Contenidos

Introducción

Hasta ahora hemos visto que cada vez que queremos usar una variable debemos reservarle un lugar de la memoria al comenzar el programa. Debemos indicar al compilador cuánta memoria vamos a usar.

Por ejemplo, si hacemos la declaración:

```
disco mis_cds[40];
```

el compilador reserva espacio para 40 cds.

➡ Si en realidad solo queremos guardar datos para 2 cds, se desperdicia mucha memoria.

➡ Si me encanta la música, dicho array se me quedará pequeño.

Introducción

Por tanto, hay ocasiones en las que no sabemos cuánta memoria vamos a necesitar hasta que no se ejecuta el programa.

C++ permite poder reservar memoria según se va necesitando, es decir, en tiempo de ejecución.

Podremos reservar memoria para almacenar 2 cds, o reservar memoria para 100 cds si el usuario que ejecuta la aplicación es un adicto a la música: esto lo sabremos en el momento en que se ejecuta el programa, no antes.

- **Ventaja de utilizar memoria dinámica:**

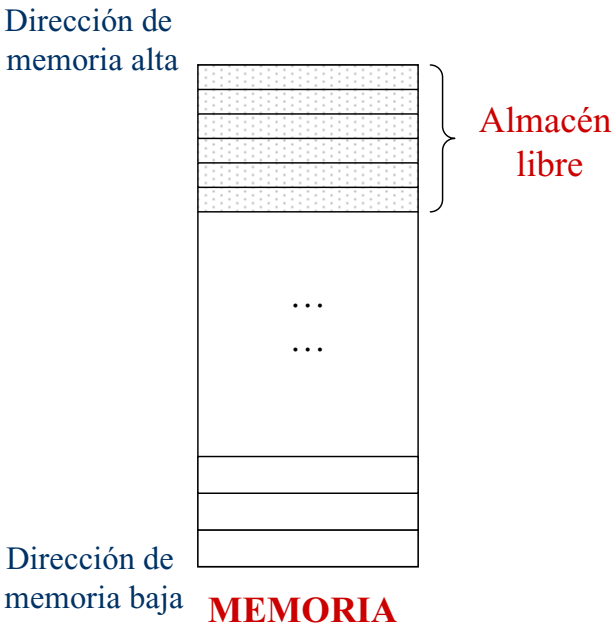
Los programas aprovecharán mejor la memoria del ordenador en el que se ejecuten, usando sólo lo necesario.

Operadores NEW y DELETE

C++ proporciona dos operadores para la gestión de memoria:

new()
permite reservar memoria del almacén libre

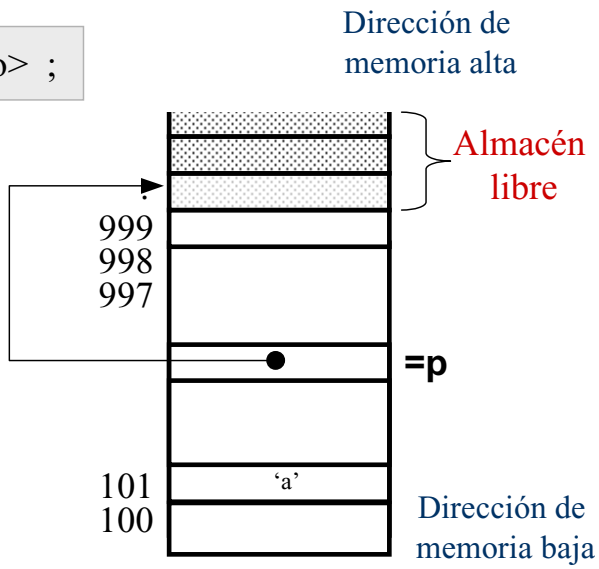
delete()
permite liberar la memoria cuándo no se necesita



El Operador NEW

El operador **new** permite reserva un bloque de memoria y devuelve la dirección de comienzo de dicho bloque. Esta dirección se almacena en un puntero.

```
<variable_puntero> = new  <tipo_dato>  ;
```



Cuando se invoca al operador ***new***, el compilador realiza una comprobación de tipos.

```
char   *p = NULL ;
p  = new char ;
```

Si la operación falla, el operador ***new***, devuelve el puntero nulo NULL.

El Operador NEW : Ejemplo

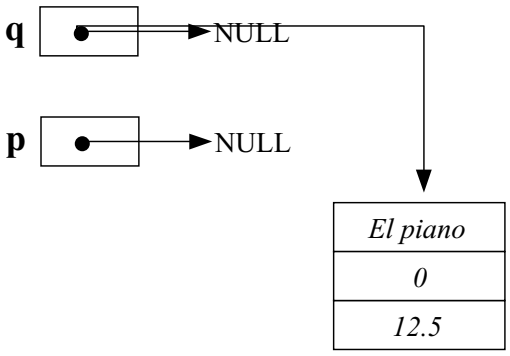
```
struct disco
     {
          char titulo[30];
          int num_canciones;
          float precio;
     };

disco *q = NULL;
int  *p = NULL;

q = new disco;
cin.getline(q->titulo,30);
q->precio = 12,5;
q->num_canciones = 0;

*p = 3;
```

ERROR !!



El compilador reserva 36 bytes de memoria que es lo que ocupa una variable de tipo **disco.**

El Operador NEW

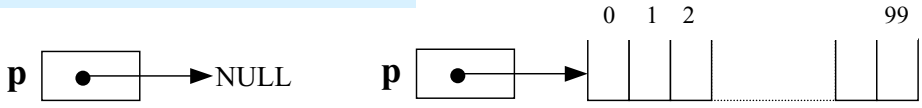
Hemos visto la sintaxis del operador **new** para la reserva de memoria para tipos de datos básicos y tipos estructura.
C++ también permite reservar bloques de memoria para arrays. Se conoce como arrays dinámicos.

```
<variable_puntero> = new  <tipo_dato> [num_componentes] ;
```

```
int   *p = NULL ;
p  = new  int [100];
```

El compilador reserva memoria para un array de 100 enteros.
En total, reserva un bloque de 200 bytes de memoria.

El puntero **p** apunta al comienzo del bloque de memoria reservado.



El Operador NEW

La zona de almacén libre se puede agotar, es decir, es posible que se solicite la reserva de memoria mediante el operador **new**, pero no quede memoria disponible en el almacén.

La operación falla, el operador ***new***,
devuelve el puntero nulo **NULL**.

Es por tanto, obligatorio comprobar que el compilador ha podido realizar la reserva de memoria. Para ello preguntaremos por el valor del puntero.

```
int   *p = NULL ;
p  = new  int [100];

if (p == NULL)
   cout << "No hay suficiente memoria en el almacén";
...
```

El Operador DELETE

Cuando ya no necesitemos más el espacio reservado debemos liberarlo, es decir, indicar al ordenador que puede destinarlo a otros fines.

Para ello utilizamos el operador ***delete***.

```
delete  <variable_puntero> ;
delete [ ] <variable_puntero> ;
```

Liberar memoria para arrays

Si no liberamos el espacio que ya no necesitamos, corremos el peligro de agotar la memoria del ordenador.

El compilador libera un bloque de 200 bytes de memoria.

```
int   *p = NULL ;
p = new  int [100];
….
delete [ ] p ;
...
```


EJEMPLO

```
void main()
   int  numero_discos;
   struct disco
    {
        char titulo[30];
        int num_canciones;
        float precio;
    };
  disco *mis_cds = NULL;
  cout << "¿Cuántos discos tienes?;
  cin >> numero_discos;
  mis_cds = new  disco[numero_discos] ;
  if (mis_cds == NULL ) return 1;
  for (int i=0; i<numero_discos; i++)
     cin.getline(mis_cds[i].titulo, 30 );
  …
  delete [ ] mis_cds;
```

```
for (int i=0; i<numero_discos; i++)
    cin.getline( (mis_cds+i)->titulo ,  30 );
…
```

```
disco *p = mis_cds;
for (int i=0; i<numero_discos; i++)
   {
      cin.getline( p->titulo ,  30 );
      p++;
   }
```

```
disco *p = mis_cds;
for (int i=0; i<numero_discos; i++)
   {
      cin.getline( (*p).titulo ,  30 );
      p++;
   }
```