Задания

1) Реализовать проект библиотеки librobo с использованием CMake. Библиотека должна содержать интерфейсный класс IRobot, объявляющий методы:

setPosition - установка текущих координат
setMotion - задание перемещение
getPosition - получить текущее местоположение

Далее необходимо создать два класса-наследника, имплементирующих указанные методы для случаев перемещения на плоскости (класс Robot2D) и в пространстве (класс Robo3D).

По сути, метод setMotion должен реализовывать вычисление нового местоположения при помощи матрицы пермещения для 2D и 3D случаев (https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_%D0%BF%D0%B5%D1%80%D0%B5%D1%85%D0%BE%D0%B4%D0%B0)

2) Для каждой из следующих программ определите результат выполнения. Если программа не скомпилируется, то объясните почему. Запускать код не нужно, вы должны определить результат/ошибки программ без помощи компилятора.

а)

```
#include <iostream>

class Parent
{
public:
    Parent()
    {
        std::cout << "Parent()\n";
    }
    ~Parent()
    {
        std::cout << "~Parent()\n";
    }
};

class Child: public Parent
{
public:
    Child()
    {
        std::cout << "Child()\n";
    }
    ~Child()
    {
        std::cout << "~Child()\n";
```

```
    }
};

int main()
{
    Child ch;
}
```

b)

```
#include <iostream>

class Parent
{
public:
    Parent()
    {
        std::cout << "Parent()\n";
    }
    ~Parent()
    {
        std::cout << "~Parent()\n";
    }
};

class Child: public Parent
{
public:
    Child()
    {
        std::cout << "Child()\n";
    }
    ~Child()
    {
        std::cout << "~Child()\n";
    }
};

int main()
{
    Child ch;
    Parent p;
}
```

c)

```
#include <iostream>

class Parent
{
private:
    int m_x;
public:
    Parent(int x): m_x(x)
    {
        std::cout << "Parent()\n";
    }
    ~Parent()
    {
        std::cout << "~Parent()\n";
    }

    void print() { std::cout << "Parent: " << m_x << '\n';  }
};

class Child: public Parent
{
public:
    Child(int y):  Parent(y)
    {
        std::cout << "Child()\n";
    }
    ~Child()
    {
        std::cout << "~Child()\n";
    }

    void print() { std::cout << "Child: " << m_x << '\n'; }
};

int main()
{
    Child ch(7);
    ch.print();
}
```

d)

```
#include <iostream>

class Parent
{
protected:
    int m_x;
public:
    Parent(int x): m_x(x)
    {
        std::cout << "Parent()\n";
    }
    ~Parent()
    {
        std::cout << "~Parent()\n";
    }

    void print() { std::cout << "Parent: " << m_x << '\n';  }
};

class Child: public Parent
{
public:
    Child(int y):  Parent(y)
    {
        std::cout << "Child()\n";
    }
    ~Child()
    {
        std::cout << "~Child()\n";
    }

    void print() { std::cout << "Child: " << m_x << '\n'; }
};

int main()
{
    Child ch(7);
    ch.print();
}
```

e)

```
#include <iostream>

class Parent
{
protected:
    int m_x;
public:
    Parent(int x): m_x(x)
    {
        std::cout << "Parent()\n";
    }
    ~Parent()
    {
        std::cout << "~Parent()\n";
    }

    void print() { std::cout << "Parent: " << m_x << '\n';  }
};

class Child: public Parent
{
public:
    Child(int y):  Parent(y)
    {
        std::cout << "Child()\n";
    }
    ~Child()
    {
        std::cout << "~Child()\n";
    }

    void print() { std::cout << "Child: " << m_x << '\n'; }
};

class D2 : public Child
{
public:
    D2(int z): Child(z)
    {
        std::cout << "D2()\n";
```

```
    }
    ~D2()
    {
        std::cout << "~D2()\n";
    }

        // Обратите внимание, здесь нет метода print()
};

int main()
{
    D2 d(7);
    d.print();
}
```