# Задача 1 (20 точки)

Изходът от програмата е:

```
Bar called !
Foo called !
Foo called !
Foo called !
```

Защо - при извикването на конструктора на `Baz` се извиква конструктора на `Bar` (`Baz` наследява `Bar`). Също така, в конструктора на `Baz` се заделя нов динамичен масив с три елемента, което от своя страна извиква 3 пъти конструктора на `Foo`

Изход от програмта - 10 точки.

Обяснение - 10 точки.

# Задача 2 (20 точки)

Грешките са:

- Липса на виртуален деструктор (5 точки) - в този случай няма да се извика деструктора на `Bar` и ще се получи memory-leak (5 точки)
- Липса на оператор= и copy конструктор (5 точки) - при копиране на обект, ще се получи така, че две инстанции споделят една и съща памет. При извикване на деструктора, първия обект ще изтрие паметта, което ще означава че втората инстация ще опита да изтрие вече изтрита памет, което ще доведе до грешка.

# Задача 3 (80 точки)

```
class IntegerOperation{
public:
    IntegerOperation(): capacity(1), size(0)
    {
        this->integers = new int[1];
    }

    IntegerOperation(const IntegerOperation& from): capacity(from.capacity),
size(from.size)
    {
        IntegerOperation::copy_memory(this->integers, from.integers);
    }

    IntegerOperation& operator=(const IntegerOperation& from)
    {
        if(this != &from)
        {
            this->capacity = from.capacity;
            this->size = from.size;

            IntegerOperation::clear_memory(this->integers);
            IntegerOperation::copy_memory(this->integers, from.integers);
```

```
        }
        return *this;
    }
    ~IntegerOperation()
    {
        IntegerOperation::clear_memory(this->integers);
    }

    void insert(const int& integer)
    {
        if(this->size == this->capacity)
        {
            this->resize();
        }

        this->integers[this->size] = integer;
        this->size++;
    }

    void sum()
    {
        this->result = 0;
        for(int i = 0; i < this->size; i++)
        {
            this->result += this->integers[i];
        }
    }

    void sub()
    {
        this->result = 0;
        for(int i = 0; i < this->size; i++)
        {
            this->result -= this->integers[i];
        }
    }

    void multiply()
    {
        this->result = 1;
        for(int i = 0; i < this->size; i++)
        {
            this->result *= this->integers[i];
        }
    }
    int get_result() const
    {
        return this->result;
    }

    int operator+(IntegerOperation& second)
    {
        this->sum();
        second.sum();

        return this->get_result() + second.get_result();
    }
```

```
    int operator-(IntegerOperation& second)
    {
        this->sub();
        second.sub();

        return this->get_result() - second.get_result();
    }

    int operator*(IntegerOperation& second)
    {
        this->multiply();
        second.multiply();

        return this->get_result() * second.get_result();
    }
private:
    void copy_memory(int* destination, const int* source)
    {
        destination = new int[this->capacity];

        for(int i = 0; i < this->size; i++)
        {
            destination[i] = source[i];
        }
    }
    static void clear_memory(int* source)
    {
        delete[] source;
    }

    void resize()
    {
        int* new_memory = new int[this->capacity * 2];

        for(int i = 0; i < this->size; i++)
        {
            new_memory[i] = this->integers[i];
        }
        this->capacity *= 2;
        clear_memory(this->integers);
        this->integers = new_memory;
    }
    int* integers;
    int result;
    int capacity;
    int size;
};
```

- Операция "събиране" - 6 точки
- Операция "изваждане" - 6 точки
- Операция "умножение" - 6 точки
- Оператор+ - 6 точки
- Оператор- - 6 точки
- Оператор* - 6 точки
- Голяма четворка (всяко по 6 точки) - 24 точки
- Метод `insert()` - 10 точки

- const методи, const & - 5 точки
- Изнасяне на логика за разширяване на динамичен масив в отделни функции - 5 точки
- Ако кодът не компилира, 50% надолу

## Задача 4 (80 точки)

```
#include <iostream>
#include <vector>
#include <string>

class Card{
public:
    Card() = default;
    Card(const std::string& initial_name, const unsigned& initial_id, const
unsigned& initial_image_id): name(initial_name), id(initial_id),
image_id(initial_image_id) {}

    void set_name(const std::string& new_name)
    {
        this->name = new_name;
    }

    void set_id(const unsigned& new_id)
    {
        this->id = new_id;
    }

    void set_image_id(const unsigned& new_image_id)
    {
        this->image_id = new_image_id;
    }
    unsigned get_id() const
    {
        return this->id;
    }
    unsigned get_image_id() const
    {
        return this->image_id;
    }
    std::string get_name() const
    {
        return this->name;
    }

    virtual void print() const
    {
        std::cout << "Name: " << this->name << ", ID: " << this->id << ", Image
ID: " << this->image_id;
    }

    virtual ~Card() = 0;
protected:
    std::string name;
    unsigned id;
    unsigned image_id;
};
```

```
class HeroCard: virtual public Card
{
public:
    HeroCard() = default;
    HeroCard(const std::string& initial_name, const unsigned& initial_id, const
unsigned& initial_image_id,
             const unsigned& initial_attack_power, const unsigned&
initial_defence_power):
             Card(initial_name, initial_id, initial_image_id),
attack_power(initial_attack_power), defence_power(initial_defence_power)
    {}

    void set_attack_power(const unsigned& new_attack_power)
    {
        this->attack_power = new_attack_power;
    }

    void set_defence_power(const unsigned& new_defence_power)
    {
        this->defence_power = new_defence_power;
    }

    unsigned get_attack_power() const
    {
        return this->attack_power;
    }

    unsigned get_defence_power() const
    {
        return this->defence_power;
    }

    void print() const override
    {
        Card::print();
        std::cout << ", Attack power: " << this->attack_power << ", Defence
power: " << this->defence_power << std::endl;
    }

    ~HeroCard() override = default;

protected:
    unsigned attack_power;
    unsigned defence_power;
};

class MagicCard: virtual public Card{
public:
    MagicCard() = default;
    MagicCard(const std::string& initial_name, const unsigned& initial_id, const
unsigned& initial_image_id,
             const std::string& initial_effect): Card(initial_name, initial_id,
initial_image_id), effect(initial_effect) {
    }

    void set_effect(const std::string& new_effect)
    {
```

```
        this->effect = new_effect;
    }

    std::string get_effect() const
    {
        return this->effect;
    }

    void print() const override
    {
        Card::print();
        std::cout << ", Effect: " << this->effect << std::endl;
    }

    ~MagicCard() override = default;

protected:
    std::string effect;
};

class SpecialCard: public HeroCard, public MagicCard{
public:
    SpecialCard() = default;
    SpecialCard(const std::string& initial_name, const unsigned& initial_id,
const unsigned& initial_image_id,
                const unsigned& initial_attack_power, const unsigned&
initial_defence_power,const std::string& initial_effect,
                const unsigned& initial_level): Card(initial_name, initial_id,
initial_image_id), level(initial_level)
    {
        HeroCard::attack_power = initial_attack_power;
        HeroCard::defence_power = initial_defence_power;
        MagicCard::effect = initial_effect;
    }

    void set_level(const unsigned& new_level)
    {
        this->level = new_level;
    }

    unsigned get_level() const
    {
        return this->level;
    }

    void print() const override
    {
        Card::print();

        std::cout << ", Attack power: " << this->attack_power
        << ", Defence power: " << this->defence_power
        << ", Effect: " << this->effect << std::endl;
    }
    ~SpecialCard() override = default;
private:
    unsigned level;
};
class Deck{
```

```
public:
    static Deck* get_instance()
    {
        return instance;
    }

    void add_card(const HeroCard& new_card)
    {
        this->cards.push_back(new HeroCard(new_card));
    }

    void add_card(const MagicCard& new_card)
    {
        this->cards.push_back(new MagicCard(new_card));
    }

    void print() const
    {
        for(auto* card: cards)
        {
            card->print();
        }
    }
private:
    static Deck* instance;
    Deck() = default;
    ~Deck()
    {
        for(auto* card: cards)
        {
            delete card;
        }
        delete instance;
    }
    std::vector<Card*> cards;
};
Deck* Deck::instance = new Deck();
```

## Клас Card (16т)

- Get/Set (всеки по 0,5 точки) - 3т
- Виртуален деструктор - 5т
- Виртуален метод за принтиране - 5т
- Стил - const методи, const &, unsigned (всяко по 1т) - 3т

## Клас HeroCard (15т)

- Виртуално наследяване - 5т
- Get/Set (всяко по 0,5 точки) - 2т
- Стил - const методи, const &, unsigned (всяко по 1т) - 3т
- Преизползване на print() от горния клас - 5т

## Клас MagicCard (14т)

- Виртуално наследяване - 5т

- Get/Set (всяко по 0,5 точки) - 1т
- Стил - const методи, const &, unsigned - (всяко по 1т) - 3т
- Преизползване на print() от горния клас - 5т

## Клас SpecialCard (14т)

- Наследява `HeroCard` и `MagicCard` - 5т
- Get/Set (всяко по 0,5 точки) - 1т
- Преизползване на print() - 5т
- Стил - const методи, const &, unsigned - (всяко по 1т) - 3т

## Клас Deck (21т)

- Хетерогнен контейнер (вектор от указатели към базов клас - 5 точки, копиране в динамичната памет - 5 точки, правилен деструктор - 5 точки) - 15 точки
- Правилен метод `print()` - 5 точки
- Стил - const методи, const & (всяко по 0,5т) - 1т

- Бонус: Singleton - 10 точки

Ако някой клас не компилира, 50% надолу (за клас)