Punteros parte II

Sofía Beatriz Pérez
Daniel Agustín Rosso

sperez@iua.edu.ar
drosso@iua.edu.ar

Centro Regional Univesitario Córdoba
Instituto Univeristario Areonáutico

Informática II - Clase número 2 - Ciclo lectivo 2022

Agenda

Arreglos de punteros

Indirección múltiple

Arreglos multidimencionales y punteros

Punteros a funciones

Disclaimer

Los siguientes slides tienen el objetivo de dar soporte al dictado de la asignatura. De ninguna manera pueden sustituir los apuntes tomados en clases y/o la asistencia a las mismas.
Es importante mencionar que todos este material se encuentra en un proceso de mejora continua.
Si encuentra bugs, errores de ortografía o redacción, por favor repórtelo a sperez@iua.edu.ar y/o drosso@iua.edu.ar. También puede abrir issues en el repositorio de este link: ▸ infoI_IUA_GitLab

Arreglos y punteros I

Recordando que los arreglos se definen de forma contigua en la memoria, supongamos las siguientes declaraciones:

```
1  int total[5]={0};
2  char letra[5]={0};
```

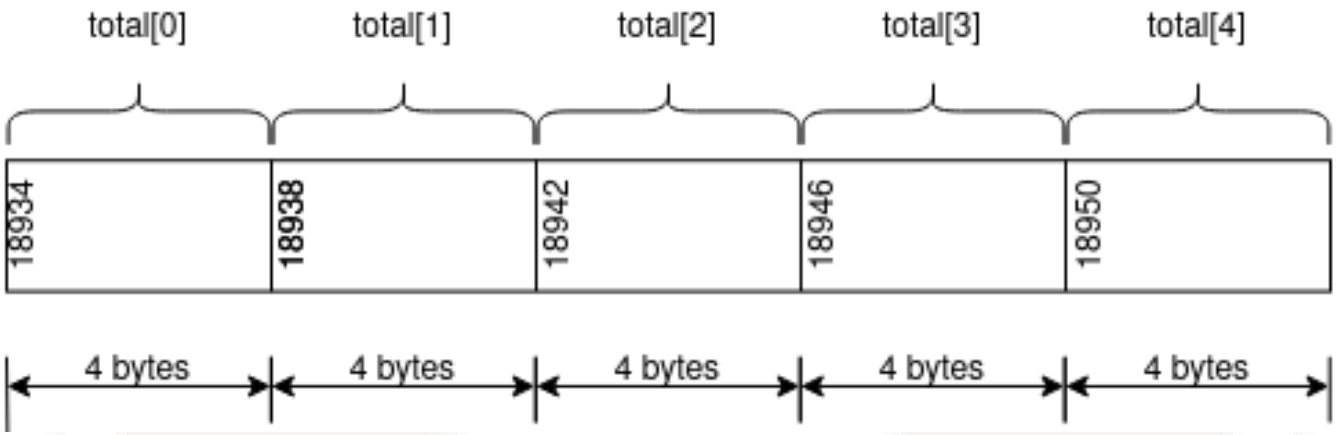

Figure: Arreglo de enteros.

Arreglos y punteros II

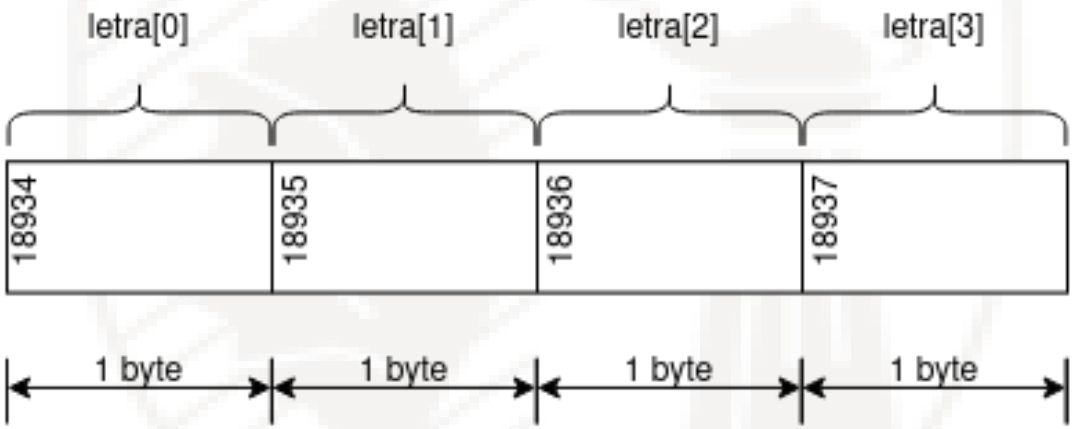

Figure: Arreglo de caracteres.

Conociendo el tamaño que ocupa en memoria cada elemento del arreglo:

Arreglos y punteros III

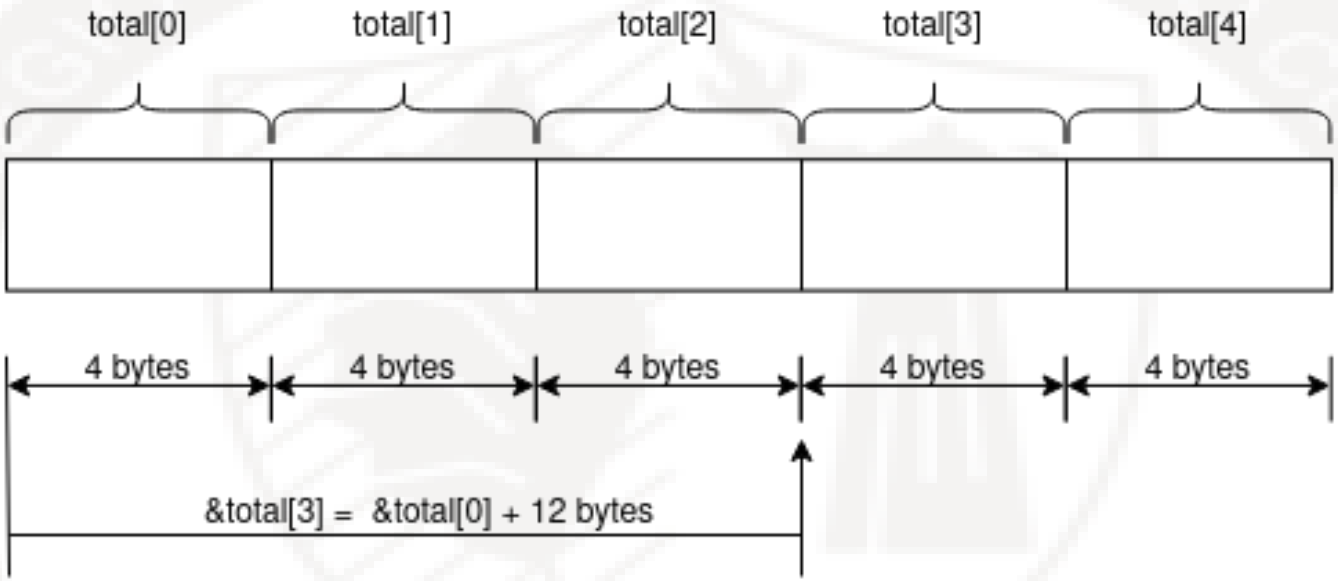

Figure: Arreglo de caracteres.

Arreglos y punteros IV

Recordando que **en C/C++ el nombre de un arreglo es un puntero al primer elemento del mismo.**. Esto implica que estas lineas son equivalentes.

```
1 int total[5] = {1};
2 int *ptrtotal;
3 ptrtotal = total;
4 ptrtotal = &total[0];
```

Considerando las operaciones de suma y resta de punteros vista anteriormente (aritmética de punteros):

Arreglos y punteros V

```
1  int total[5] = {1, 2, 3, 4, 5};
2  int *ptrtotal;
3  ptrtotal = total;
4
5  ptrtotal++;
6  total[1];
7
8  ptrtotal++;
9  total[2];
10
11 ptrtotal++;
12 total[2];
```

Arreglos y punteros VI

Las lineas 5 y 6 apuntan a la misma posición de memoria y por ende al mismo elemento del arreglo.

De forma general:

Elemento del arreglo	Notación de subíndice	Notación de puntero
Elemento 0	total[0]	*ptrtotal
Elemento 1	total[1]	*(ptrtotal+1)
Elemento 2	total[2]	*(ptrtotal+2)
Elemento 3	total[3]	*(ptrtotal+3)
Elemento 4	total[4]	*(ptrtotal+4)

Arreglos y punteros VII

```
1  #include<stdio.h>
2  int main(void)
3  {
4      int total[5]={0};
5      int ii;
6      int *p=NULL;
7      p=total;
8
9      for(ii=0;ii<5;ii++)
10     {
11         scanf("%d",(total+ii));
12     }
13
14
15
```

Arreglos y punteros VIII

```
16      for(ii=0;ii<5;ii++)
17      {
18          printf("%d",*(total+ii));
19      }
20
21      for(ii=0;ii<5;ii++)
22      {
23          printf("%d",*(p+ii));
24      }
25      return(0);
26  }
```

Arreglos de punteros I

Los punteros pueden estructurarse en arrays como cualquier otro tipo de datos. Por lo cual, la forma de operar con ellos es igual a como si fuese un arreglo de los ya conocidos:

```
1  int *ptrarray[5];
2  int a=10;
3  ptrarray[3]=&a;
```

Arreglos de punteros II

```
1  #include<stdio.h>
2  int main(void)
3  {
4      int *ptrarray[5];
5      int a=10;
6      ptrarray[3]=&a;
7      printf("%d " ,*(ptrarray[3]));
8      return(0);
9  }
```

Indirección múltiple I

Se puede hacer que un puntero apunte a otro puntero que apunte a un valor de destino. Esta situación es conocida como *indirección múltiple opunteros a punteros*.
La indirección múltiple puede llevarse a la extensión que uno desee, pero existen pocos casos en los que se necesite más de un puntero a puntero.

Indirección múltiple II

```
1  #include<stdio.h>
2  int main(void)
3  {
4      int x;
5      int *p;
6      int **q;
7
8      x=10;
9      p=&x;
10     q=&p;
11
12     printf("x vale %d",**q);
13     return(0);
14 }
```

Arreglos multidimencionales y punteros I

También puede tenerse acceso a arreglos multidimencionales usando notación de punteros. La notación se vuelve mas compleja conforme aumentan las dimensiones del arreglo.

```
1  int nums[2][3]={{16,18,20},{25,26,27}};
```

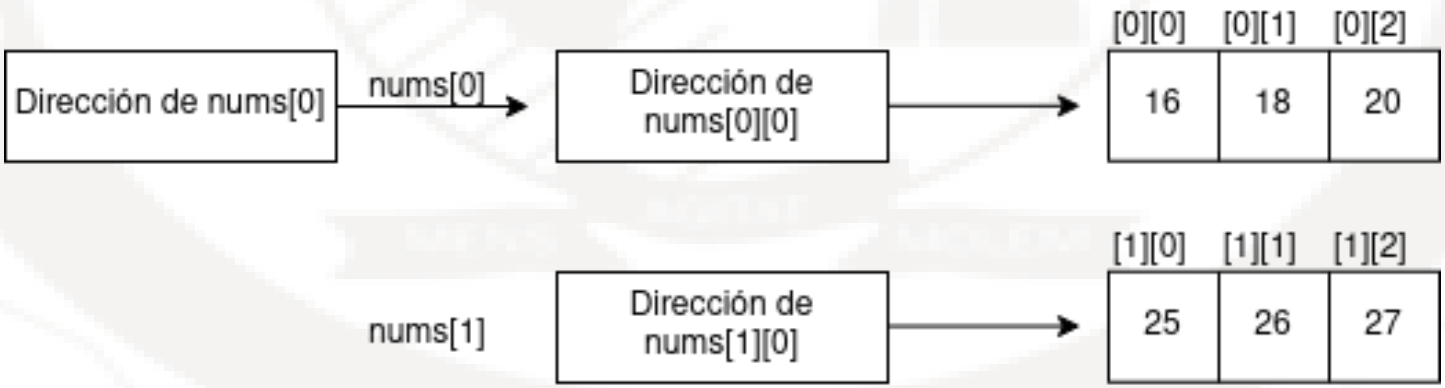

Figure: Arreglo de dos dimensiones y punteros.

Arreglos multidimencionales y punteros II

Notación de puntero	Notación de subíndice	Valor almacenado
*(*nums)	nums[0][0]	16
*(*nums +1)	nums[0][1]	18
*(*nums +2)	nums[0][2]	20
((nums+1))	nums[1][0]	25
((nums+1) +1)	nums[1][1]	26
((nums+1) +2)	nums[1][2]	27

La utilidad de esto se verá cuando se estudie el tema de creación de arreglos dinámicamente.

Punteros a funciones I

Una función tiene una ubicación física en memoria que puede asignarse a un puntero. Esa dirección es el punto de entrada a una función, es decir, la dirección que se usa cuando se la invoca. Una vez que esta dirección es almacenada en un puntero, se puede invocar a la función mediante la utilización de este puntero.
La dirección de la función se obtiene utilizando el nombre de la misma sin paréntesis ni argumentos:

```
1  void cubo(float x); //prototipo
2
3  void (*ptrAFunc) (float);   //puntero a funcion
4  ptrAFunc = cubo;            //Asignacion
5  ptrAFunc(x);                //Llamada
```

Punteros a funciones II

```c
#include<stdio.h>

void hola_mundo(void);
void chau_mundo(void);

int main(void)
{
    void (*ptr_to_func)(); //Puntero a funcion
    ptr_to_func = hola_mundo; //Puntero a hola mundo
    (*ptr_to_func)();    //Llamando
    ptr_to_func = chau_mundo;
    (*ptr_to_func)();
    return(0);
}

```

Punteros a funciones III

```
16
17  void hola_mundo(void)
18  {
19      printf("Hello IUA\n");
20  }
21
22  void chau_mundo(void)
23  {
24      printf("ByeBye IUA\n");
25  }
```

Punteros a funciones IV

```
1
2  #include<stdio.h>
3  int mul(int n1, int n2)
4  {
5      return (n1*n2);
6  }
7  void main(void)
8  {   int res;
9      int (*ptr_to_func)(int,int);
10     ptr_to_func = mul;
11     res = (*ptr_to_func)(10,5);
12     printf("10*5 = %d\n",res);
13     res = (*ptr_to_func)(100,10);
14     printf("100*10 = %d\n",res);
15 }
```

¡Muchas gracias!

Consultas:

sperez@iua.edu.ar
drosso@iua.edu.ar