1. Punteros a objetos
2. Vectores dinámicos de objetos
3. Uso de objetos dinámicos
4. Atributos dinámicos
5. Creación de objetos con atributos dinámicos
6. Destrucción de objetos con atributos dinámicos
7. Operador de asignación para objetos con atributos dinámicos

Punteros a objetos

En C++ podemos definir punteros a objetos, ya que las clases son tipos, y podemos declarar punteros de cualquier tipo.

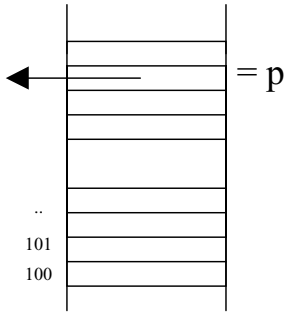
```
int  *p1;
char *c ;
fecha * f;
```

Por tanto, para la clase Punto, podemos tener:

```
Punto  *p1;
```

Esta declaración, no llama al constructor ya que es un puntero, no un Punto.

```
class Punto
{
  private:
     int coorx ;
     int coory ;
  public:
    ...
} ;
```

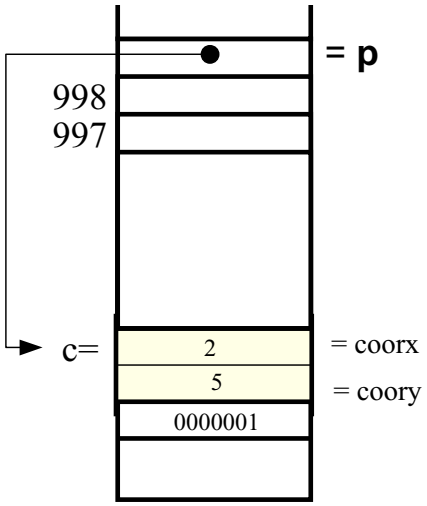


Punteros a objetos

Para trabajar con el puntero, podemos asignarle la dirección de memoria de un objeto de tipo Punto.

```
class Punto
 {
   private:
     int coorx ;
     int coory ;
   public:
      Punto(int a, int b);
      int acc_x( );
      int acc_y( );
      void mu_x( int a);
      void mu_y( int a);
      void visualizar( );
} ;
```

```
Punto c(2,3);
Punto  *p1;
p1 = &c;
p1->visualizar();
(*p1).visualizar();
p1->mu_y(8);
```

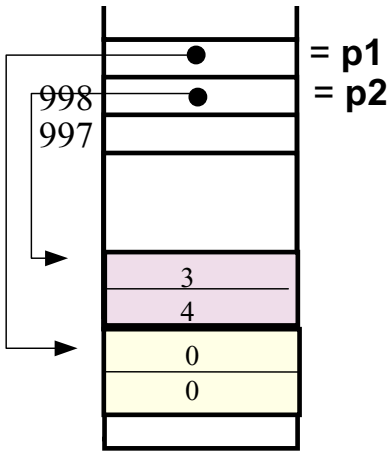


Punteros a objetos

También podemos usar otras facilidades de las variables dinámicas como son los operadores **new** y **delete**.

```
Punto  *p1, *p2;
p1 = new Punto();
p2 = new Punto(3,4);
delete p2;
```

Constructor que queremos que se ejecute



El operador **new** sigue los siguientes pasos:

1. Crea espacio para el objeto.
2. Llama al constructor especificado.

El operador **delete** realiza los pasos inversos:

1. Llama al destructor.
2. Libera la memoria ocupada por el objeto.

Vectores dinámicos de objetos

También podemos declarar vectores dinámicos de objetos:

```
Punto  *p;
p = new Punto[100];
```

El operador **new**

1. Crea espacio para los 100 puntos.
2. Llama al constructor por defecto para cada objeto (0 a 99).

```
delete [] p ;
```

El operador **delete** realiza los pasos inversos:

1. Llama al destructor para cada punto (99 a 0).
2. Libera la memoria.

Uso de objetos dinámicos

Antes de usar el objeto, pasándole mensajes a través del puntero que lo apunta, se ha de crear el objeto con **new**.

Cuando ya no se necesita el objeto, se destruye con **delete**.

```
int main()
{
    Punto  *p;
   // CREAR -----
    p = new Punto();      // creado el objeto dinámico
   // USAR -----
    p->visualizar();      // uso del objeto mediante envío de mensajes
   // DESTRUIR -----
   delete p;              // liberar memoria del objeto
}
```

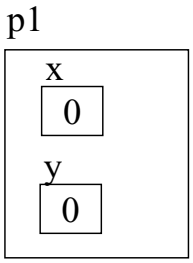
Atributos dinámicos

Tendremos atributos dinámicos, cuando en una clase se declara un atributo en forma de puntero.

Se trata de otra forma de guardar en memoria los atributos.

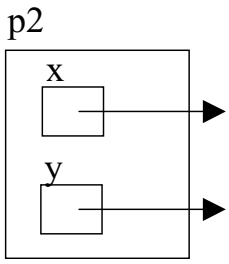
```
class Punto
{
  private:
    int x ;
    int y ;
  public:
    ...
} ;
```

```
Punto  p1;
```




```
class Punto
{
  private:
    int * x ;
    int * y ;
  public:
    ...
} ;
```

```
Punto  p2;
```



Ambas definiciones son equivalentes, pero con implementación distinta

Creación de objetos con atributos dinámicos

Cuando se crea un objeto con atributos dinámicos, se crean al mismo tiempo esos atributos.

¿Dónde?

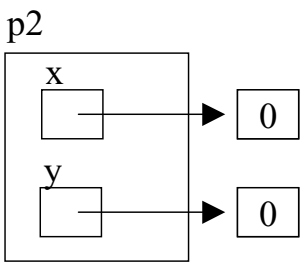
En el constructor.

```
class Punto
{
  private:
    int * x ;
    int * y ;
  public:
    Punto (int a=0, int b=0);
...
} ;
```

```
Punto  p2;
```

```
Punto :: Punto(int a, int b)
{
  x = a;
  y = b;
}
```

```
Punto :: Punto(int a, int b)
{
  x = new int;
  y = new int;
  *x = a;
  *y = b;
}
```



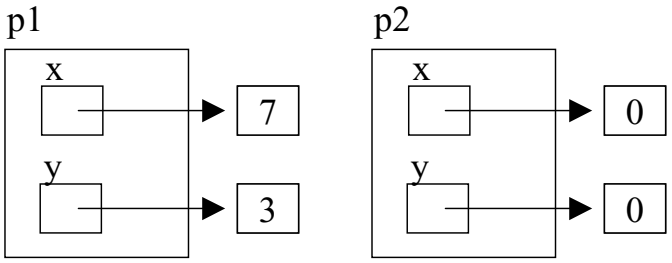
Creación de objetos con atributos dinámicos

Para cada objeto de la clase Punto que se crea, se crean a su vez sus atributos dinámicos.

```
class Punto
{
  private:
    int * x ;
    int * y ;
  public:
    Punto (int a=0, int b=0);
...
} ;
```

```
Punto :: Punto(int a, int b)
{
  x = new int;
  y = new int;
  *x = a;
  *y = b;
}
```

```
int main()
{
  Punto  p1(7,3);
  Punto p2();
  .....
}
```



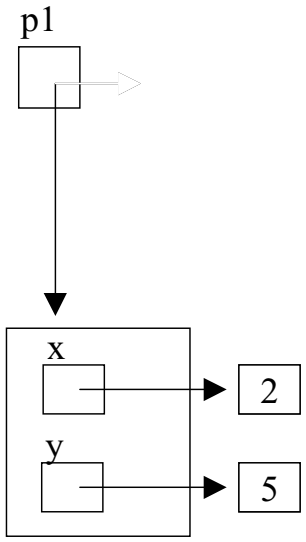
Creación de objetos con atributos dinámicos

¿Cuál es el efecto de la siguiente declaración?

```
class Punto
{
  private:
    int * x ;
    int * y ;
  public:
    Punto (int a=0, int b=0);
...
} ;
```

```
Punto :: Punto(int a, int b)
{
  x = new int;
  y = new int;
  *x = a;
  *y = b;
}
```

```
int main()
{
  Punto  *p1;
  p1 = new Punto(2,5);
  .....
}
```



Destrucción de objetos con atributos dinámicos

De la misma forma, cuando se destruye un objeto con atributos dinámicos, se destruyen al mismo tiempo esos atributos dinámicos.

¿Dónde?

En el destructor.

Es el que se ejecuta automáticamente cuando se destruyen objetos.

```
class Punto
{
  private:
     int * x ;
     int * y ;
  public:
     Punto (int a=0, int b=0);
     ~Punto( );
} ;
```

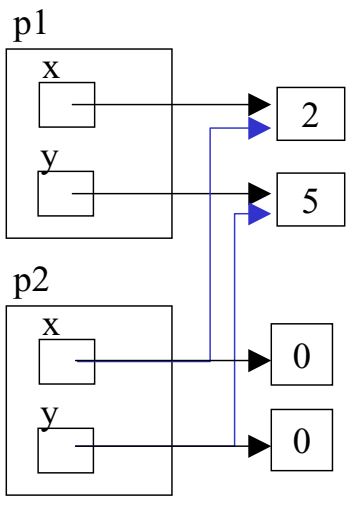
```
Punto :: Punto(int a, int b)
{
   x = new int;
   y = new int;
   *x = a;
   *y = b;
}
```

```
Punto :: ~Punto( )
{
   delete x;
   delete y;
}
```

El destructor destruye los atributos dinámicos

Operador de asignación para atributos dinámicos

Cuando se copian objetos , se ejecuta el operador de asignación de la clase. Si en la clase no está definido ningún operador de asignación, el compilador realiza una copia del objeto bit a bit.




```
int main()
{
   Punto  p1 (2,5), p2;
   p2 = p1;
}
```

El problema está en que ambos objetos comparten memoria.

Se produce una pérdida de memoria

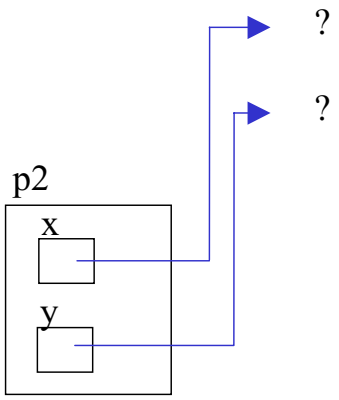
Se copian los atributos igualando los valores de los punteros (copia bit a bit).

Operador de asignación para atributos dinámicos

En ésta situación,

¿qué ocurre cuando se modifica el objeto p1?,

¿y cuándo se destruye el objeto p1?



Operador de asignación para atributos dinámicos

Cuando las clases definen atributos dinámicos, el operador de asignación se hace imprescindible. Se debe evitar la compartición de datos y la pérdida de memoria.

El operador de asignación debe copiar los contenidos de los punteros, no los punteros.

```
Punto&  Punto :: operator = ( Punto p)
{
  *x = *p.x;
  *y  = *p.y;
  return *this;
}
```

Como el objeto sobre el que se copia ya tiene creados los atributos dinámicos, simplemente se copia lo apuntado.

La clase Punto completa

El archivo Punto.h

```
class Punto
 {
    private:
      int *x ;
      int *y ;
    public:
       Punto(int a, int b);
      ~Punto( );
       Punto& operator= (Punto p);
       int acc_x( );
       int acc_y( );
       void mu_x( int a);
       void mu_y( int b);
       void visualizar( );
} ;
```


La clase Punto completa

El archivo Punto.cpp

```
Punto :: Punto(int a, int b)
{
   x = new int;
   y = new int;
   *x = a;
   *y = b;
}
```

```
Punto& Punto :: operator = ( Punto p)
{
  *x = *p.x;
  *y  = *p.y;
  return *this;
}
```

```
Punto :: ~Punto( )
{
   delete x;
   delete y;
}
```

Los accedentes y mutadores trabajan con lo apuntado por el atributo puntero

```
void Punto::mu_x(int a )
{
     *x = a;
}
```

```
int Punto ::acc_x( )
{
    return *x;
}
```