**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

Лабораторная работа № 3

по ОБЪЕКТНО-ОРИЕНТИРОВАННОМУ ПРОГРАММИРОВАНИЮ

«Наследование и полиморфизм»

Студент \_\_\_\_\_\_\_\_\_\_\_ Целищев А.Е.

Группа ПМ-21-2

Руководитель

Доцент \_\_\_\_\_\_\_\_\_\_\_ Кургасов В.В.

Липецк 2022г.

Оглавление

Цель работы 2

Задание кафедры 2

Код программы 3

Пример работы программы 6

Вывод 7

Контрольные вопросы...........................................................................................8

**Цель работы:**

Изучить механизмы наследования и полиморфизма с использованием виртуальных функций.

**Задание кафедры:**

Реализовать на языке С++ концепцию векторной функции скалярного аргумента (с привлечением механизма наследования и виртуальных функций) вида:

.

Количество элементов вектора результата и аргумент *х*, для которого производится вычисление, задаётся пользователем. Вид функции, по которому вычисляется значение каждого элемента вектора результата, указывается пользователем из заданного множества параметризованных функций (табл.). Параметры функций вводятся пользователем для каждого элемента вектора результата отдельно, при формировании вектора функций.

Элементы векторной функции скалярного аргумента

|  |  |  |
| --- | --- | --- |
| Вариант | Функция 1 | Функция 2 |
| 11 | *a\*cos(b\*x) + c* | a\* |

***Код программы***

#include <iostream>

#include <cmath>

**using** **namespace** std;

// Базовый абстрактный класс, являющийся интерфейсом для производных классов,

// определяющих различные вектор-функции (ниже - MyFunction).

**class** VectorFunction

{

// Интерфейс рассчитан на вектор-функции размера 2,

// поэтому в protected принимаются два набора коэффициентов,

// причем первая функция имеет 3 к-та, вторая - 2

**protected**:

**double** coefs\_one[3];

**double** coefs\_two[2];

**double** answer[2];

**public**:

**virtual** **void** set\_coefs\_one(**double** \*coefs) = 0; // Задать к-ты для первой функции

**virtual** **void** set\_coefs\_two(**double** \*coefs) = 0; // Задать к-ты для второй функции

**virtual** **void** calculate(**double** x) = 0; // Посчитать значение вектор-функции от аргумента х

**double** get\_ans(**int** i) // вывести i-тый (начиная с 1) элемент вектор-функции

{

**return** **this**->answer[i-1];

}

};

**class** MyFunction: **public** VectorFunction // Наследование методов и свойств из класса VectorFunction

{

**public**:

**void** set\_coefs\_one(**double** \*coefs) override // Задать к-ты для первой функции. Передаётся массив

{

**for** (**int** i = 0; i < 3; i++)

{

**this**->coefs\_one[i] = coefs[i];

}

}

**void** set\_coefs\_two(**double** \*coefs) override // Задать к-ты для второй функции. Передаётся массив

{

**for** (**int** i = 0; i < 2; i++)

{

**this**->coefs\_two[i] = coefs[i];

}

}

// Результаты вычислений записываются в результирующий вектор,

// определенный в интерфейсе через массив длины 2.

// При помощи унаследованного метода get\_ans(int i) можно затем

// обратиться к любому из элементов результирующего вектора.

**void** calculate(**double** x) override

{

**this**->answer[0] = **this**->coefs\_one[0] \* cos(x \* **this**->coefs\_one[1]) + **this**->coefs\_one[2];

**this**->answer[1] = **this**->coefs\_two[0] \* pow(x, **this**->coefs\_two[1]);

}

};

**int** main(){

MyFunction vf = MyFunction(); // Создаём объект класса MyFunction. Создание вектор-функции

// Определяем коэффициенты первой и второй функции созданной вектор-функции

**double** coefs1[3], coefs2[2];

cout << "Enter coefficients a, b, c for the first function:\n";

cin >> coefs1[0] >> coefs1[1] >> coefs1[2];

vf.set\_coefs\_one(coefs1);

cout << "Enter coefficients a, b for the second function:\n";

cin >> coefs2[0] >> coefs2[1];

vf.set\_coefs\_two(coefs2);

// Получаем массив аргументов

**int** x\_size;

cout << "Enter amount of x values:\n";

cin >> x\_size;

**double** x\_values[x\_size];

cout << "Enter x-values. If their amount is more than 1, separate them with space:\n";

**for** (**int** i = 0; i < x\_size; i++)

{

cin >> x\_values[i];

}

// Вычисляем результат для каждого аргумента и выводим в консоль поэлементно

**for** (**int** i = 0; i < x\_size; i++)

{

vf.calculate(x\_values[i]);

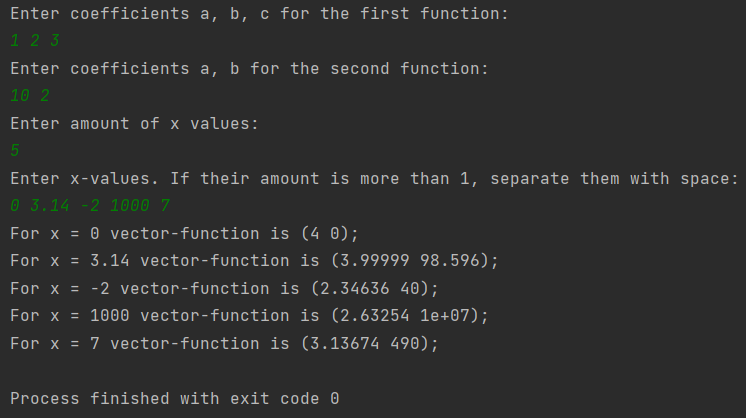
cout << "For x = " << x\_values[i] << " vector-function is (" << vf.get\_ans(1) << ' ' << vf.get\_ans(2) << ");\n";

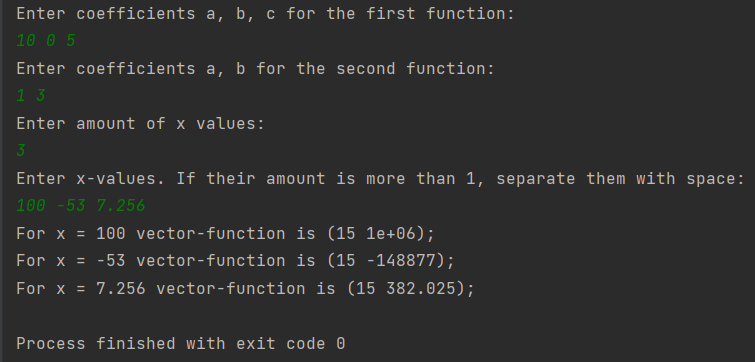
}

**return** 0;

}

***Пример работы программы***





***Вывод:***

В ходе выполнения данной работы изучил механизмы наследования и полиморфизма с использованием виртуальных функций. С использованием полученных знаний реализовал на языке С++ концепцию векторной функции скалярного аргумента, которую можно успешно применять в различных расчётах.

***Контрольные вопросы***

1. **Зачем нужен механизм наследования?**

Основное назначение - повторное использование кода, так как большинство используемых типов данных являются вариантами друг друга, и для каждого писать свой класс нецелесообразно. Таким образом, благодаря механизму наследования становится возможным создать новый класс на основе существующего (унаследовав свойства и методы этого класса), лишь переписав отличающийся функционал.

1. **Зачем используются модификаторы при наследовании классов? Какие это модификаторы?**

Модификаторы: public, protected, private.

При наследовании с модификатором public все данные родительского класса в уровнях доступа public и protected наследуются без изменения уровня доступа к ним;

При наследовании с модификатором protected все унаследованные данные становятся защищёнными (все заносятся в уровень доступа protected);

При наследовании с модификатором private все унаследованные данные становятся приватными (заносятся в уровень доступа private)

1. **Зачем нужен механизм полиморфизма?**

Механизм полиморфизма нужен для того, чтобы один и тот же класс использовать для решения общих для определённого класса действий, различие между которыми будет во входных данных. То есть в таком случае определённый класс является «общим решением» некоторой задачи. Более чёткое определение полиморфизма: полиморфизм - это механизм, позволяющий использовать один и тот же интерфейс для общего класса действий.

1. **Что понимается под динамическим полиморфизмом?**

Динамический полиморфизм заключается в использовании виртуальных функций, которые определяются в базовом классе и могут быть переопределены в классе-наследнике. То есть в наследнике переопределённая виртуальная функция будет иметь реализацию, отличную от той, что определена в родительском классе. Это особенно полезно при работе с различными типами данных, но когда для всех классов (базового и его наследников) целесообразно иметь один метод вместо нескольких. Таким образом существенно упрощается работа с кодом: «одна и та же» функция имеет одну и ту же логику использования, имея лишь отличную реализацию под конкретный тип данных. Связывание вызова таких функций с их определениями происходит во время выполнения программы, поэтому такой полиморфизм динамический.

1. **К какому из видов полиморфизма относится перегрузка оператора?**

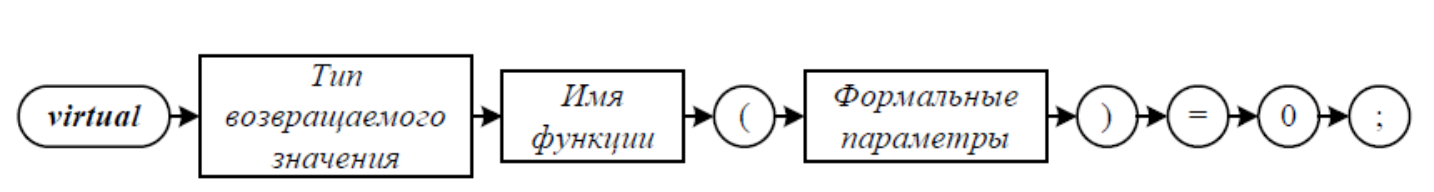
Перегрузка операторов и функций относится к статическому полиморфизму, так как они определяются во время компиляции программы.

1. **Что такое интерфейс класса?**

Интерфейс описывает поведение и возможности класса, не связываясь с его конкретной реализацией. Реализуются интерфейсы с использованием абстрактных классов. По сути, абстрактный класс может называться интерфейсом класса, так как он задаёт основную структуру и возможности для классов-наследников.

1. **Зачем нужен чисто виртуальный метод? Как он выглядит?**

Чисто виртуальный метод в общем виде выглядит так:



Пример определения такого метода:

virtual void func() = 0;

Чисто виртуальные методы используются, чтобы отложить реализацию до момента их переопределения в производных классах. Это также гарантирует, что для каждого производного класса метод будет переопределён, иначе компилятор выдаст ошибку.

1. **Какой класс называется абстрактным?**

*Абстрактный класс* - это такой класс, который содержит хотя бы одну чистую виртуальную функцию (метод). В чистой виртуальной функции тело не определено, поэтому на основе абстрактных классов нельзя создавать объекты, так как они, имея в своём составе чистые виртуальные функции, не являются полноценными типами данных.