Listas enlazadas

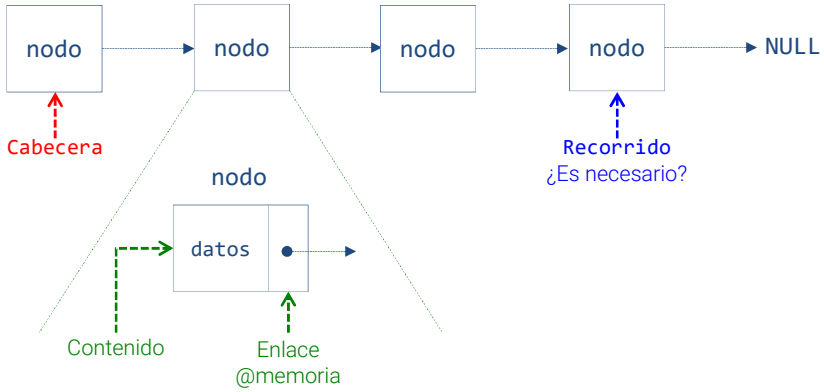
INF2240 – Estructura de datos
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Listas simplemente enlazadas

- Presentan un grado de flexibilidad mayor que las listas contiguas.
- Las acciones de insertar o eliminar un enésimo elemento no contemplan desplazamiento de los elementos restantes de la lista.
- Los elementos se almacenan en posiciones de memoria que no son continuas o adyacentes, por lo que cada elemento necesita almacenar la posición o dirección del siguiente elemento de la lista.

Listas simplemente enlazadas

— A continuación se aprecia gráficamente una lista simplemente enlazada

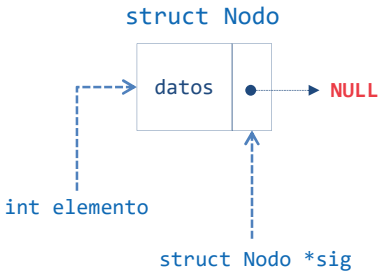


75

Listas simplemente enlazadas

— La definición de un nodo se hace de la forma:

```
#include <stdio.h>
#include <stdlib.h>

struct Nodo
{
	int elemento;
	struct Nodo *sig;
};
```



76

Funciones sobre listas simplemente enlazadas

```c
#include <stdio.h>
#include <stdlib.h>

struct Nodo
{
	int elemento;
	struct Nodo *sig;
};

struct Nodo *crearNodo()
{
	struct Nodo *nuevo;

	/*Se obtiene la memoria al nuevo nodo*/
	nuevo=((struct Nodo*)malloc(sizeof(struct Nodo)));

	/*Se completa con los datos*/
	printf("ingrese numero: ");
	scanf("%d", &nuevo->elemento);
	nuevo->sig=NULL;

	return nuevo;
}
```

→

Código en Github
https://goo.gl/v2pCpR

77

Funciones sobre listas simplemente enlazadas

```c
void enlazarNodo(struct Nodo **head, struct Nodo *nuevo)
{
	struct Nodo *rec=*head;

	if(*head==NULL)
	{
		*head=nuevo;
	}
	else
	{
		/*Como no era el primer elemento a agregar se busca el ultimo elemento*/
		while(rec!=NULL)
		{
			if(rec->sig==NULL)
			{
				rec->sig=nuevo;
				rec=nuevo;
			}
			rec=rec->sig;
		}
	}
}
```

→

Código en Github
https://goo.gl/v2pCpR

78

Funciones sobre listas simplemente enlazadas

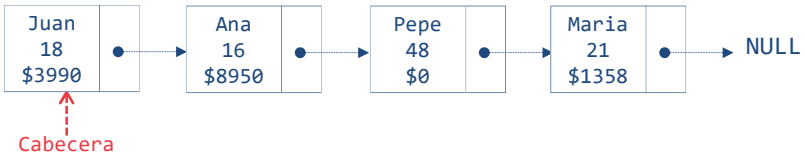
```
main()
{
	/*En este ejemplo se agregarán 3 elementos a la lista*/

	/*Declaracion de la lista*/
	struct Nodo *lista=NULL, *aux, *rec;

	/*Primer elemento*/
	aux=crearNodo();
	enlazarNodo(&lista,aux);


	/*Segundo elemento sin usar aux*/
	enlazarNodo(&lista,crearNodo());

	/*Tercer elemento sin usar aux*/
	enlazarNodo(&lista,crearNodo());

	/*Mostrar elementos de la lista*/
	rec=lista;
	while(rec!=NULL)
	{
		printf("[%d]->", rec->elemento);
		rec=rec->sig;
	}
	printf("NULL\n\n");
}
```

Código en Github
https://goo.gl/v2pCpR

79

Representación usando listas

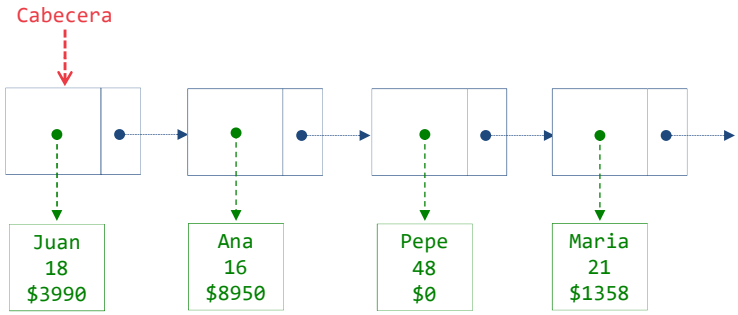
— La representación de una lista simplemente enlazada cualquiera sería de la forma:



80

Representación usando listas

— Si se considera el manejo por separado de los datos debería ser de la forma:



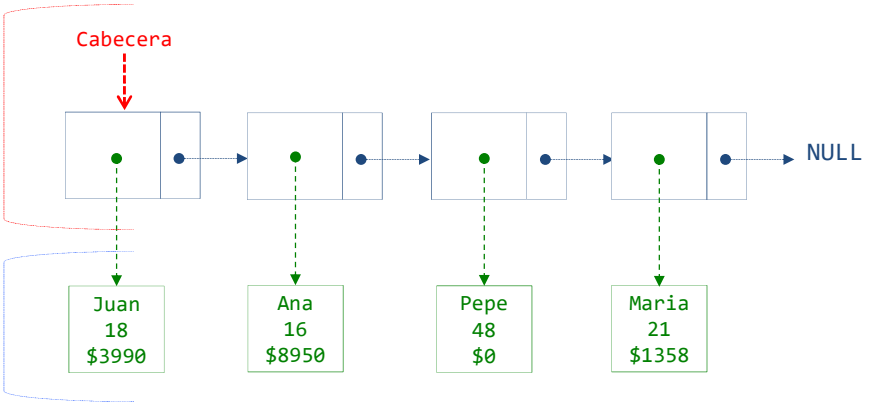
81

Representación usando listas

— Explícitamente:

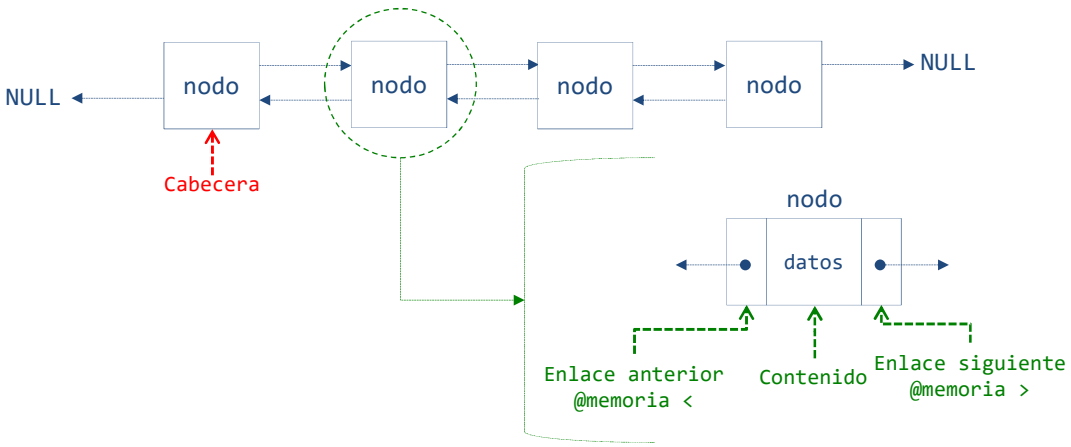
```
struct Nodo
{
    struct Persona *dato;
    struct Nodo *sig;
};
```

```
struct Persona
{
    char *nombre;
    int edad;
    int deuda;
};
```



82

Listas doblemente enlazadas



Funciones sobre listas doblemente enlazadas

```c
#include <stdio.h>
#include <stdlib.h>

struct Nodo
{
	int elemento;
	struct Nodo *ant, *sig;
};

struct Nodo *crearNodo()
{
	struct Nodo *nuevo;

	/*Se obtiene la memoria al nuevo nodo*/
	nuevo=((struct Nodo*)malloc(sizeof(struct Nodo)));

	/*Se completa con los datos*/
	printf("ingrese numero: ");
	scanf("%d", &nuevo->elemento);
	nuevo->ant=NULL;
	nuevo->sig=NULL;

	return nuevo;
}
```

→

Código en Github
https://goo.gl/pQrRkg

Funciones sobre listas doblemente enlazadas

```
void enlazarNodo(struct Nodo **head, struct Nodo *nuevo)
{
	struct Nodo *rec=*head;

	if(*head==NULL)
	{
		*head=nuevo;
	}
	else
	{
		/*Como no era el primer elemento a agregar se busca el ultimo elemento*/
		while(rec!=NULL)
		{
			if(rec->sig==NULL)
			{
				rec->sig=nuevo;
				rec->sig->ant=rec;
				rec=nuevo;
			}
			rec=rec->sig;
		}
	}
}
```

→

Código en Github
https://goo.gl/pQrRkg

85

Funciones sobre listas doblemente enlazadas

```
main()
{
	/*En este ejemplo se agregarán 3 elementos a la lista*/

	/*Declaracion de la lista*/
	struct Nodo *lista=NULL, *aux, *rec;

	/*Primer elemento*/
	aux=crearNodo();
	enlazarNodo(&lista,aux);


	/*Segundo elemento sin usar aux*/
	enlazarNodo(&lista,crearNodo());

	/*Tercer elemento sin usar aux*/
	enlazarNodo(&lista,crearNodo());

	/*Mostrar elementos de la lista*/
	rec=lista;
	while(rec!=NULL)
	{
		printf("[%d]->", rec->elemento);
		rec=rec->sig;
	}
	printf("NULL\n\n");
}
```

Código en Github
https://goo.gl/pQrRkg

86

Listas circulares

- Las listas circulares corresponden al tipo de listas que poseen el campo `sgte` del último elemento de la lista apuntando al primero.
- De esta manera es posible acceder desde un elemento a cualquiera de los elementos que le preceden.
- Hay que tener en cuenta que se pueden producir lazos o bucles infinitos, para evitar esto siempre hay que contar con un nodo que identifique el principio o el final de la lista.

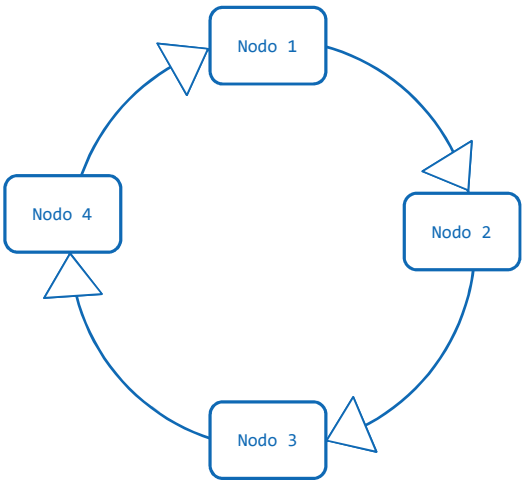
87

Listas circulares

- Contar con un nodo cabecera con un tipo de dato que permita identificar su condición (tipo de dato no válido como datos de otros elementos).
- Se debe ubicar un indicador o puntero en el nodo cabecera.

88

Listas circulares



1. Nadie apunta a NULL
2. Inserción menos compleja
3. Eliminación menos compleja
4. Recorrido menos compleja

89

Listas circulares

```
#include <stdio.h>
#include <stdlib.h>

struct Nodo
{
	int elemento;
	struct Nodo *sig;
};

struct Nodo *crearNodo()
{
	struct Nodo *nuevo;

	/*Se obtiene la memoria al nuevo nodo*/
	nuevo=((struct Nodo*)malloc(sizeof(struct Nodo)));

	/*Se completa con los datos*/
	printf("ingrese numero: ");
	scanf("%d", &nuevo->elemento);
	nuevo->sig=NULL;

	return nuevo;
}
```

→

Código en Github
https://goo.gl/xmDRoq

90

Listas circulares

```
void enlazarNodo(struct Nodo **head, struct Nodo *nuevo)
{
	struct Nodo *rec=*head;

	if(*head==NULL)
	{
		*head=nuevo;
		(*head)->sig=*head;
	}
	else
	{
		do
		{
			if(rec!=NULL)
			{
				if(rec->sig==*head)
				{
					nuevo->sig=rec->sig;
					rec->sig=nuevo;
					rec=rec->sig;
				}
			}

		}while(rec->sig!=*head);
	}
}
```

→

Código en Github
https://goo.gl/xmDRoq

91

Listas circulares

```
main()
{
	/*En este ejemplo se agregarán 3 elementos a la lista*/

	/*Declaracion de la lista*/
	struct Nodo *listaCircular=NULL, *aux, *rec;

	/*Primer elemento*/
	aux=crearNodo();
	enlazarNodo(&listaCircular,aux);


	/*Segundo elemento sin usar aux*/
	enlazarNodo(&listaCircular,crearNodo());

	/*Mostrar elementos de la lista*/
	rec=listaCircular;
	if(rec)
	{
		do
		{
			printf("[%d]->", rec->elemento);
			rec=rec->sig;
		}while(rec!=listaCircular);
	}
	printf("NULL\n\n");
}
```

Código en Github
https://goo.gl/xmDRoq

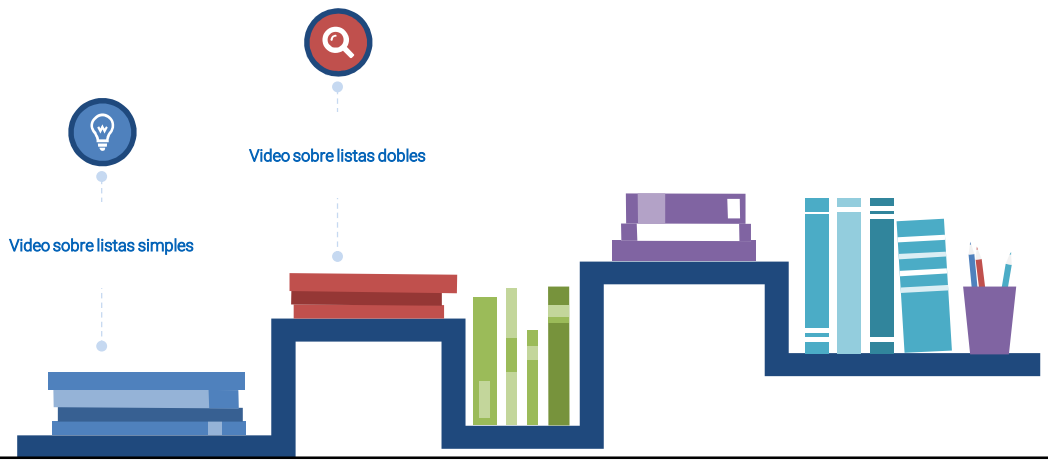
92

Listas circulares

- También podemos tener listas doblemente enlazada circulares, en donde hay que cuidar el enlace que retrocede.
- Actividad:
  - Implemente las funciones para agregar, modificar y buscar en una lista doblemente enlazada circular.

93

Lecturas del módulo



94

Especificación sobre listas

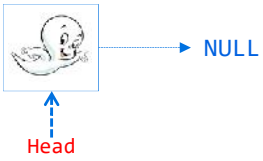
INF2240 – Estructura de datos
Escuela de Ingeniería Informática
Pontificia Universidad Católica de Valparaíso

Nodo fantasma

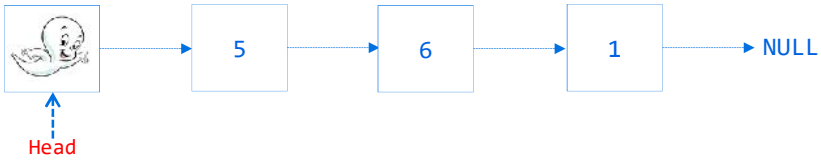
- En el esquema propuesto se deben hacer excepciones al insertar y eliminar el nodo del comienzo de la lista.
- El manejo se simplifica si se utiliza un nodo "fantasma":
  - Es un nodo siempre presente en la lista.
  - Su contenido es irrelevante (el valor u objeto contenido no forma parte de la lista)

Nodo fantasma

— Lista vacía:



— Lista con elementos:



97

Nodo fantasma

```c
#include <stdio.h>
#include <stdlib.h>

struct Dato
{
	int valor;
};
struct Nodo
{
	struct Dato *dato;
	struct Nodo *sig;
};

struct Nodo *crearNodo(struct Dato *dato)
{
	struct Nodo *nuevo;

	/*Se obtiene la memoria al nuevo nodo*/
	nuevo=((struct Nodo*)malloc(sizeof(struct Nodo)));

	nuevo->dato=dato;
	nuevo->sig=NULL;

	return nuevo;
}
```

→

Código en Github
https://goo.gl/A06FQj

98

Nodo fantasma

```
void enlazarNodo(struct Nodo **head, struct Nodo *nuevo)
{
	struct Nodo *rec=*head;

	/*No existe nodo inicial, ya que es el fantasma*/
	while(rec!=NULL)
	{
		if(rec->sig==NULL)
		{
			rec->sig=nuevo;
			rec=nuevo;
		}
		rec=rec->sig;
	}
}
```

→

Código en Github
https://goo.gl/A06FQj

99

Nodo fantasma

```
main()
{

	/*Declaracion de la lista*/
	struct Nodo *lista;
	struct Dato *dato;

	/*Se crea lista con nodo fantasma*/
	lista=crearNodo(NULL);

	/*Se agrega otro elemento*/
	dato=((struct Dato*)malloc(sizeof(struct Dato)));

	printf("Ingrese un dato");
	scanf("%d", &dato->valor);

	enlazarNodo(&lista,crearNodo(dato));
}
```

Código en Github
https://goo.gl/A06FQj

100

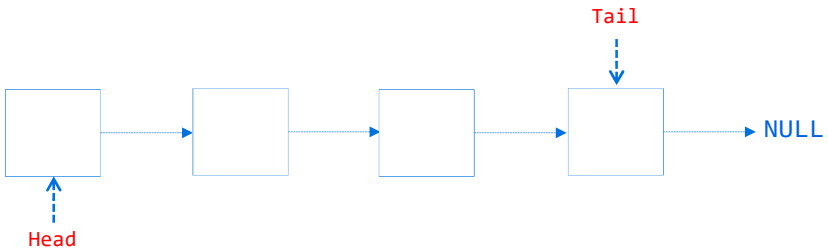
Nodo fantasma: eliminar

- Los recorridos descritos anteriormente requieren que todas las veces sea encontrado el último elemento de la lista.
- Más conveniente: tener una variable que siempre referencie al último elemento de la lista.
- Esto aplica a listas con o sin nodo fantasma (con pequeños cambios).

101

Tail

- La variable **`tail`** mantiene siempre la referencia al último elemento:



102

Tail

- El método agregarAlFinal ya no requiere recorrer la lista para ubicar el último nodo.
- La varaible tail es actualizada después de la inserción.
- Notar que el procedimiento de eliminación debe actualizar la referencia tail si se remueve el último nodo de la lista.

Tail

```c
#include <stdio.h>
#include <stdlib.h>

struct Nodo
{
	int elemento;
	struct Nodo *sig;
};

struct Nodo *crearNodo()
{
	struct Nodo *nuevo;

	/*Se obtiene la memoria al nuevo nodo*/
	nuevo=((struct Nodo*)malloc(sizeof(struct Nodo)));

	/*Se completa con los datos*/
	printf("ingrese numero: ");
	scanf("%d", &nuevo->elemento);
	nuevo->sig=NULL;

	return nuevo;
}
```

→

Código en Github
https://goo.gl/aidoFs

Tail

```
void enlazarNodo(struct Nodo **head, struct Nodo **tail, struct Nodo *nuevo)
{
	struct Nodo *rec=*head;

	if(*head==NULL)
	{
		*head=nuevo;
		*tail=*head;
	}
	else
	{
		(*tail)->sig=nuevo;
		*tail=(*tail)->sig;
	}
}
```

→

Código en Github
https://goo.gl/aidoFs

105

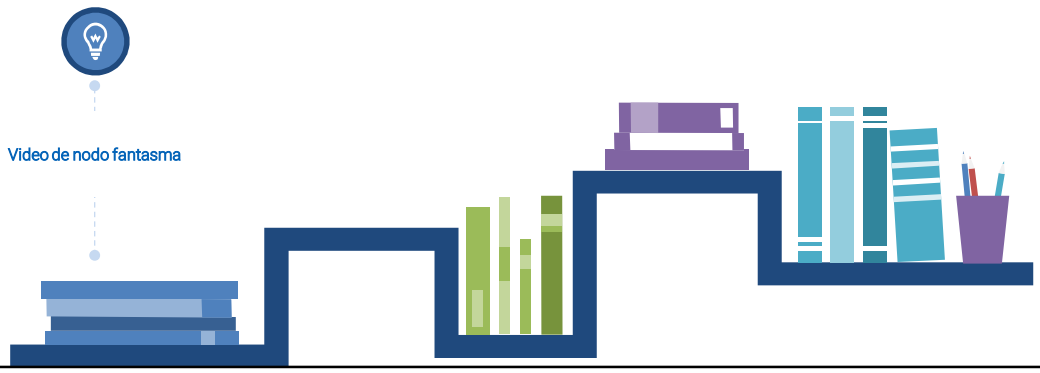
Tail

```
main()
{
	/*En este ejemplo se agregarán 3 elementos a la lista*/

	/*Declaracion de la lista*/
	struct Nodo *lista=NULL, *tail, *aux, *rec;

	/*Primer elemento*/
	aux=crearNodo();
	enlazarNodo(&lista,&tail,aux);


	/*Segundo elemento sin usar aux*/
	enlazarNodo(&lista,&tail,crearNodo());

	/*Tercer elemento sin usar aux*/
	enlazarNodo(&lista,&tail,crearNodo());

	/*Mostrar elementos de la lista*/
	rec=lista;
	while(rec!=NULL)
	{
		printf("[%d]->", rec->elemento);
		rec=rec->sig;
	}
	printf("NULL\n\n");
}
```

Código en Github
https://goo.gl/aidoFs

106

Lecturas del módulo



107