0.1. Destructores

En ocasiones la creación de un objeto emplea un recurso, generalmente la reserva de memoria dinámica. El **destructor** de una clase es una función que es llamada automáticamente cuando el objeto se destruye con el fin de liberar ese recurso cuando el objeto deje de existir (las reglas que determinan cuando un objeto de una clase se destruye son las mismas que para cualquier variable: salir del ámbito si es automática, llamada a `delete` si es dinámica, final del programa si es global o estática). El destructor tiene el mismo nombre que el constructor pero precedido de una vírgula (~) y no tiene argumentos.

```
#include<iostream>
using namespace std;
struct datos{
  int x,y,z;
};

class estructura_dinamica{
  datos *p;
public:
  estructura_dinamica(int a, int b, int c){
    //asignacion de memoria
    p=new datos;
    //utilizacion de la memoria asignada
    p->x=a;
    p->y=b;
    p->z=c;
    cout<<"Objeto creado"<<endl;
  }

  //destructor
  ~estructura_dinamica(){
    //se libera la memoria asignada a p
    delete p;
    cout<<"Objeto destruido"<<endl;
  }

  //funcion publica, accede al atributo p
  //y mediante el a los de la estructura
  int suma()
  {
    //lo mismo que si se hubiese escrito
    //(*p).x+(*p).y+(*p).z
    return p->x+p->y+p->z;
  }
};
```

```
void f(int n){
  int i;
  //llamada al constructor
  estructura_dinamica otra(1,2,3);
  //se usa la funcion publica
  cout<<n+otra.suma()<<endl;
  //al llegar al final del bloque
  //se destruye, llamada al destructor
  //automaticamente
}

int main()
{
  cout<<"primera llamada a f"<<endl;
  f(2);
  cout<<"segunda llamada a f"<<endl;
  f(3);
}
```

Salida de este programa:

```
primera llamada a f
Objeto creado
8
Objeto destruido
segunda llamada a f
Objeto creado
9
Objeto destruido
```

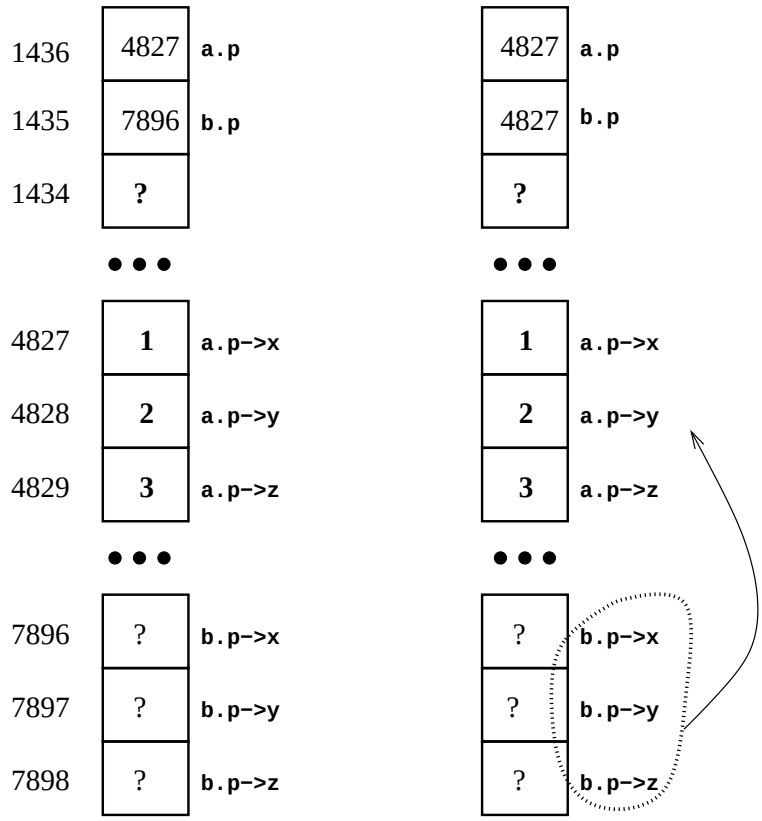
0.2. Asignación y paso por valor de objetos como argumentos

Cada variable de una clase es una representación de un tipo abstracto de datos en una tupla de valores. Esto es, el contenido de la variable está compuesto por los valores de los datos definidos dentro de la clase. Por tanto, la asignación consiste en la copia de esos valores.
En los ejemplos vistos hasta ahora, los atributos de las clases pertenecían a tipos predefinidos en C++. Si se asigna un objeto de las clases vistas a otro objeto de la misma clase, esta operación copiará los valores de los datos miembros del objeto a la derecha de la operación de asignación en sus análogos del objeto a la izquierda del operador de asignación, generándose un objeto correcto. Sin embargo, este procedimiento aplicado a la clase `estructura_dinamica` definida en la sección anterior no produce el resultado apetecido. La razón es la siguiente,

el dato miembro de esa clase es un puntero a una estructura de tipo `datos`. Si se utiliza el operador de asignación predefinido, se copiará el valor del dato miembro de una clase (`A`) en la otra. El valor de ese dato miembro es un puntero, por lo tanto lo que se copiará será la dirección de memoria en donde se almacena la estructura de tipo `datos`, no los campos de esa estructura en una estructura apuntada por el campo `A`. En la figura 0.1 se puede ver una representación del contenido de la memoria del computador en este caso, obsérvese como el estado final de la memoria no es el deseado, el contenido de la memoria asociada a `y` no se replica en el contenido de `x`. Únicamente se copia el valor de `A`, que es un puntero, pero no el contenido de la variable apuntada por el, una estructura con tres campos de tipo entero. Esta circunstancia puede producir (entre otros problemas) que el programa se interrumpa, es decir, aun compilando sin errores, estos pueden producirse en tiempo de ejecución. Por ejemplo, en el caso de la definición anterior, el siguiente `main()` produce un error en tiempo de ejecución, al llamar dos veces consecutivamente a `delete` para una misma dirección de memoria.

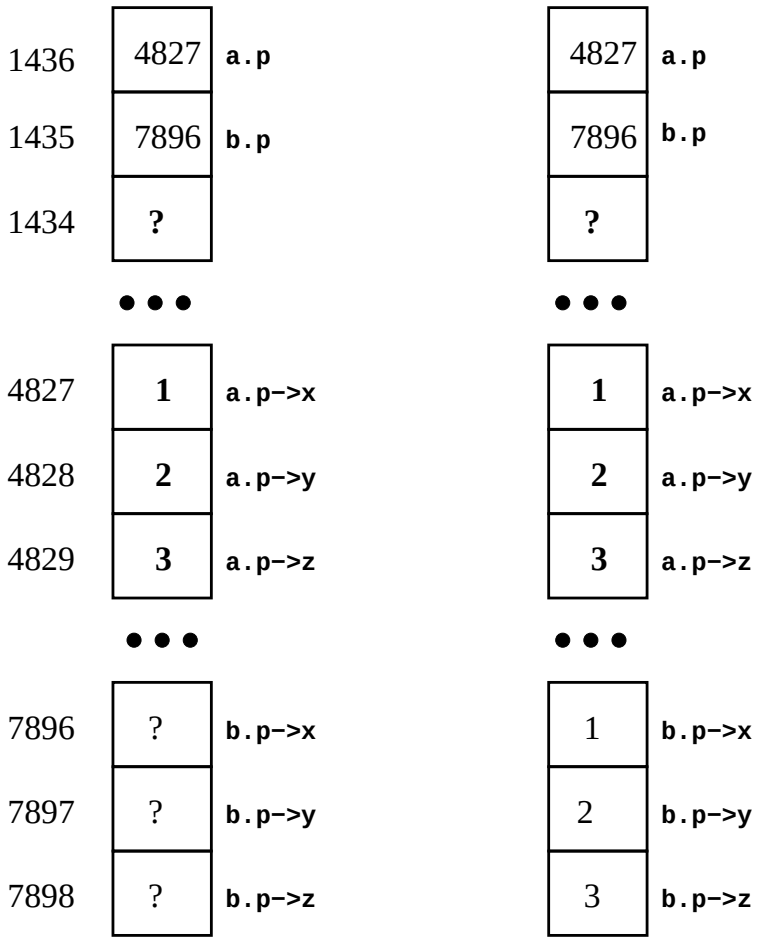
```
int main()
{
estructura_dinamica a(1,2,3),b(4,5,6);//declaracion de variables
b=a;                                  //provocará un error
//fin del programa, se llamará al destructor de a y de b
//la segunda llamada provoca un error en tiempo de ejecución
}
```

El comportamiento adecuado del operador de asignación se puede observar en la figura 0.2, obsérvese como en este caso si se copian los campos de la estructura apuntada.
Una situación similar se produce cuando se utiliza el paso por valor de variables como argumentos de funciones. En ese caso, cada parámetro real de la llamada se copia en el parámetro formal correspondiente en la definición de la función, además previamente se ha de crear un nuevo objeto, en el que se copiará el valor del parámetro real, por este motivo, si se utiliza memoria dinámica, se empleará el operador `new`. Para solucionar este problema, se sobrecarga el constructor, definiendo una versión que recibe como argumento una referencia constante a una variable del mismo tipo. Por ejemplo, el constructor de copia de una clase `T` tendrá un prototipo `T(const T&x);`. En el caso del paso por referencia no existe ningún problema, no se copia el valor de la variable sino que se usa un nombre alternativo (sinónimo) para él. En el siguiente ejemplo se muestra como redefinir el operador de asignación de forma adecuada y se escribe el constructor de copia. Nótese que en el operador de asignación no se utiliza `new` al contrario que en el caso del constructor de copia, esto es debido a que el objeto a la izquierda del '=' ya ha reservado memoria utilizando `new` cuando fue declarado (llamando al constructor correspondiente). Adicionalmente se escribe un constructor por defecto que se ejecuta cuando se declaran objetos de la clase `estructura_dinamica` sin escribir a continuación tres valores entre paréntesis.




```
estructura_automatica a(1,2,3);      b=a;
estructura_automatica b;
```

Figura 0.1: Contenido de la memoria tras la ejecución de las instrucciones que figuran al pie de cada mapa, ilustrando el comportamiento del operador de asignación si **NO** se redefine adecuadamente cuando se utiliza memoria dinámica.




```
estructura_automatica a(1,2,3);      b=a;
estructura_automatica b;
```

Figura 0.2: Contenido de la memoria tras la ejecución de las instrucciones que figuran al pie de cada mapa, ilustrando el comportamiento del operador de asignación si es redefinido adecuadamente.

```
#include<iostream>
using namespace std;
struct datos{
  int x,y,z;
};
class estructura_dinamica{
  datos *p;
public:
  estructura_dinamica(int a, int b, int c)
  {
    p=new datos;
    p->x=a;
    p->y=b;
    p->z=c;
    cout<<"Constructor a partir de enteros"<<endl;
  }

  ~estructura_dinamica()
  {
    delete p;
    cout<<"Destructor"<<endl;
  }

  estructura_dinamica()
  {
    //constructor por defecto
    p=new datos;
    p->x=0;
    p->y=0;
    p->z=0;
    cout<<"Constructor por defecto"<<endl;
  }


  //constructor de copia
  estructura_dinamica(const estructura_dinamica &a)
  {
    p=new datos;
    p->x=a.p->x;
    p->y=a.p->y;
    p->z=a.p->z;
    cout<<"Constructor de copia"<<endl;
  }

  //operador de asignacion, no se crea un
  //objeto nuevo
```

```
  void operator=(const estructura_dinamica &a)
  {
    p->x=a.p->x;
    p->y=a.p->y;
    p->z=a.p->z;
    cout<<"Operador de asignacion"<<endl;
  }

  int suma()
  {
    return p->x+p->y+p->z;
  }
};

//paso por valor, se utiliza el constructor de copia
void f_valor(estructura_dinamica a)
{
  cout<<"Al acabar esta funcion se llamara al destructor"<<endl;
  cout<<a.suma()<<endl;
}

//paso por referencia, no se utiliza ningun constructor
void f_referencia(estructura_dinamica &a)
{
  cout<<"Al acabar esta funcion NO se llamara al destructor"<<endl;
  cout<<a.suma()<<endl;
}

int main()
{
  //se llama al constructor a partir de enteros
  //y al constructor por defecto
  estructura_dinamica a(1,2,3),b;
  //se llama al operador de asignacion
  b=a;
  //se utiliza el paso por valor, por lo tanto se
  //llama al constructor de copia
  f_valor(b);
  //aqui no
  f_referencia(b);
  cout<<"se llega al final, se destruyen los dos objetos que quedan"<<endl;
}
```

Este programa muestra por pantalla lo siguiente:

```
Constructor a partir de enteros
Constructor por defecto
```

8

```
Operador de asignacion
Constructor de copia
Al acabar esta funcion se llamara al destructor
6
Destructor
Al acabar esta funcion NO se llamara al destructor
6
se llega al final, se destruyen los dos objetos que quedan
Destructor
Destructor
```