Introducción a la programación en C

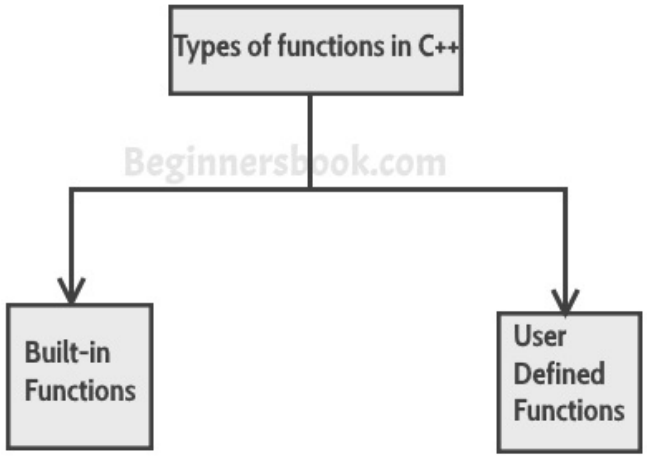
Informática I - Instituto Unviersitario Areonáutico

May 25, 2021

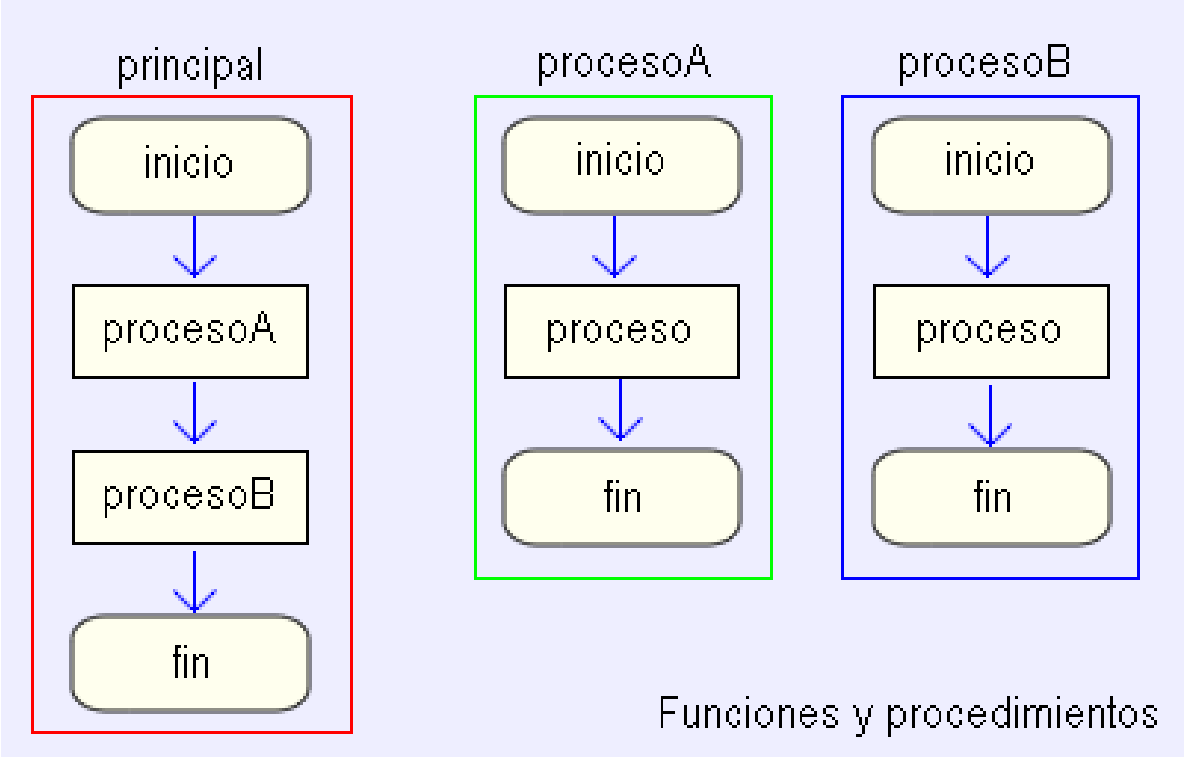
Funciones definición

Las funciones permiten a los desarrolladores dividir un programa en módulos independientes.

- Funciones pre-empaquetadas de C: permiten realizar cálculos matemáticos, operaciones con cadenas de texto, operaciones de entrada y salida de datos, etc.
- Definidas por el desarrollador: permiten realizar tareas particulares del algoritmo en cuestión.



Funciones definición



Anatomía de un función en C I

Las funciones se pueden clasificar en 4 tipos según su naturaleza:

- Si Reciben y si retornan datos: calcular el promedio de dos números
- No reciben y si retornan datos: menú de opciones
- No reciben y no retornan datos: función saludo
- Si reciben y no retornan datos: impresión de datos

Anatomía de un función en C I

Prototipo

Consiste en una presentación de la función. En el se define que tipo de dato retorna y si lo hace, el nombre de la misma y el tipo de dato de lo/los parámetros que recibe. En caso de ser mas de un parámetros, se los separa por comas.

Ejemplos de prototipos:

```
1  float   promedio (int , int);
2  int     menu (void);
3  void    saludo (void);
4  void    imprimir (int, float, char);
```

Notar que para indicar que una función no recibe y/o no retorna parámetros, se utiliza la palabra reservada **void**. Además, las funciones en C pueden recibir muchos valores y de distintos tipos, pero **sólo pueden retornar un único dato**.

Anatomía de un función en C II

Estructura general de una función en C

Luego de la declarar el prototipo de la función, se procede a la especificación formal de la misma.

La sentencia return

Dicha sentencia fuerza la salida inmediata de la función. Es decir que las sentencias que se encuentren después de una sentencia return(); no serán ejecutadas.
Esta sentencia puede ser utilizada para retornar valores, siempre y cuando el tipo de retorno **no sea void**.

Anatomía de un función en C III

Ejemplo general:

```
1  tipo_valor_de_retoro  nombre_de_la_funcion (parametros)
2  {
3     definiciones;
4     instrucciones;
5     return(); //segun el tipo de funcion
6  }
```

Ejemplos particulares:

```
1  float promedio (int n1, int n2)
2  {
3     float resultado=0;
4     resultado=(n1+n2)/2.0;
5     return(resultado)
6  }
```

Anatomía de un función en C IV

```
1  int menu (void)
2  {
3     int opcion=0;
4     printf("1-Comenzar juego\n");
5     printf("2-Guardar partida\n");
6     printf("3-Cargar partida\n");
7     printf("4-Salir\n");
8     scanf("%d",&opcion);
9     return(opcion);
10 }
```

```
1  void saludo (void)
2  {
3     printf("**********************\n");
4     printf("*Bienvenido al juego*\n");
5     printf("**********************\n");
6  }
```

Llamada a funciones I

- Se realiza con el nombre de la función
- Si la función recibe datos, estos deben ser enviados al momento de la llamada. En orden, entre paréntsis y separados por comas
- Si la función NO recibe datos, se deben colocar los paréntesis vacíos
- Si la función retorna parámetros, debemos asignar el valor de retorno a una variable
- Una llamada a una función es una sentencia de C. Por ello debe colocarse el ; al final de la misma

▸ Ver ejemplo I completo en github

Ámbito de variables

Variables locales

Se declaran dentro de una función y sólo están disponibles durante su ejecución. Cuando la función termina, son destruidas.

Variables globales

Globales: Se declaran fuera de las funciones y existen durante todo el ciclo de vida del programa.
Su uso NO es considerado una buena práctica de programación.

▸ Ver ejemplo I completo en github

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;
9     printf("Ingrese un numero \n");
10    scanf("%d",&n1);
11
12
```

Mecanismo de paso de argumentos a funciones II

```
13      printf("N1 antes de llamar a suma_tres %d \n",n1);
14      resultado=suma_tres(n1);
15      printf("N1 despues de llamar a suma_tres %d \n",n1)
16      printf("Valor de resultado %d \n",resultado);
17
18  }
19
20  int suma_tres (int n1)
21  {
22      int resultado=0;
23      n1=n1+3;
24      resultado=(n1+3);
25      return(resultado);
26  }
```

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     int resultado;
10    printf("Ingrese un numero \n");
11    scanf("%d",&n1);
12
```

Mecanismo de paso de argumentos a funciones IV

```
13      printf("N1 antes de llamar a suma_tres %d \n",n1);
14      resultado=suma_tres(&n1);
15      printf("N1 despues de llamar a suma_tres %d \n",n1);
16      printf("Valor de resultado %d \n",resultado);
17
18  }
19
20  int suma_tres (int *n1)
21  {
22      int resultado=0;
23      *n1=*n1+3;
24      resultado=(*n1+3);
25      return(resultado);
26  }
```

¿Qué significan los símbolos * y &?

Punteros I

Definición

Los punteros son variables cuyos valores son direcciones de memoria. En general, esta dirección de memoria es la ubicación de otra variable.

La declaración de una variable puntero se realiza indicando el tipo de dato, seguido de un * y el nombre de la variable:

```
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;
```

Punteros II

Operador ampersand (&)

Se lo conoce como *operador de dirección*. Devuelve la dirección de memoria de un operando.

Operador *

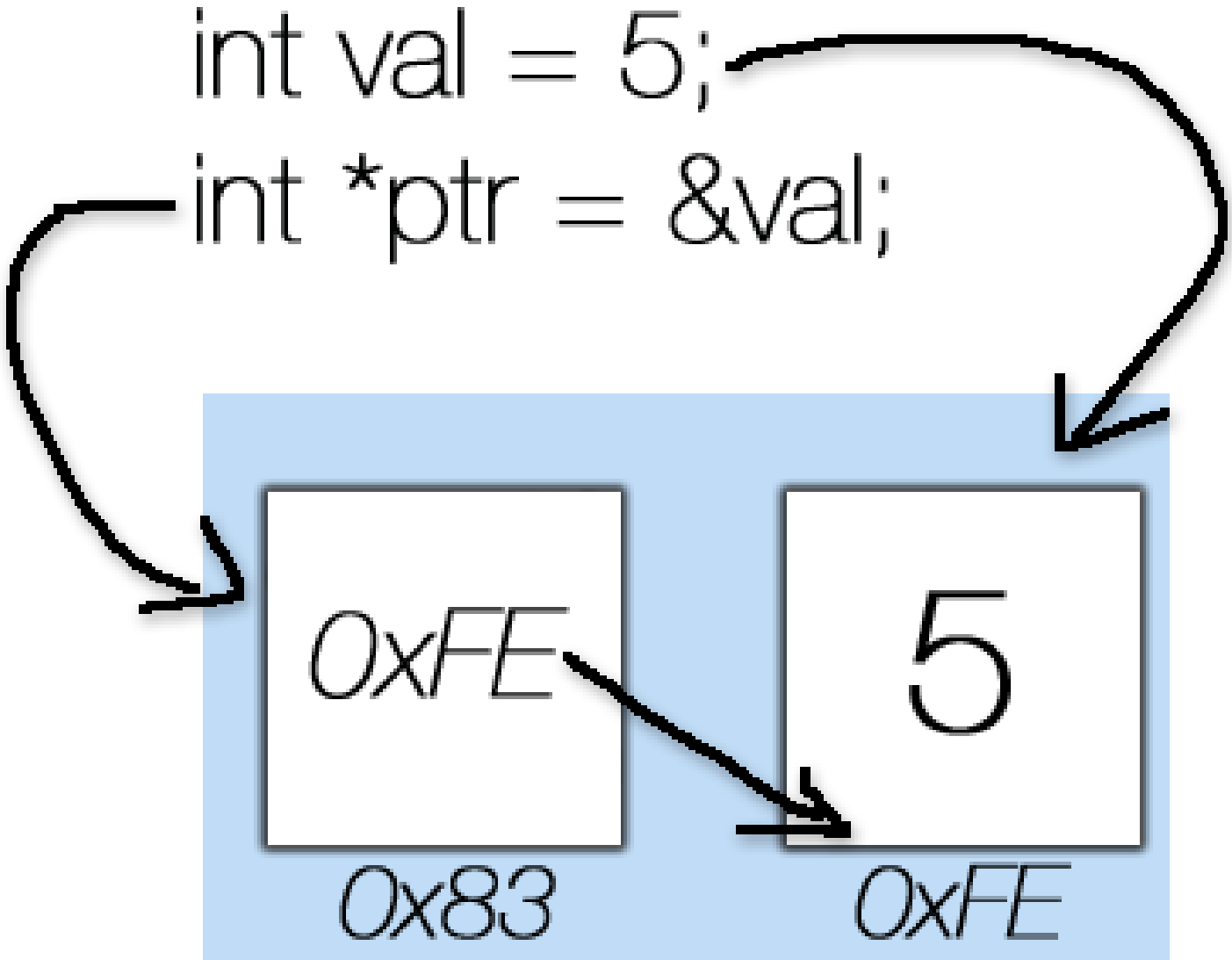
Se lo conoce como operador *operador de indirección o desreferencia*, devuelve el valor del objeto al que apunta su operando

Ejemplo:

Punteros III

```c
#include<stdio.h>
#include <stdlib.h>
int main(void)
{

  int *ptr=NULL;
  int  val=5;

  ptr=&val;

  printf("Valor de val %d\n",val);
  printf("Posicion de memoria de val %X\n",&val);
  printf("Valor de ptr %X\n",ptr);
  printf("Posicion de memoria de ptr %X\n",&ptr);
  printf("Val accedido por ptr %d\n",*ptr);

return(0);
}
```

Punteros IV



Volviendo a las funciones: paso por referencia I

```
1  #include<stdio.h>
2
3  /*PROTOTIPO*/
4  void swap(int *,int *);
5
6  int main (void)
7  {
8     int n1=0;
9     int n2=0;
10
11    printf("Ingrese un numero \n");
12    scanf("%d",&n1);
13    printf("Ingrese un numero \n");
14    scanf("%d",&n2);
15    printf("N1 antes de llamar a swap %d \n",n1);
16    printf("N2 antes de llamar a swap %d \n",n2);
17    swap(&n1,&n2);
18
```

Volviendo a las funciones: paso por referencia II

```
19      printf("N1 despues de llamar a swap %d \n",n1);
20      printf("N2 despues de llamar a swap %d \n",n2);
21      return(0);
22  }
23
24  void swap (int *n1, int *n2)
25  {
26      int auxiliar=0;
27      auxiliar=*n1;
28      *n1=*n2;
29      *n2=auxiliar;
30  }
```

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
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*).
   ▸ Ver en github

Ejemplos II

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.
   - 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 III

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.

¿Cómo le indicamos a la función que va a recibir un arreglo?

Hay tres formas de señalar esto en el prototipo de la función:

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

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

Funciones y arreglos unidimensionales II

Ejemplo utilizando un array con tipo definido y sin dimensiones:

```
1  #define TAM 10
2  /*PROTOTIPOS*/
3  void cargar_vector(int []);
4  void imprimir_vector(int []);
5
6  int main (void)
7  {
8     int array_1d[TAM]={0};
9     /*LLAMADA A FUNCIONES*/
10    cargar_vector(array_1d);
11    imprimir_vector(array_1d);
12    return(0);
13 }
14
15
16
17
```

Funciones y arreglos unidimensionales III

```
18
19
20
21
22  void cargar_vector (int vector[])
23  {
24     int ii;
25     int dato;
26     for (ii=0;ii<TAM;ii++)
27     {
28        printf("Ingrese el elemento %d\n",ii);
29        scanf("%d",&dato);
30        vector[ii]=dato;
31     }
32  }
33
34
35
```

Funciones y arreglos unidimensionales IV

```
36
37
38
39
40  void imprimir_vector (int vector[])
41  {
42     int ii;
43     for (ii=0;ii<TAM;ii++)
44     {
45        printf(" %d",vector[ii]);
46     }
47  }
```

▸ Ver ejemplo completo en github

Funciones y arreglos unidimensionales V

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
8
9  int main (void)
10 {
11    int array_1d[TAM]={0};
12    cargar_vector(array_1d);
13    imprimir_vector(array_1d);
14 }
15
16
17
```

Funciones y arreglos unidimensionales VI

```
18
19
20
21  void cargar_vector (int vector[TAM])
22  {
23     int ii;
24     int dato;
25     for (ii=0;ii<TAM;ii++)
26     {
27        printf("Ingrese el elemento %d\n",ii);
28        scanf("%d",&dato);
29        vector[ii]=dato;
30     }
31  }
32
33
34
35
```

Funciones y arreglos unidimensionales VII

```
36
37
38  void imprimir_vector (int vector[TAM])
39  {
40     int ii;
41     for (ii=0;ii<TAM;ii++)
42     {
43        printf(" %d",vector[ii]);
44     }
45  }
```

▸ Ver ejemplo completo en github

Funciones y arreglos unidimensionales VIII

Ejemplo utilizando un puntero :) :

```
1  #include<stdio.h>
2  #define TAM 10
3  /*PROTOTIPO*/
4  void cargar_vector(int *);
5  void imprimir_vector(int *);
6
7
8
9  int main (void)
10 {
11    int array_1d[TAM]={0};
12    cargar_vector(array_1d);
13    imprimir_vector(array_1d);
14 }
15
16
17
```

Funciones y arreglos unidimensionales IX

```
18
19
20
21  void cargar_vector (int *vector)
22  {
23     int ii;
24     int dato;
25     for (ii=0;ii<TAM;ii++)
26     {
27        printf("Ingrese el elemento %d\n",ii);
28        scanf("%d",&dato);
29        vector[ii]=dato;
30     }
31  }
32
33
34
35
```

Funciones y arreglos unidimensionales X

```
36
37
38  void imprimir_vector (int *vector)
39  {
40     int ii;
41     for (ii=0;ii<TAM;ii++)
42     {
43        printf(" %d",vector[ii]);
44     }
45  }
```

▸ 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);
   - Cargar un vector de 10 elementos de tipo entero.
     Prototipo: void cargar (int []);
   - Imprimir el vector
     Prototipo: void imprimir (int []);

Ejemplos II

- 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);
- 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