# 1. Лабораторная работа № 1. Изучение классов языка Си++

## 1.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в изучении основных понятий объектно- ориентированного программирования языка Си++ – классов и объектов, и овладении навыками разработки программ на языке Си++ с использованием объектно- ориентированных средств. Для достижения цели необходимо выполнить следующие задачи:

- изучить необходимые учебные материалы, посвященные основам объектно-ориентированного программирования на языке Си++;

- разработать программу на языке Си++ для решения заданного варианта задания;

- отладить программу;

- выполнить решение контрольного примера с помощью программы и ручной расчет контрольного примера;

- подготовить отчет по лабораторной работе.

## 1.2. Краткая характеристика объекта изучения

### 1.2.1. Понятие класса и объекта

Класс в языке Си++ – это  новый тип, определяемый программистом, включающий данные (поля класса) и методы (функции) для обработки этих данных. Переменные этого типа называются объектами.

Формат объявления класса:

*<Ключевое слово> <Имя_класса>*

*{*

*<список компонент>*

*};*

В качестве ключевого слова используется одно из трех ключевых слов:

- *struct*

- *class*

- *union*

Можно дать такое определение класса через структуру, которая была в языке Си. Класс – это структура, в которую введены методы для обработки полей.

Объекты – это переменные типа класса. Формат определения объектов:

*<Имя_класса> <Имя_объекта1>,…<Имя_объекта_N>;*

Обращение к полям и методам класса внутри методов класса просто по имени, а за пределами класса через имя объекта и операцию «.» или через имя указателя на объект и операцию «->». Каждый объект класса имеет в оперативной памяти свои копии полей класса.

Пример работы с полями:

```
struct A { int i; void print() { printf("i=%d", i); }    };
A a1;  A *pA=&a1;
a1.i=10; a1.print();       pA->i=10; pA->print();
a1.A::i=10; a1.A::print();       pA->A::i=10; pA->A::print();
```

### 1.2.2. Доступность компонент класса

Свойство доступности определяет возможность доступа к полям и методам за пределами класса (через имя объекта или через указатель на объект).

Существуют три статуса доступа:

- *public* (полностью доступны за пределами класса);
- *private* (не доступны за пределами класса, можно обращаться к компонентам только в методах своего класса);
- *protected* (доступны только в своем классе и в производных классах).

По умолчанию, если класс определен с ключевым словом *struct*, то все компоненты имеют статус доступа *public*. Если с ключевым словом *union*, тоже *public*, но все поля каждого объекта располагаются в памяти, начиная с одного адреса. Если класс определен с ключевым словом *class*, то все поля и методы по умолчанию имеют статус доступа *private*.

Статус доступа можно изменить с помощью соответствующих модификаторов, что продемонстрировано в следующем примере:

```
struct A
{
   …… // Статус доступа public
private:
…….. // Статус доступа private
```

```
protected:
……..// Статус доступа protected
};
class B
{
   …… // Статус доступа private
public:
…….. // Статус доступа public
protected:
……..// Статус доступа protected
};
```

### 1.2.3. Основные элементы класса

#### *Компонентные данные и функции класса*

Компонентные данные (поля класса) и функции класса (методы класса) уже во многом рассмотрены выше. Следует добавить, что методы класса могут быть определены внутри класса, в этом случае они по возможности (если нет ограничений) являются подставляемыми, но чаще всего класс содержит описание (заголовки) методов, а определения методов находятся за пределами класса. Эта возможность является удобной в проектах, состоящих из многих файлов. Создается отдельный файл с расширением .h (заголовочный файл), в котором находится описание класса вместе с полями и заголовками методов. Определения методов находится в файле реализации класса (файле с расширением .cpp). В этом случае, чтобы использовать класс в другом файле с исходным кодом, необходимо подключить заголовочный файл с описанием класса. Также внутри методов можно использовать умалчиваемые значения параметров (требования такие же, как к обычным функциям). В классах возможна перегрузка методов.

В следующем примере показаны перечисленные возможности:

```
struct Complex
{
        double real,     image; // Поля класса
        void define(double re=0.0, double im=0.0) // Определение метода
                                                  // внутри класса
        {
                real=re; image=im;// Обращение к полям внутри метода
```

```
    }
    void print(); // Описание метода
    };
void Complex::print() // Определение метода за пределами класса
{
        printf("\nreal=%f  image=%f", real, image);
}
```

### *Конструктор класса*

Конструктор класса – специальный блок операторов (инструкций), вызываемый при создании объекта. Назначение: присвоение начальных значений полям, выделение памяти, открытие файлов, сетевых соединений и т.п. Имя конструктора совпадает с именем класса, конструктор не имеет возвращаемого значения. Возможна перегрузка конструкторов. Конструктор может определяться как внутри класса, так и за пределами.

Формат определения конструктора внутри класса:

*Имя_класса(Список_формальных_параметров)*

*{ Операторы_тела конструктора }*

По умолчанию класс всегда имеет конструктор копирования вида *A(A& a) {… }* (*A* – имя класса), создающий копию объекта (происходит копирование полей), и если нет явного конструктора, то по умолчанию создается конструктор без параметров. Эти конструкторы можно переопределять.

Примеры вызовов конструкторов:

*A a1;  A* pA=new A; // Вызываются конструкторы без параметров*

*A a2(3, 4); A * pA2=new A(3, 4);*

// Вызываются конструкторы с 2-мя параметрами

Для конструктора с одним параметром можно использовать форму:

*A a1=5; A a2=a1;* вместо *A a1(5); A a2(a1);*

### *Деструктор класса*

Деструктор – специальный блок операторов (инструкций), служащий для деинициализации объекта (освобождение памяти, закрытие файлов и т.п.).

Вызывается автоматически при удаление объекта, например, оператором *delete* или при выходе из блока, в котором существует объект. Не имеет возвращаемого значения и параметров. Может определяться как внутри класса, так и за пределами. Пример деструктора:

*~имя_класса()*

*{*

*тело_деструктора*

*}*

### 1.3. Задачи и порядок выполнения работы

В работе необходимо разобраться с понятием класса некоторой предметной области и соответствующему ему класса как типа языка Си++, введенного пользователем. Также необходимо знать отличие классов от объектов и назначение и порядок использования основных элементов класса в Си++: полей, методов (функций класса), конструкторов, деструктора. Обратить внимание на способы создания массива объектов класса динамически.

Студент разрабатывает программу на языке Си++ в виде консольного приложения, в программе необходимо создать **динамический массив и последовательной контейнер STL (vector, deque или list) объектов некоторого класса**, данные об объектах читаются из текстового файла. Результаты работы программы и результаты ручного расчета (при необходимости) представляет преподавателю в отчете.

Условие задачи:

Класс автомобиль. Содержит поля (статус доступа private):

- название (марка);
- средний расход топлива (в литрах на 100 км).

Методы и конструкторы:

- конструктор для инициализации полей;
- метод для расчета требуемого объема топлива для заданного пробега (метод имеет 1 параметр, который задает требуемый пробег в километрах, возвращает требуемый объем топлива в литрах);
- метод для печати параметров объекта.

При необходимости можно использовать другие методы и конструкторы.

Требуется: создать массив и vector объектов класса, рассчитать требуемый суммарный объем топлива для заданного пробега всех автомобилей, пробег вводится с клавиатуры, вывести на печать параметры объектов и рассчитанный объем топлива.

### *Пример выполнения работы*

Следует отметить, что для создания массива объектов динамически, когда существует конструктор с параметрами для инициализации полей объекта, можно использовать два основных способа.

Особенности первого способа:

- необходимо в класс включить конструктор без параметров и отдельный метод для инициализации полей, например, с именем *set* и числом параметров, которые соответствуют числу инициализируемых полей класса;

- объявляем указатель на массив объектов, если имя класса *MyClass*, то это объявление может иметь вид: *MyClass *pOb;*

- задаем число объектов, например, вводим с клавиатуры или читаем из файла как значение целой переменой *n*, и создаем массив объектов динамически *pOb=new MyClass[n];* здесь, при создании каждого объекта, вызывается конструктор без параметров, именно для этого его и нужно включить в класс;

- в цикле для каждого объекта вводим параметры или читаем из файла для его инициализации и инициализируем уже созданный объект с помощью вызова метода *set*, это действие может иметь вид *pOb[i].set(...);* где в метод *set* вместо многоточия передаются фактические параметры.

Особенности второго способа (здесь не нужен конструктор без параметров и метод *set*):

- объявляем указатель на указатель на массив объектов, если имя класса *MyClass*, то это объявление может иметь вид: *MyClass **ppOb;*

- задаем число объектов, например, вводим с клавиатуры или читаем из файла как значение целой переменой *n*, и создаем массив указателей динамически *ppOb=new MyClass*[n];*

- в цикле для каждого объекта вводим параметры или читаем их из файла для его инициализации и создаем объект динамически с помощью вызова конструктора с параметрами, это действие может иметь вид *ppOb[i]=new MyClass(...);* где вместо многоточия передаются фактические параметры для конструктора класса.

При использовании последовательного контейнера (vector, deque или list) объект можно добавлять в конец контейнера с помощью метода push_back, или сразу создавать контейнер из n объектов и далее инициализировать каждый элемент с помощью метода set.

Ниже в листинге показаны оба способа, один из способов представлен в виде комментариев. **В листинге данные вводятся с клавиатуры, в работе их надо читать из файла.**

Листинг программы с комментариями:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
class Avt // Класс автомобиль
{
	string marka; // Марка
	double rash; // Расход топлива на 100 км
public:
	Avt()  // Конструктор без параметров создает "пустой" объект
	{
		rash = 0;
		marka = "";
	}
	void set(string mar, double r) // Функция для инициализации полей для
		// созданного "пустого" объекта
	{
		marka= mar; // Копируем строку, содержащую марку автомобиля
		rash = r; // Задаем значение расхода топлива на 100 км
	}
	Avt(string mar, double r) // Конструктор для инициализации полей
	{
		marka=mar; // Копируем строку, содержащую марку автомобиля
		rash = r; // Задаем значение расхода топлива на 100 км
	}
	double getRash(double dlina) const // Функция возвращает - сколько нужно топлива
для
		// пробега заданного расстояния
	{
		return rash * dlina / 100.;
	}
	void print() const // Функция для печати полей объекта
	{
		cout << "\nmarka: " << marka << "  rashod na 100 km=" << rash;
	}
};


int main()
{
	int n; // Неизвестное число объектов
	cout << "n="; cin >> n;  // Ввод с клавиатуры n
	double rast; // Расстояние, для которого требуется вычислить расход топлива
	cout << "rast="; cin >> rast; // Ввод с клавиатуры расстояния
	double SumRashod = 0; // Суммарный расход топлива для всех автомобилей
	string str; double r; // Вспомогательные переменные для ввода марки
	// автомобиля и расхода топлива

	/*************************************************************************
	Первый способ создаем массив "пустых" объектов и инициализируем их с
	помощью функции set
	*************************************************************************/

	Avt *pAvt; // Указатель на массив
	pAvt=new Avt[n]; // Для каждого объекта вызывается конструктор без параметров,
	// т.е. созданы "пустые" объекты
	vector<Avt> vecAvt(n); // Создаем пустые объекты, вызывается конструктор без
параметров
	// Цикл ввода данных для объектов
	for(int i=0; i<n; i++)
	{
	cout<<"Object N="<<(i+1)<<":\n"<<"marka: ";
	cin.ignore();
```

```
        getline(cin, str); // Ввод марки автомобиля
        cout<<"Rashod="; cin>>r;  // Ввод расхода топлива
        pAvt[i].set(str, r); // Вызываем функцию set для инициализации полей
        vecAvt[i].set(str, r); // Инициализация для vector
        // объектов
        }
        // Цикл печати полей для объектов
        for(int i=0; i<n; i++) pAvt[i].print();
        // Цикл для расчета суммарного расхода топлива
        for(int i=0; i<n; i++) SumRashod+=pAvt[i].getRash(rast);
        cout << "\nSumRashor=" << SumRashod;  // Вывод на печать суммарного расхода
топлива
        // Тоже делаем для vector
        SumRashod = 0;
        // Цикл печати полей для объектов
        for (const auto pos : vecAvt) pos.print();
        // Цикл для расчета суммарного расхода топлива
        for (const auto pos : vecAvt) SumRashod += pos.getRash(rast);
        cout << "\nSumRashor=" << SumRashod;  // Вывод на печать суммарного расхода
топлива
        /*************************************************************************
        Конец первого способа
        *************************************************************************/

        /*************************************************************************
        Второй способ создаем массив указателей на объекты и
        далее каждый объект создается динамически с помощью конструктора с инициализацией
        *************************************************************************/

        /*Avt **ppA; // Указатель на массив указателей
        ppA = new Avt*[n]; // Создаем массив указателей
        vector<Avt> vecAvt; // Параллельно создаем вектор автомобилей
        for (int i = 0; i < n; i++)
        {
                cout << "Object N=" << (i + 1) << ":\n" << "marka: ";
                cin.ignore();
                getline(cin, str); // Ввод марки автомобиля
                cout << "Rashod="; cin >> r; // Ввод расхода топлива
                ppA[i] = new Avt(str, r); // Создание объекта динамически
                // с вызовом конструктора с параметрами
                vecAvt.push_back(Avt(str, r)); // Добавляем объект в контейнер
        }
        // Цикл печати полей для объектов
        for (int i = 0; i < n; i++) ppA[i]->print();
        // Цикл для расчета суммарного расхода топлива
        for (int i = 0; i < n; i++) SumRashod += ppA[i]->getRash(rast);
        cout << "\nSumRashor=" << SumRashod;  // Вывод на печать суммарного расхода
топлива
        // Тоже делаем для vector
        SumRashod = 0;
        // Цикл печати полей для объектов
        for (const auto pos : vecAvt) pos.print();
        // Цикл для расчета суммарного расхода топлива
        for (const auto pos : vecAvt) SumRashod += pos.getRash(rast);
        cout << "\nSumRashor=" << SumRashod;  // Вывод на печать суммарного расхода
топлива
        /*************************************************************************
        Конец второго способа
        *************************************************************************/
        return 0;
}
```

## 1.4. Форма отчета по лабораторной работе

Отчет должен содержать: титульный лист, цель работы, условие задачи, текст программы с комментариями, ручной расчет контрольного примера и результаты решения контрольного примера программой для проверки правильности работы алгоритма, выводы по работе.

*При выполнении работы все входные данные читаются из текстового файла input.txt (создать этот файл любым текстовым редактором), результаты выводятся на консоль и в файл output.txt, для этого использовать одну функцию для вывода. В отчете представить содержимое этих файлов. Для задания строк можно использовать std::string.*

## 1.5. Вопросы для самоконтроля

1. Классы и объекты.
2. Статусы доступа полей и методов.
3. Назначение и перегрузка конструкторов.
4. Деструктор класса.

**Варианты заданий для лабораторной работы № 1**

Во всех вариантах требуется следующее:

Описать класс, включающий заданные поля и методы (функции). Разработать программу (или две программы, выбрать по желанию), которая **создает динамический массив объектов и последовательный контейнер STL (vector, deque или list выбрать самостоятельно) объектов** и выполняет требуемые действия. **Все исходные данные для работы программы читаются из текстового файла, созданного в простом редакторе типа «Блокнот».** Выходные данные выводятся **на консоль и в текстовый файл, для этого использовать одну функцию для вывода**.

Вариант 1

Класс – сотрудник предприятия. Параметры (поля класса) – ФИО, оклад, надбавка к окладу в %. Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию печати параметров, функцию вычисления зарплаты (зарплата = оклад + процентная надбавка от оклада). Вывести на печать параметры всех сотрудников и суммарную их зарплату.

Вариант 2

Класс – студент. Параметры (поля класса) – ФИО, массив из m-х оценок за последнюю сессию. Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию печати параметров, функцию проверки возможности получения студентом стипендии (все оценки без троек). Вывести на печать всех студентов, получающих стипендию.

Вариант 3

Класс – банковский вклад. Параметры (поля класса): ФИО владельца, текущая сумма, годовая процентная ставка (проценты начисляются ежегодно с капитализацией начисленных процентов с основной суммой). Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию печати параметров, функцию расчета суммы на счету через заданное число лет (число лет – параметр функции). Вывести на печать параметры всех вкладов и суммарную сумму на счетах через заданное число лет, которое вводится с клавиатуры.

Вариант 4

Класс – аппаратно- программное средство защиты (СЗ) от несанкционированного доступа (НСД). Параметры (поля класса) – название и номер класса защищенности от НСД (Существует семь классов защищенности от НСД, наивысший 1-ый, самый низкий 7, например, если требуется обеспечить защищенность по 3- му классу, то можно использовать СЗ с классами 1, 2 или 3). Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию, проверяющую можно ли это СЗ использовать для заданного класса (номер заданного класса – параметр функции), функцию печати параметров СЗ. Вывести на печать параметры тех СЗ, которые можно использовать для заданного класса защищенности, номер класса защищенности вводится с клавиатуры.

Вариант 5

Класс – криптографический метод защиты информации. Параметры (поля класса) – название и тип (симметричный или несимметричный). Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию печати параметров. Вывести на печать все методы заданного типа, тип вводится с клавиатуры.

Вариант 6

Класс – прямоугольник. Параметры (поля класса) – длина, ширина. Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию печати параметров, функцию вычисления площади. Вывести на печать все параметры прямоугольников, площади которых превышают заданное значение, которое вводится с клавиатуры.

Вариант 7

Класс – автомобиль. Параметры (поля класса) – марка, максимальная скорость (км/ч). Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию печати параметров. Вывести на печать параметры тех автомобилей, максимальная скорость которых превышает заданное значение, введенное с клавиатуры.

Вариант 8

Класс – квадратное уравнение $ax^2 + bx + c = 0$ ($a \neq 0$). Параметры (поля класса) – $a$, $b$, $c$. Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию печати параметров, функцию расчета дискриминанта. Вывести на печать параметры тех уравнений, которые имеют вещественные корни.

Вариант 9

Класс – полином $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ ($a_n \neq 0$). Параметры (поля класса): n и массив коэффициентов $a_n, a_{n-1}, \ldots, a_0$. Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию печати параметров, функцию вычисления значения полинома при заданном x (x- параметр функции). Вывести на печать параметры всех полиномов и сумму их значений при заданном значении x, вводимом с клавиатуры.

Вариант 10

Класс - книга. Параметры (поля): автор, название, количество страниц. Статус доступа всех полей private. Класс включает: конструктор, функцию печать параметров, при необходимости функции доступа к полям. Распечатать параметры книги с максимальным количеством страниц.

Вариант 11

Класс – межсетевой экран (МЭ). Параметры (поля класса) – название и номер класс защищенности (Существует пять классов защищенности МЭ, наивысший 1-ый, самый низкий 5, например, если требуется использовать МЭ 3- го класса защищенности, то можно использовать МЭ с классами 1, 2 или 3). Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию, проверяющую можно ли МЭ использовать для заданного класса (номер заданного класса – параметр функции), функцию печати параметров МЭ. Вывести на печать параметры тех МЭ, которые можно использовать для заданного класса защищенности, номер класса защищенности вводится с клавиатуры.

Вариант 12

Класс – персональный компьютер (ПК). Параметры (поля класса): название процессора, тактовая частота. Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию печати параметров.

Вывести на печать параметры всех компьютеров в порядке невозрастания тактовой частоты.

Вариант 13

Класс – программа- антивирус. Параметры (поля класса): название, число вредоносных программ в базе. Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию печати параметров. Вывести на печать параметры антивирусов и антивирус с самой большой базой вредоносных программ.

Вариант 14

Класс - вещественное число, записанное с точкой. Параметры - его значение (вещественный тип) и запись (строковое представление числа в десятичной системе счисления с точкой). Статус доступа всех полей private. Класс включает: конструктор, при необходимости функции доступа к полям, функцию, определяющую количество цифр в целой части числа в десятичной записи, функцию печати параметров. Напечатать все числа, сумму введенных чисел и суммарное количество цифр в целых частях всех чисел.

Вариант 15

Класс - предложение. Параметры: массив слов (n<10) и их количество. Статус доступа всех полей private. Класс включает: конструктор и функцию, определяющую количество слов, длиннее 5 букв, при необходимости функции доступа к полям, функцию печати параметров. Напечатать параметры предложений и процент слов длиннее 5 букв в заданном тексте.

Вариант 16

Класс - выражение, состоящее из целых чисел и знаков операций (скобок нет).

Параметры: массив значений чисел (n<10), количество чисел и массив знаков операций (тип char). Статус доступа всех полей private. Класс включает: конструктор и функцию, вычисляющую результат (приоритеты операций не учитывать, считать приоритет операций одинаковым), при необходимости функции доступа к полям, функцию печати параметров. Ввести несколько выражений и вывести результаты в порядке, обратном вводу.

Вариант 17

Класс - некоторый товар в магазине. Параметры: наименование, количество и стоимость. Статус доступа всех полей private. Класс включает: конструктор и функцию, определяющую суммарную стоимость товара, при необходимости функции доступа к полям, функцию печати параметров. Напечатать параметры всех товаров и суммарную стоимость всех товаров в магазине.

Вариант 18

Класс - некоторый товар в магазине. Параметры: наименование, количество и закупочная цена. Статус доступа всех полей private. Класс включает: конструктор и функцию, определяющую стоимость товара исходя из заданного процента прибыли (процент прибыли – параметр функции), при необходимости функции доступа к полям, функцию печати параметров. Напечатать параметры всех товаров и суммарную стоимость всех товаров в магазине с учетом заданного процента прибыли, который вводится с клавиатуры.

Вариант 19

Класс - студент. Параметры (поля): ФИО, массив экзаменационных оценок ($m$=4). Статус доступа всех полей private. Класс включает: конструктор и функцию определения среднего балла, при необходимости функции доступа к полям, функцию печати параметров. Напечатать параметры всех студентов и 3-х самых сильных студентов группы.

Вариант 20

Класс - ангар. Параметры (поля): ширина и длина. Статус доступа всех полей private. Класс включает: конструктор и функцию, определяющую площадь помещения, при необходимости функции доступа к полям, функцию печати параметров. Напечатать параметры ангаров и площадь склада, состоящего из этих ангаров.

Вариант 21

Класс - квартира. Параметры (поля): общая площадь и стоимость одного квадратного метра. Статус доступа всех полей private. Класс включает: конструктор и функцию, определяющую стоимость квартиры, при необходимости функции доступа к полям, функцию печати параметров. Напечатать параметры всех квартир и все квартиры, стоимость которых не превышает заданной суммы (сумма вводится с клавиатуры).

Вариант 22

Класс - квартира. Параметры (поля): стоимость и количество комнат. Статус доступа всех полей private. Класс включает: конструктор и функцию, определяющую среднюю стоимость одной комнаты, при необходимости функции доступа к полям, функцию печати параметров. Напечатать параметры всех квартир и все квартиры, стоимость одной комнаты в которых не превышает заданной суммы (сумма вводится с клавиатуры).

Вариант 23

Класс - выставочные экспонаты. Параметры (поля): название, время экспонирования (в днях), стоимость одного дня экспонирования. Статус доступа всех полей private. Класс включает: конструктор и функцию определения стоимости экспонирования, при необходимости функции доступа к полям, функцию печати параметров. Вывести на печать параметры экспонатов и экспонат, стоимость экспонирования которого максимальна.

Вариант 24

Класс - книга. Параметры (поля): автор, название, количество экземпляров и количество желающих ее прочитать читателей. Статус доступа всех полей private. Класс включает: конструктор и функцию определения средней длины очереди на чтение каждого экземпляра, при необходимости функции доступа к полям, функцию печати параметров. Напечатать параметры книг и наиболее читаемую книгу в библиотеке.

Вариант 25

Класс - выражение, состоящее из целых чисел и знаков операций (скобок нет).

Параметры (поля): строка, содержащая выражение. Статус доступа всех полей private. Класс включает: конструктор и функцию, определяющую количество операций, при необходимости функции доступа к полям. Ввести несколько выражений и определить суммарное количество операций в них.

Вариант 26

Класс – вектор на плоскости. Параметры (поля): координаты конца вектора: x, y (начало вектора в точке с координатами 0, 0), Статус доступа всех полей private. Класс включает: конструктор, функцию печати параметров, при необходимости функции

доступа к полям, функцию вычисления длины вектора. Вывести на печать все вектора и вектор с наибольшей длиной.

Вариант 27

Класс - скаковая лошадь. Параметры: кличка и массив рекордов, содержащий 5 лучших результатов, показанных лошадью на скачках (результат определяется временем). Статус доступа всех полей private. Класс включает: конструктор и функцию, определяющую среднее время, показанное лошадью, при необходимости функции доступа к полям, функцию печати параметров. Вывести на печать параметры лошадей и среднее время по всей конюшне.

# 2. Лабораторная работа № 2. Изучение перегрузки стандартных операций в языке Си++

## 2.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си++, использующих перегрузку стандартных операций. Для достижения цели необходимо выполнить следующие задачи:

- изучить необходимые учебные материалы, посвященные перегрузке стандартных операций в языке Си++ ;
- разработать программу на языке Си++ для решения заданного варианта задания;
- отладить программы;
- выполнить решение контрольного примера с помощью программы и ручной расчет контрольного примера;
- подготовить отчет по лабораторной работе.

## 2.2. Краткая характеристика объекта изучения

Перегрузка операций в языке Си++ это возможность распространения действия стандартных операций на операнды, для которых эти операции первоначально не предназначались. Это возможно, если хотя бы один из операндов является объектом класса, для этого создается специальная, так называемая, оператор- функция, которая может быть как членом класса, так и функцией, не принадлежащей классу.

Формат определения оператор- функции имеет вид:

*<тип_возвращаемого_значения>*
*operator <знак_операции>*
*(спецификация_параметров)*
*{*
*операторы_тела_функции*
*}*

Существует три способа перегрузки:

- оператор-функция определяется как функция, не принадлежащая классу;
- оператор-функция определяется как функция класса;

- оператор-функция определяется как дружественная функция класса.

Особенности перегрузки операций:

- можно перегружать только стандартные операции, например, нельзя перегрузить операцию ‘**’ (возведение в степень в языке Фортран, отсутствует в Си++);
- не допускают перегрузки операции: *‘.’, ‘.*’, ‘?:’, ‘::’, ‘sizeof’, ‘#’, ‘##’*;
- при перегрузке сохраняется арность операций (унарная операция остается унарной, а бинарная – бинарной);
- бинарная операция перегружается либо как функция, не принадлежащая классу с двумя параметрами, один обязательно объект (ссылка на объект) класса, или как функция класса с одним параметром, первым операндом операции выступает объект класса, для которого вызывается функция;
- бинарные операции *‘=‘, ‘[]’, ‘->’* должны обязательно определяться как компонентные функции класса;
- унарная операция перегружается либо как функция, не принадлежащая классу с одним параметром - объектом (ссылкой на объект) класса, или как функция класса без параметров, операндом операции выступает объект класса, для которого вызывается функция.

### 2.3. Задачи и порядок выполнения работы

Студент в работе создает класс и необходимыV1.print();

```
V2.print();
```

е оператор- функции для перегрузки заданных в своем варианте операций. Особое внимание обратить на способы перегрузки унарных и бинарных операций и параметры оператор- функций для этих операций, когда оператор функция является членом класса и когда не является. Знать те случаи, когда оператор функция должна быть обязательно членом класса и случаи, когда оператор- функция обязательно не принадлежит классу. При защите работы необходимо обосновать выбор способа определения оператор-функции – внутри класса или вне его. При необходимости студент выполняет ручной расчет для проверки работы программы для задачи небольшой размерности. Результаты работы программы, ручного расчета представляются в отчете.

Условие задачи:

Дан класс (например, с именем *Vector*), задающий вектор размерности *n*. Поля класса: указатель на массив, задающий вектор (тип элемента *double*), массив должен

создаваться динамически; число элементов (размерность) вектора (тип *int*). Класс включает: конструктор без параметров, задающий пустой вектор (число элементов равно 0); конструктор, создающий объект вектор на основе обычного одномерного массива размерности *n*; конструктор копирования, конструктор перемещения, деструктор.

Необходимо перегрузить операции и продемонстрировать их работу. Перегрузить операцию *[]* (обращение к элементу вектора по индексу), операцию = (копирование вектора или создание копии вектора), операцию * (умножение числа на вектора), на выходе вектор такой же размерности, каждый элемент которого равен произведению соответствующего элемента исходного вектора на число.

***Пример выполнения работы***

Исходный код консольного представлен ниже.

Листинг программы с комментариями:

```
#include <iostream>
using namespace std;
class Vector // Класс- вектор
{
    double *p=nullptr; // Указатель на массив (вектор)
    int n=0; // Размерность вектора (число элементов) массива
public:
    Vector(double *p, int n) // Коструктор на входе массив, задающий вектор
    {
        this->n = n; // Задаем число элементов
        this->p = new double[n]; // Выделяем память
        for (int i = 0; i < n; i++) this->p[i] = p[i]; // Копируем один
массив в другой
        cout<<"Vector(double *p, int n)"<<endl;
    }
    Vector(int n): n(n) // Конструктор - выделяем память без инициализации
    {
        p=new double[n];
        cout<<"Vector(int n)"<<endl;
    }
    Vector(const Vector & V)
    { // Конструктор копирования
        n = V.n;
        p = new double[n];
        for (int i = 0; i < n; i++)
            p[i] = V.p[i];
        cout<<"Vector(const Vector & V) n="<<n<<endl;
    }
    // Конструктор перемещения (подробная форма)
  /*  Vector(Vector&& V)  // Параметр - правосторонняя ссылка
        : p(V.p), n(V.n)   // Можно инициализировать так
    {
        //p = M.p;  // Или инициализировать так
        //n = M.n;
        // Присвойте данным-членам исходного объекта значения по умолчанию.
Это не позволяет деструктору освобождать память
        V.p = nullptr;
        V.n = 0;
    }*/
    // Конструктор перемещения (краткая форма)
    Vector(Vector&& V)  // Параметр - правосторонняя ссылка
```

```
    {
        std::swap(p, V.p);
        std::swap(n, V.n);
         cout<<"Vector(Vector&& V)"<<endl;
    }
    void print() const // Печать вектора (массива), заменить на перегрузку <<
    {
        cout<<"n="<<n<<endl;
        for (int i = 0; i < n; i++)
            std::cout << p[i] << " ";
        std::cout << std::endl;
    }
    Vector() { p = nullptr; n = 0;
     cout<<"Vector()"<<endl;
     } // Конструктор без параметров, задает "пустой" объект
    double& operator[](int index) // Оператор- функция (перегрузка операции
        // обращения к элементу)
    {
        return p[index];
    }
    Vector & operator =(const Vector& v2) // Оператор- функция копирования
объекта
    {
        if (this!=&v2) // Запрет копирования вектора самого в себя
        {
            n = v2.n;
            if (p != nullptr) delete[] p; // Освобождаем память старого
вектора
            p = new double[n]; // Выделяем память для нового вектора
            for (int i = 0; i < n; i++) p[i] = v2.p[i];
        }
        cout<<"Vector & operator =(const Vector& v2)"<<endl;
        return *this; // Возвращаем ссылку на текущий объект
    }
    Vector & operator =(Vector&& v2) // Оператор- функция перемещения объекта
    {
        if (this!=&v2) // Запрет перемещения вектора самого в себя
        {
            /* Подробная форма
            n = v2.n;
            if (p != nullptr) delete[] p; // Освобождаем память старого
вектора
            p = v2.p; // Перемещаем данные
            v2.p=nullptr; // Перемещаемый вектор пуст
            */
            // Краткая форма
            std::swap(p, v2.p);
            std::swap(n, v2.n);
        }
        cout<<"Vector & operator =(Vector&& v2)"<<endl;
        return *this; // Возвращаем ссылку на текущий объект
    }
    ~Vector() // Деструктор
    {
        cout<<"~Vector() n="<<n <<endl;
        if (p!=nullptr) delete[] p; // Освобождаем память
    }
    friend Vector operator *(double x, Vector& v2); // Дружественная функция,
    friend Vector operator +(const Vector& v1, const Vector& v2);
    // определенная вне класса
};
// Умножение числа на вектор (первый операнд не объект класса,
// функция обязательно определяется вне класса)
Vector operator *(double x, Vector& v2) // Оператор- функция вне класса
```

```
{
    Vector V(v2.n); // Создаем новый объекта заданной размерности
    for (int i = 0; i < v2.n; i++) V.p[i] = x * v2.p[i]; // Заполняем массив
    return V; // Возвращаем объект
}

Vector operator +(const Vector& v1, const Vector& v2) // Оператор- функция
вне класса
{
    Vector V(v1.n+v2.n); // Создаем новый объект заданного размера
    for (int i = 0; i < v1.n; i++) V.p[i] = v1.p[i]; // Заполняем массив
    for (int i = 0; i < v2.n; i++) V.p[i+v1.n] = v2.p[i]; // Заполняем массив
    return V; // Возвращаем объект
}

int main()
{
    double m1[] = { 1, 2, 3, 4.5, 7 };
    Vector V1(m1, 5); // Создаем объект
    Vector V2(m1, 5); // Создаем объект
    Vector V3=V1+V2;
    V3.print();
    return 0;
}
```

### 2.4. Форма отчета по лабораторной работе

Отчет должен содержать: титульный лист, цель работы, условие задачи, текст программы с комментариями, при необходимости ручной расчет контрольного примера и результаты решения контрольного примера программой для проверки правильности работы алгоритма, выводы по работе.

### 2.5. Вопросы для самоконтроля

1. Способы перегрузки операций в Си++.
2. Перегрузка бинарной операции, когда первый операнд не является объектом класса.
3. Перегрузка операции с помощью дружественной функции класса.
4. Операции, для перегрузки которых оператор- функция обязательно должна принадлежать классу.

**Варианты заданий для лабораторной работы № 2**

Дан класс (например, с именем *Vector*), задающий вектор размерности *n*. Поля класса: указатель на массив, задающий вектор (тип элемента *int* или *double* в зависимости от варианта), массив должен создаваться динамически, число элементов (размерность) вектора (тип *int*). Класс включает: конструктор без параметров, задающий пустой вектор (число элементов равно 0), конструктор, создающий объект вектор на основе обычного одномерного массива размерности *n*, конструктор копирования, конструктор перемещения, деструктор.

Необходимо перегрузить операции и продемонстрировать их работу. Перегрузить операцию [] (обращение к элементу вектора по индексу), операцию = (присваивание с копированием), операцию = (присваивание с перемещением), а также операцию вставки (<<) объекта в поток cout или в файл (объект класса ostream) и операцию извлечения (>>) объекта из потока cin или из файла (объект класса istream). *Также продемонстрировать разницу между конструктором копирования и конструктором перемещения и между операциями присваивания с копированием и перемещением. Исходные коды класса разместить в двух файлах: в заголовочном файле класса и файле реализации класса.*

*При выполнении работы все входные данные читаются из текстового файла input.txt (создать этот файл любым текстовым редактором), результаты выводятся в файл output.txt. В отчете представить содержимое этих файлов.*

Индивидуальные операции для перегрузки каждым студентом представлены в таблице 1.

Таблица 1 – Варианты заданий

| Описание операции перегруженной операции | Тип элемента вектора (массива) | Типы операндов и результата для перегруженной операции | | | № варианта |
|---|---|---|---|---|---|
| | | Первый операнд | Второй операнд | Результат | |
| + сложение векторов одинаковой размерности, на выходе вектор такой же размерности элемент которого равен сумме соответствующих элементов двух векторов | double | Vector | Vector | Vector | 1 |
| | | Vector | double * | Vector | 2 |
| | | double * | Vector | Vector | 3 |
| + сложение векторов, на выходе вектор, длина которого сумме длин векторов вначале идут элементы первого вектора, затем второго, если один из векторов задан обычным массивом, то считать, что его длина равна длине вектора, заданным объектом класса | double | Vector | Vector | Vector | 4 |
| | | Vector | double * | Vector | 5 |
| | | double * | Vector | Vector | 6 |
| Постфиксная форма – (декремент), при этом каждый элемент исходного вектора уменьшается на 1, результат операции «старый» (неизмененный) вектор. | int | Vector | _____ | Vector | 7 |
| * умножение вектора на число, на выходе вектор такой же размерности, каждый элемент которого равен произведению соответствующего элемента исходного вектора на число | double | Vector | double | Vector | 8 |
| | | double | Vector | Vector | 9 |
| Префиксная форма ++ (инкремент), при этом каждый элемент исходного вектора увеличивается на 1, результат операции - измененный вектор. | int | Vector | _____ | Vector | 10 |
| * скалярное произведение векторов (одинаковой размерности), на выходе значение этого произведения | double | Vector | Vector | double | 11 |
| | | Vector | double * | double | 12 |
| | | double * | Vector | double | 13 |
| Постфиксная форма ++ (инкремент), при этом каждый элемент исходного вектора увеличивается на 1, результат операции «старый» (неизмененный) вектор. | int | Vector | _____ | Vector | 14 |
| ^ побитовая операция исключающая ИЛИ с двумя векторами одинаковой размерности, на выходе вектор такой же размерности элемент, которого равен битовой операции ^ соответствующих элементов двух векторов | int | Vector | Vector | Vector | 15 |
| | | Vector | int * | Vector | 16 |
| | | int * | Vector | Vector | 17 |
| Префиксная форма -- (декремент),, при этом | int | Vector | _____ | Vector | 18 |

| | | | | | |
|---|---|---|---|---|---|
| каждый элемент исходного вектора уменьшается на 1, результат операции измененный вектор. | | | | | |
| ^ логическая операция (исключающая ИЛИ) с двумя векторами одинаковой размерности, на выходе вектор такой же размерности элемент, которого равен логическая операции ^ соответствующих элементов двух векторов | bool | Vector | Vector | Vector | 19 |
| | | Vector | bool * | Vector | 20 |
| | | bool * | Vector | Vector | 21 |
| < сравнение двух векторов (массивов) сравнение проводится последовательно по элементам. Результат истинно, если элемент первого операнда меньше элемента второго элемента, если элемент первого операнда больше элемента второго элемента, результат ложь, в случае равенства переход к следующий паре элементов. Если в каком либо операнде элементы закончились, то результат ложь. Если один из векторов задан обычным массивом, то считать, что его длина равна длине вектора, заданным объектом класса. | int | Vector | Vector | bool | 22 |
| | | Vector | int * | bool | 23 |
| | | int * | Vector | bool | 24 |

# 3. Лабораторная работа № 3. Изучение возможностей наследования классов

## 3.1. Цель и задачи работы, требования к результатам ее выполнения

Цель работы состоит в овладении навыками разработки программ на языке Си++, использующих возможности наследования классов для решения различных задач. Для достижения цели необходимо выполнить следующие задачи:

- изучить необходимые учебные материалы, посвященные наследованию классов в языке Си++;
- разработать программу на языке Си++ для решения заданного варианта задания;
- отладить программы;
- представить результаты работы программы;
- подготовить отчет по лабораторной работе.

## 3.2. Краткая характеристика объекта изучения

### 3.2.1 Общие сведения о наследовании классов

Основная идея, используемая при наследовании классов, заключается в том, что на основе существующего класса (класс родитель или базовый класс), создается производный класс (класс- наследник, дочерний класс), который включает в себя поля и функции базового класса (наследует их), и содержит дополнительные поля (обладает новыми свойствами) и функции.

Примеры определения производных классов:

*class S: X, Y, Z*

*{ ... } ;*

*class B: public A*

*{....};*

*class D: public X, protected B*

*{....};*

### 3.2.2 Статусы доступа при наследовании классов

Перед именем базового класса может указываться статус доступа наследования (одно из ключевых слов *public, protected, private*). Статус доступа наследования определяет статус доступа наследуемых полей и функций из базового класса внутри производного класса.

Производный класс может определяться с ключевым словом *struct* или *class* (с ключевым словом *union* производный класс не определяется). Если производный класс определен с ключевым словом *class*, то по умолчанию статус доступа наследования *private*.

Если производный класс определен с ключевым словом *struct*, то по умолчанию статус доступа наследования *public*.

Статусы доступа при наследовании классов представлены в таблице 1.

Таблица 1- Статусы доступа при наследовании классов

| **Тип наследования доступа** | **Доступность в базовом классе** | **Доступность компонент базового класса в производном** |
|---|---|---|
| public | public | public |
| | protected | protected |
| | private | не доступны |
| protected | public | protected |
| | protected | protected |
| | private | не доступны |
| private | public | private |
| | protected | private |
| | private | не доступны |

### 3.2.3. Особенности конструкторов при наследовании

Конструктор производного класса в первую очередь всегда должен вызывать конструктор базового класса. Если это действие не выполняется явно, то по умолчанию вызывается конструктор без параметров (если он есть, если его нет, будет ошибка). Если класс имеет несколько базовых, то конструкторы базовых классов должны вызываться в порядке перечисления этих классов в списке базовых.

### 3.2.4. Особенности деструкторов при наследовании

Деструктор производного класса всегда неявно по умолчанию после выполнения своего тела вызывает деструкторы базовых классов. Причем порядок разрушения объекта (вызовов деструкторов) обратен порядку создания (вызова конструкторов).

### 3.2.5. Переопределение функций. Виртуальные функции

Если в производном классе объявлена функция с именем, типом возвращаемого значения и количеством и типами параметров, такая же, как в базовом классе, то данная функция является переопределенной. (Не путать с перегрузкой функций). С помощью простых переопределенных функций реализуется механизм статического полиморфизма. (Полиморфизм – возможность функции в производном классе работать по-другому).

Суть статического связывания: когда указатель одного типа ссылается на объект другого типа при наследовании классов, то выбор переопределенного метода определяется типом указателя, а не типом объекта.

Суть динамического связывания: когда указатель одного типа ссылается на объект другого типа при наследовании классов, то выбор переопределенного метода определяется типом объекта, а не типом указателя, для этого переопределенный метод должен быть объявлен виртуальным в базовом классе. С помощью динамического связывания реализуется механизм динамического полиморфизма.

Динамический полиморфизм при переопределении метода в производном классе обеспечивается объявлением заголовка метода с ключевым словом virtual в базовом классе. Встретив у функции модификатор *virtual*, компилятор создает для класса таблицу виртуальных функций, а в класс добавляет новый скрытый для программиста член — указатель на эту таблицу.

Таблица виртуальных функций хранит в себе адреса всех виртуальных методов класса (по сути, это массив указателей), а также всех виртуальных методов базовых классов этого класса. За счет этих указателей на функции обеспечивается динамическое связывание: при вызове метода через указатель, который имеет тип базового класса, но содержит адрес объекта производного класса (такое преобразование допустимо и выполняется неявно), вызывается именно метод производного класса (вызов метода определяется типом объекта).

### 3.3. Задачи и порядок выполнения работы

В работе первоначально необходимо создать базовый класс и на его основе создать производный класс. При этом необходимо использовать вызов конструктора базового класса внутри конструктора производного класса, а также возможности переопределения методов (функций) класса. Необходимо разобраться с понятием виртуального метода, а также статическим и динамическим полиморфизмом. Продемонстрировать данные возможности в выполняемом примере.

Условие задачи:

Создать базовый класс «точка на плоскости». Элементы класса: поля, задающие координаты точки; конструктор для инициализации полей; функция для печати значений полей. Создать производный класс «точка в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для печати значений полей (внутри переопределенной функции в первую очередь должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить динамический полиморфизм, показать его особенности в программе.

***Пример выполнения работы***

Листинг программы с комментариями:

```
#include <stdlib.h>
#include <iostream>
using namespace std;
class point // Базовый класс - "Точка на плоскости"
{
	double x, y; // Координаты точки
public:
	point(double x, double y) // Конструктор для инициализации полей
	{
		this->x=x; this->y=y;
	}
	virtual void print() // Метод для печати полей (виртуальный)
	{
		cout<<"\nx="<<x<<" y="<<y; // Печатаем значения полей
	}
};
class point3d: public point // Производный класс - "Точка в пространстве"
```

```
{
	double z; // Новое поле - координата z
public:
	point3d(double x, double y, double z): // Конструктор
	  point(x, y) // Явный вызов конструктора базового класса
	  {
		this->z=z;
	  }
	  void print() // Переопределенный метод print
	  {
		point::print(); // Вызов в переопределенном методе метода
				// базового класса
		cout<<" z="<<z; // Допечатывает поле z
	  }
};

int main(int argc, char* argv[])
{
	point p1(1, 2); // Создается объект с вызовом конструктора
	point3d p3(3, 4, 5); // Создается объект с вызовом конструктора
	point *pp; // Указатель типа базового класса
	pp=&p1; // Настраиваем на объект базового класса
	pp->print(); // Вызов метода через указатель
	pp=&p3; // Настриваем указатель на объект производного класса
		// (преобразование типа допустимо)
	pp->print(); // Вызов метода через указатель, вызывается метод класса point3d
	// Если метод print в классе point был объявлен без virtual,
	// то вызывался бы метод print класс point
	system("pause"); // Останавливаем программу до нажатия любой клавиши
	return 0;
}
```

### 3.4. Форма отчета по лабораторной работе

Отчет должен содержать: титульный лист, цель работы, условие задачи, текст программы с комментариями, вывод результатов работы программы, выводы по работе.

### 3.5. Вопросы для самоконтроля

1. Производные классы, статусы доступа наследуемых полей и методов в производных классах.

2. Особенности конструкторов с производных классах.

3. Особенности деструкторов с производных классах.

4. Переопределение функций в производных классах, виртуальные функции.

5. Статическое и динамическое связывание.

### Варианты заданий для лабораторной работы № 3

Вариант № 1

Создать базовый класс «вектор на плоскости». Элементы класса: поля, задающие координаты точки (статус доступа *protected*), определяющей конец вектора (начало вектора находится в точке с координатами 0, 0); конструктор для инициализации полей; функция для вычисления длины вектора, функция для печати полей и длины вектора. Создать производный класс «вектор в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для вычисления длины вектора; переопределенная функция для печати полей и длины вектора. Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 2

Создать базовый класс «точка на плоскости». Элементы класса: поля, задающие координаты точки (статус доступа *protected*); конструктор для инициализации полей; функция для печати значений полей. Создать производный класс «точка в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для печати значений полей (внутри переопределенной функции в первую очередь должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 3

Создать базовый класс «квадрат». Элементы класса: поле, задающее длину стороны (статус доступа *protected*); конструктор для инициализации поля; функция для вычисления площади квадрата; функция для печати поля и площади квадрата. Создать производный класс «куб». Элементы класса: конструктор для инициализации поля; переопределенная функция для вычисления объема куба (вместо площади) (внутри переопределенной функции должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 4

Создать базовый класс «прямоугольник». Элементы класса: поля, задающие длины сторон (статус доступа *protected*); конструктор для инициализации полей; функция для вычисления площади прямоугольника; функция для печати полей и значения площади. Создать производный класс «прямоугольный параллелепипед». Элементы класса: дополнительное поле, задающее высоту; конструктор для инициализации полей; переопределенная функция для вычисления объема (вместо площади) (внутри переопределенной функции должна вызываться функция из базового класса); переопределенная функция для печати полей и значения объема. Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 5

Создать базовый класс «круг». Элементы класса: поле, задающее радиус; конструктор для инициализации поля (статус доступа *protected*); функция для вычисления площади круга (площадь круга $\pi r^2$); функция для печати полей и площади. Создать производный класс «шар». Элементы класса: конструктор для инициализации поля; переопределенная функция для вычисления объема (вместо площади круга) шара (площадь шара $\frac{4}{3}\pi r^3$). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 6

Создать базовый класс «автомобиль». Элементы класса: поле, содержащее наименование модели автомобиля; поле, содержащее значение максимальной скорости (статус доступа *protected*); конструктор для инициализации полей; функция для печати параметров автомобиля. Создать производный класс «грузовой автомобиль». Элементы класса: дополнительно поле, содержащее грузоподъемность автомобиля в тоннах; конструктор для инициализации полей; переопределенная функция печати параметров автомобиля (внутри переопределенной функции должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 7

Создать базовый класс «вещественное число». Элементы класса: поле, задающее значение числа (статус доступа *protected*); конструктор для инициализации поля; функция для вычисления модуля числа; функция для печати поля и модуля числа. Создать производный класс «комплексное число». Элементы класса: дополнительно поле, задающее значение мнимой части числа; конструктор для инициализации полей; переопределенная функция для вычисления модуля числа (модуль числа – корень квадратный из суммы квадратов вещественной и мнимой частей числа); переопределенная функция для печати полей и модуля числа. Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 8

Создать базовый класс «вектор на плоскости». Элементы класса: поля, задающие координаты точки (статус доступа *protected*), определяющей конец вектора (начало вектора находится в точке с координатами 0, 0); конструктор для инициализации полей; функция для печати координат вектора. Создать производный класс «вектор в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для печати координат вектора (внутри переопределенной функции должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов

созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 9

Создать базовый класс «квадрат». Элементы класса: поле, задающее длину стороны (статус доступа *protected*); конструктор для инициализации поля; функция для вычисления периметра квадрата; функция для печати длины стороны и периметра. Создать производный класс «прямоугольник». Элементы класса: дополнительное поле, задающее другую сторону; конструктор для инициализации полей; переопределенная функция для вычисления периметра прямоугольника; переопределенная функция для печати длин сторон и периметра. Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 10

Создать базовый класс «автомобиль». Элементы класса: поле, содержащее наименование модели автомобиля; поле, содержащее значение максимальной скорости (статус доступа *protected*); конструктор для инициализации полей; функция для печати параметров автомобиля. Создать производный класс «автобус». Элементы класса: дополнительно поле, содержащее максимальное число перевозимых пассажиров; конструктор для инициализации полей; переопределенная функция печати параметров автобуса (внутри переопределенной функции должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 11

Создать базовый класс «школа». Элементы класса: поле, содержащее название школы; поле, содержащее значение числа обучаемых в школе (статус доступа *protected*); конструктор для инициализации полей; функция для печати параметров школы. Создать производный класс «специализированная школа». Элементы класса: дополнительно поле, содержащее название специализации школы; конструктор для инициализации полей; переопределенная функция печати параметров школы (внутри переопределенной функции должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить

и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 12

Создать базовый класс «круг». Элементы класса: поле, содержащее значение радиуса круга (статус доступа *protected*); конструктор для инициализации поля; функция для печати радиуса круга. Создать производный класс «эллипс». Элементы класса: дополнительно поле, содержащее значение второй полуоси эллипса (для задания первой полуоси использовать наследуемое поле радиуса круга); конструктор для инициализации полей; переопределенная функция печати параметров эллипса (внутри переопределенной функции должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 13

Создать базовый класс - сотрудник предприятия. Компоненты класса: поля: ФИО, оклад, надбавка за стаж (в процентах от оклада за 1 год), стаж (в годах), статус доступа полей *protected*;

конструктор для инициализации полей;

функция для вычисления зарплаты;

функция для печати параметров сотрудника.

Создать производный класс - начальник подразделения.

Дополнительные поля: процентная надбавка к окладу за выполнение обязанностей начальника и название подразделения.

Переопределить функцию для вычисления зарплаты и функцию для печати параметров начальника. Внутри переопределенных функций вызывать соответствующие функции из базового класса.

Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 14

Создать базовый класс – простой счет в банке. Компоненты класса: поля: ФИО владельца, начальная сумма счета, ставка вклада (проценты в год), время существования вклада в годах, статус доступа полей *protected*;

конструктор для инициализации полей;

функция для вычисления суммы на счете с учетом начисленных процентов за время существования вклада;

функция для печати параметров счета.

Создать производный класс – привилегированный счет.

Дополнительные поля: процент кредита предоставляемому по счету (проценты от доступной на счете суммы с учетом времени существования вклада).

Переопределенная функция для вычисления суммы на счете с учетом доступного кредита.

Переопределенная функция для печати параметров счета.

Внутри переопределенных функций вызывать соответствующие функции из базового класса.

Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 15

Создать базовый класс «вектор на плоскости». Элементы класса: поля, задающие координаты точки (статус доступа *protected*), определяющей конец вектора (начало вектора находится в точке с координатами 0, 0); конструктор для инициализации полей; функция для вычисления длины вектора, функция для печати полей и длины вектора. Создать производный класс «вектор в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для вычисления длины вектора; переопределенная функция для печати полей и длины вектора. Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 16

Создать базовый класс «точка на плоскости». Элементы класса: поля, задающие координаты точки (статус доступа *protected*); конструктор для инициализации полей;

функция для печати значений полей. Создать производный класс «точка в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для печати значений полей (внутри переопределенной функции в первую очередь должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 17

Создать базовый класс «квадрат». Элементы класса: поле, задающее длину стороны (статус доступа *protected*); конструктор для инициализации поля; функция для вычисления площади квадрата; функция для печати поля и площади квадрата. Создать производный класс «куб». Элементы класса: конструктор для инициализации поля; переопределенная функция для вычисления объема куба (вместо площади) (внутри переопределенной функции должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 18

Создать базовый класс «прямоугольник». Элементы класса: поля, задающие длины сторон (статус доступа *protected*); конструктор для инициализации полей; функция для вычисления площади прямоугольника; функция для печати полей и значения площади. Создать производный класс «прямоугольный параллелепипед». Элементы класса: дополнительное поле, задающее высоту; конструктор для инициализации полей; переопределенная функция для вычисления объема (вместо площади) (внутри переопределенной функции должна вызываться функция из базового класса); переопределенная функция для печати полей и значения объема. Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 19

Создать базовый класс «круг». Элементы класса: поле, задающее радиус; конструктор для инициализации поля (статус доступа *protected*); функция для вычисления площади круга (площадь круга $\pi r^2$); функция для печати полей и площади. Создать производный класс «шар». Элементы класса: конструктор для инициализации поля; переопределенная функция для вычисления объема (вместо площади круга) шара (площадь шара $\frac{4}{3}\pi r^3$). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 20

Создать базовый класс «автомобиль». Элементы класса: поле, содержащее наименование модели автомобиля; поле, содержащее значение максимальной скорости (статус доступа *protected*); конструктор для инициализации полей; функция для печати параметров автомобиля. Создать производный класс «грузовой автомобиль». Элементы класса: дополнительно поле, содержащее грузоподъемность автомобиля в тоннах; конструктор для инициализации полей; переопределенная функция печати параметров автомобиля (внутри переопределенной функции должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 21

Создать базовый класс «вещественное число». Элементы класса: поле, задающее значение числа (статус доступа *protected*); конструктор для инициализации поля; функция для вычисления модуля числа; функция для печати поля и модуля числа. Создать производный класс «комплексное число». Элементы класса: дополнительно поле, задающее значение мнимой части числа; конструктор для инициализации полей; переопределенная функция для вычисления модуля числа (модуль числа – корень квадратный из суммы квадратов вещественной и мнимой частей числа); переопределенная функция для печати полей и модуля числа. Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 22

Создать базовый класс «вектор на плоскости». Элементы класса: поля, задающие координаты точки (статус доступа *protected*), определяющей конец вектора (начало вектора находится в точке с координатами 0, 0); конструктор для инициализации полей; функция для печати координат вектора. Создать производный класс «вектор в трехмерном пространстве». Элементы класса: дополнительное поле, задающее дополнительную координату; конструктор для инициализации полей; переопределенная функция для печати координат вектора (внутри переопределенной функции должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 23

Создать базовый класс «квадрат». Элементы класса: поле, задающее длину стороны (статус доступа *protected*); конструктор для инициализации поля; функция для вычисления периметра квадрата; функция для печати длины стороны и периметра. Создать производный класс «прямоугольник». Элементы класса: дополнительное поле, задающее другую сторону; конструктор для инициализации полей; переопределенная функция для вычисления периметра прямоугольника; переопределенная функция для печати длин сторон и периметра. Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 24

Создать базовый класс «автомобиль». Элементы класса: поле, содержащее наименование модели автомобиля; поле, содержащее значение максимальной скорости (статус доступа *protected*); конструктор для инициализации полей; функция для печати параметров автомобиля. Создать производный класс «автобус». Элементы класса: дополнительно поле, содержащее максимальное число перевозимых пассажиров; конструктор для инициализации полей; переопределенная функция печати параметров автобуса (внутри переопределенной функции должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 25

Создать базовый класс «школа». Элементы класса: поле, содержащее название школы; поле, содержащее значение числа обучаемых в школе (статус доступа *protected*); конструктор для инициализации полей; функция для печати параметров школы. Создать производный класс «специализированная школа». Элементы класса: дополнительно поле, содержащее название специализации школы; конструктор для инициализации полей; переопределенная функция печати параметров школы (внутри переопределенной функции должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 26

Создать базовый класс «круг». Элементы класса: поле, содержащее значение радиуса круга (статус доступа *protected*); конструктор для инициализации поля; функция для печати радиуса круга. Создать производный класс «эллипс». Элементы класса: дополнительно поле, содержащее значение второй полуоси эллипса (для задания первой полуоси использовать наследуемое поле радиуса круга); конструктор для инициализации полей; переопределенная функция печати параметров эллипса (внутри переопределенной функции должна вызываться функция из базового класса). Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 27

Создать базовый класс - сотрудник предприятия. Компоненты класса: поля: ФИО, оклад, надбавка за стаж (в процентах от оклада за 1 год), стаж (в годах), статус доступа полей *protected*;

конструктор для инициализации полей;

функция для вычисления зарплаты;

функция для печати параметров сотрудника.

Создать производный класс - начальник подразделения.

Дополнительные поля: процентная надбавка к окладу за выполнение обязанностей начальника и название подразделения.

Переопределить функцию для вычисления зарплаты и функцию для печати параметров начальника. Внутри переопределенных функций вызывать соответствующие функции из базового класса.

Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.

Вариант № 28

Создать базовый класс – простой счет в банке. Компоненты класса: поля: ФИО владельца, начальная сумма счета, ставка вклада (проценты в год), время существования вклада в годах, статус доступа полей *protected*;

конструктор для инициализации полей;

функция для вычисления суммы на счете с учетом начисленных процентов за время существования вклада;

функция для печати параметров счета.

Создать производный класс – привилегированный счет.

Дополнительные поля: процент кредита предоставляемому по счету (проценты от доступной на счете суммы с учетом времени существования вклада).

Переопределенная функция для вычисления суммы на счете с учетом доступного кредита.

Переопределенная функция для печати параметров счета.

Внутри переопределенных функций вызывать соответствующие функции из базового класса.

Создать по 1 объекту каждого из классов. Показать вызов созданных функций. При переопределении функций обеспечить и продемонстрировать два варианта: статический полиморфизм и динамический полиморфизм.