## ¿Cómo gestionar la dependencia circular en C++?

La dependencia circular ocurre cuando dos clases se han de incluir la una a la otra. Para que el compilador no emita un error la forma de implementar las clases es la siguiente:

## A.h

```
#ifndef _A_H_
#define _A_H_
#include <string>

class B;  // Declaración forward de la clase B

class A {
public:
  B getBfromA();
  std::string who_are_you();
};

#endif
```

## B.h

```
#ifndef _B_H_
#define _B_H_
#include <string>

class A;  // Declaración forward de la clase A

class B {
public:
  A getAfromB();
  std::string who_are_you();
};

#endif
```

## A.cpp

```
#include "A.h"
#include "B.h"

B A::getBfromA() {
  return B();
}

std::string A::who_are_you() {
  return "I am A";
}
```

## B.cpp

```
#include "B.h"
```

```
#include "A.h"

A B::getAfromB() {
  return A();
}

std::string B::who_are_you() {
  return "I am B";
}
```

## main.cpp

```
#include "A.h"
#include "B.h"

#include <iostream>

int main() {
  A a;
  B b;

  A ab = b.getAfromB();
  B ba = a.getBfromA();

  std::cout << "ab is: " << ab.who_are_you() << endl;
  std::cout << "ba is: " << ba.who_are_you() << endl;
  return 0;
}
```

Cuando se utiliza dependencia circular hay ciertas prácticas que no se pueden hacer. Supongamos el código **A.cpp** donde la clase B está declarada pero no definida. Veamos que se puede hacer:

- Utilizar B como puntero o referencia, por ejemplo:

  ```
  class A {
      B *pt;
      B &pt;
  };
  ```

- Declarar funciones o métodos cuyos parámetros o valores devueltos sean clases incompletas.

  ```
  class A {
    void f1(B);
    B    f2();
  };
  ```

- Definir funciones o métodos que acepten/retornen punteros/referencias a los tipos incompletos (pero sin usar sus miembros)

  ```
  class A {
    void f3(B*, B&) {}
    B&   f4()       {}
    B*   f5()       {}
  };
  ```

Lo que no se puede hacer:

- Utilizar B como clase base:

```
class A : B {} // error de compilación
```

- Usar B para declarar un miembro.

```
class A {
    B m; // error de compilación
};
```

- Definir funciones o métodos usando el tipo B.

```
class A {
  void f1(B b) {} // error de compilación
  B    f2()    {} // error de compilación
};
```

- Usar sus métodos o campos, en general intentar referenciar una variable con un tipo incompleto.

```
class A {
    B *m;
    void method()
    {
        m->someMethod();      // error de compilación
        int i = m->someField; // error de compilación
    }
};
```