

Estructuras de Datos Dinámicas: Listas

Teoría: Programación I

<http://proguno.unsl.edu.ar>

proguno@unsl.edu.ar

Listas

- Capacidad: dinámica
- Orden: no cronológico. El orden puede ser definido por el programador. Ej. Orden alfabético, orden numérico, etc...
- Implementaciones:
 - Listas estáticas
 - Listas dinámica

Estructuras de Datos Dinámicas:

Listas ⁽¹⁾

- Capacidad: dinámica, crece y disminuye con las inserciones y supresiones
- Orden: No tiene orden cronológico de inserción o supresión.
- Secuencia.
 - unidireccional, 1º \Rightarrow último

Estructuras de Datos Dinámicas:

Listas ⁽²⁾

- Elementos de una lista unidireccional o secuencia, llamados *nodos*, constan de *dos partes*:
 - *Variable de Información Propiamente Dicha (VIPD)*
 - *puntero* al elemento (nodo) siguiente en la lista.
 - último elemento, el puntero no apunta a un elemento y se dice que su valor es *nil*

Estructuras de Datos Dinámicas:

Listas ⁽³⁾

Operaciones sobre la lista

- Inserción
- Supresión
- Copia
- Predicados: isEmpty, isFull

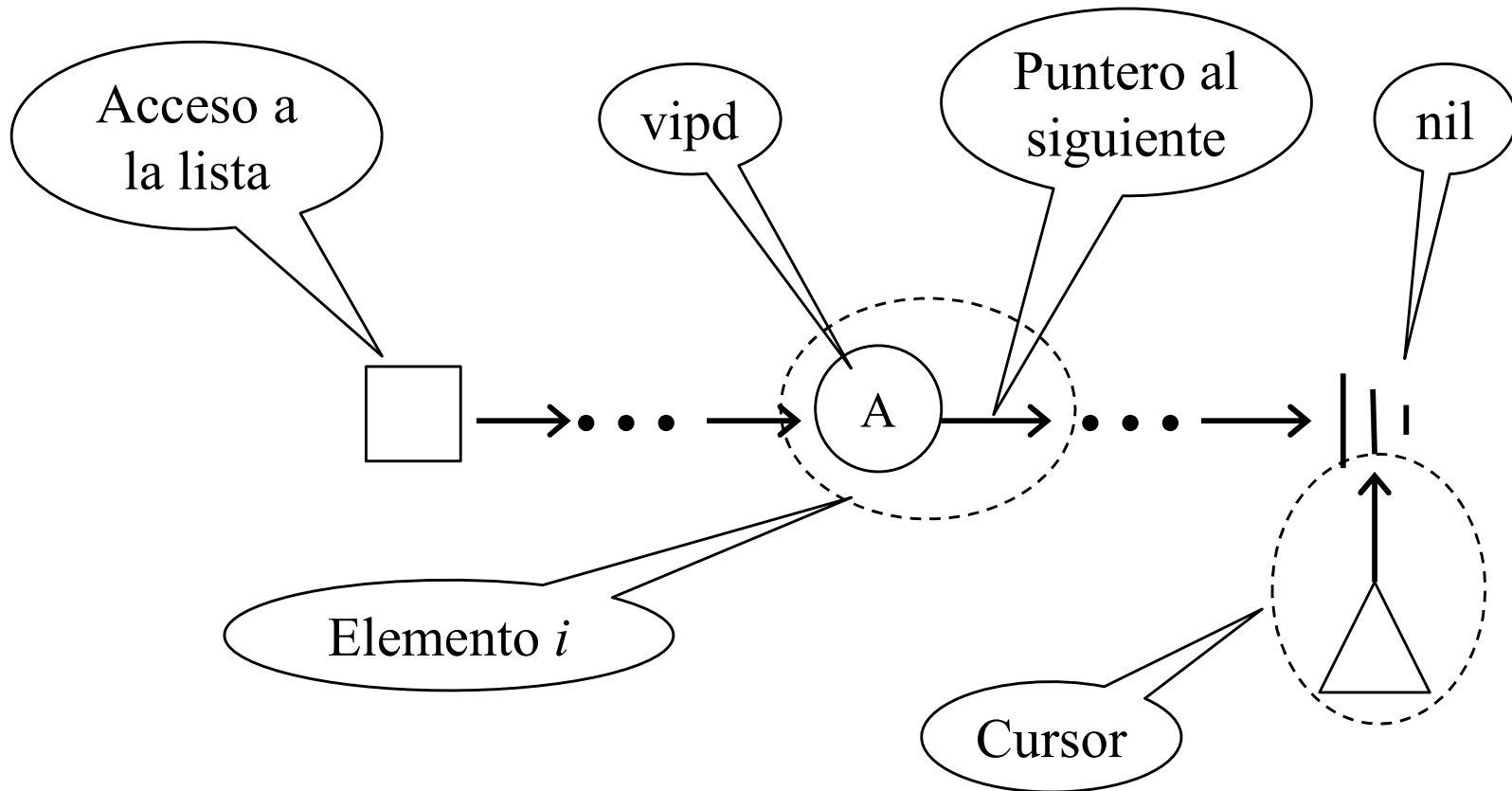
Estructuras de Datos Dinámicas:

Listas ⁽⁴⁾

- Selector de la lista: implícito \Rightarrow cursor
- Operaciones sobre el cursor de la lista
 - Ir al primero: reset
 - Avanzar: forwards
 - Predicado:
Fuera de la estructura (cursor = *nil*): isOos

Estructuras de Datos Dinámicas:

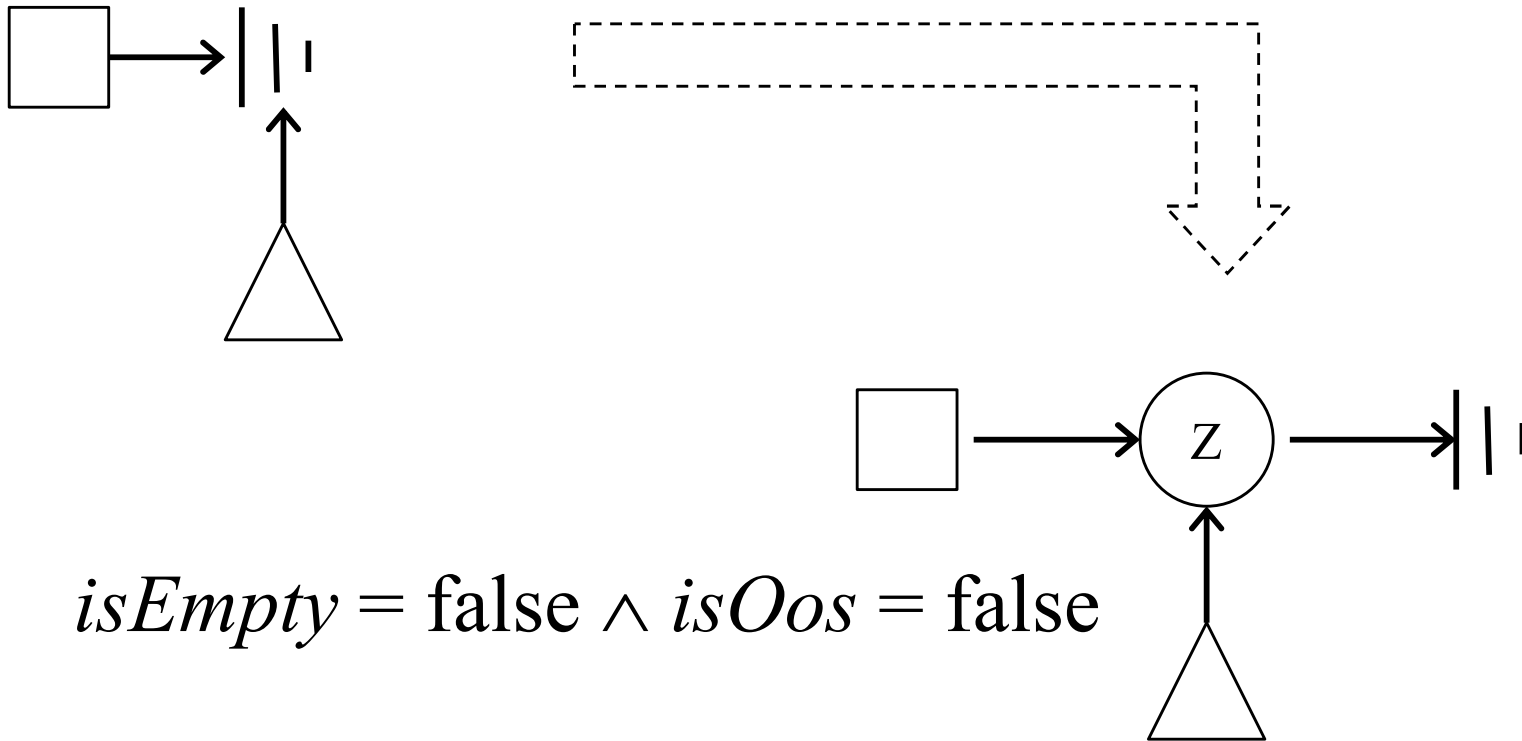
Listas: Representación gráfica (4)



Cursor

insert con lista vacía

- $isEmpty = true \wedge isOos = true$

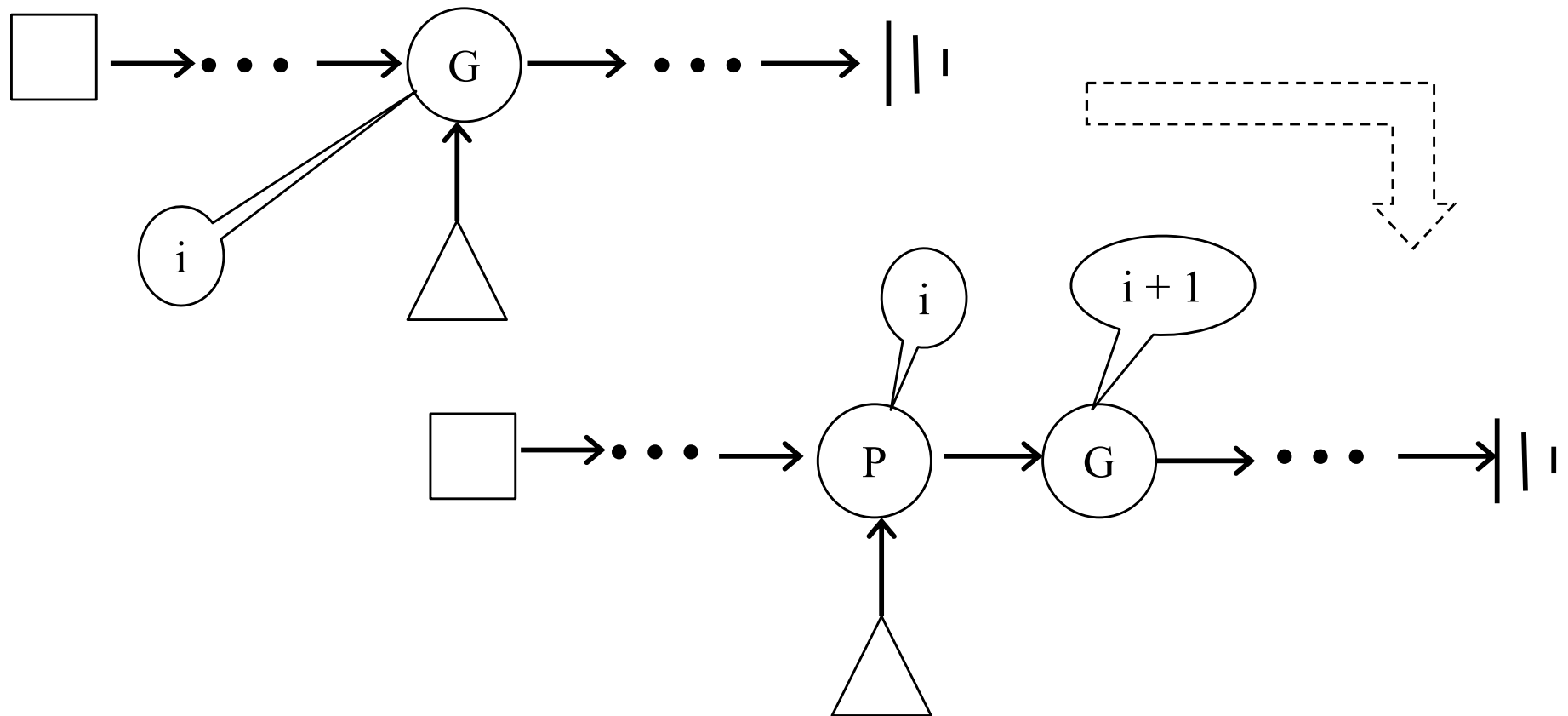


- $isEmpty = false \wedge isOos = false$

Cursor

insert con lista NO vacía

– $isEmpty = false \wedge isOos = false$

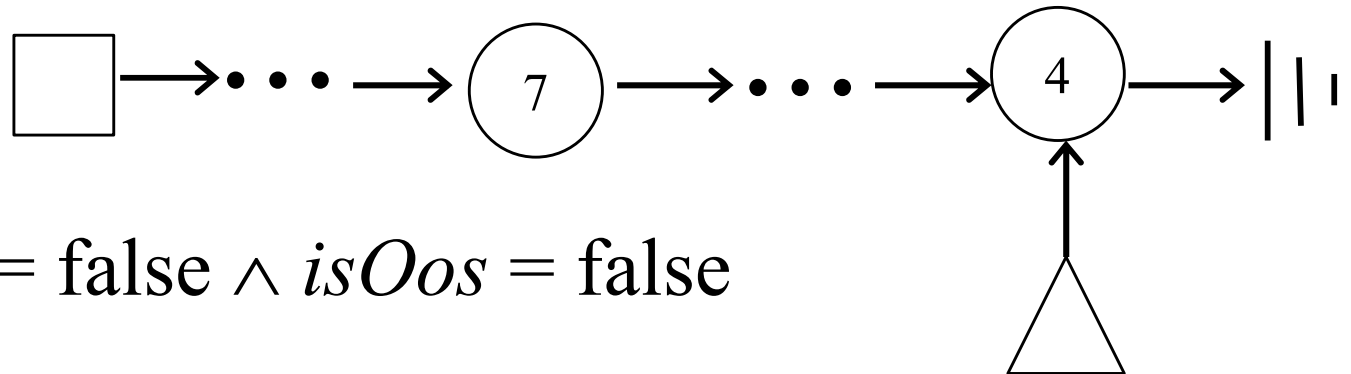
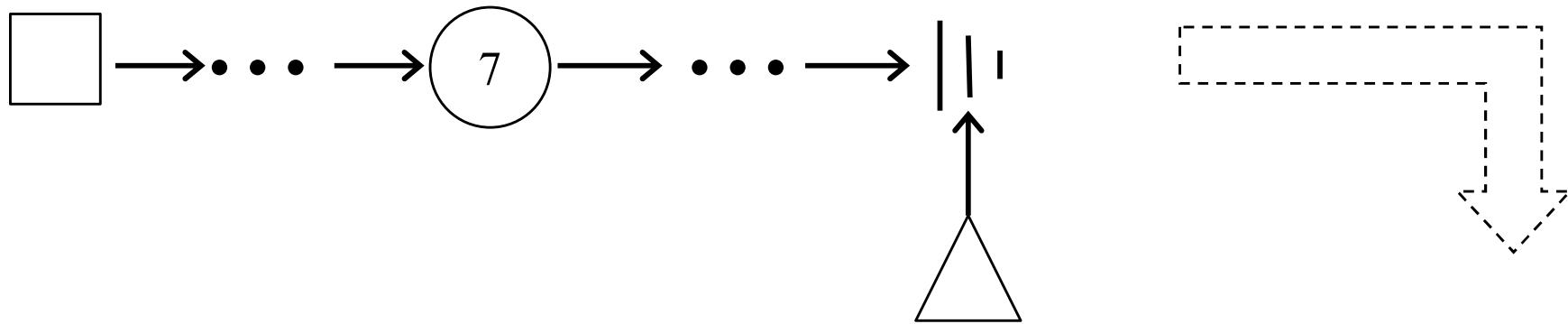


– $isEmpty = false \wedge isOos = false$

Cursor

insert con lista NO vacía

– $isEmpty = false \wedge isOos = true$

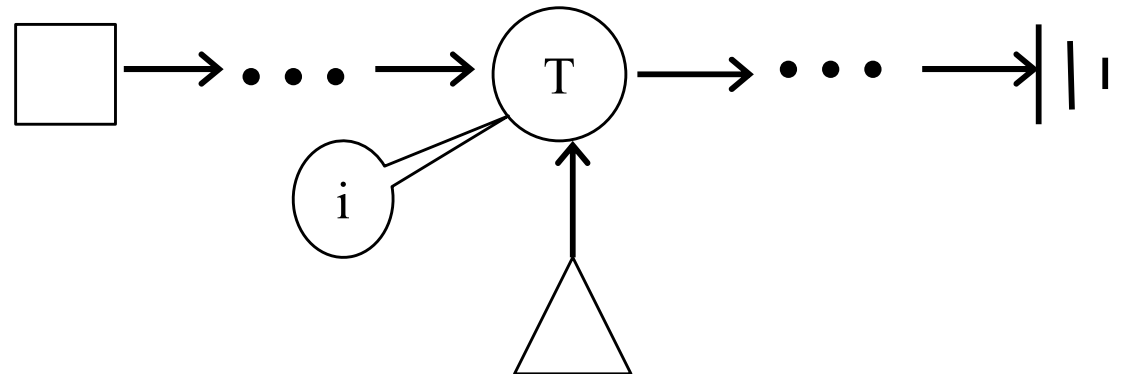
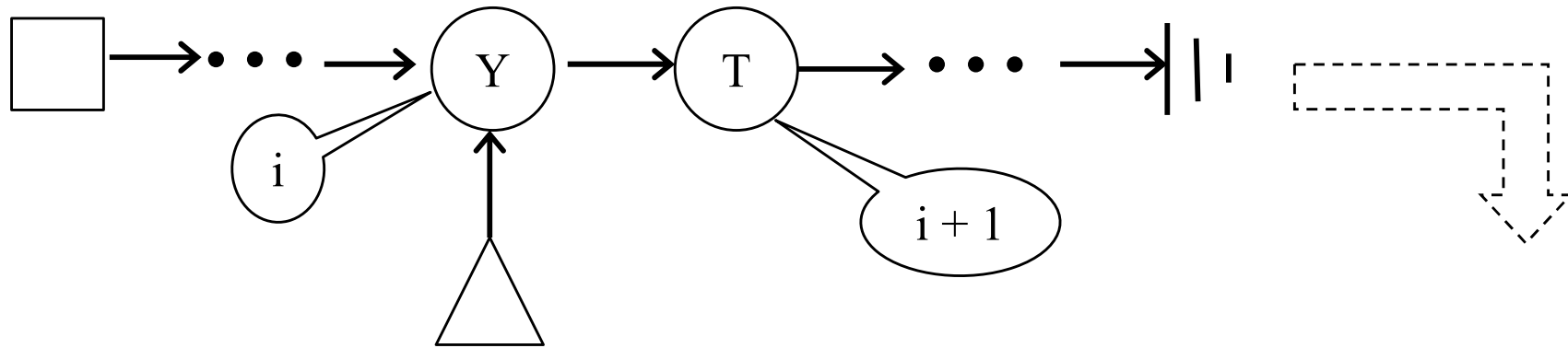


– $isEmpty = false \wedge isOos = false$

Cursor

suppress con lista NO vacía

– $isEmpty = false \wedge isOos = false$

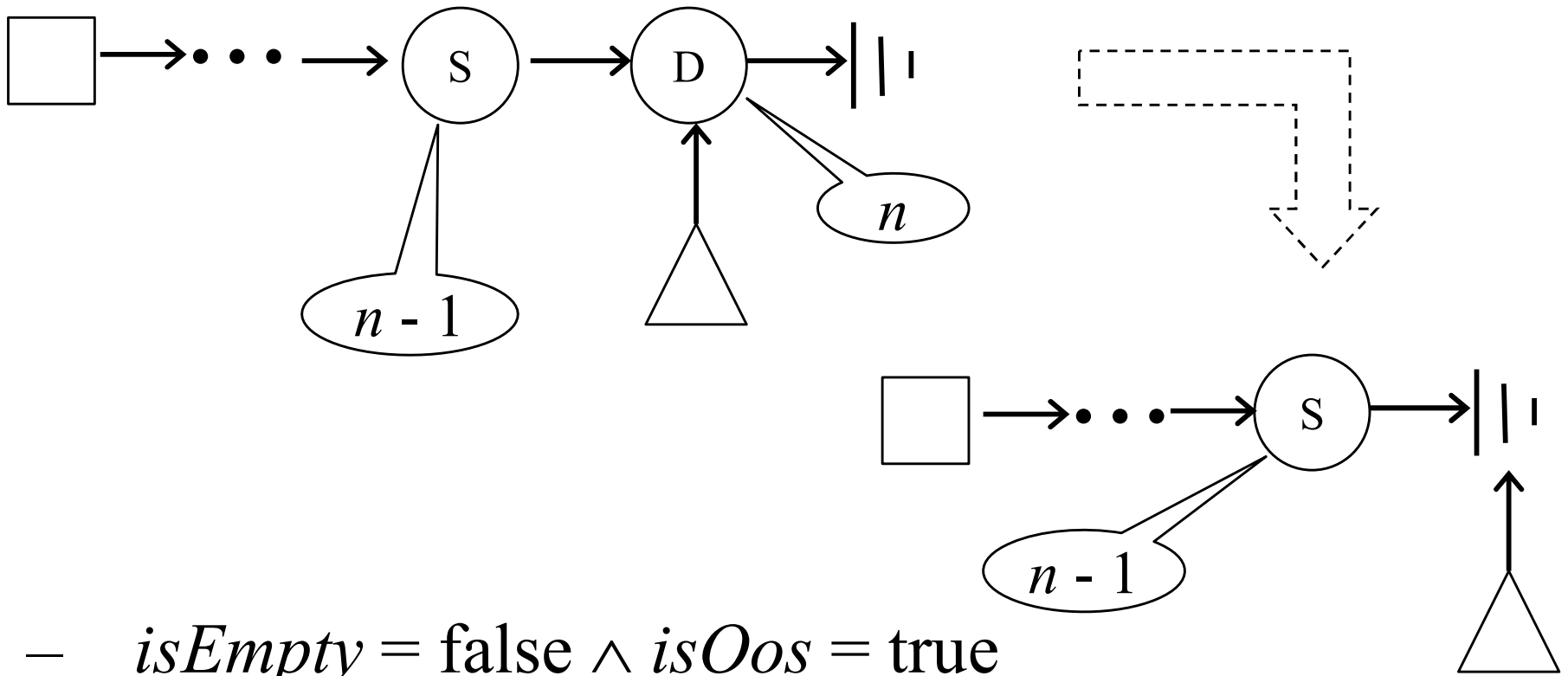


– $isEmpty = false \wedge isOos = false$

Cursor

suppress con lista NO vacía

– $isEmpty = false \wedge isOos = false$

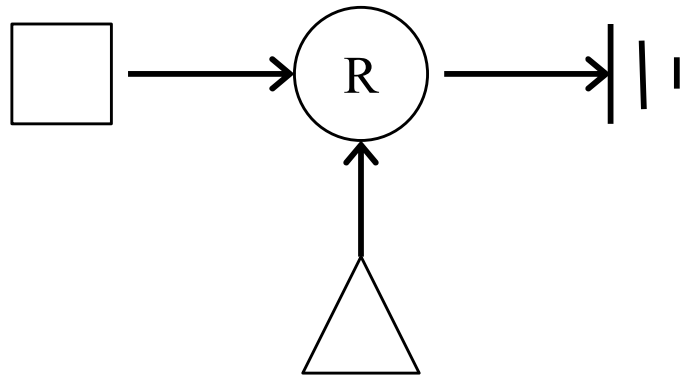


– $isEmpty = false \wedge isOos = true$

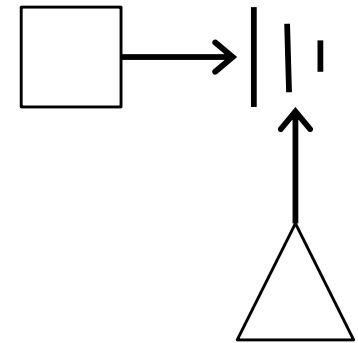
Cursor

suppress con lista NO vacía

- $isEmpty = false \wedge isOos = false$



- $isEmpty = true \wedge isOos = true$



Tipo de Datos Abstracto (TDA) (1)

Ejemplo de uso (1)

```
#include "list_of_int.h"
...
/** Imprime lista de enteros */
void printListaInt (list_of_int x) {
    reset(&x);
    while (!isOos(x)) {
        printf("%d ", copy(x));
        forwards(&x);
    }
    printf("\n");
} /* fin de printListaInt */
```

Tipo de Datos Abstracto (TDA) (2)

Ejemplo de uso (2)

```
#include "list_of_char.h"
...
/* ***** Buscar en lista de char ***** */
void buscaListaChar(list_of_char x, char y) {
    reset(&x);
    while (!isOos(x) && copy(x) != y) {
        forwards(&x);
    }
    if (!isOos(x))
        printf("%c esta en la lista\n", y);
    else
        printf("%c NO esta en la lista\n", y);
} /* fin de buscaListaChar */
```

Tipo de Datos Abstracto (TDA) ⁽³⁾

Ejemplo de uso ⁽³⁾

```
#include "list_of_int.h"
...

/* ***** Copiar lista de int ***** */
void copiaListaInt (lista_of_int *x,
                   lista_of_int y) {
    reset(&y);
    while (!isOos(y)) {
        insert(x, copy(y));
        forwards(&y);
    }
} /* fin de copiaListaInt */
```


¿Pensemos?

- ¿Es posible implementar una Pila dinámica?
- ¿Y una Fila?
- ¿Cómo implemento la lista en forma dinámica? Memoria?

¿Cómo?

Programación I

Datos Recursivos

<http://proguno.unsl.edu.ar>

proguno@unsl.edu.ar

Definiciones recursivas

Repaso

- La **definición** de un concepto es **recursiva** si el concepto es definido en términos de sí mismo.
- En una **definición recursiva**, en general, distinguimos dos partes:
 - Caso(s) base o elemental(es).
 - Definición recursiva o caso general.

Recursividad en Computación

Repaso

- Se encuentra presente en:
 - Definiciones recursivas de módulos.
 - Definiciones recursivas de datos.

Definiciones Recursivas de Datos

- Ejemplo 1: Definición recursiva de una lista

Lista { elemento seguido de una **Lista** (Caso General)
Lista vacía (Caso base)

Definiciones Recursivas de Datos

- Ejemplo 2: Definición recursiva de un árbol binario

Árbol {
 Árbol derecho + Árbol izquierdo (Caso Gral.)
 Árbol vacío (Caso base)

Datos Recursivos en C

- Diversas estructuras de datos pueden ser implementadas por medio de datos recursivos.
- En esta materia:
 - Listas
 - Pilas
 - Colas

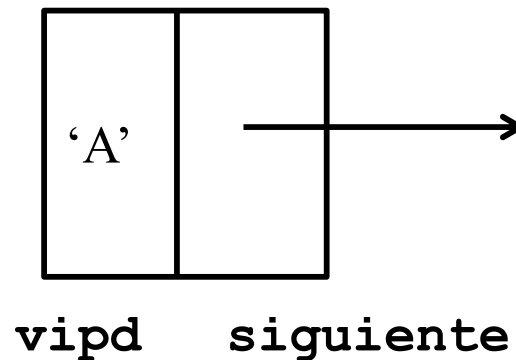
Datos Recursivos en C

- Se pueden implementar en C por medio de structs que se autoreferencian por medio de campos de tipo puntero.

Datos Recursivos en C - LISTAS

```
struct nodo{  
    char vipd;  
    struct nodo *siguiente;  
};
```

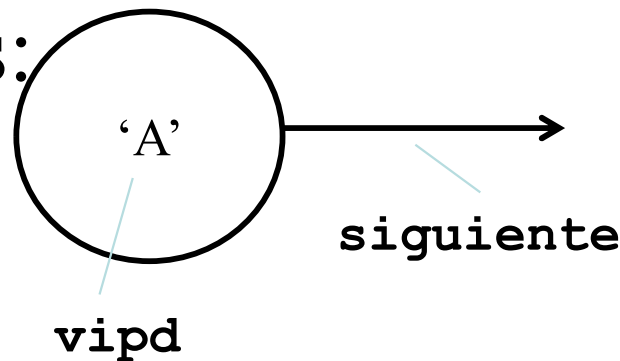
- Cada **struct nodo** es un registro con dos campos (**vipd** y **siguiente**), que tiene la siguiente pinta:



Datos Recursivos en C -LISTAS

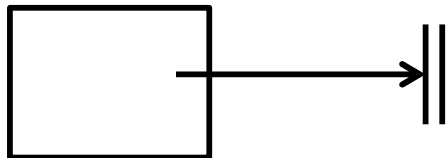
```
struct nodo{  
    char vipd;  
    struct nodo *siguiente;  
};
```

- Podemos también representar gráficamente un **struct nodo** de la forma que estamos más habituados:



Datos Recursivos en C -LISTAS

```
struct nodo{
    char vipd;
    struct nodo *next;
}
typedef struct nodo Nodo;
typedef Nodo * list_of_char;
list_of_char lis = NULL; /*lista vacia*/
```



lis

Datos Recursivos en C -LISTAS

- ¿Nos basta con definir el tipo `list_of_char` como `Nodo *`?
- Sí y no
 - Si, porque es una posible representación para una lista.
 - No, porque queremos implementar las listas unidireccionales tal como lo hemos venido haciendo, es decir, necesitaremos mantener información no solo sobre el acceso a la lista sino también sobre sus cursores.



Implementación de un TDA para Listas Unidireccionales

- 1) Definir el **tipo de dato** para soportar las listas.
- 2) Definir las **funciones típicas** para operar con las listas y a partir de las cuales podremos definir nuevas funciones:
 - **init**
 - **isEmpty**
 - **isFull**
 - **reset**
 - **forward**
 - **isOos**
 - **copy**
 - **insert**
 - **suppress**

Implementación de un TDA para **Listas Unidireccionales**

- 1) Definición del **tipo de dato**
 - Para soportar una lista unidireccional tal como lo hemos venido haciendo necesitaremos mantener información sobre:
 - Acceso a la lista
 - Cursor de la lista
 - Cursor auxiliar



Implementación de un TDA para Listas Unidireccionales

```
struct nodo{  
    tipoBase vipd;  
    struct nodo *next;  
};
```

*Acá vendrá el tipo de
los elementos de la
lista*

```
typedef struct nodo Nodo;  
typedef struct {  
    Nodo *acc;    /* acceso a la lista */  
    Nodo *cur;    /* cursor de la lista */  
    Nodo *aux;    /* cursor auxiliar */  
} List_of_char;
```

Implementación de un TDA para Listas Unidireccionales de **char**

```
struct nodo{
    char vipd;
    struct nodo *next;
};
typedef struct nodo Nodo;
typedef struct {
    Nodo *acc;    /* acceso a la lista */
    Nodo *cur;    /* cursor de la lista */
    Nodo *aux;    /* cursor auxiliar */
} List_of_char;
```

En este caso estamos definiendo el tipo List_of_char para soportar listas de caracteres, pero podrían ser listas de cualquier tipo simple o estructurado, por ejemplo fechas, como se ve en la próxima transparencia.

Implementación de un TDA para Listas Unidireccionales de Fecha

```
#include Fecha.h /* TDA Fecha */  
  
struct nodo{  
    Fecha vipd;  
    struct nodo *next;  
};  
  
typedef struct nodo Nodo;  
  
typedef struct {  
    Nodo *acc; /* acceso a la lista */  
    Nodo *cur; /* cursor de la lista */  
    Nodo *aux; /* cursor auxiliar */  
} ListaDeFechas;
```

En este caso estamos definiendo el tipo ListaDeFechas para soportar listas cuyo tipo base es el tipo Fecha, definido en el TDA Fecha

Implementación de un TDA para **Listas Unidireccionales de char**

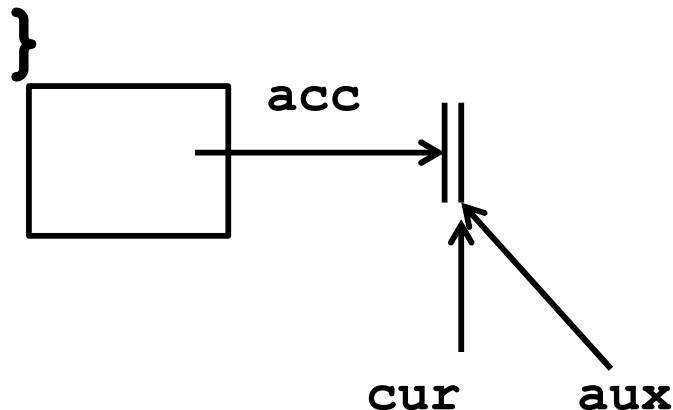
- 2) Definición de las **operaciones** del TDA:
 - `init`
 - `reset`
 - `forwards`
 - `isOos`
 - `copy`
 - `insert`
 - `suppress`
 - `isEmpty`
 - `isFull`



Implementación de un TDA para Listas Unidireccionales de **char**

Inicialización de la lista (en vacío)

```
void init(list_of_char *l) {  
    (*l).acc = NULL;  
    (*l).cur = NULL;  
    (*l).aux = NULL;  
}
```



Implementación de un TDA para Listas Unidireccionales de Fecha

Inserción de una fecha en la posición apuntada por el cursor

```
void insert(list_of_char *l, char c) ;
```

- Necesitaremos, entre otras cosas, pedir espacio al compilador para el nuevo elemento a insertar (un caracter, en este caso):
 - **malloc**(*n*) /* asigna *n* bytes de memoria y devuelve el puntero a la dirección del lugar asignado, sino retorna NULL.
*/
- ¿Cuántos bytes pedimos?
 - **sizeof**(*Tipo*) /* devuelve el tamaño en bytes ocupado por un objeto de datos de tipo *T*. */

Implementación de un TDA para Listas Unidireccionales de Fecha

Supresión de la fecha corriente (apuntada por el cursor)

```
void suppress(list_of_char *l) ;
```

- Necesitaremos, entre otras cosas, devolver el espacio liberado:

- `free(vble_ptr) ;`

Devuelve al sistema los bytes apuntados por *vble_ptr*.

Fin ... por suerte ... ¿no? ;-)