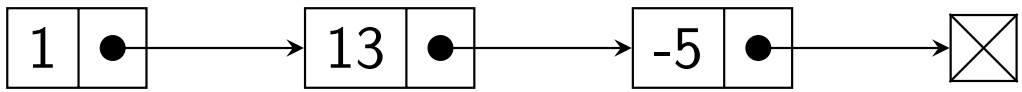
Listas enlazadas

Algoritmos y Estructuras de Datos II

Listas simplemente enlazadas

Una *lista simplemente enlazada* es una estructura que sirve para representar una secuencia de elementos.

Gráficamente



Listas simplemente enlazadas

Sus principales características son:

- Permiten una manejo más fino del uso de memoria
- Permiten insertar al principio (y potencialmente al final) de forma eficiente
- Son eficientes para reacomodar elementos (útil para ordenar)

Lista de Enteros

Implementemos la clase `ListaDeEnt`, sobre una lista simplemente enlazada, con los siguientes métodos:

```
ListaDeEnt();
~ListaDeEnt();
void agregarAtras(int x);
int longitud() const;
int iesimo(int i) const;
```

Constructores por copia y memoria dinámica

¿Qué pasa si no implementamos nuestro propio constructor por copia?

```
1 int main() {
2     ListaDeEnt l1;
3     l1.agregarAtras(1);
4     ListaDeEnt l2(l1);
5     l2.agregarAtras(2);
6     l1.longitud(); // ??
7 }
```

Constructores por copia y memoria dinámica

C++ usa un constructor por copia por defecto que hace una copia **únicamente** de los campos de la clase.

Si usamos memoria dinámica muy posiblemente tengamos entonces *aliasing* entre instancias. Como consecuencia, tenemos peligro de romper otras instancias y de perder memoria.

Constructores por copia y memoria dinámica

C++ usa un constructor por copia por defecto que hace una copia **únicamente** de los campos de la clase.

Si usamos memoria dinámica muy posiblemente tengamos entonces *aliasing* entre instancias. Como consecuencia, tenemos peligro de romper otras instancias y de perder memoria.

Agreguemos entonces:

```
ListaDeEnt(const ListaDeEnt& o);
```

Operador de asignación y memoria dinámica

¿Y con la asignación?

Operador de asignación y memoria dinámica

¿Y con la asignación?
También usa C++ una asignación por defecto que copia los campos de la clase. En este caso es aún peor, porque si teníamos algún valor lo acabamos de perder en el éter!

Operador de asignación y memoria dinámica

¿Y con la asignación?
También usa C++ una asignación por defecto que copia los campos de la clase. En este caso es aún peor, porque si teníamos algún valor lo acabamos de perder en el éter!

Agreguemos:

```
ListaDeEnt& operator= (const ListaDeEnt& o);
```

Lista de Enteros: Una posible solución

```
1  #include <cstdio>
2
3  class ListaDeEnt {
4  public:
5      ListaDeEnt();
6      ListaDeEnt(const ListaDeEnt& o);
7      ~ListaDeEnt();
8
9      void agregarAtras(int x);
10     int longitud() const;
11     int iesimo(int i) const;
12
13     ListaDeEnt& operator=(const ListaDeEnt& o);
14
15 private:
16     struct Nodo {
17         int valor;
18         Nodo* sig;
19
20         Nodo(int v, Nodo* s) : valor(v), sig(s) { }
21     };
22
23     Nodo* prim;
24
25     void copiarNodos(const ListaDeEnt &o);
26     void destruirNodos();
27 };
```

Lista de Enteros: Una posible solución (2)

```
1  ListaDeEnt::ListaDeEnt() : prim(NULL) { }
2
3  ListaDeEnt::ListaDeEnt(const ListaDeEnt& o) : prim(NULL) {
4      copiarNodos(o);
5  }
6
7  ListaDeEnt::~ListaDeEnt() {
8      destruirNodos();
9  }
10
11 void ListaDeEnt::agregarAtras(int x) {
12     Nodo* nuevo = new Nodo(x, NULL);
13     if (prim == NULL) {
14         prim = nuevo;
15         return;
16     }
17
18     Nodo* actual = prim;
19     while(actual->sig != NULL) {
20         actual = actual -> sig;
21     }
22     actual->sig = nuevo;
23 }
24
25 int ListaDeEnt::longitud() const {
26     Nodo* actual = prim;
27     int contador = 0;
28     while (actual != NULL) {
29         contador++;
30         actual = actual->sig;
31     }
32     return contador;
33 }
```

Lista de Enteros: Una posible solución (3)

```
1  int ListaDeEnt::iesimo(int i) const {
2      Nodo* actual = prim;
3      for (int j = 0; j < i; ++j) {
4          actual = actual->sig;
5      }
6      return actual->valor;
7  }
8
9  ListaDeEnt& ListaDeEnt::operator=(const ListaDeEnt& o) {
10     destruirNodos();
11     copiarNodos(o);
12     return *this;
13 }
14
15 void ListaDeEnt::copiarNodos(const ListaDeEnt &o) {
16     Nodo* actual = o.prim;
17     while (actual != NULL) {
18         agregarAtras(actual->valor);
19         actual = actual->sig;
20     }
21 }
22
23 void ListaDeEnt::destruirNodos() {
24     Nodo* actual = prim;
25     while (actual != NULL) {
26         Nodo* siguiente = actual->sig;
27         delete actual;
28         actual = siguiente;
29     }
30     prim = NULL;
31 }
```