Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (ПНИПУ)

Электротехнический факультет

Дисциплина: основы алгоритмизации и программирования, 2 семестр

ОТЧЁТ

«Лабораторная работа ООП № 4. Простое наследование.

Принцип подстановки»

Выполнил

Студент группы РИС-22-2Б

Хохряков А.С.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь 2023

# Содержание отчета

1. Постановка задачи.
2. Анализ задачи.
3. UML-Диаграмма
4. Ответы на вопросы
5. Текст программы

Постановка задачи

1. Определить пользовательский класс.

2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

3. Определить в классе деструктор.

4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).

5. Перегрузить операцию присваивания.

6. Перегрузить операции ввода и вывода объектов с помощью потоков.

7. Определить производный класс.

8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

9. Реализовать функции, получающие и возвращающие объект базового класса. Продемонстрировать принцип подстановки.

Вариант 3:  
Базовый класс:

ПАРА ЧИСЕЛ (PAIR)

Первое число (first) - int

Второе число (second) - int

Определить методы изменения полей и вычисления произведения чисел.

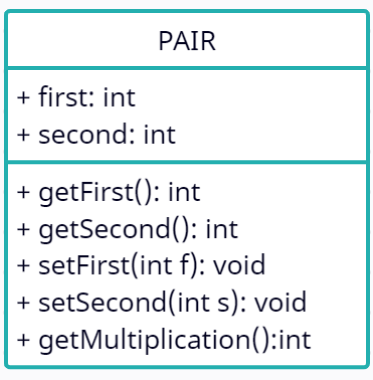
Создать производный класс ПРЯМОУГОЛЬНЫЙ ТРЕУГОЛЬНИК

((RIGHT ANGLED), с полями-катетами. Определить метод вычисления гипотенузы.

Анализ задачи

1. Объявление класса PAIR с защищенными полями first и second и публичными методами для получения и изменения значений полей, а также методом для вычисления произведения значений полей.
2. Объявление класса RIGHTANGLED, который наследует от класса PAIR и имеет дополнительный метод для вычисления гипотенузы по значениям полей first и second.
3. В функции main() объявление двух целочисленных переменных f и s.
4. Вывод сообщения "Enter two numbers: ".
5. Чтение значений f и s с помощью объекта cin.
6. Создание объекта класса PAIR с передачей значений f и s в конструктор.
7. Вывод значений полей объекта PAIR с помощью методов getFirst(), getSecond() и getMultiplication().
8. Создание объекта класса RIGHTANGLED с передачей значений f и s в конструктор.
9. Вывод значений полей объекта RIGHTANGLED (катеты) с помощью методов getFirst() и getSecond(), а также вычисленной гипотенузы с помощью метода getHypotenuse().

UML – Диаграмма



Ответы на вопросы

1. Механизм наследования в объектно-ориентированном программировании используется для создания новых классов на основе существующих классов. Наследование позволяет унаследовать свойства и методы базового класса в производный класс и переопределить или дополнить их по необходимости.
2. Компоненты класса, описанные со спецификатором public, наследуются в производный класс без изменений, то есть они остаются открытыми и доступными для использования из любого места программы.
3. Компоненты класса, описанные со спецификатором private, не наследуются в производный класс и не доступны для использования в нем.
4. Компоненты класса, описанные со спецификатором protected, наследуются в производный класс и остаются защищенными, то есть доступны только в базовом и производном классах.
5. Производный класс описывается с помощью ключевого слова "class" или "struct", за которым следует имя производного класса и двоеточие, а затем имя базового класса, от которого он наследуется.
6. Конструкторы базового класса наследуются в производный класс, но не могут быть вызваны напрямую из производного класса. Вместо этого они вызываются автоматически при создании объекта производного класса.
7. Деструкторы базового класса также наследуются в производный класс и вызываются автоматически при удалении объекта производного класса.
8. Объекты производных классов конструируются в порядке от базового класса к производному классу. Сначала вызывается конструктор базового класса, а затем конструктор производного класса.
9. Объекты производных классов уничтожаются в порядке от производного класса к базовому классу. Сначала вызывается деструктор производного класса, а затем деструктор базового класса.
10. Виртуальные функции - это функции, которые могут быть переопределены в производных классах. Механизм позднего связывания позволяет вызывать функцию, определенную в производном классе, даже если указатель на объект имеет тип базового класса. Это позволяет динамически определять, какая функция будет вызываться во время выполнения программы.
11. Конструкторы могут быть виртуальными, но это не рекомендуется, так как конструктор не может быть переопределен в производных классах. Деструкторы, напротив, часто делаются виртуальными, чтобы гарантировать правильное освобождение памяти при удалении объектов производных классов через указатель на базовый класс.
12. Спецификатор virtual не наследуется, но виртуальная функция, объявленная в базовом классе, может быть переопределена как виртуальная в производном классе.
13. Открытое наследование устанавливает отношение "является" между базовым и производным классами. Производный класс является расширением базового класса и наследует его свойства и методы.
14. Закрытое наследование устанавливает отношение "реализует" между базовым и производным классами. Производный класс реализует интерфейс базового класса, но не является его расширением и не наследует его свойства и методы.
15. Принцип подстановки (Liskov substitution principle) заключается в том, что объекты производных классов должны быть способны заменять объекты базовых классов без нарушения корректности программы. Это означает, что поведение производного класса должно быть совместимо с поведением базового класса и не должно приводить к ошибкам или непредсказуемым результатам.
16. Объект x класса Teacher будет иметь следующие компонентные данные:

* age (наследовано от класса Student);
* name (наследовано от класса Student);
* post (унаследовано от класса Employee);
* stage.

1. Конструкторы без параметров:

class Student {

public:

Student() : age(0), name("") {}

int age;

string name;

};

class Employee : public Student {

public:

Employee() : post("") {}

string post;

};

class Teacher : public Employee {

public:

Teacher() : stage(0) {}

int stage;

};

1. Конструкторы с параметрами:

class Student {

public:

Student(int a, string n) : age(a), name(n) {}

int age;

string name;

};

class Employee : public Student {

public:

Employee(int a, string n, string p) : post(p) {}

string post;

};

class Teacher : public Employee {

public:

Teacher(int a, string n, string p, int s) : stage(s) {}

int stage;

};

1. Конструкторы копирования:

class Student {

public:

Student(const Student& other) : age(other.age), name(other.name) {}

int age;

string name;

};

class Employee : public Student {

public:

Employee(const Employee& other) : post(other.post) {}

string post;

};

class Teacher : public Employee {

public:

Teacher(const Teacher& other) : stage(other.stage) {}

int stage;

};

1. Операции присваивания:

class Student {

public:

Student& operator=(const Student& other) {

if (this != &other) {

age = other.age;

name = other.name;

}

return \*this;

}

int age;

string name;

};

class Employee : public Student {

public:

Employee& operator=(const Employee& other) {

if (this != &other) {

Student::operator=(other);

post = other.post;

}

return \*this;

}

string post;

};

class Teacher : public Employee {

public:

Teacher& operator=(const Teacher& other) {

if (this != &other) {

Employee::operator=(other);

stage = other.stage;

}

return \*this;

}

int stage;

};

Текст программы

#include <iostream>

#include <cmath>

using namespace std;

class PAIR {

protected:

int first, second;

public:

PAIR() : first(0), second(0) {}

PAIR(int f, int s) : first(f), second(s) {}

PAIR(const PAIR& p) : first(p.first), second(p.second) {}

~PAIR() {}

int getFirst() const {

return first;

}

int getSecond() const {

return second;

}

void setFirst(int f) {

first = f;

}

void setSecond(int s) {

second = s;

}

int getMultiplication() const {

return first \* second;

}

};

class RIGHTANGLED : public PAIR {

public:

RIGHTANGLED() : PAIR() {}

RIGHTANGLED(int f, int s) : PAIR(f, s) {}

RIGHTANGLED(const PAIR& p) : PAIR(p) {}

double getHypotenuse() const {

return sqrt(first \* first + second \* second);

}

};

int main() {

int f, s;

cout << "Enter two numbers: ";

cin >> f >> s;

PAIR pair(f, s);

cout << "First number: " << pair.getFirst() << ", second number: " << pair.getSecond() << ", multiplication: " << pair.getMultiplication() << endl;

RIGHTANGLED rightangled(f, s);

cout << " Cathetus: " << rightangled.getFirst() << ", " << rightangled.getSecond() << ", hypotenuse: " << rightangled.getHypotenuse() << endl;

return 0;

}