

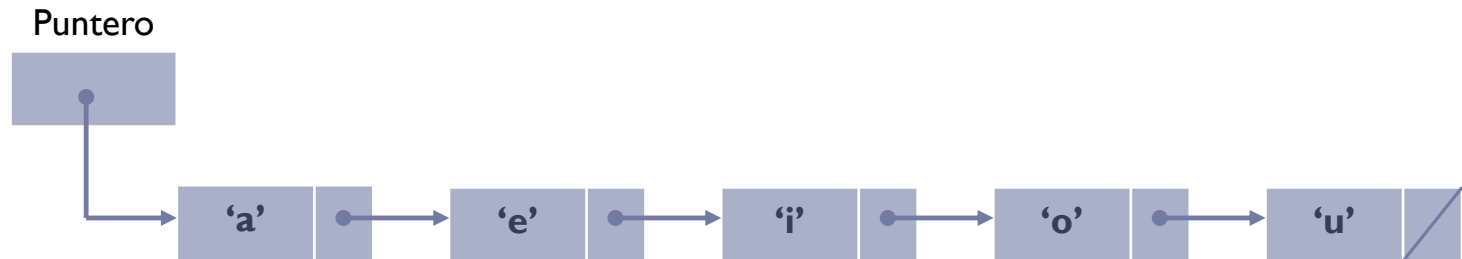
Listas, Pilas y Colas implementadas con Punteros

AED II – 2019

Práctico G1

Listas implementadas con Punteros

- ▶ Una **Lista** 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 struct nodo {  
    int elem;  
    struct nodo *siguiente;  
}tLista;
```

```
tLista * v_lista;
```



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

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



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  
    v_lista = NULL;          a NULL*/  
}
```

► Lista vacía.

```
bool listaVacía() {          /*Devuelve verdadero o falso, según si la  
    if (v_lista == NULL) {    lista está vacía o no.*/  
        return true;  
    } else {  
        return false;  
    }  
}
```



Funciones básicas: Insertar

```
Void insertarPrimero(int pElem) {           /*Inserta el primer nodo en la lista, incorporando el dato
    v_lista = malloc(sizeof(tLista));       que recibió por parámetro*/
    v_lista->elem = pElem;
    v_lista->siguiente = NULL;
}

void insertarAdelante(int pElem) { /*Inserta un nodo adelante de la lista. */
    tLista * nuevoNodo;
    nuevoNodo = malloc(sizeof(tLista));
    nuevoNodo->elem = pElem;
    nuevoNodo->siguiente = v_lista;
    v_lista = nuevoNodo;
}

void insertarElemento(int pElem) { /*La función insertarElemento se encarga de evaluar si inserta
    if (v_lista == NULL){                el primer nodo ó uno adelante, e invocar a
        insertarPrimero(pElem);          la función correcta según corresponda. */
        printf("Primer elemento insertado!\n");
    }else{
        insertarAdelante(pElem);
        printf("Elemento insertado!\n");
    }
}
```



Funciones básicas: Insertar K

```
Void insertarK(int k, int nuevoDato) {  
    tLista * nuevoNodo, * aux;  
    int i;  
    aux = v_lista;  
  
    for(i = 1; i < k-1; i++) {  
        aux = aux->siguiente;    /*El bucle avanza aux hasta el nodo k-1,  
                                posicionando el puntero en la posición correcta para  
                                la inserción*/  
    }  
    nuevoNodo = malloc(sizeof(tLista));  
    nuevoNodo->elem = nuevoDato;  
    nuevoNodo->siguiente = aux->siguiente;    /*Se actualizan los punteros*/  
    aux->siguiente = nuevoNodo;  
}
```



Funciones básicas: Eliminar primer nodo

```
void eliminarPrimero() {  
    tLista * aux;  
    aux = v_lista;  
    v_lista = v_lista ->siguiente;  
    free(aux);  
    printf("Primer elemento eliminado!\n");  
}
```

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



Funciones básicas: Eliminar K

```
void eliminarK(int k) {  
    tLista * nodoSuprimir, * aux;  
    int i;  
    aux = v_lista;  
    for(i = 1; i < k-1; i++) {  
        aux = aux->siguiente; /*El bucle avanza aux hasta el nodo k-1,  
                                posicionando el puntero en la posición correcta para  
                                la inserción*/  
    }  
    nodoSuprimir = aux->siguiente;  
    aux->siguiente = nodoSuprimir->siguiente; /*Se actualizan los punteros*/  
    free(nodoSuprimir); /*Se libera la memoria*/  
  
    printf("Elemento de la posición %d eliminado\n", k);  
}
```



Funciones básicas: Visualizar elementos

```
void visualizarElementos() {  
    tLista *aux;    /*Se genera un auxiliar para recorrer la lista*/  
    aux = v_lista;  
    if (listaVacia() == false)    {  
        while(aux != NULL) {  
            printf("%d ", aux->elem);  
            aux = aux->siguiente;  
        }  
        /*Si la lista no esta vacía,  
        Se recorre hasta que el auxiliar  
        es Null*/  
    }else printf( "\nLa lista esta vacia!!\n" );  
}
```




Pilas implementadas con Punteros

- ▶ Una pila es un tipo de lista en el que todas las inserciones y eliminaciones de elementos se realizan por el mismo extremo de la lista. (LIFO).
- ▶ Definición:

```
typedef struct nodo {  
    int elem;  
    struct nodo *siguiente;  
}tPila;
```

```
tPila * v_pila;
```

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



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

Funciones básicas de las Pilas

- ▶ Inicializar pila.
- ▶ Saber si la pila esta vacía.
- ▶ Insertar nodo.
- ▶ Eliminar nodo.
- ▶ Visualizar elementos.
- ▶ Cima.



Funciones básicas de las Pilas

► Inicializar pila.

```
void inicializarPila() { /*Inicializa la pila igualando la variable  
                           v_pila = NULL; a NULL*/  
}
```

► Pila vacía.

```
bool pilaVacía() { /*Devuelve verdadero o falso, según si la  
                    pila está vacía o no.*/  
    if (v_pila == NULL) {  
        return true;  
    } else {  
        return false;  
    }  
}
```



Funciones básicas: Insertar

```
void apilar(int pElem) {  
    tPila * aux;           /*Inserta un nodo en la pila, incorporando el dato que  
    aux = v_pila;          recibió por parámetro*/  
    v_pila = malloc(sizeof(tPila));  
    v_pila->elem = pElem;  
    v_pila->siguiente = aux;  
  
    printf("Elemento insertado!\n");  
}
```



Funciones básicas: Eliminar

```
void desapilar() {  
    tPila * aux;  
    aux = v_pila;  
    v_pila = v_pila->siguiente;  
    free(aux);  
    printf("Elemento de la cima eliminado!\n");  
}
```

*/*Elimina un nodo de la pila, desplazando el
puntero al siguiente nodo y liberando la memoria
(free)*/*



Funciones básicas: Visualizar elementos

```
void visualizarElementos() {  
    tPila *aux;      /*Se genera un auxiliar para recorrer la pila*/  
    aux = v_pila;  
    if (pilaVacía() == false) {  
        printf( "\nElementos en la pila: \n" );  
        while(aux != NULL) {  
            printf("%c \n", aux->elem);  
            aux = aux->siguiente;  
        }  
        /*Si la pila no está vacía,  
        Se recorre hasta que el auxiliar  
        es Null*/  
    } else printf( "\nLa pila está vacía!!\n" );  
}
```



Funciones básicas: Cima

Opción 1: Devuelve el dato que está en la cima de la pila, correspondiente al primer nodo

```
int cima() {  
    return v_pila->elem;  
}
```

Opción 2: Devuelve el nodo que está en la cima de la pila. Puede utilizarse esta opción para los casos donde el nodo contiene más de un campo con datos.

```
tPila cima() {    /*La función devuelve un nodo (tipo tPila)*/  
    tPila auxiliar;  
    auxiliar.elem =v_ pila->elem;  
    auxiliar.siguiente = v_pila->siguiente;  
    return auxiliar;  
}
```



Colas implementadas con Punteros

- ▶ La Cola es una lista en la que las inserciones se realizan por un extremo(**final**) y las eliminaciones se realizan por el otro extremo (**principio de la lista o frente**).
- ▶ El primero en llegar es el primero en salir; por esto, las colas también se llaman listas **FIFO**.

```
typedef struct nodo {  
    int elem;  
    struct nodo *siguiente;  
} tApNodo;
```

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


```
typedef struct {  
    tApNodo * principio;  
    tApNodo * final;  
} tCola;
```

```
tCola v_colas;
```

v_colas: es una variable registro compuesta por dos punteros, uno apunta al principio de la cola y el otro al final.

Ejemplo:

```
typedef struct nodo {  
    int codprod;  
    int stock;  
    float precio;  
    struct nodo * siguiente;  
}tApNodo;
```



DATOS

```
typedef struct {  
    tApNodo * principio;  
    tApNodo * final;  
}tCola;
```

```
tCola vCola;
```



Colas implementadas con Punteros

- ▶ Inicializar cola.
- ▶ Saber si la cola esta vacía.
- ▶ Insertar nodo.
- ▶ Eliminar nodo.
- ▶ Visualizar elementos.
- ▶ Primer elemento.



Funciones básicas de las Colas

► Inicializar cola.

```
void inicializarCola() {  
    v_cola.principio = NULL;  
    v_cola.final = NULL;  
}
```

*/*Inicializa la cola igualando los punteros principio y final a NULL*/*

► Cola vacía.

```
bool colaVacía() {  
    if (v_cola.final == NULL)  
        return true;  
    else  
        return false;  
}
```

*/*Devuelve verdadero o falso, según si el puntero que indica el final de la cola es Null*/*



Funciones básicas: Insertar

```
void push(int pElem) {
    tApNodo * nuevoNodo;
    nuevoNodo = malloc (sizeof(tApNodo));
    nuevoNodo->elem = pElem;
    nuevoNodo->siguiente = NULL;

    if (colaVacia() == true) {
        v_cola.principio = nuevoNodo;
        v_cola.final = nuevoNodo;
    } else {
        v_cola.final->siguiente = nuevoNodo;
        v_cola.final = nuevoNodo;
    }
    printf("Elemento insertado!\n");
}
```

*/*Se genera un nuevo nodo para insertar en la cola, incorporando los datos que se recibieron por parámetros*/*

*/*Se actualizan los punteros. Si la cola está vacía, deben actualizarse principio y final.*

Si la cola ya contiene nodos, sólo se actualiza final./*



Funciones básicas: Eliminar

```
void pop() {                                     /*Se define un nodo auxiliar que se utilizará para liberar la memoria*/
    tApNodo * nodoAux;
    if (colaVacia() == true) {
        printf("No hay elementos en cola\n");
    } else {
        if (v_cola.principio == v_cola.final) {   /*Si la cola no está vacía, se controla si hay
                                                    uno o más elementos.
                                                    Si hay un solo nodo, se libera la memoria y
                                                    tanto principio como final serán igual a Null*/
            nodoAux = v_cola.principio;
            free(nodoAux);
            v_cola.principio = NULL;
            v_cola.final = NULL;

        } else {
            nodoAux = v_cola.principio;
            v_cola.principio = nodoAux->siguiente;
            free(nodoAux);

        }
    }
}
```



Funciones básicas: Visualizar elementos

```
void visualizarElementos() {  
    tApNodo * colaPrincipio;    /*Se genera un auxiliar para recorrer la cola*/  
    if (colaVacia() == true){  
        printf("No hay elementos para mostrar!\n");  
    } else {  
        printf("Elementos en la cola: \n");  
        colaPrincipio = vCola.principio;    /*Si la cola no esta vacía,  
        Se recorre hasta que el auxiliar  
        es Null*/  
        while (colaPrincipio != NULL) {  
            printf("%.2ft", colaPrincipio->elem);  
            colaPrincipio = colaPrincipio->siguiente;  
        }  
        printf("\n\n");  
    }  
}
```



Funciones básicas: Primer Elemento

Opción 1: Devuelve el dato que está en el frente de la cola, correspondiente al primer nodo

```
int primerElemento() {  
    return vCola.principio->elem;  
}
```

Opción 2: Devuelve el nodo que está en el frente de la cola. Puede utilizarse esta opción para los casos donde el nodo contiene más de un campo con datos.

```
tApNodo primerElemento() {    /*La función devuelve un nodo (tipo tApNodo)*/  
    tApNodo auxiliar;  
    auxiliar.elem = vCola.principio->elem;  
    auxiliar.siguiente = vCola.principio->siguiente;  
    return auxiliar;  
}
```



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.

