Apuntadores

- Es una variable que contiene una dirección de memoria.
- Generalmente, esa dirección es la dirección de otra variable de memoria.
- Si una variable contiene la dirección de otra variable, entonces se dice que la primera apunta a la segunda.

Apuntadores

Declaración:

```
tipo_dato *nombre_variable;
```

Operadores

- `&` Devuelve la dirección de memoria de su operando. Por ejemplo, en la línea $m = \&cuenta$; m guarda la dirección de la variable cuenta.
- `*` Devuelve el valor de la variable localizada en la dirección de memoria a la que se apunta. Por ejemplo: si a la variable $cuenta$ le asignamos el valor de 1200, entonces $*m$ tendrá el valor de 1200.

Apuntadores

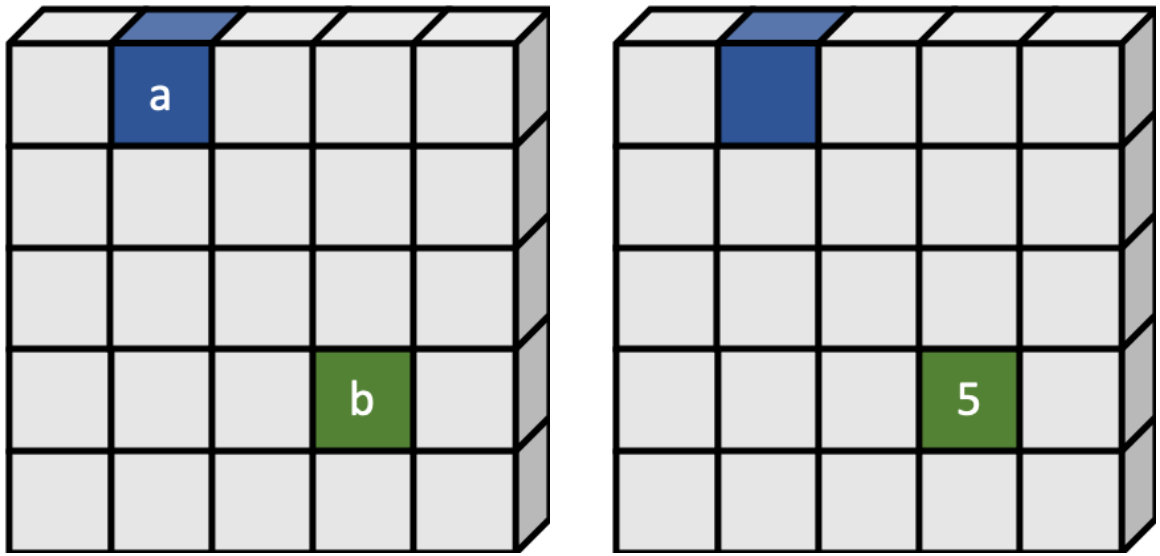
- El **tipo_dato** del apuntador determina el tipo de variables a las que puede apuntar el apuntador.

Apuntadores

Por ejemplo:

- Supongamos que existen el apuntador a y la variable b.

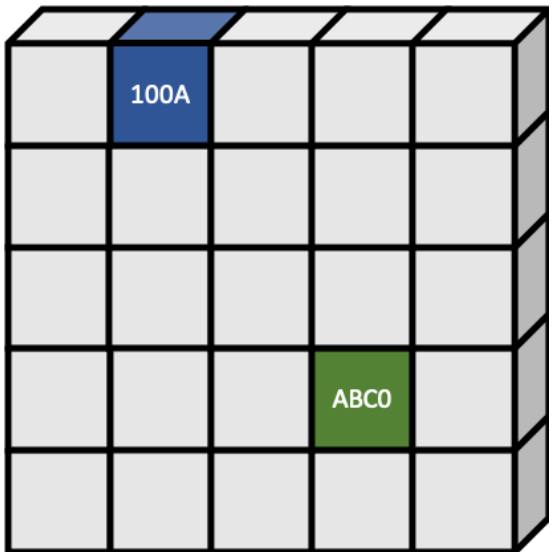
```
int *a;
int b=5;
```



Apuntadores

Por ejemplo:

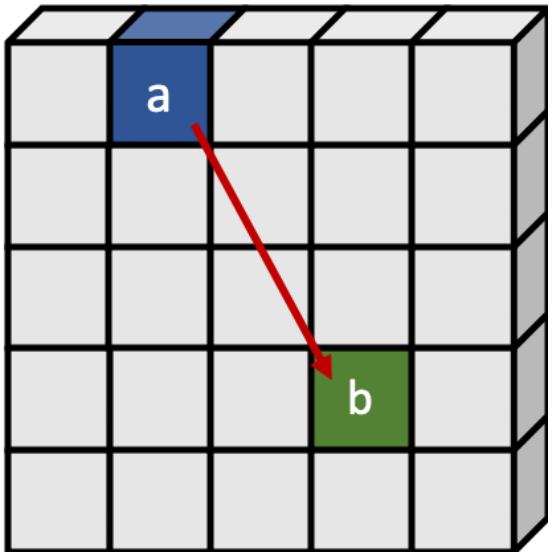
- Ahora digamos que las variables a y b están ubicadas en las direcciones 100A y ABC0, respectivamente.



Apuntadores

Por ejemplo:

- Si se ejecuta la línea $a = \&b$; entonces la celda 100A contendrá la dirección de b.



Apuntadores

Los apuntadores se utilizan en situaciones en las que pasar valores es difícil o indeseable, como podrían ser los siguientes casos:

- Para regresar más de un valor de una función.
- Pasar arreglos fácilmente de una función a otra.
- Manipular arreglos más fácilmente, manipulando punteros a ellos, o a partes de ellos, en vez de mover los arreglos en sí.
- Crear estructuras de datos complejas, como son las listas encadenadas a y árboles binarios, donde una estructura de datos debe tener referencias a otras estructuras de datos.
- Para comunicar información acerca de la memoria.

Ejemplo: manejo de apuntadores

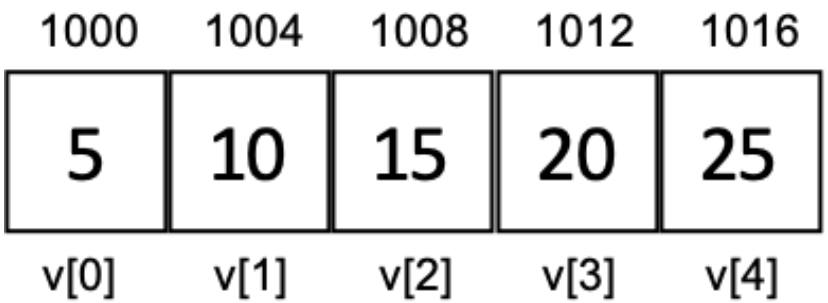
```c
#include<stdio.h>
#include<conio.h>

int main()
{
     int *direccion;
     int edad=0;
     printf("Cuantos anios tienes");
     scanf("%d", &edad);
     direccion = &edad;
     printf("\nTu edad esta guardada en la posicion de memoria: %p",
          direccion);
     printf("\nTu edad es: %d", *direccion);
     printf("\nEl apuntador a tu edad esta guardado en la posicion de memoria
          : %p", &direccion);
     getch();
}
```

Apuntadores y vectores

- Un vector es un apuntador que contiene la dirección de la primera (0-ésima) posición del vector.

```
int v[5] = {5,10,15,20,25};
int *dir;
```



- v = 1000

Apuntadores y vectores

- Por lo tanto las siguientes expresiones son equivalente:
  - dir = &v[0] $\leftrightarrow$ dir = v
  - x = *dir $\leftrightarrow$ x = v[0]
  - *(v + 1) $\leftrightarrow$ v[1]
  - *(v + i) $\leftrightarrow$ v[i]

Aritmética de apuntadores

```
int v[5], *dir;
dir = v;
```

- dir $\rightarrow$ Apunta a la posición inicial del vector.
- dir + 0 $\rightarrow$ Apunto a la posición inicial del vector.
- dir + 1 $\rightarrow$ Apunta a la segunda posición del vector.
- dir + i $\rightarrow$ Apunta a la posición i + 1 del vector.

```
dir = &v[4];
```

- dir $\rightarrow$ Apunta a la última posición del vector.
- dir - 1 $\rightarrow$ Apunta a la penúltima posición del vector.
- dir - i $\rightarrow$ se refiere a la posición 4 - i en v.

Ejemplo: manejo de vectores con apuntadores

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

void imprimir(int *v)
{
     int i=0;
     do
     {
          printf("%d \n", *(v+i));
          i++;
     }while(i<5);
}

void main()
{
     int vector[5] = {3,0,2,2,4};
     imprimir(vector);
     getch();
}
```

Ejercicio

- Elabora un programa que pida dos vectores en $R3$ y que implemente una función para calcular el producto punto.

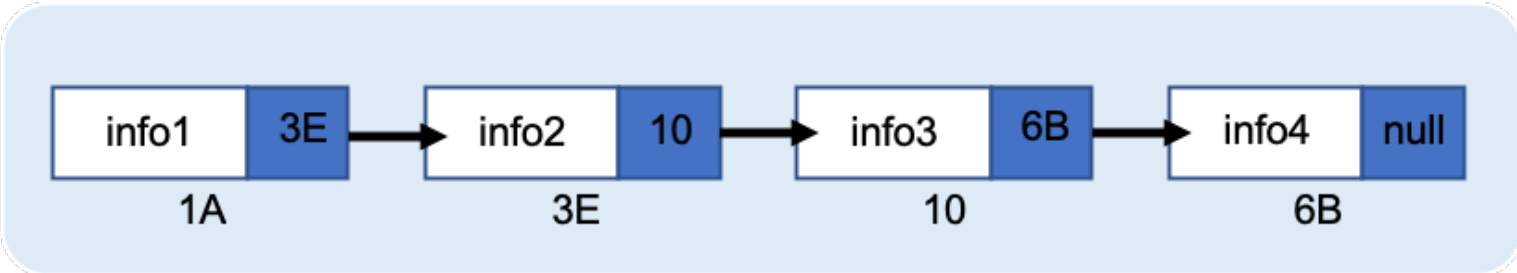
Ejercicios

- Elaborar un programa que implemente una pila estática. El programa debe mostrar un menú de opciones donde el usuario pueda elegir la operación que desea realizar, y cada operación debe ser ejecutada a través de una función. Implemente las operaciones básicas y las complementarias de pilas.
- Elaborar un programa que implemente una cola estática. El programa debe mostrar un menú de opciones donde el usuario pueda elegir la operación que desea realizar, y cada operación debe ser ejecutada a través de una función. Implemente las operaciones básicas y las complementarias de colas.

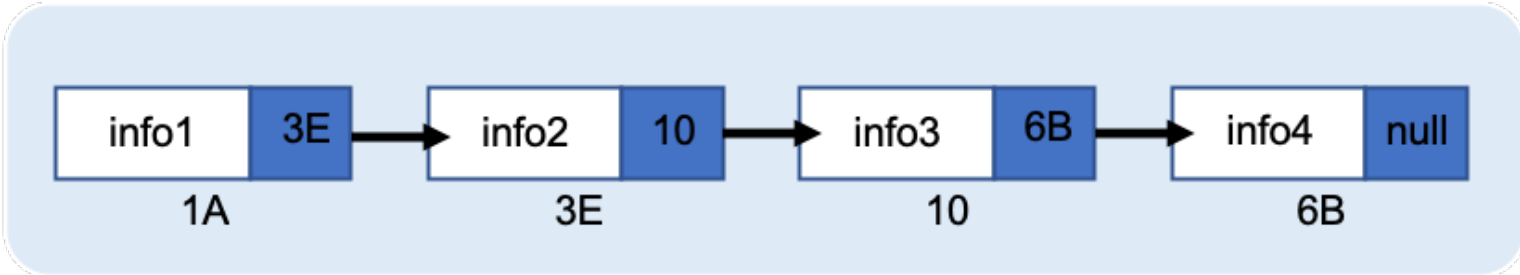
Nota: Utilice los conceptos de apuntadores para realizar ambos programas.

Lista enlazada

- Conjunto de nodos conectados entre sí, situados en direcciones de memoria no consecutivas.
- Cada nodo se compone de una sección de datos y una referencia al siguiente nodo de la lista.



Lista enlazada



- Para insertar un elemento cualquiera debemos ir recorriendo la lista.
- Ventajas con respecto a vectores: una inserción en medio de la lista no requiere mover todos los elementos que se encuentran después del punto de inserción, mientras que en un vector es necesario recorrer todos los elementos para abrir espacio al nuevo elemento.
- Si se permite el acceso a la lista sólo por el primer elemento, entonces se comporta como una **pila**.
- Si las inserciones se realizan por el último elemento, y los accesos sólo por el inicio, entonces se comporta como una **cola**.

Lista enlazada: operaciones básicas

- Insertar elementos
- Mostrar elementos
- Eliminar elementos

Lista enlazada: implementación para pilas (1/5)

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

struct Nodo
{
    int dato;
    struct Nodo *ant;
};

int vacia(struct Nodo *tope)
{
    return (!tope)? 1:0;
}

struct Nodo *crearNodo(int n)
{
    //PASO 1: Creamos el nuevo nodo
    struct Nodo *nuevo_nodo;
    nuevo_nodo = (struct Nodo*)malloc(sizeof(struct Nodo));
    //PASO 2: Asignar el dato al nuevo nodo
    nuevo_nodo->dato = n;
    nuevo_nodo->ant = NULL;
    return(nuevo_nodo);
}
```

Lista enlazada: implementación para pilas (2/5)

```
struct Nodo *push(struct Nodo *tope, int n)
{
     struct Nodo *aux;
     aux = crearNodo(n);
     aux->ant = tope;
     tope = aux;
     return(tope);
}

void mostrar(struct Nodo *tope)
{
     struct Nodo *aux;
     if(!tope)
          printf("La pila esta vacia");
     else
     {
          aux = tope;
          do{
               printf("%d ", aux->dato);
               aux = aux->ant;
          }while(aux != NULL);
     }
     getch();
}
```

Lista enlazada: implementación para pilas (3/5)

```
struct Nodo *pop(struct Nodo *tope [redacted])
{
     struct Nodo *aux;
     [redacted]
     aux = tope;
     [redacted]
     tope = tope->ant;
     free(aux);
     return(tope);
}
```

Lista enlazada: implementación para pilas (4/5)

```
void main()
{
    struct Nodo *tope = NULL;
    int dato, op;
    do{
        printf("PROGRMA QUE IMPLEMENTA PILAS CON LISTAS ENLAZADAS\n\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Mostrar\n");
        printf("4. Salir\n");
        printf("Opcion: ");
        scanf("%d", &op);
        switch(op)
        {
            case 1:   printf("Introduce un numero: ");
                      scanf("%d", &dato);
                      if(tope == NULL)
                          tope = crearNodo(dato);
                      else
                          tope = push(tope,dato);
                      break;
```

Lista enlazada: implementación para pilas (5/5)

```
            case 2:    if(!vacia(tope))
                         {
                              printf("El dato eliminado es: %d\n", tope->dato);
                              tope = pop(tope, &dato);
                              printf("El nuevo dato en la cima es: %d\n", tope->
                                  dato);
                         }
                         else
                              printf("Pila vacia\n");
                         getch();
                         break;
            case 3:    if(!vacia(tope))
                              mostrar(tope);
                         else printf("Pila vacia\n");
                         break;
            case 4:    break;
        }

    }while(op != 4);
}
```