

# TEMA 3:

Fundamentos para la construcción de código a partir del algoritmo

# TEMA 3:

## Apuntadores

# Objetivo

- El alumno construirá programas utilizando el lenguaje de programación C a través de un análisis y modelado algorítmico previo.

# Apuntador

- Es una variable que **contiene la dirección de memoria** de otra variable.
- Se puede apuntar a cualquier tipo de dato y variable.
- El operador unario & **devuelve la dirección de memoria de la variable a la que apunta**.
- El operador de referencia \* **devuelve o da acceso al contenido de la variable a la que se apunta**.
- El nombre de un arreglo es siempre un apuntador a la dirección de memoria del primer elemento del arreglo.

# Errores comunes en apuntadores

- No asignar en primer lugar una dirección de memoria válida al apuntador antes de usarlo.
- No inicializar el apuntador en NULL;

INCORRECTO	CORRECTO
<pre>int *apu; *apu= 100;</pre>	<pre>int *apu= NULL; //int* apu= NULL;  int y=5; apu= &amp;y; *apu= 100;</pre>

# Proceso 1

variables

dato

Memoria  
física

2

Direcciones  
de memoria

100

apunt

?

200

- int dato= 2;
- int \*apunt;

variables

dato

Memoria  
física

2

Direcciones  
de memoria

100

- apunt= &dato;

apunt

100

200

- \*apunt= 5;

variables

dato

Memoria  
física

5

Direcciones  
de memoria

100

apunt

100

200

```

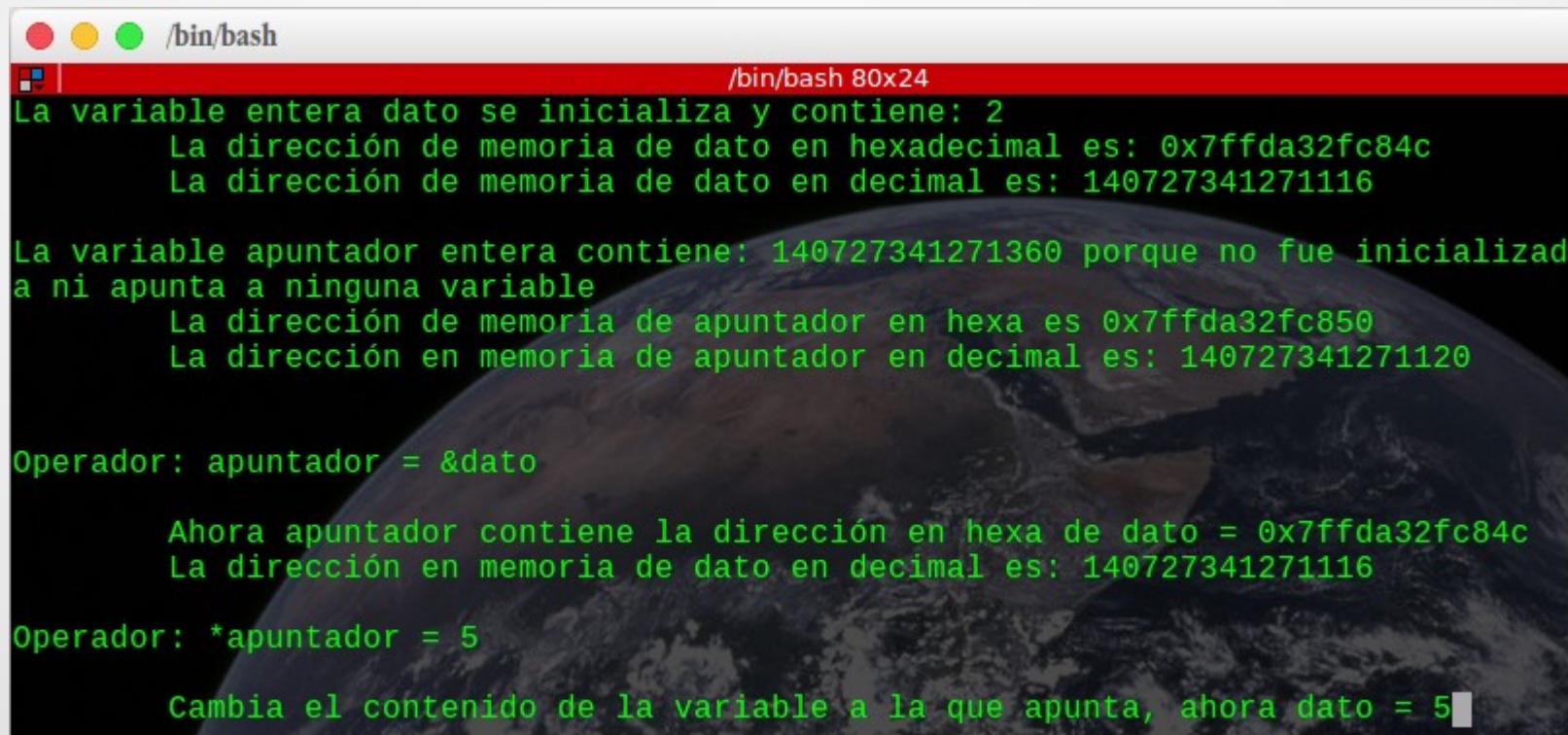
1  /*programa que apunta a una variable
2   * hecho por huicho*/
3
4  #include <stdio.h> //para printf y getchar
5
6  int main(int argc, char* argv[])
7  {
8      int dato=2; //variable dato de tipo entera inicializada en 2
9
10     printf("La variable entera dato se inicializa y contiene: %d",dato);
11     printf("\n\tLa direcci%cn de memoria de dato en hexadecimal es: %p", 162, &dato);
12     printf("\n\tLa direcci%cn de memoria de dato en decimal es: %ld", 162, (long int)&dato);
13
14     //el tipo del apuntador debe ser igual al tipo de la variable a la que apunta
15     int *apuntador;
16     //int *apuntador=NULL;
17
18     printf("\n\nLa variable apuntador entera contiene: %ld", (long int)apuntador);
19     printf(" porque no fue inicializada ni apunta a ninguna variable ");
20     printf("\n\tLa direcci%cn de memoria de apuntador en hexa es %p", 162, &apuntador);
21     printf("\n\tLa direcci%cn en memoria de apuntador en decimal es: %ld\n", 162, (long int)&apuntador);
22
23     //el operador unario & devuelve la dirección de memoria de una variable
24     apuntador = &dato;
25
26     printf("\n\nOperador: apuntador = &dato\n");
27     printf("\n\tAhora apuntador contiene la direcci%cn en hexa de dato = %p", 162, apuntador);
28     //%p muestra la dirección en hexadecimal de una variable o apuntador
29     printf("\n\tLa direcci%cn en memoria de dato en decimal es: %ld", 162, (long int)apuntador);
30
31     //el operador de indirección * devuelve el contenido de la variable apuntada
32     *apuntador = 5;
33
34     printf("\n\nOperador: *apuntador = 5\n");
35     printf("\n\tCambia el contenido de la variable a la que apunta, ahora dato = %d",dato);
36
37     getchar();
38     return 0;
39 }

```

Hecho por Huicho :)

# Proceso 1

- Imprimir dirección de memoria de una variable en hexadecimal:  
`printf("%p", &dato);`
- Imprimir dirección de memoria de una variable en decimal:  
`printf("%ld", (long int)&dato);`



```
/bin/bash
/bin/bash 80x24
La variable entera dato se inicializa y contiene: 2
    La dirección de memoria de dato en hexadecimal es: 0x7ffda32fc84c
    La dirección de memoria de dato en decimal es: 140727341271116

La variable apuntador entera contiene: 140727341271360 porque no fue inicializada ni apunta a ninguna variable
    La dirección de memoria de apuntador en hexa es 0x7ffda32fc850
    La dirección en memoria de apuntador en decimal es: 140727341271120

Operador: apuntador = &dato

    Ahora apuntador contiene la dirección en hexa de dato = 0x7ffda32fc84c
    La dirección en memoria de dato en decimal es: 140727341271116

Operador: *apuntador = 5

    Cambia el contenido de la variable a la que apunta, ahora dato = 5
```

# Proceso 2

- **long int x= 1;**
- **long int y= 2;**
- **long int \*apunt;**

- 
- **apunt= &x;**

- 
- **y= \*apunt;**

- 
- **x= (long int)apunt;**

- 
- **\*apunt= 3;**

```
● ○ ● /bin/bash
          /bin/bash 80x35
La variable entera X se inicializa y contiene: 1
      La dirección en memoria de X en decimal es: 140724613828720
La variable entera Y se inicializa y contiene: 2
      La dirección en memoria de Y en decimal es: 140724613828728
La variable apuntador entera contiene: 140724613828976
porque no se ha inicializado ni apuntado a ninguna variable
y su dirección inicial de memoria es 140724613828736

Operador: apuntador = &X
Ahora apuntador contiene la dirección en memoria de X: 140724613828720

Operador: y = *apuntador
La variable Y recibe el contenido de la variable X a la que se apunta
X vale: 1
Y ahora vale: 1

Operador: x = apuntador
La variable X recibe el contenido de apuntador que es la dirección de X
La dirección en memoria de X en decimal es: 140724613828720
X ahora vale: 140724613828720

Operador: *apuntador = 3
Cambia el contenido de la variable a la que apunta
X ahora vale: 3
```

# Direcciones de memoria en arreglo unidimensional estático

	Memoria física
arre[0]	2 <b>140726147255680</b>
arre[1]	3 <b>140726147255684</b>
arre[2]	5 <b>140726147255688</b>

arreglo

OCUPADA POR EL SISTEMA OPERATIVO

Direcciones de memoria reales

```
#define MAX 100;  
float arre[MAX];
```

- El **incremento en 4** de las direcciones de memoria reales se debe a que cada casilla o localidad de memoria se conforma de 8 bits, pero al ser el arreglo de tipo **float** cada elemento **necesita 4 cuadritos para completar los 32 bits** que le corresponden.



The screenshot shows a terminal window with the following content:

```
/bin/bash  
/bin/bash 80x24  
Programa que lee e imprime arreglo de n elementos  
Ingresá la cantidad de elementos: 3  
Arreglo a:  
Ingresá el valor del elemento[0]: 1  
Ingresá el valor del elemento[1]: 2  
Ingresá el valor del elemento[2]: 3  
Arreglo a:  
Elemento[0]: 1 esta en 140726147255680  
Elemento[1]: 2 esta en 140726147255684  
Elemento[2]: 3 esta en 140726147255688
```

# Direcciones de memoria en arreglo bidimensional estático

	Memoria física
aBidi[0][0]	2 140731192337312
aBidi[0][1]	3 140731192337316
aBidi[1][0]	5 140731192337320
aBidi[1][1]	7 140731192337324

arreglo bidimensional

OCUPADA POR EL SISTEMA OPERATIVO

float aBidi[2][2];

Direcciones de memoria reales

- El incremento en 4 de las direcciones de memoria reales se debe a que cada casilla o localidad de memoria se conforma de 8 bits, pero al ser el arreglo de tipo **float** cada elemento necesita 4 cuadritos para completar los 32 bits que le corresponden.

```
/bin/bash
/bin/bash 80x24
Programa que lee e imprime arreglo bidimensional de 2x2

Matriz a:

Ingrese el valor del elemento[0][0]: 1
Ingrese el valor del elemento[0][1]: 2
Ingrese el valor del elemento[1][0]: 3
Ingrese el valor del elemento[1][1]: 4

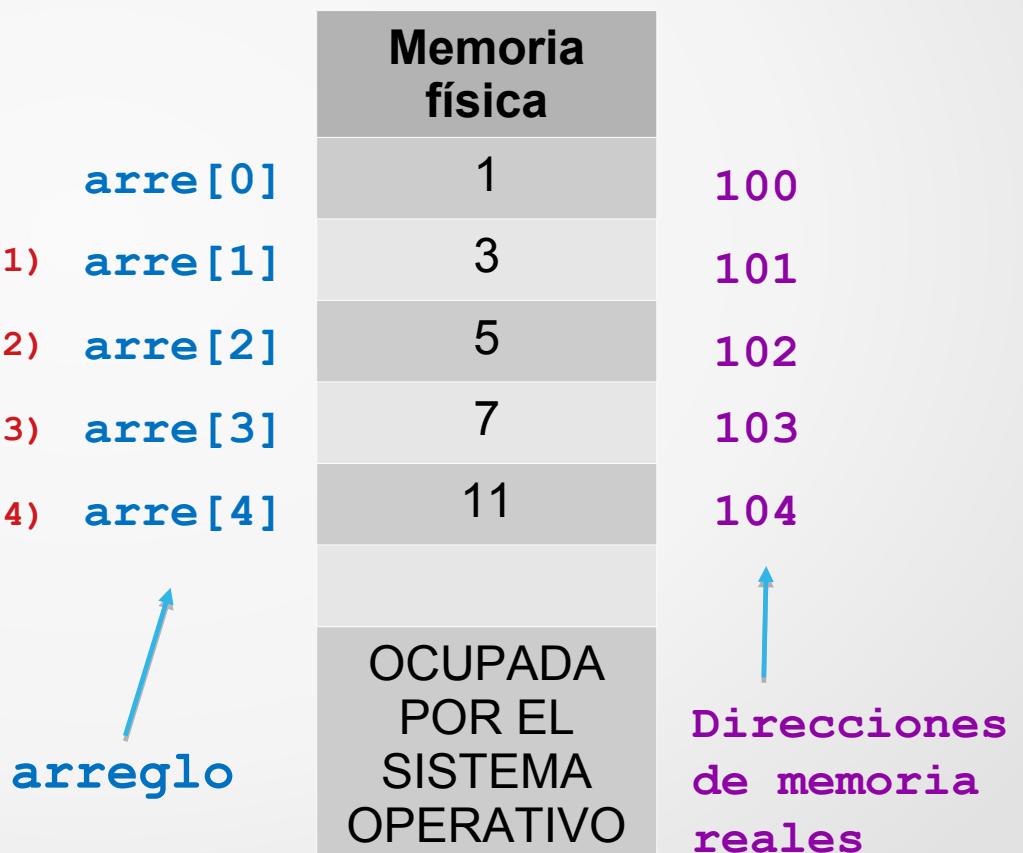
Matriz a:

140731192337312 140731192337316
140731192337320 140731192337324
```

# Aritmética de apuntadores

- Si apu es un apuntador a una variable, al **sumar 1 a su contenido devolverá la dirección de memoria del siguiente elemento**, al sumar 2 al apuntador dará la dirección del siguiente y así sucesivamente sin importar el tipo de dato y su tamaño en bits.
- `* (apu + 1)`
- `* (apu + i)`
- `* (apu + conde)`
- `* (apu++)`
- `* (apu+=conde)`

```
float arre[5]={1,3,5,7,11};  
float *apu= NULL;  
apu= &arre[0]; //apu= arre;
```



# Cómo leer con aritmética de apuntadores?

- Bloque de código para leer incluyendo ciclo, petición y lectura.
- El tope del ciclo es el tamaño o una variable con la cantidad dada.

```
float a[5]; //los 5 datos son reales  
  
short conde; //el contador es entero corto  
  
float *apuntador= NULL; //apuntador a real  
  
apuntador= &a[0]; //se da dirección del primero al apuntador  
  
  
for(conde=0; conde<=4; conde++)  
{  
    printf("Ingrese elemento[%hd] : ", conde);  
    scanf("%f", (apuntador+conde));  
}
```

# Cómo imprimir con aritmética de apuntadores?

- Bloque de código para imprimir incluyendo ciclo.
- El tope del ciclo es el tamaño o una variable con la cantidad dada.

```
float a[5]; //los 5 datos son reales  
  
short conde; //el contador es entero corto  
  
float *apuntador= NULL; //apuntador a real  
  
apuntador= &a[0]; //se da dirección del primero al apuntador  
  
  
for(conde=0; conde<=4; conde++)  
{  
    printf("\n\t Elemento[%hd] : %f", conde, *(apuntadort+conde));  
}
```

# Apuntador a un arreglo unidimensional estático

```
1  /*programa que lee y muestra arreglo de n elementos con ciclos for
2   * hecho por huicho*/
3
4  #include <stdio.h> //para printf, scanf y getchar
5
6  #define MAX 100
7
8  int main(int argc, char* argv[])
9  {
10    float arreglo[MAX]; //arreglo para reales con tamaño máximo de 100
11    float *dedo= NULL;
12    short conde; //contador que recorre cada elemento del arreglo
13    short cuantos; //guarda la cantidad de elementos del usuario
14
15    dedo= &arreglo[0];
16
17    printf("Programa que lee e imprime arreglo de n elementos \n\n");
18    printf("Ingresa la cantidad de elementos: ");
19    scanf("%hd", &cuantos);
20
21    printf("\n Arreglo a: \n\n");
22    for(conde=0; conde<=cuantos-1; conde++) //también conde<cuantos
23    {
24      printf("Ingrese el valor del elemento[%hd]: ", conde);
25      //scanf("%f", &arreglo[conde]);
26      scanf("%f", (dedo+conde));
27    }
28
29    printf("\n Arreglo a: \n");
30    for(conde=0; conde<=cuantos-1; conde++) //también conde<cuantos
31    {
32      //printf("\n\t Elemento[%hd]: %g esta en %ld", conde, arreglo[conde], (long)&arreglo[conde]);
33      printf("\n\t Elemento[%hd]: %g esta en %ld", conde, *(dedo+conde), (long)(dedo+conde));
34    }
35
36    getchar();
37    getchar();
38    return 0;
39 }
```

# Apuntador a un arreglo bidimensional estático

```
1  /*programa que lee y muestra arreglo bidimensional de mxn con ciclos for
2   * hecho por huicho*/
3
4  #include <stdio.h> //para printf, scanf y getchar
5
6  #define MAX 100
7
8  int main(int argc, char* argv[])
9  {
10    float arregloBidi[MAX][MAX]; //arreglo bidimensional para reales
11    float *dedo= NULL;
12    short reng, col=0; //contadores que recorren renglones y columnas
13    short renglones; //guarda la cantidad de renglones del usuario
14    short columnas; //guarda la cantidad de columnas del usuario
15
16    dedo= &arregloBidi[0][0];
17
18    printf("Programa que lee e imprime arreglo bidimensional de mxn \n\n");
19    printf("Ingresa la cantidad de renglones: ");
20    scanf("%hd", &renglones);
21    printf("Ingresa la cantidad de columnas: ");
22    scanf("%hd", &columnas);
23
24    printf("\n Matriz a: \n\n");
25    for(reng=0; reng<=(renglones*columnas)-1; reng++) //ciclo para apuntador
26    {
27      printf("Ingrese el valor del elemento[%hd]: ", reng);
28      scanf("%f", (dedo+reng));
29    }
30
31    printf("\n Matriz a: \n\n");
32    for(reng=0; reng<=(renglones*columnas)-1; reng++) //ciclo para apuntador
33    {
34      printf("%5g", *(dedo+reng));
35      col++;
36
37      if(col==columnas)
38      {
39        printf("\n");
40        col=0;
41      }
42    }
43
44    printf("\n Matriz a: \n");
45    for(reng=0; reng<=(renglones*columnas)-1; reng++) //ciclo para apuntador
46    {
47      printf("\n\t %g esta en %ld", *(dedo+reng), (long int)(dedo+reng));
48    }
49
50    getchar();
51    getchar();
52    return 0;
53 }
```

# Apuntador a un arreglo bidimensional estático

```
● ● ● /bin/bash
/bin/bash 80x36
Programa que lee e imprime arreglo bidimensional de mxn

Ingresa la cantidad de renglones: 3
Ingresa la cantidad de columnas: 3

Matriz a:

Ingrese el valor del elemento[0]: 1
Ingrese el valor del elemento[1]: 2
Ingrese el valor del elemento[2]: 3
Ingrese el valor del elemento[3]: 4
Ingrese el valor del elemento[4]: 5
Ingrese el valor del elemento[5]: 6
Ingrese el valor del elemento[6]: 7
Ingrese el valor del elemento[7]: 8
Ingrese el valor del elemento[8]: 9

Matriz a:

    1    2    3
    4    5    6
    7    8    9

Matriz a:

    1 esta en 140737056681312
    2 esta en 140737056681316
    3 esta en 140737056681320
    4 esta en 140737056681324
    5 esta en 140737056681328
    6 esta en 140737056681332
    7 esta en 140737056681336
    8 esta en 140737056681340
    9 esta en 140737056681344
```

```

1  /*programa que muestra y modifica arreglo con apuntador
2   * hecho por huicho*/
3
4  #include <stdio.h> //para printf y getchar
5
6  int main(int argc, char* argv[])
7  {
8      int arreglo[5] = {1,3,5,7,11};
9      int *apuntador= NULL;
10     int valor;
11     short conde;
12     printf("Aritm%ctica de apuntadores en arreglos\n\n", 130);
13
14     printf("\nLa direcci%cn del primer elemento es: %ld", 162, (long int)&arreglo[0]);
15
16     //apuntador apunta a la direcci%on del elemento arreglo[0]
17     apuntador= &arreglo[0];
18
19     //otra forma de pasar la direcci%on
20     apuntador= arreglo;
21
22     printf("\n\nAhora el apuntador tiene la direcci%cn del arreglo: %ld", 162, (long int)apuntador);
23
24     //a valor se le asigna el contenido de apuntador (el elemento 0 de arreglo)
25     valor= *apuntador;
26     printf("\n\nEl elemento 0 de arreglo usando apuntador es: %d \n\n", valor);
27
28     printf("Recorriendo el arreglo sumando 1 al apuntador: \n");
29     for(conde=0; conde<=4; conde++) //recorriendo arreglo con ciclo y apuntador
30     {
31         printf("\n\tElemento[%d]: %d", conde, *(apuntador+conde));
32     }
33
34     printf("\n\nModificando elemento del arreglo por su apuntador: \n");
35     //cambiar el valor del elemento arreglo[2] con apuntador
36     *(apuntador+2)= 100;
37
38     //recorriendo arreglo con ciclo y apuntador para ver el cambio
39     for(conde=0; conde<=4; conde++)
40     {
41         printf("\n\tElemento[%d]: %d", conde, *(apuntador+conde));
42     }
43
44     getchar();
45     return 0;
46 }
```

```

/bin/bash
Aritmética de apuntadores en arreglos

La dirección del primer elemento es: 140728539030528
Ahora el apuntador tiene la dirección del arreglo: 140728539030528
El elemento 0 de arreglo usando apuntador es: 1
Recorriendo el arreglo sumando 1 al apuntador:
    Elemento[0]: 1
    Elemento[1]: 3
    Elemento[2]: 5
    Elemento[3]: 7
    Elemento[4]: 11

Modificando elemento del arreglo por su apuntador:
    Elemento[0]: 1
    Elemento[1]: 3
    Elemento[2]: 100
    Elemento[3]: 7
    Elemento[4]: 11

```

# TEMA 3:

## Memoria dinámica

# Memoria dinámica

- Se usa cuando se sabe exactamente la cantidad de memoria a usar durante la ejecución y con esto evitar su desperdicio al declarar un exceso de espacio en un arreglo estático.
- Se requiere un apuntador para almacenar la dirección de memoria del primer elemento que devuelve la función **calloc** o **malloc** al reservar en tiempo de ejecución.
- Puede emplearse para arreglo unidimensional y bidimensional.
- Es posible liberar el bloque de memoria reservado después de usarla aún cuando no se haya finalizado la ejecución del programa.

# Función malloc

- Pedirla a la biblioteca `<stdlib.h>`
- Prototipo: `void *malloc(tamaño en bytes)`
- Tamaño en bytes de un tipo de dato: `sizeof(double) ;`
- Es indispensable convertir el apuntador devuelto al tipo de dato usado mediante **cast**:
- `int *apuntador= NULL;`
- `apuntador= (int*)malloc(10*sizeof(int)) ;`
- `apuntador= (int*)malloc(cantidad*sizeof(int)) ;`
- **Si no hay memoria suficiente el apuntador devuelve NULL.**

# Función calloc

- Pedirla a la biblioteca `<stdlib.h>`
- Prototipo: `void *calloc(cantidad, tamaño en bytes)`
- Tamaño en bytes de un tipo de dato: `sizeof(double) ;`
- Reserva memoria como malloc pero inicializa en 0 cada elemento
- Convertir el apuntador devuelto al tipo de dato usado con `cast:`
- `int *apuntador= NULL;`
- `apuntador= (int*)calloc(10,sizeof(int)) ;`
- `apuntador= (int*)calloc(cantidad,sizeof(int)) ;`
- **Si no hay memoria suficiente el apuntador devuelve `NULL`.**

# Diferencia entre malloc y calloc

```
C:\Program Files (x86)\Dev-Cpp\ConsolePausuer.exe
Memoria dinámica para arreglo con malloc

Los elementos del arreglo son:
    Elemento[0]: 46399680
    Elemento[1]: 46404984
    Elemento[2]: 0

Memoria dinámica para arreglo con calloc

Los elementos del arreglo son:
    Elemento[0]: 0
    Elemento[1]: 0
    Elemento[2]: 0 .
```

# Función realloc

- Pedirla a la biblioteca `<stdlib.h>`
- Prototipo: `void *realloc(apuntador, nueva cantidad)`
- Tamaño en bytes de un tipo de dato: `sizeof(double)` ;
- Es indispensable convertir el apuntador devuelto al tipo de dato usado mediante **cast**:
- `int *apuntador= NULL;`
- `apuntador= (int*)malloc(cantidad*sizeof(int));`
- `apuntador= (int*)realloc(apuntador,nuevaCantidad);`
- **Si no hay memoria suficiente el apuntador devuelve `NULL`.**

# Cómo leer con aritmética de apuntadores?

- Bloque de código para leer incluyendo ciclo, petición y lectura.
- El tope del ciclo es el tamaño o una variable con la cantidad dada.

```
float *apuntador= NULL; //apuntador a real  
short conde; //el contador es entero corto  
apuntador= (float*)malloc(cant*sizeof(float)); //reserva  
  
for(conde=0; conde<=cant-1; conde++)  
{  
    printf("Ingrese elemento[%hd]: ", conde);  
    scanf("%f", (apuntador+conde));  
}
```

# Cómo imprimir con aritmética de apuntadores?

- Bloque de código para imprimir incluyendo ciclo.
- El tope del ciclo es el tamaño o una variable con la cantidad dada.

```
float *apuntador= NULL; //apuntador a real  
short conde; //el contador es entero corto  
apuntador= (float*)malloc(cant*sizeof(float)); //reserva  
  
for(conde=0; conde<=cant-1; conde++)  
{  
    printf("\n\t Elemento [%hd]: %f", conde, *(apuntador+conde));  
}
```

# malloc

- Los datos a almacenar son enteros `int` por tanto el apuntador también es `int`.
- Se compara el apuntador con `NULL` para comprobar si pudo reservar la memoria y si no terminar la ejecución.
- La lectura y la impresión se realiza con aritmética de apuntadores.

The terminal window shows the execution of a C program. The command `./bin/bash` is run, followed by the program name `/bin/bash`. The program asks for the number of elements (Cuántos elementos desea: 3), reads three integers (Ingrrese el elemento[0]: 1, Ingrrese el elemento[1]: 2, Ingrrese el elemento[2]: 3), prints them (Los elementos del arreglo son: Elemento[0]: 1, Elemento[1]: 2, Elemento[2]: 3), and then frees the allocated memory.

```
./bin/bash
/bin/bash
Memoria dinámica para arreglo con malloc

Cuántos elementos desea: 3
Ingrrese el elemento[0]: 1
Ingrrese el elemento[1]: 2
Ingrrese el elemento[2]: 3

Los elementos del arreglo son:
  Elemento[0]: 1
  Elemento[1]: 2
  Elemento[2]: 3
```

```
1  /*programa que lee y muestra arreglo de n elementos dinámico
2   * hecho por huicho*/
3
4  #include <stdio.h> //para printf, scanf y getchar
5  #include <stdlib.h> //para exit
6
7  int main(int argc, char* argv[])
8  {
9      int *apuntador= NULL;
10     int cantidad;
11     int conde;
12
13     printf("Memoria din%cmica para arreglo con malloc \n\n", 160);
14     printf("Cu%cntos elementos desea: ", 160);
15     scanf("%d", &cantidad);
16
17     apuntador = (int*)malloc(cantidad*sizeof(int));
18     if(apuntador==NULL) //valida si se pudo reservar memoria
19     {
20         printf("\n\t No pude reservar la memoria :( ");
21         getchar();
22         getchar();
23         exit(0); //termina la ejecución justo ahí
24     }
25
26     for(conde=0; conde<=cantidad-1; conde++) //ciclo para leer
27     {
28         printf("Ingrese el elemento[%d]: ", conde);
29         scanf("%d", (apuntador+conde));
30     }
31
32     printf("\n\nLos elementos del arreglo son: ");
33     for(conde=0; conde<=cantidad-1; conde++) //ciclo para imprimir
34     {
35         printf("\n\t Elemento[%d]: %d ", conde, *(apuntador+conde));
36     }
37     free(apuntador); //libera la memoria solicitada
38
39     getchar();
40     getchar();
41     return 0;
42 }
```

Hecho por Huicho :)

# calloc

- Los datos son **int** por tanto apuntador también es **int**.
- La cantidad es un argumento en la función junto con el tamaño.
- Si apuntador contiene **NULL** no pudo reservar la memoria termina la ejecución.
- Lectura e impresión se realizan con aritmética de apuntadores.

```
● ● ● /bin/bash
[ ] |                               /bin/
Memoria dinámica para arreglo con calloc
Cuántos elementos desea: 3
Ingrrese el elemento[0]: 1
Ingrrese el elemento[1]: 2
Ingrrese el elemento[2]: 3

Los elementos del arreglo son:
  Elemento[0]: 1
  Elemento[1]: 2
  Elemento[2]: 3
```

```
1  /*programa que lee y muestra arreglo de n elementos dinámico
   * hecho por huicho*/
2
3
4  #include <stdio.h> //para printf, scanf y getchar
5  #include <stdlib.h> //para exit
6
7  int main(int argc, char* argv[])
8  {
9      int *apuntador= NULL;
10     int cantidad;
11     int conde;
12
13     printf("Memoria din%cmica para arreglo con calloc \n\n", 160);
14     printf("Cu%ntos elementos desea: ", 160);
15     scanf("%d", &cantidad);
16
17     apuntador = (int*)calloc(cantidad,sizeof(int));
18     if(apuntador==NULL) //valida si se pudo reservar memoria
19     {
20         printf("\n\t No pude reservar la memoria :( ");
21         getchar();
22         getchar();
23         exit(0); //termina la ejecución justo ahí
24     }
25
26     for(conde=0; conde<=cantidad-1; conde++) //ciclo para leer
27     {
28         printf("Ingrese el elemento[%d]: ", conde);
29         scanf("%d", (apuntador+conde));
30     }
31
32     printf("\n\nLos elementos del arreglo son: ");
33     for(conde=0; conde<=cantidad-1; conde++) //ciclo para imprimir
34     {
35         printf("\n\t Elemento[%d]: %d ", conde, *(apuntador+conde));
36     }
37     free(apuntador); //libera la memoria solicitada
38
39     getchar();
40     getchar();
41     return 0;
42 }
```

Hecho por Huicho :)

# realloc

- La memoria del arreglo dinámico se genera inicialmente con `malloc`.
- El arreglo se rellena de números aleatorios por `rand` de acuerdo a la cantidad dada por el usuario.
- `cantidad/=2` equivale a `cantidad= cantidad/2`
- `cantidad*=3` equivale a `cantidad= cantidad*3`
- Los argumentos de `realloc` son el apuntador al arreglo que se desea cambiar de tamaño y la nueva cantidad.

```
1  /*programa que lee, muestra y cambia tamaño de arreglo dinámico
2   * hecho por huicho*/
3
4  #include <stdio.h> //para printf, scanf y getchar
5  #include <stdlib.h> //para exit, srand y rand
6  #include <time.h> //para time
7
8  int main(int argc, char* argv[])
9  {
10    int *apuntador= NULL;
11    int cantidad;
12    int conde;
13
14    srand(time(NULL));
15
16    printf("Memoria din%cmica para arreglo de aleatorios con malloc \n\n", 160);
17    printf("Cu%cntos elementos desea: ", 160);
18    scanf("%d", &cantidad);
19
20    apuntador = (int*)malloc(cantidad*sizeof(int));
21    if(apuntador==NULL) //valida si se pudo reservar memoria
22    {
23      printf("\n\t No pude reservar la memoria :( ");
24      getchar();
25      getchar();
26      exit(0); //termina la ejecuci%on justo ah%í
27    }
28
29    printf("\n Los %d elementos del arreglo son: ", cantidad);
30    for(conde=0; conde<=cantidad-1; conde++) //ciclo para imprimir
31    {
32      *(apuntador+conde)= rand() % 51;
33      printf("\n\t Elemento[%d]: %d ", conde, *(apuntador+conde));
34    }
35
36    printf("\n\n De los %d elementos, ahora el arreglo a la mitad: ", cantidad);
37    apuntador = (int*)realloc(apuntador, cantidad/=2); //cantidad= cantidad/2
38    for(conde=0; conde<=cantidad-1; conde++) //ciclo para imprimir
39      printf("\n\t Elemento[%d]: %d ", conde, *(apuntador+conde));
40
41    printf("\n\n De los %d elementos, el triple del arreglo: ", cantidad);
42    apuntador = (int*)realloc(apuntador, cantidad*=3); //cantidad= cantidad*3
43    for(conde=0; conde<=cantidad-1; conde++) //ciclo para imprimir
44      printf("\n\t Elemento[%d]: %d ", conde, *(apuntador+conde));
45
46    free(apuntador); //libera la memoria solicitada
47
48    getchar();
49    getchar();
50  }
```

# realloc

- La memoria del arreglo dinámico se genera inicialmente con `malloc`.
- El arreglo se rellena de números aleatorios por `rand` de acuerdo a la cantidad dada por el usuario (4).
- `realloc` redimensiona el arreglo a la mitad y mantiene el valor de los que quedan.
- `realloc` redimensiona el arreglo al triple del anterior, mantiene el valor de los que estaban y el de los nuevos es indeterminado.

```
● ● ○ /bin/bash
          /bin/bash 80x24
Memoria dinámica para arreglo de aleatorios con malloc
Cuántos elementos desea: 4
Los 4 elementos del arreglo son:
  Elemento[0]: 18
  Elemento[1]: 41
  Elemento[2]: 36
  Elemento[3]: 44
De los 4 elementos, ahora el arreglo a la mitad:
  Elemento[0]: 18
  Elemento[1]: 41
De los 2 elementos, el triple del arreglo:
  Elemento[0]: 18
  Elemento[1]: 41
  Elemento[2]: 36
  Elemento[3]: 44
  Elemento[4]: 0
  Elemento[5]: 0
```

# Pasar de estático a dinámico

```
char nombre[10];
```

variable
nombre[0]
nombre[1]
nombre[2]
nombre[3]
nombre[4]
nombre[5]
nombre[6]
nombre[7]
nombre[8]
nombre[9]

Memoria física

'L'

'u'

'i'

's'

'\0'

b

b

b

b

b

```
char *apu= NULL;  
apu= (char*)malloc(longi+1*sizeof(char));
```

variable

apu[0]
apu[1]
apu[2]
apu[3]
apu[4]

Memoria física

'L'

'u'

'i'

's'

'\0'

b

b

b

b

longi=strlen(nombre);

Sumar 1 a longi para almacenar también el carácter de terminación nulo

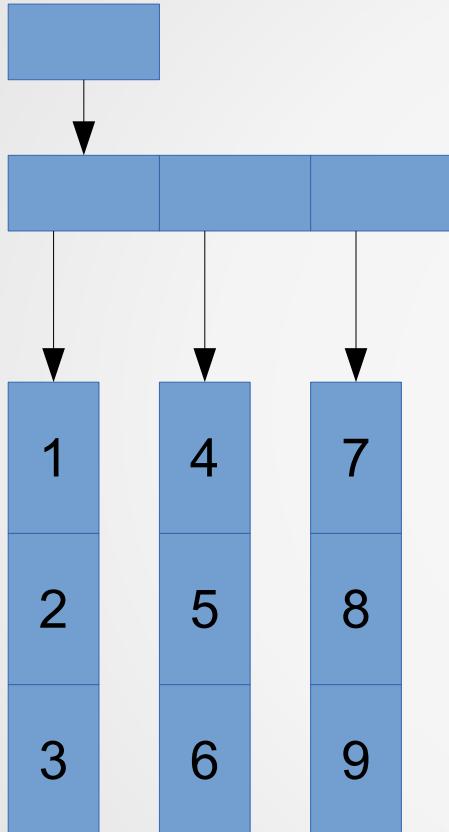
Después de reservar el espacio, cómo pasar de un arreglo al otro?

# Apuntador de apuntadores

- Se declara el apuntador de apuntadores de acuerdo al tipo de dato de los elementos y las variables para la cantidad de renglones y columnas a reservar.
- Se declara un contador para recorrer el arreglo de apuntadores.
- El tope del ciclo es el número de renglones.

```
float **matrix= NULL; //apuntador de apuntadores a real  
int renglones, columnas; //cantidades son enteras  
int conde; //contador es entero  
  
matrix= (float**)malloc(renglones*sizeof(float*));  
for(conde=0; conde<=renglones-1; conde++)  
{  
    matrix[conde]= (float*)malloc(columnas*sizeof(float));  
}
```

# Apuntador de apuntadores



```
/bin/bash
/bin/bash 80x45
Memoria dinámica para arreglo bidimensional con malloc

Cuántos renglones desea: 3
Cuántas columnas desea: 3
Ingrese elemento[0][0]: 1
Ingrese elemento[0][1]: 2
Ingrese elemento[0][2]: 3
Ingrese elemento[1][0]: 4
Ingrese elemento[1][1]: 5
Ingrese elemento[1][2]: 6
Ingrese elemento[2][0]: 7
Ingrese elemento[2][1]: 8
Ingrese elemento[2][2]: 9

Apuntador de apuntadores contiene: 28432432
Elemento 0 del arreglo de ap contiene 28432464 y está en 28432432
Elemento 1 del arreglo de ap contiene 28432496 y está en 28432440
Elemento 2 del arreglo de ap contiene 28432528 y está en 28432448

Los elementos del arreglo bidimensional son:
    1   2   3
    4   5   6
    7   8   9

Los elementos y su dirección son:
Elemento[0][0]: 1 y está en 28432464
Elemento[0][1]: 2 y está en 28432468
Elemento[0][2]: 3 y está en 28432472

Elemento[1][0]: 4 y está en 28432496
Elemento[1][1]: 5 y está en 28432500
Elemento[1][2]: 6 y está en 28432504

Elemento[2][0]: 7 y está en 28432528
Elemento[2][1]: 8 y está en 28432532
Elemento[2][2]: 9 y está en 28432536
```

# Cómo leer con aritmética de apuntadores?

- Bloque de código para leer incluyendo ciclo, petición y lectura.
- El tope de los ciclos es el número de renglones y columnas.

```
float **matrix= NULL; //apuntador de apuntadores a real  
int reng, col; //contadores son enteros  
  
for(reng=0; reng<=renglones-1; reng++)  
{  
    for(col=0; col<=columnas-1; col++)  
    {  
        printf("Ingrese elemento[%d] [%d]: ", reng, col);  
        scanf("%f", (* (matrix+reng)+col));  
    }  
}
```

# Cómo imprimir con aritmética de apuntadores?

- Bloque de código para imprimir como matriz.
- El tope de los ciclos es el número de renglones y columnas.

```
float **matrix= NULL; //apuntador de apuntadores a real  
int reng, col; //contadores son enteros  
  
for(reng=0; reng<=renglones-1; reng++)  
{  
    for(col=0; col<=columnas-1; col++)  
    {  
        printf("%5f", *(*(matrix+reng)+col));  
    }  
    printf("\n");  
}
```

```

1  /*programa que lee y muestra arreglo bidimensional dinámico
2   * hecho por huicho*/
3
4  #include <stdio.h> //para printf, scanf y getchar
5  #include <stdlib.h> //para exit
6
7  int main(int argc, char* argv[])
8  {
9      int **matrix= NULL; //apuntador de apuntadores a entero
10     int renglones, columnas;
11     int conde;
12     int reng, col;
13
14     printf("Memoria din%cmica para arreglo bidimensional con malloc \n\n", 160);
15     printf("Cu%cntos renglones desea: ", 160);
16     scanf("%d", &renglones);
17     printf("Cu%cntas columnas desea: ", 160);
18     scanf("%d", &columnas);
19
20     //se reserva el arreglo de apuntadores
21     matrix= (int**)malloc(renglones*sizeof(int*));
22
23     if(matrix==NULL)
24     {
25         printf("\n\t No pude reservar la memoria :( ");
26         getchar();
27         getchar();
28         exit(0); //termina la ejecución justo ahí
29     }
30
31     //se recorren los renglones y se reservan las columnas de cada uno
32     for(conde=0; conde<renglones; conde++)
33     {
34         matrix[conde]= (int*)malloc(columnas*sizeof(int));
35
36         if(matrix[conde]==NULL)
37         {
38             printf("\n\t No pude reservar la memoria :( ");
39             getchar();
40             getchar();
41             exit(0); //termina la ejecución justo ahí
42         }
43     }
44
45     for(reng=0; reng<=renglones-1; reng++)
46     {
47         for(col=0; col<=columnas-1; col++)
48         {
49             printf("Ingrese elemento[%d][%d]: ", reng, col);
50             //scanf("%d", &matrix[reng][col]);
51             scanf("%d", (*(matrix+reng)+col));
52         }
53     }
54
55     printf("\n\n Los elementos del arreglo bidimensional son: \n\n");
56     for(reng=0; reng<=renglones-1; reng++)
57     {
58         for(col=0; col<=columnas-1; col++)
59         {
60             //printf("%5d", matrix[reng][col]);
61             printf("%5d", (*(matrix+reng)+col));
62         }
63         printf("\n");
64     }
65
66     //liberando memoria del arreglo de apuntadores primero
67     for(conde=0; conde<columnas; conde++)
68     {
69         free(matrix[conde]);
70     }
71     free(matrix); //al final se libera la del apuntador de apuntadores
72
73     getchar();
74     getchar();
75 }
76

```

# Apuntador de apuntadores

- Los datos son `int` por tanto el apuntador de apuntadores también es `int`.
- Si apuntador de apuntadores contiene `NULL` no pudo reservar la memoria y termina la ejecución.
- Si cada elemento del arreglo de apuntadores contiene `NULL` no pudo reservar la memoria y termina la ejecución.
- La memoria del arreglo dinámico se genera con `malloc`.
- El arreglo tiene exactamente las dimensiones dadas por el usuario evitando el desperdicio de memoria de un arreglo estático.
- Lectura e impresión se realizan con aritmética de apuntadores.
- La liberación de memoria se realiza primero del arreglo de apuntadores y al final del apuntador de apuntadores.

The screenshot shows a terminal window titled '/bin/bash' with a red header bar containing three colored dots (red, yellow, green) and the path '/bin/bash'. The window title is '/bin/bash 80x24'. The terminal displays the following text:

```
Memoria dinámica para arreglo bidimensional con malloc

Cuántos renglones desea: 3
Cuántas columnas desea: 3
Ingrese elemento[0][0]: 1
Ingrese elemento[0][1]: 2
Ingrese elemento[0][2]: 3
Ingrese elemento[1][0]: 4
Ingrese elemento[1][1]: 5
Ingrese elemento[1][2]: 6
Ingrese elemento[2][0]: 7
Ingrese elemento[2][1]: 8
Ingrese elemento[2][2]: 9

Los elementos del arreglo bidimensional son:
    1   2   3
    4   5   6
    7   8   9
```

The background of the terminal window features a photograph of Earth from space.