Listas

E. Rivas

Marzo de 2014

Resúmen

Listas

Listas enlazadas

Listas de otros sabores

TADs vs. estructuras de datos

Interfaz en C

Listas - defínción

lista. un tipo de dato que implementa una colección ordenada de valores, donde un valor puede aparecer más de una vez.

A los valores de las listas se les llaman *elementos*, y son un análogo de las secuencias finitas en la matemática.

Ejemplo: $\langle 12, 99, 37 \rangle$ es una lista de enteros.

Listas - propiedades y operaciones

Las listas tienen algunas propiedades que las describen:

- Longitud.
- Orden lineal.

Listas - propiedades y operaciones

Las listas tienen algunas propiedades que las describen:

- Longitud.
- Orden lineal.

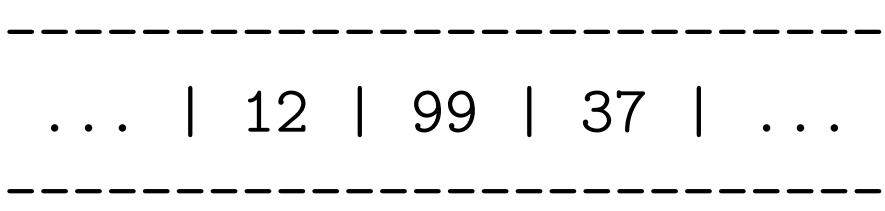
También tienen operaciones (acciones) que queremos ejecutar sobre esas listas:

- Calcular longitud.
- Recorrer elementos.
- Agregar elementos.
- Acceder a sus elementos.

Listas - arreglos

Una posible implementación de listas es mediante arreglos.

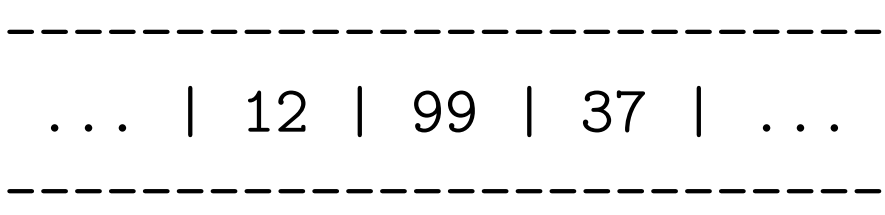
Los elementos de la lista están en correspondencia con las posiciones del arreglo.



Listas - arreglos

Una posible implementación de listas es mediante arreglos.

Los elementos de la lista están en correspondencia con las posiciones del arreglo.

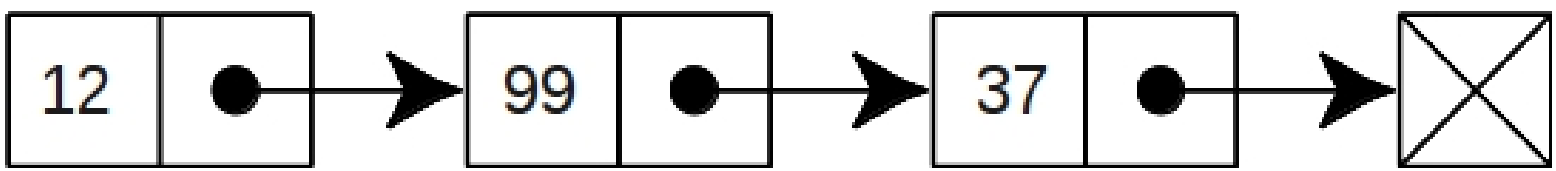


Ejercicio: realizar una estructura para representar listas con arreglos.

Listas - enlazadas

Otra posible implementación de listas es mediante *listas enlazadas*.

Pensamos a una lista compuesta por nodos, donde cada nodo contiene el dato correspondiente, y una referencia al siguiente nodo.



Listas enlazadas - C

En C utilizamos una estructura para representarlas.

```
struct nodo {
  int data;
  struct nodo *next;
};

typedef struct nodo SList;
```

Una lista es la pensamos como un puntero a un nodo inicial (`SList *`).

Listas enlazadas - creación

Podemos crear una lista creando los nodos, y luego uniendolos.

```
struct nodo n1, n2, n3;
SList *lst = &n1;

n1.data = 12;
n1.next = &n2;

n2.data = 99;
n2.next = &n3;

n3.data = 37;
n3.next = NULL;
```

Listas enlazadas - recorrido

Podemos recorrer una lista `l` con un `for`:

```
for (; l != NULL; l = l->next) {
  printf("dato: %d\n", l->data);
}
```

Ya que `l` es un puntero a una estructura, para acceder al campo de la estructura apuntada, usamos la abreviación `l->data` para `(*l).data`.

Listas enlazadas - preagregado

Si tenemos una lista `l` y queremos agregar un nodo con el valor 33 al inicio, podemos hacer:

```
struct nodo *new = malloc(sizeof(struct nodo));
new->data = 33;
new->next = l;
```

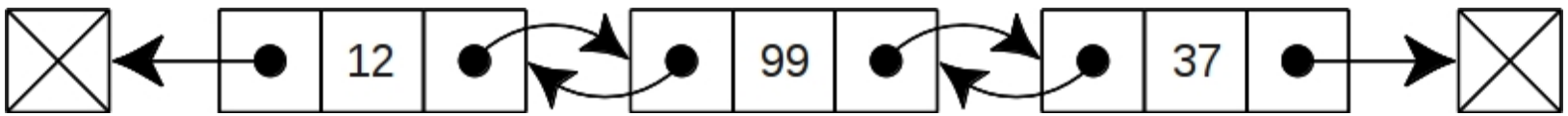
Ahora `new` es el nodo inicial de la lista con el 33 al inicio.

Listas enlazadas - ejercicios

- Implementar una función **SList *prepend(SList *, int)** que toma una lista, y un entero. La función crea un nuevo nodo cuyo dato es el entero pasado como parámetro, y cuyo siguiente es la lista parámetro. Finalmente retorna el nuevo nodo.
- De un ejemplo de cómo usar la función `prepend` para construir una lista con los elementos: 12, 99, 37 a partir de la lista vacía (`NULL`).

Listas doblemente enlazadas - idea

En una lista doblemente enlazada es que cada nodo lleva un puntero al siguiente nodo, y además un puntero al nodo anterior.




```
struct nodeDList {
  int data;
  struct nodeDList *next, *prev;
};
```

Listas doblemente enlazadas - diferencias

¿Qué ventajas trae una lista doblemente enlazada sobre una simplemente enlazada?

Listas doblemente enlazadas - diferencias

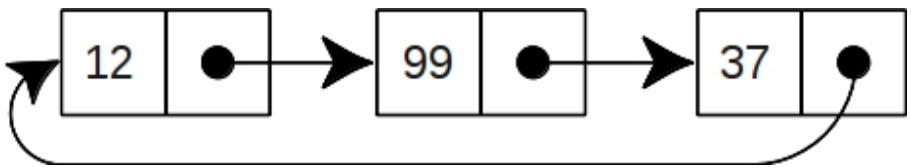
¿Qué ventajas trae una lista doblemente enlazada sobre una simplemente enlazada?

¡Se pueden recorrer en ambas direcciones!

Como contra, es necesario almacenar algo más de datos por nodo.

Listas circulares - idea

En una lista circular el último elemento tiene como siguiente al primer elemento de la lista.



Algunos problemas son inherentemente circulares, por ej. un sistema que va ejecutando una tarea sobre cada nodo y se repite indefinidamente.

TADs vs. estructuras de datos

¿Cuál es la relación entre una lista y los arreglos, listas enlazadas simples, dobles, circulares?

TADs vs. estructuras de datos

¿Cuál es la relación entre una lista y los arreglos, listas enlazadas simples, dobles, circulares?

- Las listas son *TADs*: un tipo abstracto de datos, es decir, un concepto abstracto.
- Los arreglos, listas enlazadas simples, dobles, circulares son *estructuras de datos*: maneras de organizar los datos, junto con la implementación de sus operaciones. Son concretas.

TADs vs. estructuras de datos

¿Cuál es la relación entre una lista y los arreglos, listas enlazadas simples, dobles, circulares?

- Las listas son *TADs*: un tipo abstracto de datos, es decir, un concepto abstracto.
- Los arreglos, listas enlazadas simples, dobles, circulares son *estructuras de datos*: maneras de organizar los datos, junto con la implementación de sus operaciones. Son concretas.

Varias estructuras de datos pueden concretizar un TAD.

Interfaz

Nos interesa abstraer la interfaz de una lista: las operaciones que podemos utilizar sobre estas.

```
SList *slist_prepend(SList *list, int data);

SList *slist_append(SList *list, int data);

void  slist_foreach(SList *list,
                    VisitorFuncInt visit,
                    void *extra_data);

void  slist_destroy(SList *list);
```

Interfaz - ocultación

Podemos evitar acceder directamente a los campos de las estructuras, utilizando macros auxiliares:

```
#define slist_data(l)        (l)->data
#define slist_next(l)        (l)->next
```

De esta manera, si luego se cambia la estructura, pero se mantienen macros con sentido, nuestros programas siguen siendo válidos.

Interfaz - sobre C

Cuando uno implementa una estructura de datos, conviene distribuir la información en diferentes archivos.

- `SList.h`: archivo con la estructura y prototipos de la interfaz.
- `SList.c`: implementación de las funciones.

Si otro fichero `.c` desea usar nuestra implementación, simplemente incluye el archivo `SList.h` con `include`.

Interfaz - compilación

Una vez implementadas las operaciones, compilamos el fuente con:

```
gcc -c SList.c
```

Esto generará el archivo `SList.o`. Luego, cuando compilemos otro programa que utiliza nuestras listas (e incluye `SList.h`):

```
gcc -c prog.c
gcc -o prog prog.o SList.o
```