Estructuras de datos básicas

AED I

2020

UBP UNIVERSIDAD BLAS PASCAL

Tipos de Datos Abstractos

Tipos Abstractos de Datos (TADs)

- Identificar los tipos abstractos de datos surge de analizar detenidamente el problema a resolver
- Identificarlos y especificarlos es una tarea que siempre es recompensada
- Antes de hablar de los tipos abstractos, hablaremos de los tipos concretos habituales
- Saltearemos tipos sencillos conocidos: booleanos, enteros, char, reales
- Abordaremos tipos más complejos como arreglos, listas y tuplas

Estructuras secuenciales: Listas

lista enlazada

- Es una estructura de datos dinámica que usa punteros para su implementación
- Esencialmente la lista enlazada funciona como un array que puede agrandarse y encogerse en la medida en que sea necesario.
- Es un conjunto de nodos almacenados dinámicamente, organizados de manera tal que cada nodo contiene un valor y un puntero. El puntero apunta al nodo siguiente de la lista. Si el puntero es nulo indica que es el último nodo.

lista enlazada

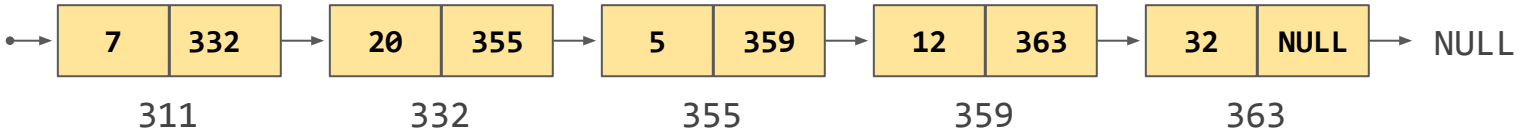
Comparada con el array tiene las siguientes **ventajas**:

- Se pueden agregar o remover items en cualquier parte de la lista
- No hay necesidad de definir un tamaño inicial

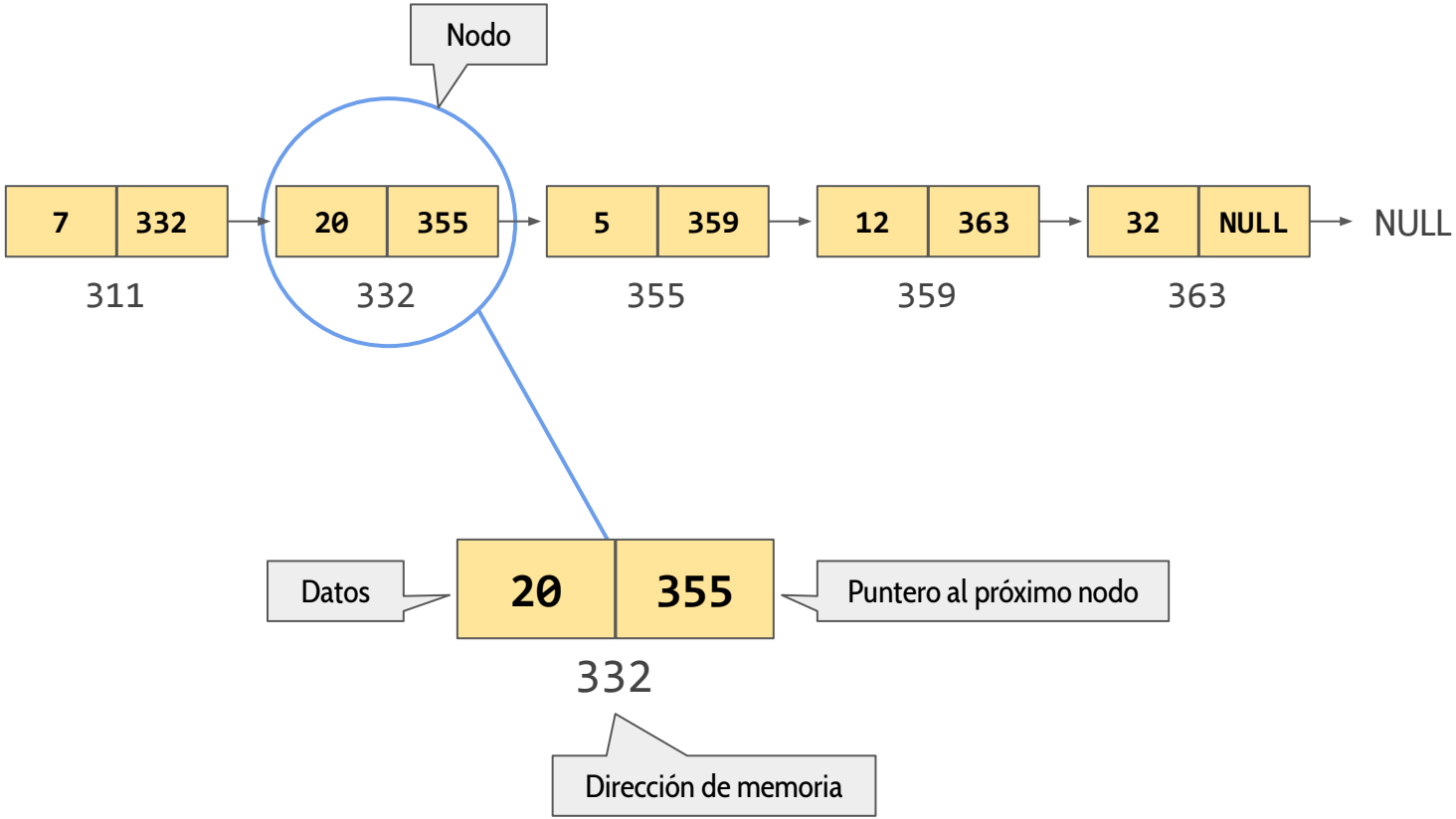
y las siguientes **desventajas**:

- No poseen acceso aleatorio. Para alcanzar el elemento n-esimo debemos recorrer todos los elementos que lo preceden
- Necesitamos hacer uso de punteros y acceso dinámico de la memoria, lo cual aumenta la posibilidad de errores
- Requieren de una cantidad importante de memoria extra que no contendrá información significativa. Igualmente se debe considerar el costo de redimensionar o sobredimensionar un array

lista enlazada



lista enlazada



lista enlazada

	Array	**Lista Enlazada**
Costo de Acceder a un elemento	O(1)	O(n)
Costo de insertar un elemento	al inicio: O(n) al final: si está lleno O(n) sino O(1) en la posición i: O(n)	al inicio: O(1) al final: O(n) en la posición i: O(n)
Uso de memoria	Tamaño Fijo. Espacio sin usar.	Espacio extra para almacenar los punteros
Asignación de memoria	Todos el array debe estar en el mismo bloque de memoria (elementos contiguos)	Flexible. Se puede usar memoria fragmentada.

lista enlazada: implementación

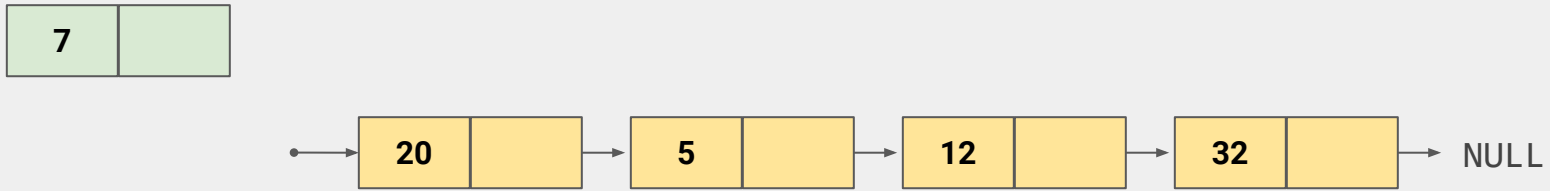
```
struct nodo{
     int dato;
     nodo * siguiente;
};
```

lista enlazada: operaciones

```
class Lista{
    private:
          struct nodo{
               int dato;
               nodo * siguiente;
          };
          nodo* inicio;
    public:
          Lista();
          void insertar(int);
          void mostrar();
          bool estaVacia();
          int longitud();
          ~Lista();
};
```

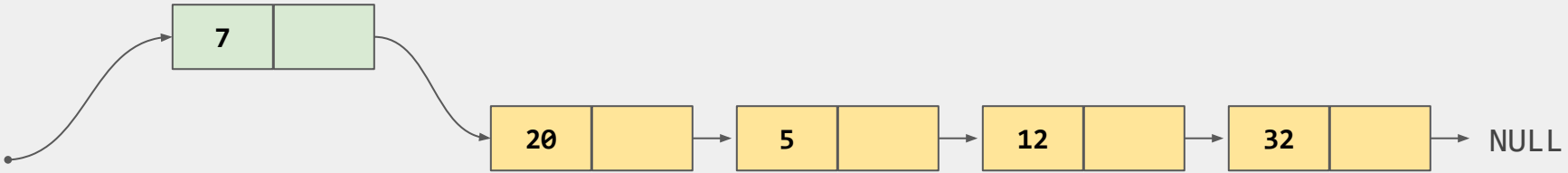
lista enlazada: insertar

```
void Lista::insertar(int dato){
    nodo* nuevoNodo = new nodo;
    nuevoNodo->dato = dato;
    nuevoNodo->siguiente = inicio;
    inicio = nuevoNodo;
}
```



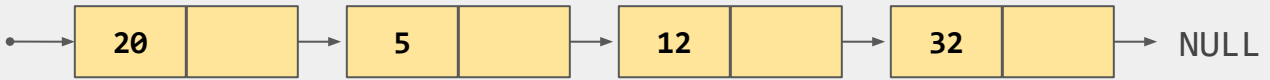
lista enlazada: insertar

```
void Lista::insertar(int dato){
    nodo* nuevoNodo = new nodo;
    nuevoNodo->dato = dato;
    nuevoNodo->siguiente = inicio;
    inicio = nuevoNodo;
}
```



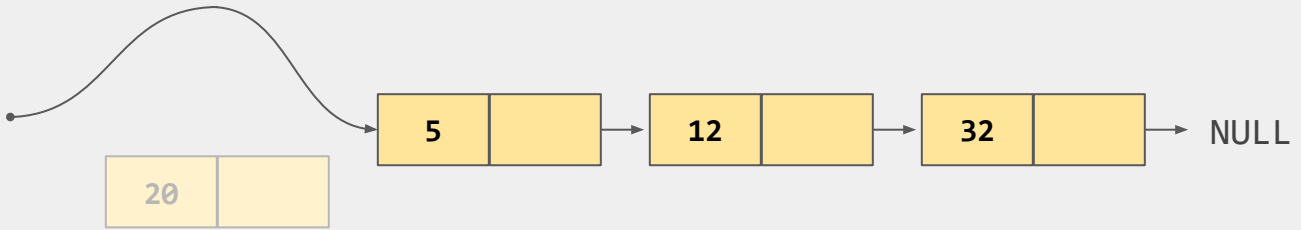
lista enlazada: remover

```
void Lista::remover(){
        nodo* nodoARemover;
        if (!estaVacia()){
                nodoARemover = inicio;
                inicio = inicio->siguiente;
                delete nodoARemover;
        }
}
```



lista enlazada: remover

```
void Lista::remover(){
        nodo* nodoARemover;
        if (!estaVacia()){
                nodoARemover = inicio;
                inicio = inicio->siguiente;
                delete nodoARemover;
        }
}
```



análisis dinámico con valgrind

```
// Compilar con la opción -g

$g++ -g -Wall Lista.cpp -o Lista

// Usar valgrind para analizar la asignación de memoria

$valgrind --leak-check=yes ./Lista
```

lista enlazada: más operaciones

```
class Lista{
    private:
       nodo* inicio;
    public:
       Lista();
       bool estaVacia();
       void insertar(int);
       void insertarUltimo(int);
       void insertarEn(int,int);
       void remover();
       void removerUltimo();
       void removerEn(int);
       int longitud();
       void invertir();
       void mostrar();
       ~Lista();
};
```

Ejercicio 2.03.

Implementá la clase Lista con las siguientes operaciones de una Lista Enlazada Simple:

```
Lista();
~Lista();
bool estaVacia();
void insertar(int);
void insertarUltimo(int);
void insertarEn(int,int);
void remover
```

```
void removerUltimo();
void removerEn(int);
*nodo buscar(int);
*nodo obtenerNodo(int);
int longitud();
void invertir();
void mostrar();
```

Implemente versiones recursivas para los siguientes métodos:

```
int longitud();
void mostrar();
void removerUltimo();
```

Ejercicio 2.04.

Modificá el programa del ejercicio 1.03. y reimplementá el atributo calificaciones como una Lista enlazada en vez de un array. Deberás reimplementar los métodos `agregarCalificacion(int)` y `getPromedio()` también.