentConvalescents << "% от всех пациентов в больнице." << endl;

cout << "Больных в больнице " << totalPecentSick << "% от всех пациентов в больнице." << endl;

cout << "Симулянтов в больнице " << totalPecentSimulators << "% от всех пациентов в больнице." << endl;

}

Листинг 3 – pacient.cpp

Результат работы программы на рисунке 1.

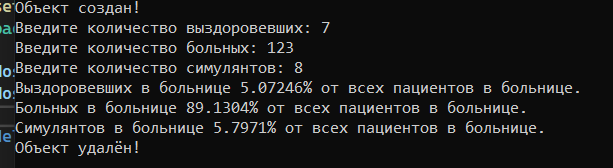


Рисунок 1 – работа программы 1

2 задание

Вы получили статистику пациентов. А что насчет персонала? Сколько в целом людей в вашей организации? Это необходимо подсчитать в процентном от- ношении.

Ваша задача: добавить 3 поля (тип int) – Doctor, Nurse, MedicalOrderly – и ме- тод, который выведет количество сотрудников каждого типа в процентах от общего количества сотрудников.

На листинге 4 размещён код файла task1.cpp с функцией main.

#include <iostream>

#include "employee.h"

int main()

{

setlocale(LC\_ALL, "Russian");

employeers\* Hospital = new employeers(0, 0, 0);

Hospital->inputEmployeers();

Hospital->outputEmployeers();

delete Hospital;

}

Листинг 4 – task2.cpp

На листинге 5 размещён код файла employee.h с созданным классом и созданными прототипами методов.

#pragma once

class employeers

{

private:

int Doctor;

int Nurse;

int MedicalOrderly;

public:

employeers(int Doctor, int Nurse, int MedicalOrderly);

~employeers();

int getDoctor()const;

int getNurse()const;

int getMedicalOrderly()const;

void setDoctor(int Doctor);

void setNurse(int Nurse);

void setMedicalOrderly(int MedicalOrderly);

void coutSetDoctorMessage()const;

void coutSetNurseMessage()const;

void coutSetMedicalOrderlyMessage()const;

void inputEmployeers();

void outputEmployeers()const;

};

Листинг 5 – employee.h

На листинге 6 размещён код файла employee.cpp с определением функций класса.

#include <iostream>

#include "employee.h"

using namespace std;

employeers::employeers(int Doctor, int Nurse, int MedicalOrderly)

{

this->Doctor = Doctor;

this->Nurse = Nurse;

this->MedicalOrderly = MedicalOrderly;

cout << "Объект создан!"<< endl;

}

employeers::~employeers()

{

cout << "Объект удалён!"<< endl;

}

//Getters

int employeers::getDoctor() const

{

return Doctor;

}

int employeers::getNurse() const

{

return Nurse;

}

int employeers::getMedicalOrderly() const

{

return MedicalOrderly;

}

//Setters

void employeers::setDoctor(int Doctor)

{

this->Doctor = Doctor;

}

void employeers::setNurse(int Nurse)

{

this->Nurse = Nurse;

}

void employeers::setMedicalOrderly(int MedicalOrderly)

{

this->MedicalOrderly = MedicalOrderly;

}

void employeers::coutSetDoctorMessage() const

{

cout << "Введите количество докторов: ";

}

void employeers::coutSetNurseMessage() const

{

cout << "Введите количество медсестёр: ";

}

void employeers::coutSetMedicalOrderlyMessage() const

{

cout << "Введите количество санитаров: ";

}

//Special methods

void employeers::inputEmployeers()

{

int Doctor;

int Nurse;

int MedicalOrderly;

coutSetDoctorMessage();

cin >> Doctor;

setDoctor(Doctor);

coutSetNurseMessage();

cin >> Nurse;

setNurse(Nurse);

coutSetMedicalOrderlyMessage();

cin >> MedicalOrderly;

setMedicalOrderly(MedicalOrderly);

}

void employeers::outputEmployeers() const

{

double total;

double totalPecentDoctor;

double totalPecentNurse;

double totalPecentMedicalOrderly;

total = getDoctor() + getNurse() + getMedicalOrderly();

totalPecentDoctor = getDoctor() \* 100 / total;

totalPecentNurse = getNurse() \* 100 / total;

totalPecentMedicalOrderly = getMedicalOrderly() \* 100 / total;

cout << "Докторов в больнице " << totalPecentDoctor << "% от всех сотрудников в больнице." << endl;

cout << "Медсестёр в больнице " << totalPecentNurse << "% от всех сотрудников в больнице." << endl;

cout << "Санитаров в больнице " << totalPecentMedicalOrderly << "% от всех сотрудников в больнице." << endl;

}

Листинг 6 – employee.cpp

Результат работы программы на рисунке 2.

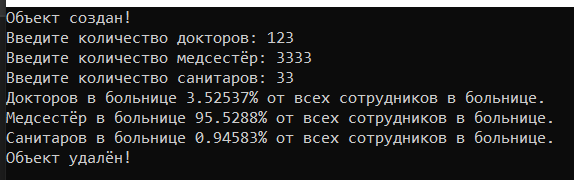


Рисунок 2 – работа программы 2

**Контрольные вопросы**

1. Что является основным элементом ООП? Что входит в его состав?

Основной элемент ООП – класс. У него есть поля, методы, конструкторы, деструкторы, модификаторы доступа и многое другое.

1. Перечислите виды классов и укажите их различия.

Простой класс, абстрактный, интерфейс, базовый, наследник.

1. Укажите способы защиты элементов, входящих в состав класса.

Модификаторы доступа, модификаторы класса, модификаторы поведения.

1. Что такое операция привязки? Приведите пример.

Процесс, который определяет какой именно метод будет вызван при обращении к объекту.

1. Дайте понятие полного квалификационного имени.

Полное квалифика- ционное имя X::а означает, что элемент **а** принадлежит классу **X**. Запись X::f1() означает, что метод **f1** принадлежит классу **X**.