```cpp
#include <iostream>
#include <set>
#include <map>

using namespace std;

/*********************************/
/*           APARTADO A          */
/*********************************/

class A{

    class C;     //Declaración adelantada

    public:

        typedef set<C*> objetosC;
        void rel1(C& c_);
        const objetosC rel1() const noexcept;

    private:

        objetosC c;      //rel1
};

class C{

    class D;     //Declaración adenlantada

    public:
        C();
        void rel4(D& d_);
        const D& rel4() const noexcept;

    private:
        D* d;        //rel4
};

class D{

    class B;     //Declaración adelantada

    public:
        typedef set<C*> objetosC;
        typedef set<B*> objetosB;

        void rel4(C& c_);
        void rel3(B& b_);

        const objetosC& rel4() const noexcept;
        const objetosB& rel3() const noexcept;
        const A& rel2() const noexcept;
        int cualificador() const noexcept;         //Devuelve d1

    private:
        objetosC c;      //rel4
```

```
57          objetosB b;       //rel3
58          A* a;             //rel2
59          int d1;           //cualificador de d
60  };
61
62  class B{
63
64      public:
65          typedef map<int,D*> objetosD;
66          void rel3(D& d_);
67          const objetosD& rel3() const noexcept;
68
69      private:
70          objetosD d;  //rel3
71  };
72
73
74  /*********************************/
75  /*           APARTADO B          */
76  /*********************************/
77
78  C::C():d(nullptr){}
79
80  /*********************************/
81  /*           APARTADO C          */
82  /*********************************/
83  class A{
84
85      public:
86
87          typedef map<C*, X> objetosC;      //Siendo un X un tipo de dato cualesquiera (int, string...)
88          void rel1(C& c_, X x_);
89          const objetosC& rel1() const noexcept;
90
91      private:
92
93          objetosC c;       //rel1
94  };
95
96
97
```