

FUNCIONES Y CLASES AMIGAS Y RESERVA DE MEMORIA



Funciones y clases amigas y reserva de memoria.

Introducción.

- Ruptura de la “ocultación de información” con funciones y clases amigas (friend)
- Motivos:
 - No hay otra solución (sobrecarga de algunos operadores)
 - Por cuestiones de claridad.
 - Por cuestiones de rendimiento del programa.

Funciones y clases amigas y reserva de memoria.

Declaración de amistad (I)

- Declaración de una *clase/función A* amiga de otra *clase B*:
 - Dentro de la declaración de la clase B
 - Anteponiendo la palabra **friend** a la función o clase:
 - **friend class** *nombrecloseA*;
 - **friend** *tipo nombrefuncionA* (*parámetros*);

Funciones y clases amigas y reserva de memoria.

Declaración de amistad (II)

Ejemplo 4.1

```
1 class TCoordenada {  
2     friend class TLinea;  
3     friend float Distancia(TCoordenada, TCoordenada);  
4  
5     public:  
6         TCoordenada();  
7         TCoordenada(int, int, int);  
8         TCoordenada(const TCoordenada &);  
9         ~TCoordenada();  
10  
11         void setX(int);  
12         void setY(int);  
13         void setZ(int);
```

Declaración de clase amiga

Declaración de función amiga

```
14  
15     int getX(void);  
16     int getY(void);  
17     int getZ(void);  
18  
19     void Imprimir(void);  
20  
21     private:  
22         int x, y, z;  
23 };
```

Funciones y clases amigas y reserva de memoria.

Declaración de amistad (III)

- Características básicas:
 - Funciones amigas: **NO** son funciones miembro de una clase.
 - Clase A amiga de Clase B → Todas funciones miembro Clase A amigas Clase B
 - Si sólo unas funciones de una clase son amigas de otra, se indican a través de '::'
 - Declaración de amistad **en cualquier parte** de la declaración de la clase (se suelen colocar al principio).
 - La amistad **NO** se **toma**, se **otorga**.
 - **NO** es una propiedad simétrica ni transitiva.

Funciones y clases amigas y reserva de memoria. Declaración de amistad (IV)

- Implementación de la función Distancia() en tcoordenada.cc

Ejemplo 4.2

```
1 #include <cmath>
2
using namespace std;

float
Distancia(TCoordenada a, TCoordenada b) {
    float d;

    d = pow((a.x - b.x), 2);
    d += pow((a.y - b.y), 2);
    d += pow((a.z - b.z), 2);

    return sqrt(d);
}
```

Salida ejemplo 4.2

```
tcoordenada.cc: In function 'float Distancia(TCoordenada, TCoordenada)':
tcoordenada.cc:72: call of overloaded 'pow(int, int)' is ambiguous
/usr/include/bits/mathcalls.h:154: candidates are: double pow(double,
double)
/usr/include/c++/3.2.2/cmath:427:      long double std::pow(long
double, int)
/usr/include/c++/3.2.2/cmath:423:      float std::pow(float, int)
/usr/include/c++/3.2.2/cmath:419:      double std::pow(double, int)
/usr/include/c++/3.2.2/cmath:410:      long double std::pow(long
double, long double)
/usr/include/c++/3.2.2/cmath:401:      float std::pow(float, float)
tcoordenada.cc:73: call of overloaded 'pow(int, int)' is ambiguous
/usr/include/bits/mathcalls.h:154: candidates are: double pow(double,
double)
/usr/include/c++/3.2.2/cmath:427:      long double std::pow(long
double, int)
/usr/include/c++/3.2.2/cmath:423:      float std::pow(float, int)
/usr/include/c++/3.2.2/cmath:419:      double std::pow(double, int)
/usr/include/c++/3.2.2/cmath:410:      long double std::pow(long
double, long double)
/usr/include/c++/3.2.2/cmath:401:      float std::pow(float, float)
tcoordenada.cc:74: call of overloaded 'pow(int, int)' is ambiguous
/usr/include/bits/mathcalls.h:154: candidates are: double pow(double,
double)
/usr/include/c++/3.2.2/cmath:427:      long double std::pow(long
double, int)
/usr/include/c++/3.2.2/cmath:423:      float std::pow(float, int)
/usr/include/c++/3.2.2/cmath:419:      double std::pow(double, int)
/usr/include/c++/3.2.2/cmath:410:      long double std::pow(long
double, long double)
/usr/include/c++/3.2.2/cmath:401:      float std::pow(float, float)
```

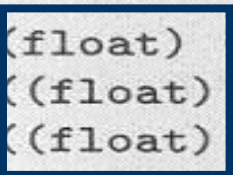

Funciones y clases amigas y reserva de memoria.

Declaración de amistad (V)

- Corregido el error:

Ejemplo 4.3

```
1 #include <cmath>
2
3 using namespace std;
4
5 float
6 Distancia(TCoordenada a, TCoordenada b) {
7     float d;
8
9     d = pow((float) (a.x - b.x), 2);
10    d += pow((float) (a.y - b.y), 2);
11    d += pow((float) (a.z - b.z), 2);
12
13
14    return sqrt(d);
15 }
```



Funciones y clases amigas y reserva de memoria. Declaración de amistad (VI)

- Declaración de la clase **TLinea** en fichero *tlinea.h*

Ejemplo 4.4

```
1 #include "tcoordenada.h"
2
3 class TLinea {
4     public:
5         TLinea();
6         TLinea(const TCoordenada &, const TCoordenada &);
7         TLinea(const TLinea &);
8         ~TLinea();
9         float Longitud(void);
10
11     private:
12         TCoordenada p1, p2;
13 };
```

Necesario

Funciones y clases amigas y reserva de memoria. Declaración de amistad (VII)

- Implementación de la clase **TLinea** en *tlinea.cc*

Ejemplo 4.5

```
1 #include "tlinea.h"
2
3 TLinea::TLinea() {
4     p1.x = 0;
5     p1.y = 0;
6     p1.z = 0;
7
8     p2.x = 0;
9     p2.y = 0;
10    p2.z = 0;
11 }
12
13 TLinea::TLinea(const TCoordenada & a, const TCoordenada & b) {
14     p1.x = a.x;
15     p1.y = a.y;
16     p1.z = a.z;
17
18     p2.x = b.x;
19     p2.y = b.y;
20     p2.z = b.z;
21 }
```

Todas las funciones son amigas de la clase TCoordenada.

```
23 TLinea::TLinea(const TLinea & l) {
24     p1.x = l.p1.x;
25     p1.y = l.p1.y;
26     p1.z = l.p1.z;
27
28     p2.x = l.p2.x;
29     p2.y = l.p2.y;
30     p2.z = l.p2.z;
31 }
32
33 TLinea::~TLinea() {
34     p1.x = 0;
35     p1.y = 0;
36     p1.z = 0;
37
38     p2.x = 0;
39     p2.y = 0;
40     p2.z = 0;
41 }
42
43 float
44 TLinea::Longitud(void) {
45     return Distancia(p1, p2);
46 }
```

Funciones y clases amigas y reserva de memoria. Declaración de amistad (VIII)

- Un posible **programa principal** de uso de la clase TLinea se guardaría en *main.cc*

Ejemplo 4.6

```
1  #include <iostream>
2
3  using namespace std;
4
5  #include "tcoordenada.h"
6  #include "tlinea.h"
7
8  int
9  main(void)
10 {
11     TCoordenada p1;
12     TCoordenada p2(1, 2, 3);
13     TCoordenada p3(p2);
14
15     TLinea l1, l2(p1, p2);
16
17     cout << Distancia(p1, p2) << endl;
18
19     return 0;
20 }
```

Funciones y clases amigas y reserva de memoria.

Declaración de amistad (IX)

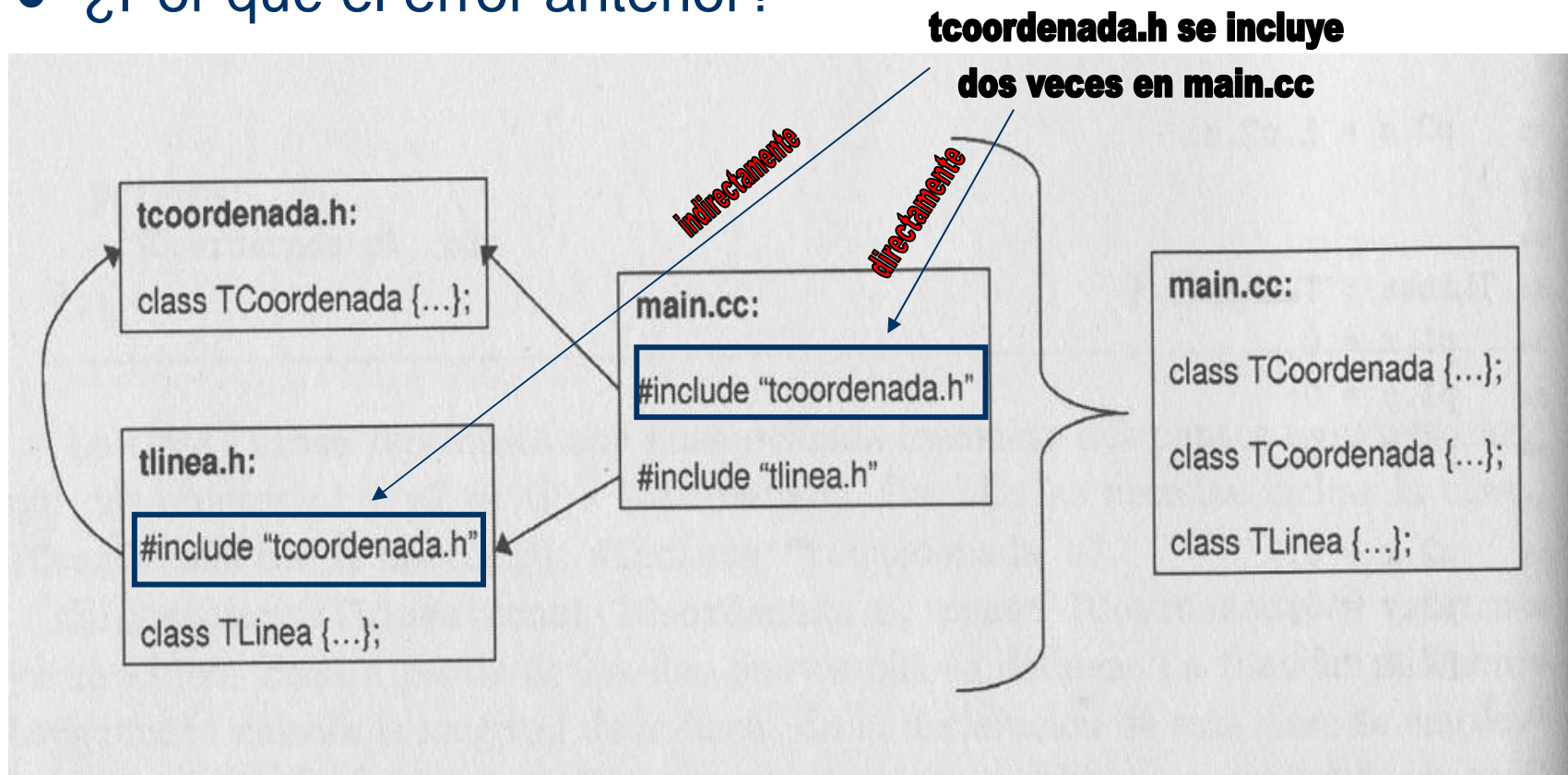
- Al compilar main.cc sale el siguiente error:

Salida ejemplo 4.6

```
1 In file included from tlinea.h:1,  
2 from main.cc:6:  
3 tcoordenada.h:2: redefinition of 'class TCoordenada'  
4 tcoordenada.h:2: previous definition of 'class TCoordenada'  
5 tcoordenada.h:4: warning: 'float Distancia(TCoordenada, TCoordenada)'  
6 is already a friend of class 'TCoordenada'  
7 tcoordenada.h:4: warning: previous friend declaration of 'float  
8 Distancia(TCoordenada, TCoordenada)'
```

Funciones y clases amigas y reserva de memoria. Declaración de amistad (X)

- ¿Por qué el error anterior?



Funciones y clases amigas y reserva de memoria.

Guardas de inclusión (I)

- ¿Cómo solucionar el problema de la doble inclusión?
→ con las instrucciones del preprocesador...

– #define <simbolo/macro>	...
– #undef <simbolo/macro>	- #if define (<simbolo>) y
– #if y #endif ,	#if !defined (<simbolo/macro>)
– #ifdef <simbolo>	respectivamente
#ifndef <simbolo>	- #elif y #else
equivalen a ...	

...se crean **guardas de inclusión**

Funciones y clases amigas y reserva de memoria.

Guardas de inclusión (II)

- En el fichero de definición de la clase (.h):

```
#ifndef _NombreClase_H_  
#define _NombreClase_H_  
<se declara la clase>  
  
...  
#endif
```

Funciones y clases amigas y reserva de memoria. Guardas de inclusión (III)

- tcoordenada.h y tlinea.h quedarían como sigue:

Ejemplo 4.7

```
1 #ifndef __TCOORDENADA__
2 #define __TCOORDENADA__
3
4 class TCoordenada {
5     friend class TLinea;
6     friend float Distancia(TCoordenada, TCoordenada);
7
8     public:
9         TCoordenada();
10        TCoordenada(int, int, int);
11        TCoordenada(const TCoordenada &);
12
13        void setX(int);
14        void setY(int);
15        void setZ(int);
16
17        int getX(void);
18        int getY(void);
19        int getZ(void);
20
21        void Imprimir(void);
22
23     private:
24         int x, y, z;
25 };
26 #endif
```

Guardas de inclusión

Ejemplo 4.8

```
1 #ifndef __TLINEA__
2 #define __TLINEA__
3
4 #include "tcoordenada.h"
5
6 class TLinea {
7     public:
8         TLinea();
9         TLinea(const TCoordenada &, const TCoordenada &);
10        TLinea(const TLinea &);
11        ~TLinea();
12        float Longitud(void);
13
14     private:
15         TCoordenada p1, p2;
16 };
17
18 #endif
```

Funciones y clases amigas y reserva de memoria. **Administración de memoria dinámica (I)**

- Con **new** (reserva) y **delete** (liberación)
- Reserva (new)
 - //Constructores por defecto
 - int *ptrInt = new int;
 - TCoordenada *ptrCoor = new TCoordenada
 - //Otros constructores
 - int *ptrInt = new int (123);
 - TCoordenada *ptrCoor = new TCoordenada(7,5,3)
- Si falla la reserva → new devuelve **NULL**.
Comprobar siempre después si ha sido así.

Funciones y clases amigas y reserva de memoria. **Administración de memoria dinámica (II)**

```
1  int *ptrInt = new int(123);
2  if(ptrInt == NULL)
3      cout << "Error" << endl;
4
5  TCoordenada *ptrCoor = new TCoordenada(7, 5, 3);
6  // Otra forma de comprobar ptrCoor == NULL
7  if(!ptrCoor)
8      cout << "Error" << endl;
```

- Liberación (delete)
 - El programador ha de liberar la memoria reservada
 - ¿Cómo? **delete** <puntero>
 - Se invoca al **destructor** y se **libera** la memoria reservada
 - Antes de liberar: Comprobar que el puntero **NO** sea **NULL**
 - Después de liberar: <puntero> = **NULL**;

Funciones y clases amigas y reserva de memoria. **Administración de memoria dinámica (III)**

- **Importante:** Muchos fallos (**segmentation fault**) con puntero se deben a no atender a estas comprobaciones.

```
1  int *ptrInt = new int(123);
2  TCoordenada *ptrCoor = new TCoordenada(7, 5, 3);
3
4  /*
5
6      Algo de código
7  */
8
9  // Se libera la memoria que ocupa
10 if(ptrInt) {
11     delete ptrInt;
12     ptrInt = NULL;
13 }
14
15 // Se libera la memoria que ocupa y se invoca
16 // al destructor de la clase TCoordenada
17 if(ptrCoor) {
18     delete ptrCoor;
19     ptrCoor = NULL;
20 }
```


Funciones y clases amigas y reserva de memoria.

Administración de memoria dinámica y array de objetos (I)

- Eliminación array de objetos:
 - Estático → se invoca al destructor implícitamente para cada elemento:

```
1  void
2  UnaFuncion(void) {
3      TCoordenada a[10];
4
5      /*
6       Algo de código
7      */
8
9      // Se invoca 10 veces al destructor de TCoordenada
10     // para los 10 objetos de a[10]
11 }
```

Funciones y clases amigas y reserva de memoria.

Administración de memoria dinámica y array de objetos (II)

- Dinamico (new)→ se ha de invocar al destructor explícitamente:

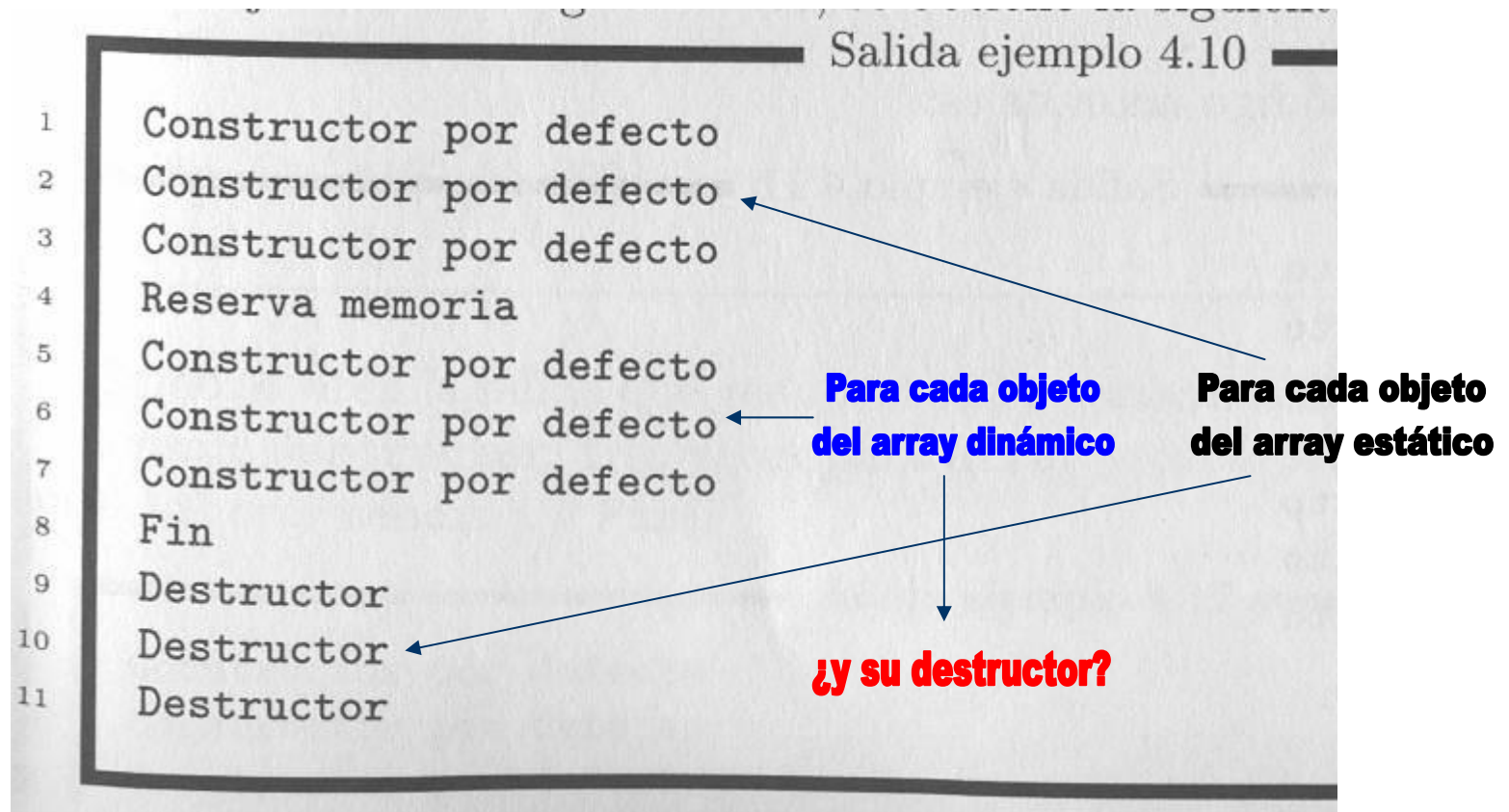
_____ Ejemplo 4.9 _____

```
1 TCoordenada::TCoordenada() {  
2     cout << "Constructor por defecto" << endl;  
3     x = y = z = 0;  
4 }  
5  
6 TCoordenada::~~TCoordenada() {  
7     cout << "Destructor" << endl;  
8 }
```

_____ Ejemplo 4.10 _____

```
1 #include <iostream>  
2  
3 using namespace std;  
4  
5 #include "tcoordenada.h"  
6  
7 int  
8 main(void)  
9 {  
10     TCoordenada array[3];  
11     TCoordenada *ptr;  
12  
13     cout << "Reserva memoria" << endl;  
14     ptr = new TCoordenada[3];  
15     if(ptr == NULL)  
16         return;  
17  
18     cout << "Fin" << endl;  
19 }
```

Funciones y clases amigas y reserva de memoria. Administración de memoria dinámica y array de objetos (III)



Funciones y clases amigas y reserva de memoria. Administración de memoria dinámica y array de objetos (IV)

- El programador ha de liberar esa memoria:

Ejemplo 4.11

```
1 #include <iostream>
2
3 using namespace std;
4
5 #include "tcoordenada.h"
6
7 int
8 main(void)
9 {
10     TCoordenada array[3];
11     TCoordenada *ptr;
12
13     cout << "Reserva memoria" << endl;
14     ptr = new TCoordenada[3];
15     if(ptr == NULL)
16         return;
17
18     cout << "Libera memoria" << endl;
19     delete ptr;
20     ptr = NULL;
21
22     cout << "Fin" << endl;
23 }
```

Funciones y clases amigas y reserva de memoria. Administración de memoria dinámica y array de objetos (V)

- Y la salida sería:

Salida ejemplo 4.11 ■

```
1 Constructor por defecto
2 Constructor por defecto
3 Constructor por defecto
4 Reserva memoria
5 Constructor por defecto
6 Constructor por defecto
7 Constructor por defecto
8 Libera memoria
9 Destructor
10 Fin
11 Destructor
12 Destructor
13 Destructor
```

¿Sólo una vez?

Funciones y clases amigas y reserva de memoria.

Administración de memoria dinámica y array de objetos (VI)

- Sólo libera el primer objeto del array
- Para liberar todos: **delete[] puntero**

Ejemplo 4.12

```
1  #include <iostream>
2
3  using namespace std;
4
5  #include "tcoordenada.h"
6
7  int
8  main(void)
9  {
10     TCoordenada array[3];
11     TCoordenada *ptr;
12
13     cout << "Reserva memoria" << endl;
14     ptr = new TCoordenada[3];
15     if(ptr == NULL)
16         return;
17
18     cout << "Libera memoria" << endl;
19     delete [] ptr;
20     ptr = NULL;
21
22     cout << "Fin" << endl;
23 }
```

Funciones y clases amigas y reserva de memoria. **Administración de memoria dinámica y array de objetos (VII)**

- Ejemplos de uso correcto e incorrecto de delete:

```
1 void
2 UnaFuncion(void) {
3     TCoordenada *a = new TCoordenada;
4     TCoordenada *b = new TCoordenada[5];
5     TCoordenada *c = new TCoordenada;
6     TCoordenada *d = new TCoordenada[5];
7
8     // Puntero a un solo objeto
9     delete a;          // Correcto
10    // Puntero a un array de objetos
11    delete [] b;        // Correcto
12    // Puntero a un solo objeto
13    delete [] c;        // Incorrecto
14    // Puntero a un array de objetos
15    delete d;           // Incorrecto
16 }
```

Funciones y clases amigas y reserva de memoria.

Compilación condicional (I)

- Uso de instrucciones de preprocesador para evitar compilar ciertas partes de código
- Útil en los procesos de depuración de errores.
- Dos formas:
 - Definiendo constantes simbólicas:

```
#define simbolo
#ifdef simbolo
    cout<<"Mensaje"<<endl;
#endif
```

Funciones y clases amigas y reserva de memoria.

Compilación condicional (II)

- Directivas condicionales:

```
#define DEPURA1
#define DEPURA2
#define DEPURA3
...
#ifdef DEPURA1
    cout << "Un mensaje" << endl;
#elif defined (DEPURA2)
    cout << "Otro mensaje" << endl;
#elif defined (DEPURA3)
    cout << "El último mensaje" << endl;
#else //No hacemos nada
#endif
```

Funciones y clases amigas y reserva de memoria. Compilación condicional (III)

Ejemplo 4.13

```
1 #include "tcoordenada.h"
```

```
2
```

```
3 #define DEPURACION
```

```
4
```

```
5 TCoordenada::TCoordenada() {
```

```
6
```

```
7 #ifdef DEPURACION
```

```
8     cout << "Constructor por defecto" << endl;
```

```
9 #endif
```

```
10
```

```
11     x = y = z = 0;
```

```
12 }
```

```
13
```

```
14 TCoordenada::~TCoordenada() {
```

```
15 #ifdef DEPURACION
```

```
16     cout << "Destructor" << endl;
```

```
17 #endif
```

```
18 }
```

Definimos el símbolo

**Se compilan y ejecutan
sólo si el símbolo existe**

Funciones y clases amigas y reserva de memoria. Compilación condicional (IV)

- Mediante la directiva **#warning** o **#error**

Ejemplo 4.14

```
1 #include "tcoordenada.h"
2
3 TCoordenada::TCoordenada() {
4 #warning Eliminar esta instrucción
5     cout << "Constructor por defecto" << endl;
6
7     x = y = z = 0;
8 }
9
10 TCoordenada::~~TCoordenada() {
11 #warning Eliminar esta instrucción
12     cout << "Destructor" << endl;
13 }
```

Salida ejemplo 4.14

```
1 tcoordenada.cc:4:2: warning: #warning Eliminar esta instrucción
2 tcoordenada.cc:11:2: warning: #warning Eliminar esta instrucción
```