Funciones: continuación
Introducción a punteros :)

María Alejandra Boggio - Sofía Beatriz Pérez
Daniel Agustín Rosso

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

Centro Regional Univesitario Córdoba
Instituto Univeristario Areonáutico

Clase número 9 - Ciclo lectivo 2022

Agenda

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 mbgogio@iua.edu.ar, sperez@iua.edu.ar y/o drosso@iua.edu.ar. También puede abrir issues en el repositorio de este link: ▸ infoI_IUA_GitLab

Introducción a los punteros

Motivación e importancia

Una de las ventajas de C/C++ es que permite al desarrollador entrar en forma directa al funcionamiento de una computadora, ya que provee acceso a las direcciones de memoria utilizadas para almacenar las variables de un programa.

Definición

De manera fundamental, los punteros son tan sólo variables que se usan para almacenar direcciones de memoria.

Operador de dirección & I

Para desplegar la dirección donde una variable está almacenada, C nos provee el operador de dirección (&), el cual significa "la dirección de".

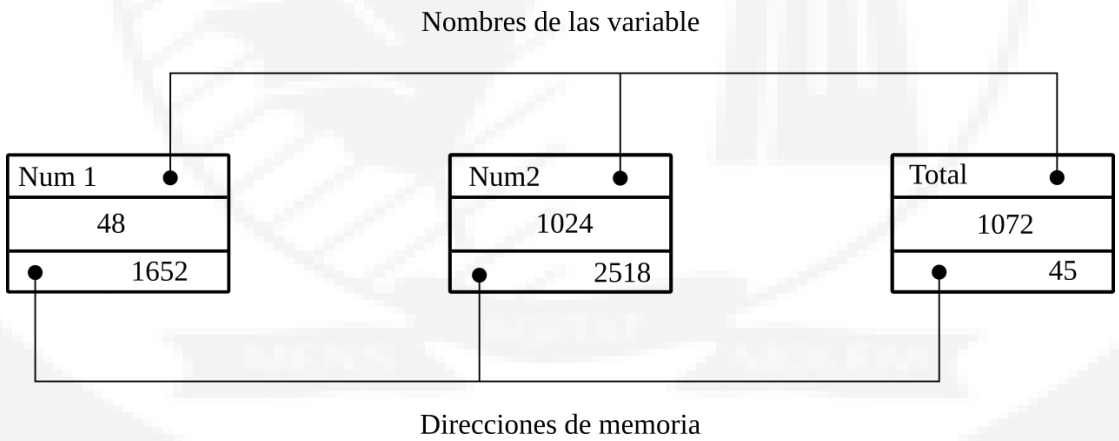

Figure: Declaración de variables

Operador de dirección & II

```
1  #include <stdio.h>
2  int main()
3  {   int num1=48;
4      int num2=1024;
5      int total=num1+num2;
6      printf("Valor de num1 %d\n",num1);
7      printf("Posicion de memoria de num1 %X\n",&num1);
8      printf("Valor de num2 %d\n",num2);
9      printf("Posicion de memoria de num2 %X\n",&num2);
10     printf("Valor de total %d\n",total);
11     printf("Posicion de memoria de total %X\n",&total);
12     return (0);
13 }
```

Almacenamiento de direcciones de memoria I

Ademas de imprimir la dirección o posición de memoria de una variable, es posible almacenarla en otra variable para poder operar con ella. Los tipos de datos capaces de almacenar direcciones de memoria, se los conoce como **punteros**.

```
1  /*Puntero a entero*/
2  int *dato=NULL;
3  /*Puntero a float*/
4  float *dato=NULL;
5  /*Puntero a char*/
6  char *dato=NULL;
7  /*Puntero a puntero*/
8  int **dato=NULL;
```

Almacenamiento de direcciones de memoria II

```
1  #include <stdio.h>
2  int main()
3  {   int num1=48;
4      int *dir_num_1;
5
6      /*uso de la variable de tipo puntero*/
7      dir_num_1=&num1;
8
9      printf("Valor de num1 %d\n",num1);
10     printf("Posicion de memoria de num1 %X\n",&num1);
11     printf("Valor de dir_num_1 %X\n",dir_num_1);
12     printf("Posicion de memoria de num1 %X\n",&dir_num_1);
13
14     return (0);
15 }
```

Operador de indirección * I

Cuando el símbolo * es seguido de una variable puntero, significa: "la variable cuya dirección de memoria está almacenada en". En otras palabras, devuelve el valor del objeto al que apunta su operando.

```
1 #include <stdio.h>
2 int main()
3 {   int num1=48;
4     int *dir_num_1;
5     dir_num_1=&num1;
6 }
```

Operador de indirección * II

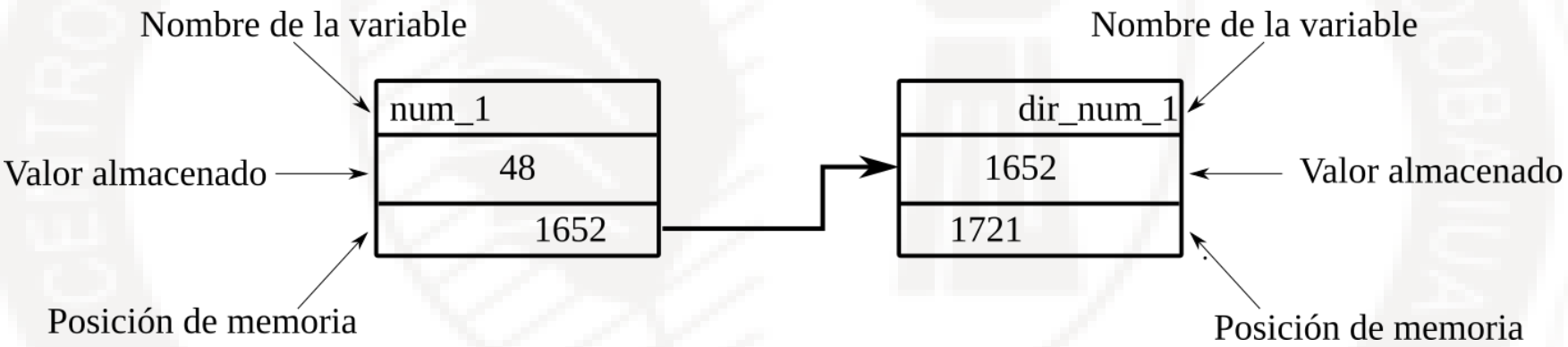

Figure: Uso de punteros

Operador de indirección * III

```
1 #include <stdio.h>
2 int main()
3 {   int num1=48;
4     int *dir_num_1;
5     dir_num_1=&num1;
6     *dir_num_1=10;
7     *dir_num_1++;
8 }
```

Operador de indirección * IV

```
1  #include <stdio.h>
2  int main()
3  {   int num1=48;
4      int *dir_num_1;
5      dir_num_1=&num1;
6      printf("Valor de num1 %d\n",num1);
7      printf("Posicion de memoria de num1 %X\n",&num1);
8      printf("Valor de dir_num_1 %X\n",dir_num_1);
9      printf("Posicion de memoria de num1 %X\n",&dir_num_1);
10     *dir_num_1=10;
11     printf("Valor de num1 %d\n",num1);
12     return (0);
13 }
```

Mas ejemplos en vivo

Mecanismo de paso de argumentos a funciones I

Paso por valor

El valor del argumento es **copiado** en el parámetro de la subrutina, por lo cual si se realizan cambios en el mismo dentro de la función, el valor original no es modificado.

```
1  #include<stdio.h>
2
3  /*PROTOTIPO*/
4  int suma_tres(int);
5
6  int main (void)
7  {
8     int n1=0,rdo;
9     printf("Ingrese un numero \n");
```

Mecanismo de paso de argumentos a funciones II

```
10      scanf("%d",&n1);
11      printf("N1 antes de llamar a suma_tres %d \n",n1);
12      rdo=suma_tres(n1);
13      printf("N1 despues de llamar a suma_tres %d \n",n1);
14      printf("Valor de resultado %d \n",rdo);
15
16  }
17
18  int suma_tres (int num)
19  {
20      num=(num+3);
21      return(num);
22  }
```

Mecanismo de paso de argumentos a funciones III

Paso referencia

Se copia la *dirección de memoria* del argumento como parámetro de la función. En este caso, al realizar cambios en parámetro formal este si se ve afectado.

```
1  #include<stdio.h>
2
3  /*PROTOTIPO*/
4  int suma_tres(int *);
5
6  int main (void)
7  {
8     int n1=0;
9     printf("Ingrese un numero \n");
10    scanf("%d",&n1);
```

Mecanismo de paso de argumentos a funciones IV

```
11    int rdo;
12
13    printf("N1 antes de llamar a suma_tres %d \n",n1);
14    rdo=suma_tres(&n1);
15    printf("N1 despues de llamar a suma_tres %d \n",n1);
16    printf("Valor de resultado %d \n",rdo);
17
18  }
19
20  int suma_tres (int *num)
21  {
22    int resultado;
23    resultado = *num + 3;
24    return(resultado);
25  }
```

Ejemplos I

1. Diseñar y codificar un programa que recibiendo desde la función main los catetos de un triángulo rectángulo, imprima en la función el resultado de la hipotenusa.
   ▸ Ver en github
2. Modificar el programa anterior para que el resultado sea impreso en la función main.
   ▸ Ver en github
3. Modificar el programa anterior para que el valor de los catetos sea pasado a la función por referencia y la impresión sea realizada en main. ▸ Ver en github

Ejemplos II

4. Modificar el programa anterior para que la impresión siga siendo realizada en main, pero la función debe tener el siguiente prototipo: void calculoHipotenusa(int *, int*, float *). ▸ Ver en github

Ejemplos III

5. Diseñar y codificar un programa que implemente las siguientes funciones:
   - Impresión de datos personales del desarrollador: debe ejecutarse al inicio del programa. Esta función no recibe ni retorna datos.
   - Menú de opciones: debe permitirle al operador seleccionar entre las siguientes opciones:
     1. Sumar dos números
     2. Restar dos números
     3. Imprimir mayor
     4. Imprimir menor

     La opción seleccionada debe ser retornada a main.

Ejemplos IV

- Implementar las funciones del apartado anterior. Los dos números deben ser enviados desde main por valor. El resultado de la suma y resta debe ser impreso en la función. La impresión del mayor y el menor debe hacerse en main().

▸ Ver en github

Ejemplos V

6. Modificar el programa anterior para que cada uno de los números sea enviado por referencia.

▸ Ver en github

Funciones y arreglos unidimensionales I

A diferencia de las variables en las que podemos elegir pasarlas a una función por valor o referencia, los arreglos sólo se pasan a funciones **mediante referencia**.
Es decir que la función trabajará **SIEMPRE** con el arreglo original y NO CON UNA COPIA DEL MISMO.[a]

[a] Piense en un arreglo de 1.000.000 elementos y analice desde el punto de vista del rendimiento: ¿es óptimo copiar tantos datos cada vez que se llama a la función?.

Funciones y arreglos unidimensionales II

En C/C++ el nombre de un arreglo es un puntero al primer elemento del mismo.

```
1  int arreglo[5] = {1};       //Arreglo inicializado a 1
2  int *ptr_entero;            //Puntero a entero
3  ptr_entero = arreglo;       //Alternativa 1
4  ptr_entero = &arreglo[0];   //Alternativa 2
```

Es decir, las últimas dos lineas son equivalentes.

Entonces... ¿cómo le indicamos a la función que va a recibir un arreglo?

- Como un array con tipo definido y sin dimensiones
- Como un array con tipo y dimensiones definidas
- Como un puntero

Funciones y arreglos unidimensionales III

Ejemplo utilizando un array con tipo definido y sin dimensiones:

```
1  #include<stdio.h>
2  #define TAM 10
3  /*PROTOTIPO*/
4  void cargar_vector(int []);
5  void imprimir_vector(int []);
6
7  int main (void)
8  {
9     int array_1d[TAM]={0};
10    cargar_vector(array_1d);
11    imprimir_vector(array_1d);
12 }
13
```

Funciones y arreglos unidimensionales IV

```
14  void cargar_vector (int vector[])
15  {
16    int ii;
17    int dato;
18    for (ii=0;ii<TAM;ii++)
19    {
20      printf("Ingrese el elemento %d\n",ii);
21      scanf("%d",&dato);
22      vector[ii]=dato;
23    }
24  }
25
26
27
28
```

Funciones y arreglos unidimensionales V

```
29  void imprimir_vector (int vector[])
30  {
31    int kk;
32    for (kk=0;kk<TAM;kk++)
33    {
34      printf(" %d",vector[kk]);
35    }
36  }
```

▸ Ver ejemplo completo en github

Funciones y arreglos unidimensionales VI

Ejemplo utilizando un array con tipo y dimensiones definidas:

```
1  #include<stdio.h>
2  #define TAM 10
3  /*PROTOTIPO*/
4  void cargar_vector(int vector[TAM]);
5  void imprimir_vector(int vector[TAM]);
6
7  int main (void)
8  {
9     int array_1d[TAM]={0};
10    cargar_vector(array_1d);
11    imprimir_vector(array_1d);
12 }
13
14
```

Funciones y arreglos unidimensionales VII

```
15
16  void cargar_vector (int vector[TAM])
17  {
18    int ii;
19    int dato;
20    for (ii=0;ii<TAM;ii++)
21    {
22      printf("Ingrese el elemento %d\n",ii);
23      scanf("%d",&dato);
24      vector[ii]=dato;
25    }
26  }
27
28
29
```

Funciones y arreglos unidimensionales VIII

```
30
31  void imprimir_vector (int vector[TAM])
32  {
33     int kk;
34     for (kk=0;kk<TAM; ii++)
35     {
36        printf(" %d",vector[kk]);
37
38     }
39  }
```

▸ Ver ejemplo completo en github

Funciones y arreglos unidimensionales IX

Ejemplo utilizando un puntero =D :

```
1  #include<stdio.h>
2  #define TAM 10
3
4  /*PROTOTIPO*/
5  void cargar_vector(int *);
6  void imprimir_vector(int *);
7
8
9  int main (void)
10 {
11    int array_1d[TAM]={0};
12    cargar_vector(array_1d);
13    imprimir_vector(array_1d);
14 }
```

Funciones y arreglos unidimensionales X

```
15
16
17  void cargar_vector (int *vector)
18  {
19    int ii;
20    int dato;
21    for (ii=0;ii<TAM;ii++)
22    {
23      printf("Ingrese el elemento %d\n",ii);
24      scanf("%d",&dato);
25      vector[ii]=dato;
26    }
27  }
28
29
```

Funciones y arreglos unidimensionales XI

```
30
31  void imprimir_vector (int *vector)
32  {
33     int ii;
34     for (ii=0;ii<TAM;ii++)
35     {
36        printf(" %d",vector[ii]);
37     }
38  }
```

Funciones y arreglos unidimensionales XII

Escritura de una función genérica para trabajar con arreglos del mismo tipo, pero distinta longitud.

```
1  #include<stdio.h>
2  /*PROTOTIPO*/
3  void cargar_vector(int [], int);
4  void imprimir_vector(int [],int);
5
6
7
8
9
10
11
12
```

Funciones y arreglos unidimensionales XIII

```
13  int main (void)
14  {
15     int array_1[10]={0};
16     int array_2[3]={0};
17     printf("Carga del primer arreglo\n");
18     cargar_vector(array_1,10);
19     printf("Carga del segundo arreglo\n");
20     cargar_vector(array_2,3);
21     printf("Impresion del primer arreglo\n");
22     imprimir_vector(array_1,10);
23     printf("Impresion del segundo arreglo\n");
24     imprimir_vector(array_2,3);
25     printf("Reutilizamos las funciones!!!\n");
26     return(0);
27  }
```

Funciones y arreglos unidimensionales XIV

```
28
29  void cargar_vector (int vector[], int tam)
30  {
31    int ii;
32    int dato;
33    for (ii=0;ii<tam;ii++)
34    {
35      printf("Ingrese el elemento %d\n",ii);
36      scanf("%d",&dato);
37      vector[ii]=dato;
38    }
39  }
40
41
42
```

Funciones y arreglos unidimensionales XV

```
43  void imprimir_vector (int vector[], int tam)
44  {
45    int ii;
46    for (ii=0;ii<tam;ii++)
47    {
48      printf(" %d",vector[ii]);
49    }
50    printf("\n");
51  }
```

▸ Ver ejemplo completo en github

Ejemplos I

1. Diseñar y codificar un programa que implemente las siguientes funciones:
   - Saludo: imprimir los datos personales del desarrollador.
     Prototipo: void saludo(void);
   - Menú: permitir al operador seleccionar una de las siguientes opciones:
     - Cargar un vector de 10 elementos de tipo entero
     - Imprimir el vector
     - Imprimir el mayor elemento dentro del arreglo
     - Imprimir el menor elemento dentro del arreglo
     - Imprimir la media de todos los elementos del arreglo
     - Imprimir los elementos mayores a la media
     - Imprimir los elementos menores a la media

     Prototipo: int menu(void);

Ejemplos II

- Cargar un vector de 10 elementos de tipo entero.
  Prototipo: void cargar (int []);
- Imprimir el vector
  Prototipo: void imprimir (int []);
- Imprimir el mayor elemento dentro del arreglo.
  Prototipo: void imprimeMayor (int []);
- Impimir el menor elemento dentro del arreglo
  Prototipo: void imprimeMenor (int []);
- Imprimir en main la media de todos los elementos del arreglo.
  Prototipo: float calculaMedia (int []);
- Imprimir los elementos mayores a la media, donde la media es recibida como parámetro desde la función main()
  Prototipo: void imprimeMayoresMedia (int [],float);

Ejemplos III

- Imprimir los elementos menores a la media, donde la media es recibida como parámetro desde la función main()
  Prototipo: void imprimeMenoresMedia (int [],float);

▸ Ver en github

Funciones y arreglos bidimensionales I

Para el caso de los arreglos bidimensionales, es necesario indicar en el prototipo de la función la cantidad de columnas que tiene arreglo. Este parámetro es utilizado por el compilador para poder determinar la posición de memoria de cada elemento y así poder operar con ellos.

```
1  #include<stdio.h>
2  #define FIL 2
3  #define COL 3
4
5  /*PROTOTIPOS*/
6  void cargar_matriz(int [][COL]);
7  void imprimir_matriz(int [][COL]);
8
```

Funciones y arreglos bidimensionales II

```
 9
10  int main (void)
11  {
12    int array_2d[FIL][COL];
13
14    /*LLAMADA A FUNCIONES*/
15    cargar_matriz(array_2d);
16    imprimir_matriz(array_2d);
17    return(0);
18  }
19
20
21
22
23
```

Funciones y arreglos bidimensionales III

```
24  void cargar_matriz (int matriz[][COL])
25  {
26     int ii;
27     int jj;
28     int dato;
29     for (ii=0;ii<FIL;ii++)
30     {
31        for(jj=0;jj<COL;jj++)
32        {
33        printf("Ingrese el elemento %d %d\n",ii,jj);
34        scanf("%d",&dato);
35        matriz[ii][jj]=dato;
36        }
37     }
38  }
```

Funciones y arreglos bidimensionales IV

```
39
40  void imprimir_matriz (int matriz[][COL])
41  {
42    int ii, jj;
43    int dato;
44    for (ii=0;ii<FIL;ii++)
45    {
46      for(jj=0;jj<COL;jj++)
47      {
48      printf("%d \t",matriz[ii][jj]);
49      }
50      printf("\n");
51    }
52  }
```

▸ Ver en github

Ejemplos I

1. Diseñar y codificar un programa declare en main un arreglo de 7 filas por 9 columnas y luego implemente las siguientes funciones:
   - Cargar arreglo
   - Imprimir arreglo
   - Imprimir la traza de la matriz
   - Imprimir todas las columnas de una fila especifica. El valor de la fila es recibido por valor desde main.
   - Imprimir todas las filas de una columna especifica. El valor de la fila es recibido por referencia desde main..
   - Menú con las opciones citadas anteriormente

▸ Ver en github

Ejemplos II

2. Una estación meteorológica toma muestras de temperatura y humedad una vez por hora durante 30 días Dicha información se almacena en dos arreglos bidimensionales declarados de la siguiente manera:

   ```
   float temperatura [30][24];
   float humedad [30][24];
   ```

   Diseñar y codificar un programa en C que implemente las siguientes funciones:

   - Cargar arreglos: debe cargar valores aleatorios en los arreglos. La temperatura debe variar entre -10^oC y 50^oC y la humedad entre 0% y 100%

Ejemplos III

- Imprimir las 24 muestras de un día. El número de día se recibe por parámetro desde la función main
- Imprimir la mayor temperatura de un día. El número de día se recibe por referencia desde la función main
- Imprimir la menor temperatura de un día. El número de día se recibe por referencia desde la función main
- Imprimir el día y la hora de la temperatura máxima y mínima
- Imprimir el promedio de temperaturas y humedades de cada uno de los dias

▸ Ver en github

¡Muchas gracias!

Consultas:

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