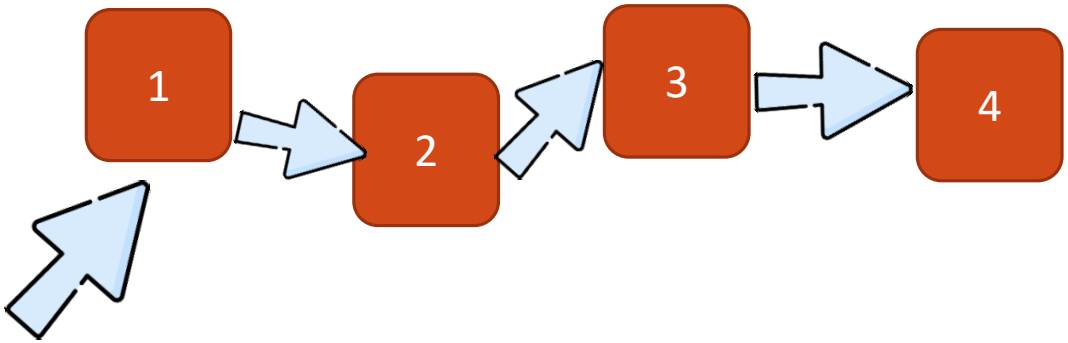


Listas, Pilas y Colas implementadas con Punteros

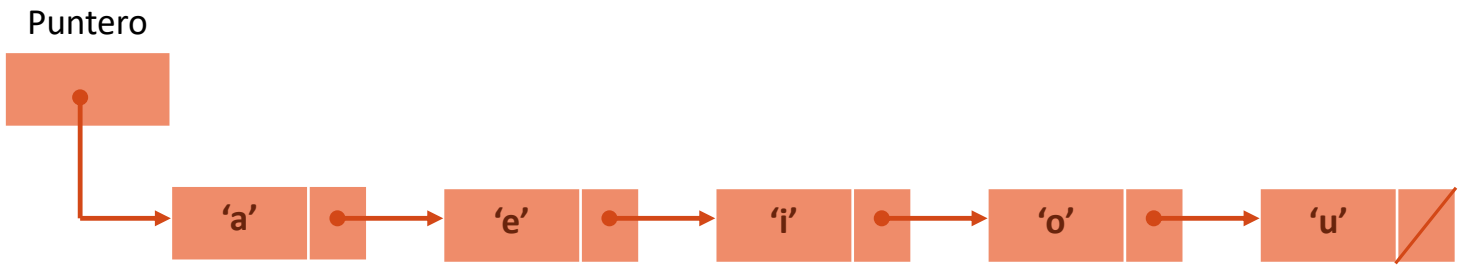
AED II – 2022

PRÁCTICO AQUINO – BURGHARDT – PRINCICH

Listas implementadas con Punteros

Una L**ista** es una colección lineal de elementos que se llaman **Nodos.**

Las representaciones para Listas con **Punteros** nos permitirán insertar y borrar elementos más fácil y eficientemente que en una implementación estática usando el tipo de datos **array**.



Listas implementadas con Punteros

Esencialmente, una lista será representada como un puntero que señala al principio (o cabeza) de la lista.

Definición:

```
typedef int tElem;

typedef struct nodo {
        tElem elem;
        struct nodo * siguiente;
}tLista;

tLista * lista;
```

Se define una estructura de tipo nodo, que contendrá dos partes:
1 – Datos.
2 – Puntero al siguiente nodo.

lista: es una variable de tipo puntero, que señala el primer nodo de la lista.

Funciones básicas de listas enlazadas

- ✔ Inicializar lista.
- ✔ Saber si la lista esta vacía.
- ✔ Insertar primer elemento.
- ✔ Inserte un elemento adelante.
- ✔ Insertar elemento (evalúa si insertar el primero o adelante).
- ✔ Eliminar el primer elemento.
- ✔ Visualizar elementos.
- ✔ Insertar elemento k-esimo.
- ✔ Eliminar elemento k-esimo.

Funciones básicas de listas enlazadas

- Inicializar lista.

```
void inicializarLista() {    /*Inicializa la lista igualando la variable
      lista = NULL;                    a NULL*/
}
```

- Lista vacía.

```
bool  listaVacia(t_lista * pLista) {   /*Devuelve verdadero o falso, según si
      return ( pLista == NULL );        la lista está vacía o no.*/
}
```

Funciones básicas: insertarPrimero

Inserta el primer nodo en la lista, incorporando el dato que recibió por parámetro.

```
void insertarPrimero( tElem pElem ) {
    /* Se crea el nodo que se va a insertar */
    tLista * nuevoNodo;

    /* Se asigna memoria al nodo */
    nuevoNodo = ( tLista * ) malloc( sizeof( tLista ));

    /* Se asigna el dato recibido al componente
    correspondiente al elemento */
    nuevoNodo->elem = pElem;

    /* Se indica que el primer nodo apunta a NULL */
    nuevoNodo->siguiente = NULL;

    /* Se agrega el nodo a la lista: la lista debe apuntar a
    nuevoNodo */
    lista = nuevoNodo;

    printf("Primer elemento insertado!\n");
}
```

Funciones básicas: insertarAdelante

Inserta un nodo adelante de la lista.

```
void insertarAdelante( tElem pElem ) {
    /* Se crea el nodo que se va a insertar */
    tLista * nuevoNodo;

    /* Se asigna memoria al nodo */
    nuevoNodo = ( tLista * ) malloc( sizeof( tLista ) );

    /* Se asigna el dato recibido al componente
    correspondiente al     elemento */
    nuevoNodo->elem = pElem;

    /* Como la inserción es por la parte de adelante de la
    lista, se indica que al nuevo nodo le sigue el resto de la
    lista */
    nuevoNodo->siguiente = lista;

    /* Como en nuevoNodo quedó la lista completa, nos
    queda indicar que la lista que se recibe como parámetro
    es igual a nuevoNodo */
    lista = nuevoNodo;

    printf("Elemento insertado!\n");
}
```

Funciones básicas: insertarElemento

Esta función se encarga de evaluar si inserta el primer nodo o uno adelante, e invocar a la función correcta según corresponda.

```
void insertarElemento( tElem pElem ) {
        if ( lista == NULL )
                insertarPrimero( pElem );
        else
                insertarAdelante( pElem );
}
```

Funciones básicas: insertarK

Esta función se encarga de insertar un elemento en la k-ésima posición.

```
void insertarK( int k, tElem nuevoDato ) {
        tLista * nuevoNodo, * aux;
        int i;
        aux = lista;

        /* El bucle avanza aux hasta el nodo K-1 */
        for(i = 1; i < k-1; i++) {
                aux = aux->siguiente;
        }

        /* Se reserva espacio para el nodo a insertar */
        nuevoNodo = malloc(sizeof(tLista));

        /* Se asigna el dato recibido al componente
        correspondiente al elemento */
        nuevoNodo->elem = nuevoDato;

        /* Se actualizan los punteros */
                /* 1. Se indica a qué nodo tiene que apuntar nuevoNodo: al
                siguiente de aux */
        nuevoNodo->siguiente = aux->siguiente;
                /* 2. Se indica a qué nodo tiene que apuntar aux: a
                nuevoNodo */
        aux->siguiente = nuevoNodo;

        printf("Elemento insertado en la posicion %d!\n", k);
}
```

Funciones básicas: eliminarPrimero

Elimina el primer nodo de la lista, desplazando el puntero al siguiente nodo y liberando la memoria (free).

```
void eliminarPrimero() {
        tLista * nodoSuprimir;

        /* Se guarda en una variable auxiliar el primer nodo de la
        lista */
        nodoSuprimir = lista;

        /* Se avanza el puntero una vez, es decir se pasa al
        siguiente nodo de la lista */
        lista = lista->siguiente;

        /* Se libera la memoria del nodo a suprimir que contenía el
        primer elemento de la lista  */
        free( nodoSuprimir );

        /* Se asigna NULL a la variable auxiliar que guarda el nodo
        a suprimir */
        nodoSuprimir = NULL;

        printf("Primer elemento eliminado!\n");
}
```

Funciones básicas: eliminarK

Esta función se encarga de eliminar el elemento ubicado en la k-ésima posición.

```
void eliminarK( int k ) {
        tLista * nodoSuprimir, * aux;
        int i;
        aux = lista;

        /* El bucle avanza aux hasta el nodo K-1 */
        for(i = 1; i < k-1; i++) {
                aux = aux->siguiente;
        }
        /* Se resguarda el nodo que se va a suprimir en la variable
        nodoSuprimir */
        nodoSuprimir = aux->siguiente;

        /* Se indica a qué nodo tiene que apuntar aux: al siguiente del
        que se va a   eliminar */
        aux->siguiente = nodoSuprimir->siguiente;

        /* Se libera la memoria del nodo a suprimir que contenía el
        elemento de la posición K de la lista  */
        free( nodoSuprimir );

        /* Se asigna NULL a la variable auxiliar que guarda el nodo a
        suprimir */
        nodoSuprimir = NULL;

        printf("Elemento de la posicion %d eliminado\n", k);
}
```

Funciones básicas: visualizarElementos

Recorre la lista para acceder a cada elemento.

```
void visualizarElementos( tLista * pLista ) {
        /* Se deberá utilizar una variable auxiliar para recorrer la lista */
        tLista * aux;
        aux = pLista;

        if ( !listaVacia( pLista ) ) {
                /* Se puede recorrer la lista */
                printf( "\n*** Detalle de elementos en la lista ***\n" );
                while(aux != NULL) {
                        printf("%d ", aux->elem);
                aux = aux->siguiente;
                }
        }else {
                printf( "\nLa lista esta vacia!!\n" );
        }
        printf("\n\n" );
}
```

Bibliografía

Material teórico de la catedra “Algoritmos y Estructuras de Datos II”.

Pablo A. Sznajdleder. Algoritmos a fondo, con implementaciones en C y Java. Alfaomega. 2012.

Gustavo López, Ismael Jeder, Augusto Vega. Análisis y diseño de algoritmos. Implementaciones en C y Pascal. Alfaomega. 2009.

Hemant Jain. Problem Solving in Data Structures & Algorithms. Using C. First Edition. 2017.