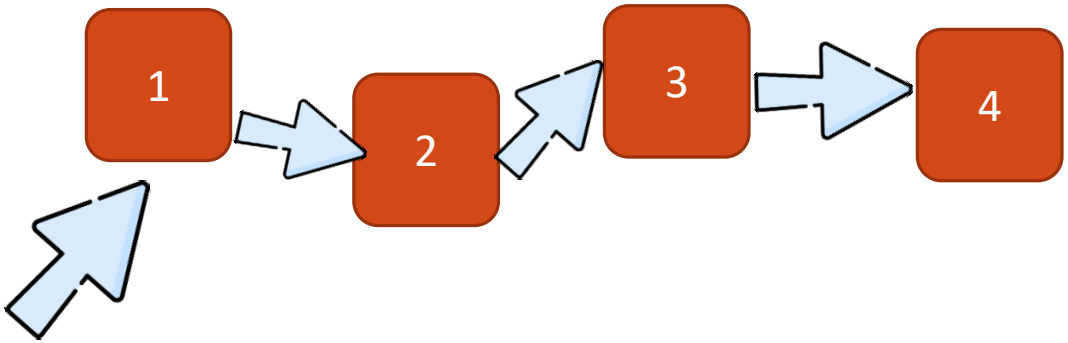


Listas, Pilas y Colas implementadas con Punteros

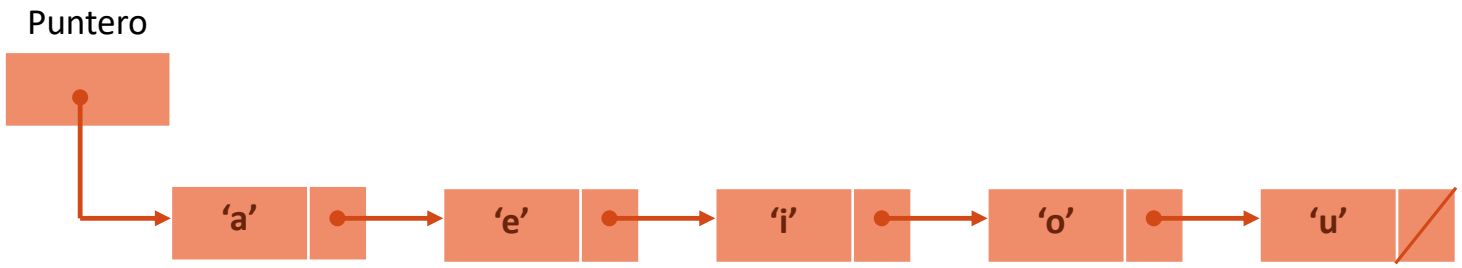
AED II – 2022

PRÁCTICO AQUINO – BURGHARDT - PRINCICH

Recordando…
Listas enlazadas

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;
```

Funciones básicas de la Lista Enlazada:

- ✔ 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.

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 char tString [25];
typedef struct {
        int dni;
        tString nombre;
} tElem;
typedef struct nodo {
        tElem datos;
        struct nodo * siguiente;
} tPila;
tPila * pila;
```

Funciones básicas de la Pila con punteros:

- ✔ Inicializar Pila.
- ✔ Saber si la pila esta vacía.
- ✔ Inserte un nodo en la pila.
- ✔ Eliminar el un nodo de la pila.
- ✔ Visualizar elementos.
- ✔ Cima.

Cola

¿Cómo implementamos una estructura de Cola con punteros?

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).
FIFO.

Definición:

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

El nodo esta compuesto por:
1- Datos
2 – Puntero al siguiente nodo

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

La estructura tCola contendrá los apuntadores al principio y final de la cola.

```
tCola v_cola;
```

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

Funciones Básicas de Colas implementadas con punteros

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

Funciones básicas

INICIALIZAR COLA

```
void inicializarCola(){

/*Se inicializan en NULL los dos campos de la cola, correspondientes a principio y final.*/

        v_cola.principio = NULL;

        v_cola.final = NULL;

}
```

COLA VACÍA

```
bool  colaVacia(tCola pCola){

/*Devuelve verdadero o falso, según si la cola está vacía o no.*/

        return (pCola.final == NULL);

}
```

Insertar un nodo: push()

```
void push(tNodo pDatos) {
        tNodo * nuevoNodo;
        nuevoNodo = malloc (sizeof(tNodo));
        nuevoNodo->codprod = pDatos.codprod;
        nuevoNodo->stock = pDatos.stock;
        nuevoNodo->precio = pDatos.precio;
        nuevoNodo->siguiente = NULL;
        if (colaVacia(v_cola)== true){
                v_cola.principio = nuevoNodo;
                v_cola.final = nuevoNodo;
         } else {
                v_cola.final->siguiente = nuevoNodo;
                v_cola.final = nuevoNodo;
        }
         printf("Elemento insertado!\n");
}
```

Eliminar un nodo: pop()

```
void pop() {
        tNodo * nodoAux;
        if (colaVacia(v_cola) == true){
                printf("No hay elementos en cola\n");
        } else {
                if (v_cola.principio == v_cola.final){
                        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);
                }
        }
}
```

Visualizar elementos

```
void visualizarElementos(tCola pCola){
        tNodo * colaPrincipio;
        if (colaVacia(pCola) == true){
                printf("No hay elementos para mostrar!\n");
         } else {
                printf("Productos: \n");
                printf("Codigo     Stock     Precio unitario: \n");
                colaPrincipio = pCola.principio;
                while (colaPrincipio != NULL) {
                        printf("%.2d\t", colaPrincipio->codprod);
                        printf("%.2d\t", colaPrincipio->stock);
                        printf("%.2f\n", colaPrincipio->precio);
                        colaPrincipio = colaPrincipio->siguiente;
                }
                printf("\n\n");
         }
}
```

Función que retorna el primer elemento.

```
tNodo primerNodo(tCola pCola){
    tNodo auxiliar;
    auxiliar.codprod = pCola.principio->codprod;
    auxiliar.stock = pCola.principio->stock;
    auxiliar.precio = pCola.principio->precio;
    auxiliar.siguiente = pCola.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.