



FACULTAD DE INFORMÁTICA

Algunas notas sobre punteros y listas enlazadas

Baltasar Fernández Manjón

<http://www.fdi.ucm.es/profesor/balta/>



Punteros

- Un puntero es una variable cuyo contenido es una dirección de memoria

*Tipo * variablePuntero*

declara una variable llamada *variablePuntero*

que puede almacenar la **dirección de un objeto de tipo *Tipo***

- & es el operador de dirección
- * es el operador de contenido de una dirección apuntada por un puntero

```
#include <iostream>
using namespace std;

int main()
{
    int i = 11, j = 22;
    double d = 3.3, e = 4.4;
    int * iptr, * jptr; // variables puntero :
                       // que almacenan direcciones de int
    double * dptr, * eptr; // de doubles

    iptr = &i; // valor de iptr es dirección de i
    jptr = &j; // valor de jptr es dirección de j

    dptr = &d;
    eptr = &e;

    cout << "&i = " << iptr << endl
         << "&j = " << jptr << endl
         << "&d = " << dptr << endl
         << "&e = " << eptr << endl;
}
```

Salida producida:

```
&i = 0x7ffbb7f4
&j = 0x7ffbb7f0
&d = 0x7ffbb7e8
&e = 0x7ffbb7e0
```



Funcionamiento de los punteros

- Ejemplo de intercambio usando referencias

```
void intercambio(int &a, int &b)
{ int temp = a; a = b; b = temp; }
```

- Llamada a la función *intercambio(x,y);*
- Usando punteros

```
void intercambio(int *a, int *b)
{ int temp = *a; *a = *b; *b = temp; }
```

- Llamada a la función *intercambio(&x, &y);*



Listas enlazadas

- Colección de objetos de clases autoreferenciadas (nodos) enlazados mediante punteros (enlaces)
 - Se accede utilizando el puntero al primer nodo de la lista
 - A partir de dicho nodo se tiene acceso secuencial al resto
 - El último nodo tiene un enlace a `null`
 - Indica el final de la lista
 - Se emplea el almacenamiento dinámico
 - Los nodos se crean cuando son necesarios
 - No existe un tamaño máximo predeterminado



Gestión dinámica de memoria: New y Delete

- Operador `new`
 - Crea una nueva instancia de la clase especificada
 - Devuelve el puntero al objeto creado
 - `Nodo * punteroObjeto = new Nodo(5);`
 - Devuelve `bad_alloc` si no se dispone de memoria
- Operador `delete`
 - `delete punteroObjeto;`
 - Libera la memoria obtenida por `new` mediante la llamada al destructor del objeto
 - No se elimina el puntero `punteroObjeto` si no la memoria a la que hace referencia



Funcionamiento de new y delete

- `New ClaseDatos`
 - 1. Reserva la memoria necesaria para `ClaseDatos`
 - 2. Invoca al constructor para hacer la inicialización adecuada
- `Delete ClaseDatos`
 - 1. Invoca al destructor de la clase
 - 2. Libera la memoria asignada



Destructor

- Se encarga de liberar los recursos que la `ClaseDatos` tenga asignados
 - E.g. la memoria que se ha reservado en tiempo de ejecución
- Cuando el compilador encuentra un objeto que deja de ser válido (queda fuera de ámbito) realiza una llamada a su destructor antes de liberar dicha memoria

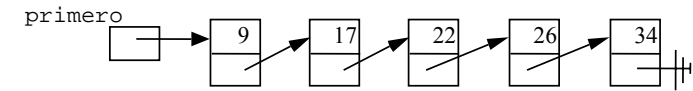


Clases con gestión de memoria dinámica

- Deben incorporar un destructor
- Hay que prestar especial cuidado para no compartir estructura entre objetos a la definición de:
 - Constructor de copia
 - Operador de asignación =



Lista enlazada simple y ejemplo de nodo



```
class Nodo {
public:
    Nodo( int );
    void setInfo( int );
    int getInfo() const;
    void setSig( Node * );
    const Node *getSig() const;
private:
    int info;
    Node* sig;
};
```



Lista doblemente enlazada

