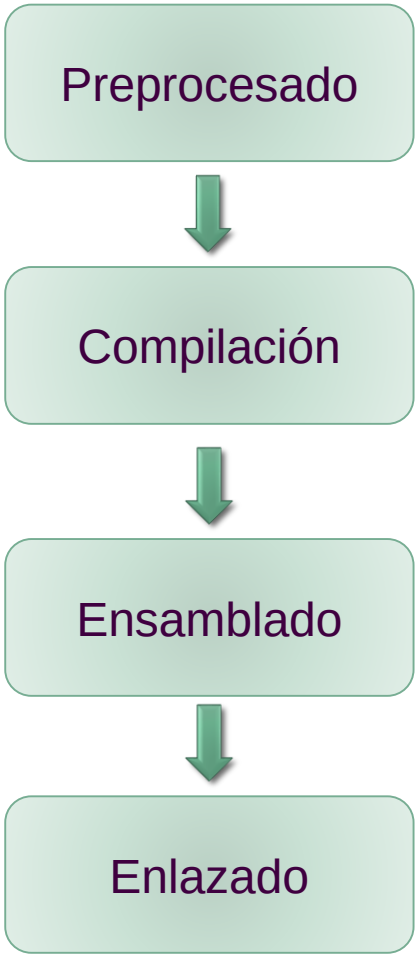
INTRODUCCIÓN AL LENGUAJE DE PROGRAMACIÓN C

Aránzazu Jurío Munárriz

ALGORITMIA

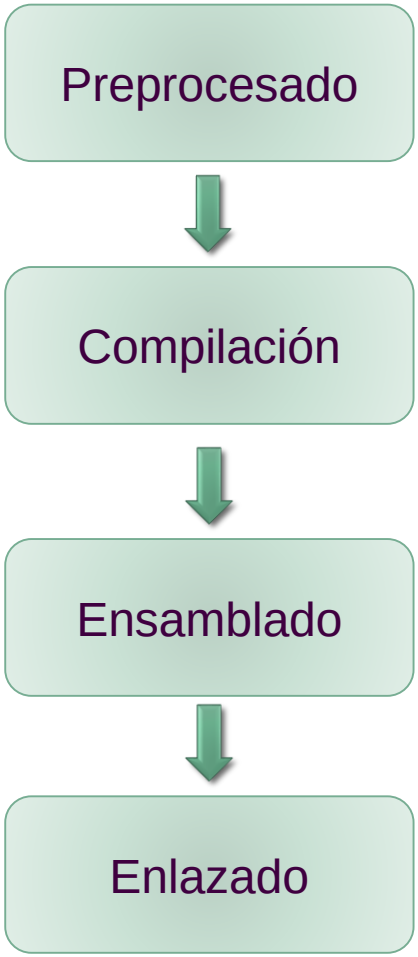
2018/2019

Compilador



- El preprocesado actúa sobre el programa fuente antes de que empiece la compilación propiamente dicha, para realizar ciertas operaciones. Por ejemplo, las variables inicializadas con #define (constantes simbólicas) son sustituidas en el código por su valor en todos los lugares donde aparece su nombre.
- La compilación transforma el código C en lenguaje ensamblador propio del procesador de nuestra máquina (lenguaje máquina).
- El ensamblado transforma el programa escrito en lenguaje ensamblador a código objeto, un archivo binario en lenguaje máquina ejecutable por el procesador.
- El enlazado (linkaje) consiste en añadir rutinas (propias o bibliotecas existentes en el mercado) que también están en código máquina, es decir, están en objeto. Una vez enlazado el programa objeto, tenemos un programa ejecutable por el ordenador.

Compilador



- Si trabajamos con un programa en un único archivo fuente, todo el proceso anterior puede hacerse en un solo paso:

gcc − o ejecutable codigo.c

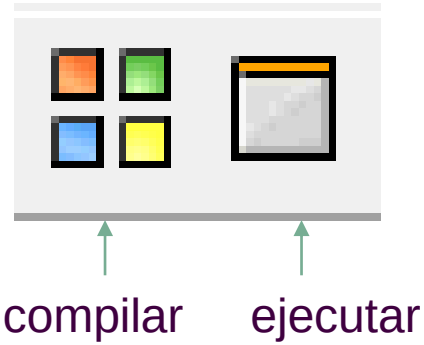
- Para ejecutar el programa utilizamos:

./ejecutable

Estas instrucciones se deben ejecutar en la consola de comandos

Entorno de desarrollo

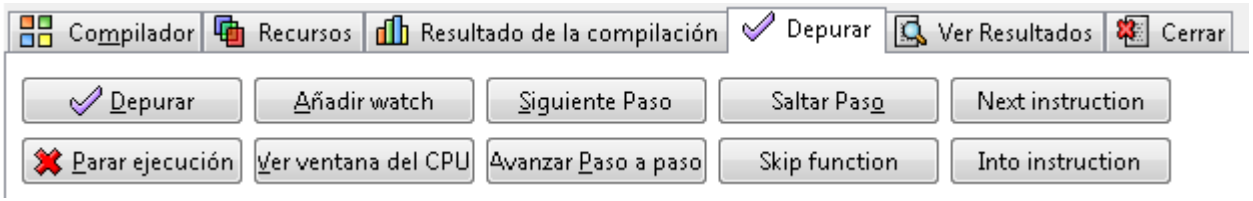
- Vamos a trabajar en el Entorno de Desarrollo Integrado (IDE) Dev-Cpp (Dev C++)



Si al ejecutar un programa se cierra automáticamente la consola, podemos añadir al final del programa principal la sentencia system(“pause”) de la librería stdlib.

Entorno de desarrollo

- Depurador
  - Para depurar nuestros códigos utilizamos la herramienta que provee el IDE
    - Permite colocar puntos de ruptura
    - Permite avanzar paso a paso en el programa
    - Permite ver los valores de las variables en cada momento, mediante “añadir watch”



Estructura de un programa (opción 1)

```
/*   Autor y Fecha
       Descripción del programa
*/

//Cabecera
//Incluir librerías estándar
#include <stdio.h>
//Declaración de constantes
#define PI 3.1416
//Declaración de tipos
typedef...
//Prototipos de funciones
int suma (int a, int b);
//Declaración de variables globales
int max=10;
```

```
//Programa principal
void main (void){
    //Variables del p. principal
    char nombre[25];
    int edad=20;
    //Código del p. principal
}

//Implementación de procedimientos y funciones
int suma (int a, int b){
    //Variables de la función
    int sum;
    //Código de la función
    sum=a+b;
    return sum;
}
```

Estructura de un programa (opción 1)

```
/*   Autor y Fecha
       Descripción del programa
*/

//Cabecera
//Incluir librerías estándar
#include <stdio.h>
//Declaración de constantes
#define PI 3.1416
//Declaración de tipos
typedef...
//Prototipos de funciones
int suma (int a, int b);
//Declaración de variables globales
int max=10;
```

```
//Programa principal
void main (void){
    //Variables del p. principal
    char nombre[25];
    int edad=20;
    //Código del p. principal
}

//Implementación de procedimientos y funciones
int suma (int a, int b){
    //Variables de la función
    int sum;
    //Código de la función
    sum=a+b;
    return sum;
}
```

Estructura de un programa (opción 1)

```
/*   Autor y Fecha
       Descripción del programa
*/

//Cabecera
//Incluir librerías estándar
#include <stdio.h>
//Declaración de constantes
#define PI 3.1416
//Declaración de tipos
typedef...
//Prototipos de funciones
int suma (int a, int b);
//Declaración de variables globales
int max=10;
```

```
//Programa principal
void main (void){
    //Variables del p. principal
    char nombre[25];
    int edad=20;
    //Código del p. principal
}

//Implementación de procedimientos y funciones
int suma (int a, int b){
    //Variables de la función
    int sum;
    //Código de la función
    sum=a+b;
    return sum;
}
```

Estructura de un programa (opción 1)

```
/*   Autor y Fecha
       Descripción del programa
*/

//Cabecera
//Incluir librerías estándar
#include <stdio.h>
//Declaración de constantes
#define PI 3.1416
//Declaración de tipos
typedef...
//Prototipos de funciones
int suma (int a, int b);
//Declaración de variables globales
int max=10;
```

```
//Programa principal
void main (void){
    //Variables del p. principal
    char nombre[25];
    int edad=20;
    //Código del p. principal
}

//Implementación de procedimientos y funciones
int suma (int a, int b){
    //Variables de la función
    int sum;
    //Código de la función
    sum=a+b;
    return sum;
}
```

Estructura: ejemplo (opción 1)

```
#include<stdio.h>                    //Librería para mostrar por pantalla
int suma(int a, int b);              //Declaración de  función
main(){                              //Programa principal
        int valor1,valor2;           //Variables del p. principal
        int resul;
        valor1=8;                    //Instrucciones
        valor2=3;
        resul=suma(valor1,valor2);
        printf("%d",resul);          //Mostrar por pantalla
}
int suma(int a, int b){              //Implementación de la función
        int r;                       //Variables de la función
        r=a+b;
        return r;                    //Valor que devuelve la función
}
```

Estructura de un programa (opción 2)

```
/*   Autor y Fecha
       Descripción del programa
*/

//Cabecera
//Incluir librerías estándar
#include <stdio.h>
//Declaración de constantes
#define PI 3.1416
//Declaración de tipos
typedef...
//Declaración de variables globales
int max=10;
```

```
//Implementación de procedimientos y funciones
int suma (int a, int b){
    //Variables y código de la función
    int sum;
    sum=a+b;
    return sum;
}

//Programa principal
void main (void){
    //Variables y código del p. principal
    char nombre[25];
    int edad=20;
}
```

Estructura: ejemplo (opción 2)

```
#include<stdio.h>                  //Librería para mostrar por pantalla
int suma(int a, int b){            //Implementación de la función
        int r;
        r=a+b;
        return r;                  //Valor que devuelve la función
}
main(){                            //Programa principal
        int valor1,valor2;         //Variables del p. principal
        valor1=8;                  //Instrucciones
        valor2=3;
        int resul;                 // Más variables del p. principal
        resul=suma(valor1,valor2);
        printf("%d",resul);        //Mostrar por pantalla
}
```

Tipos de datos

- int: números enteros
- float: números en coma flotante
- double: más grandes que float
- char: caracteres

- Cuantificadores
  - short
  - long
  - signed
  - unsigned

short int	2 bytes	-32.768 - +32.767
unsigned short int	2 bytes	0 - +65.535
unsigned int	4 bytes	0 - 4.294.967.295
Int	4 bytes	-2.147.483.648 - +2.147.483.647

Variables

- Cada variable tiene asociado un nombre, un tipo y un valor
- Una declaración asocia un tipo de datos a una variable
- Todas las variables deben ser declaradas antes de utilizarlas
- Se puede asignar valores iniciales en la declaración

```
int edad = 24;
float real1,real2;
char letra=’a’;
char frase[20];
```

Instrucciones

- Todas las instrucciones terminan en ;
- Operadores aritméticos: +, −,*, /, %
- Operadores monarios: ++, −−
- Operadores relacionales y lógicos: <, <=, >, >=, ==, ! =, &&, ||
- Operadores de asignación: =, + =, − =, *=, / =, % =

```
a=b*c;
a+=b;
a>=b;
a++;
```

Librerías

- <ctype.h>
  - tolower (int c)
  - toupper (int c)
- <stdlib.h>
  - int abs (int i)
  - int rand (void)
- <string.h>
  - char *strcopy (char *s1, const char *s2)
  - int strcmp(const char *s1, const char *s2)
- <math.h>
  - double ceil (double x)
  - double floor (double x)
  - double sin (double x)
  - double cos (double x)
  - double tan (double x)
  - double sqrt (double x)
  - double fabs (double x)
  - double pow (double x, double y)

http://www.acm.uiuc.edu/webmonkeys/book/c_guide/

Entrada / Salida

Funciones estándar de C para realizar entrada de datos por teclado y salida de datos hacia pantalla. Son funciones definidas en la librería estándar. Para usarlas es necesario incluir el fichero de cabecera:

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

Estas funciones son:

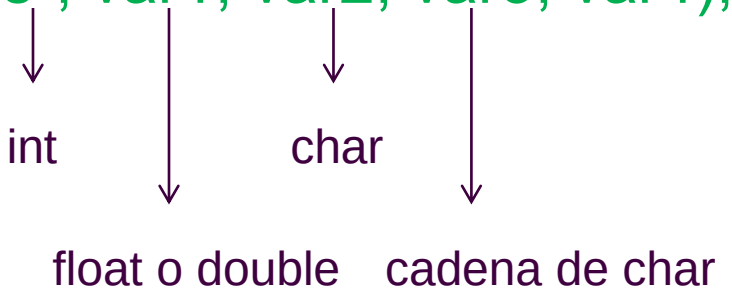
- printf( ) → Salida de datos con formato.
- scanf( ) → Entrada de datos con formato.
- getchar( ) → Entrada de 1 carácter.
- putchar( ) → Salida de 1 carácter.
- fflush( ) → Borrado del buffer del teclado.

Printf

printf (cadena de control, variables)

cadena de control: cadena de caracteres, entre comillas dobles, que especifica cómo va a ser la salida

printf(“4 variables: %d, %f, %c, %s”, var1, var2, var3, var4);

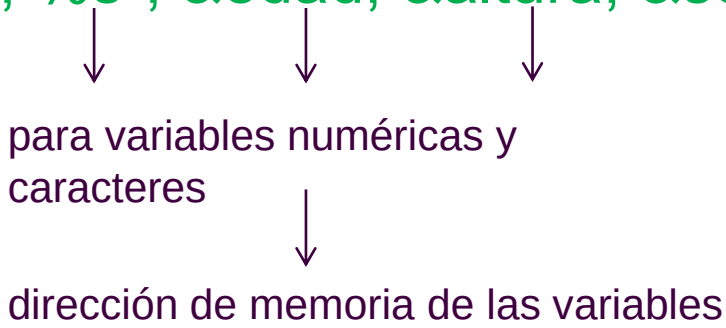


Scanf

scanf (cadena de control, variables)

cadena de control: cadena de caracteres, entre comillas dobles, que especifica cómo van a ser introducidos los datos

scanf(“%d, %f, %c, %s”, &edad, &altura, &sexo, nombre);



Ejemplo

- Escribir un programa que pida al usuario su nombre y su edad, y devuelva el año en que nació.

```
Introduce tu nombre: Maria
Introduce tu edad: 16
Maria, naciste en 2000
```

Solución

```
#include <stdio.h>

main(){
        char nombre[10];
        int edad;
        printf("Introduce tu nombre: ");
        scanf("%s",nombre);
        printf("Introduce tu edad: ");
        scanf("%d",&edad);
        printf("%s, naciste en %d", nombre, 2016-edad);
}
```

Instrucciones de control

- Sentencias de selección
  - if – else
  - switch
- Sentencias de iteración (bucles)
  - while
  - do – while
  - for
- Sentencias de salto
  - break
  - continue

Sentencias de selección

```
if
(condición1)
   sentencia1;
```

```
if (condición1)
   sentencia1;
else
   sentencia2;
```

```
if
(condición1){
   sentencia1;
   sentencia2;
}
else
   sentencia3;
```

```
if (condición1)
   if (condición2)
      sentencia1;
   else
      sentencia2;
else
   sentencia3;
```

```
if (condición1)
   if (condición2)
      sentencia1;
   else
      sentencia2;
```

```
if (condición1){
   if (condición2)
      sentencia1;
}
else
   sentencia2
```

Ejemplo

- Escribe un programa que pida un número al usuario y devuelva ese número multiplicado por 10 si el número es menor que 10, el mismo número si su valor está entre 10 y 19, y el número menos 5 si es mayor de 19.

```
Introduce un numero:
15
Resultado: 15
```

```
Introduce un numero:
5
Resultado: 50
```

```
Introduce un numero:
20
Resultado: 15
```

Solución

```
#include <stdio.h>

main(){
        int num;
        printf("Introduce un numero: ");
        scanf("%d",&num);
        if (num < 10)
                printf("Resultado: %d\n", num*10);
        else
                if (num < 20)
                        printf("Resultado: %d\n", num);
                else
                        printf("Resultado: %d\n", num-5);
}
```

Sentencias de selección

```
switch (expresion1)
{
   case constante 1:
      secuencia de sentencias;
      break;
   case constante 2:
      secuencia de sentencias;
      break;
   ...
   default:
      secuencia de sentencias;
}
```

- break para no comprobar el resto de casos
- sólo puede comparar igualdades, mientras que el if puede evaluar expresiones relacionales o lógicas

Ejemplo

- Escribir un programa que lea un número entero entre 1 y 5, y devuelva por pantalla el nombre de dicho número

```
Introduce un numero:
3
tres
```

Solución

```
#include <stdio.h>

main(){
        int num;
        printf ("Introduce un numero: ");
        scanf("%d",&num);
        switch (num){
                case 1:
                        printf("Uno\n");
                        break;
                case 2:
                        printf("Dos\n");
                        break;
                case 3:
                        printf("Tres\n");
                        break;
                case 4:
                        printf("Cuatro\n");
                        break;
                case 5:
                        printf("Cinco\n");
                        break;
                default:
                        printf("Numero no valido\n");
        }
}
```

Sentencias de iteración

```
while (condición){
   sentencias
}
```

```
do{
   sentencias
} while (condición);
```

- do – while analiza la condición al final del bucle → el bloque sentencias se ejecuta al menos una vez
- aunque solo haya una sentencia en el do – while es recomendable utilizar “{“ “}” para no confundir con el while

```
for (inicialización; condición; incremento){
   sentencias
}
```

- for (i=0; i<=19; i++)

Ejemplo

- Escribir un programa que lea números enteros hasta que el usuario introduzca un 3.

```
Introduce un numero: 5
Introduce un numero: 4
Introduce un numero: 3
```

- Escribir un programa que lea números enteros hasta que la suma obtenida sobrepase 20.

```
Introduce un numero: 5
Introduce un numero: 18
La suma es 23
```

- Escribir un programa que muestre los números múltiplos de 3 que estén entre 1 y 15.

```
Multiplos de 3: 3 6 9 12 15
```

Solución 1

```
#include <stdio.h>

main(){
        int num;
        do{
                printf("Introduce un numero: ");
                scanf("%d",&num);
        }while(num!=3);
}
```

Solución 2

```
#include <stdio.h>

main(){
        int num;
        int suma=0;
        while(suma<=20){
                printf("Introduce un numero: ");
                scanf("%d",&num);
                suma=suma+num;
        }
        printf("La suma es %d\n",suma);
}
```

Solución 3

```
#include <stdio.h>

main(){
        int i;
        printf("Multiplos de 3: ");
        for(i=1;i<=15;i++)
                if(i%3==0)
                        printf("%d ",i);
}
```

Sentencias de salto

- break
  - finalizar un case en una sentencia switch
  - forzar la terminación inmediata de un bucle
- continue
  - dentro de un bucle, ignora el resto de sentencias y comienza una nueva iteración

Ejemplo

- ¿Qué muestran cada uno de estos programas?

```
#include <stdio.h>

main(){
   int i=0;
   while (i < 5){
      i++;
      if (i%2==0)
         break;
      printf("i=%d\n",i);
   }
}
```

```
#include <stdio.h>

main(){
   int i=0;
   while (i < 5){
      i++;
      if (i%2==0)
         continue;
      printf("i=%d\n",i);
   }
}
```

Solución

i=1

i=1

i=3

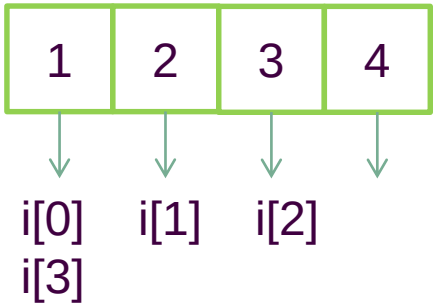
i=5

Tablas

tipo nombre_variable[numero_de_elementos];

float numeros[25];
int valores[10];

int i[4]={1,2,3,4};



Tablas multidimensionales

tipo nombre_variable [dimension1] [dimension2] [dimension3] ...

int i[2][3]={{1,2,3},{4,5,6}};

1	2	3
4	5	6

i[0][0]=1
i[1][2]=6

- En C no se puede operar con todo un vector o toda una matriz como una única entidad, sino que hay que tratar sus elementos uno a uno por medio de bucles for o while.

Ejemplo

- Escribe un programa en el que se tenga almacenada una matriz de 3 filas y dos columnas con los números del 1 al 6. El programa debe mostrar por pantalla dicha matriz.
- Escribe un programa que lea 5 números en una tabla. Posteriormente debe sumar uno a cada uno de los elementos, y mostrar los nuevos valores de la tabla por pantalla.

Solución 1

```
#include<stdio.h>

main(){
        int matriz[3][2]={{1,2},{3,4},{5,6}};
        int i,j;
        for(i=0;i<3;i++){
                for(j=0;j<2;j++)
                        printf("%d\t",matriz[i][j]);
                printf("\n");
        }
}
```

Solución 2

```
#include<stdio.h>

main(){
        int i;
        int tabla[5];
        for(i=0;i<5;i++){
                printf("Introduce un elemento: ");
                scanf("%d",&tabla[i]);
        }
        for(i=0;i<5;i++){
                tabla[i]=tabla[i]+1;
        }
        for(i=0;i<5;i++){
                printf("%d ",tabla[i]);
        }
        printf("\n");
}
```

Cadenas de caracteres

- Una cadena se define como un array de caracteres que termina en un carácter nulo ‘\0’.
- No es necesario añadir dicho carácter explícitamente, pero al declarar una variable hay que tenerlo en cuenta para la longitud máxima.

```
char cad[8]=“La casa”;
char cad[8]={‘L’, ’a’, ’ ‘, ‘c’, ‘a’, ‘s’, ‘a’, ‘\0’};
char cad[]=“La casa”;
```

- #include <string.h>
  - strcmp: compara dos cadenas lexicográficamente.
  - strcomp: compara dos cadenas lexicográficamente sin considerar mayúsculas o minúsculas.
  - strcpy: copia una cadena en otra
  - strlen: devuelve el número de caracteres de la cadena.
  - strcat: concatena dos cadenas

Ejemplo

- Escribir un programa que pida una cadena de caracteres al usuario y compruebe si es palíndroma o no.

  > Introduce una cadena: ana

  La cadena es palindroma

  > Introduce una cadena: casa

  La cadena no es palindroma

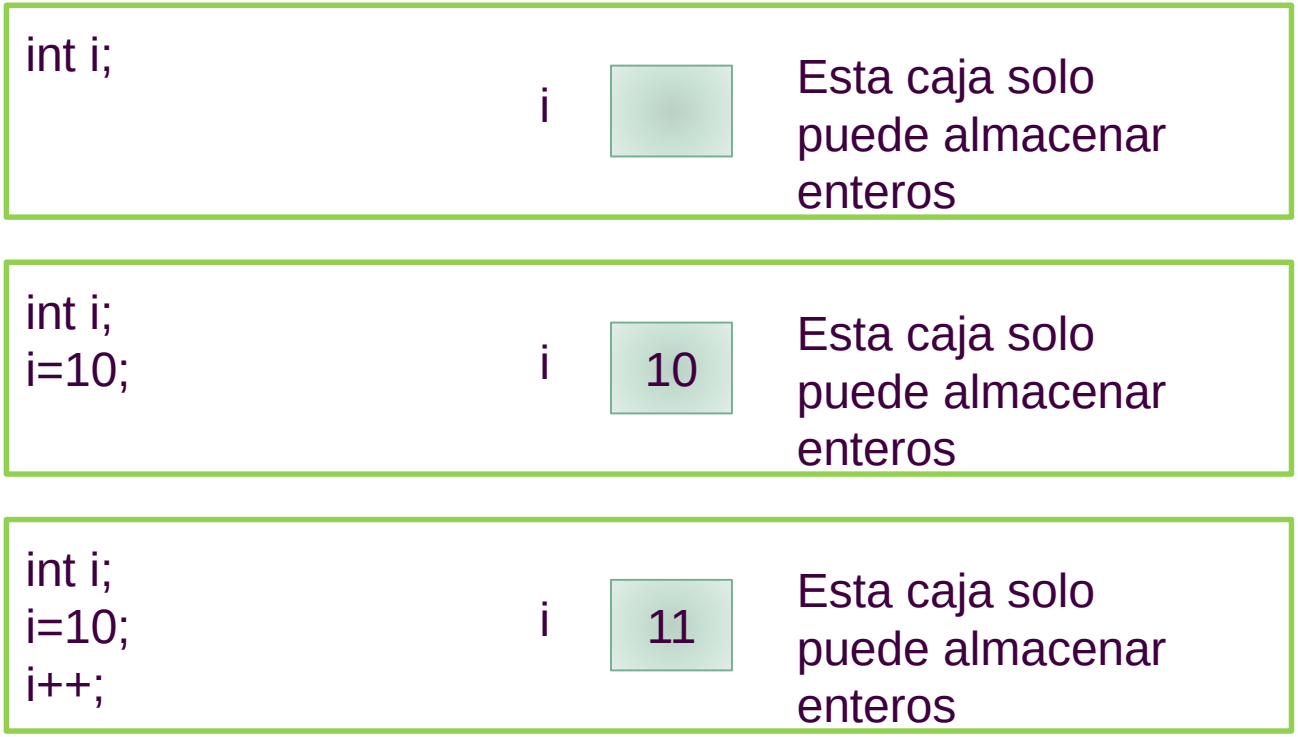
Para leer una cadena de caracteres hasta que introduzcamos un salto de línea, en lugar de “%s” (lee hasta un espacio en blanco) utilizamos **“%[^\n]”**

Solución

```
#include<stdio.h>
#include<string.h>

main(){
        char cadena[20];
        int i=0, distintos=0;
        printf("Introduce una cadena: ");
        scanf("%s",cadena);
        while(distintos==0 && i<strlen(cadena)/2){
                if (cadena[i]!=cadena[strlen(cadena)-1-i])
                        distintos=1;
                i++;
        }
        if (distintos==0)
                printf("La cadena es palindroma\n");
        else
                printf("La cadena no es palindroma\n");
}
```

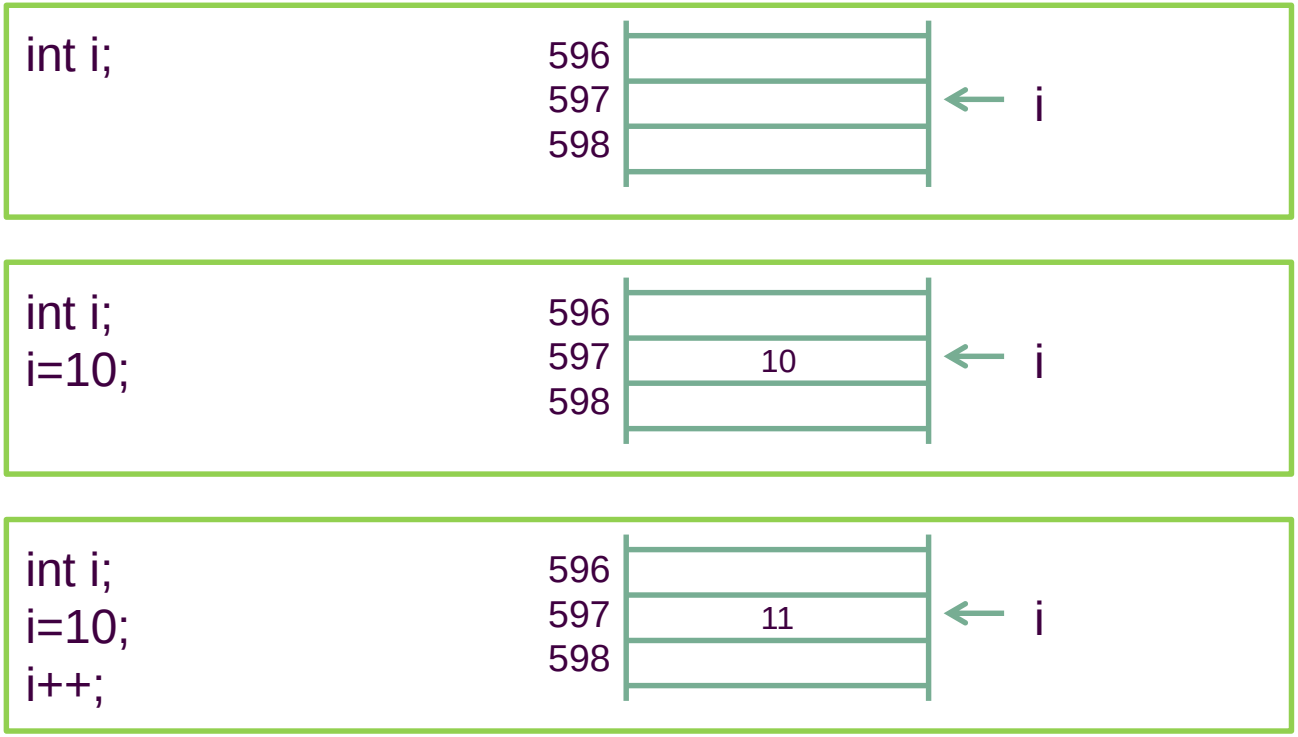
Variables y punteros

- Entendemos una variable como una caja que almacena un valor, de modo que al hacer referencia a ella trabajamos con su contenido.



Variables y punteros

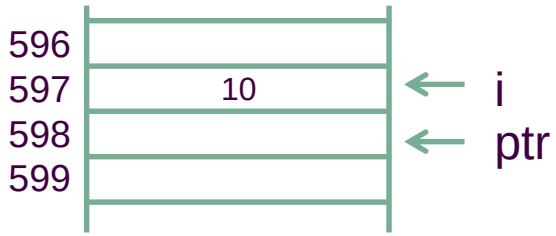
- Internamente, una posición de memoria se queda asignada a una variable.



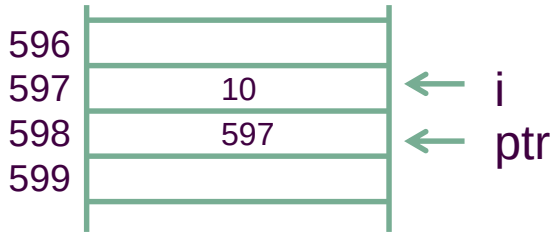
Variables y punteros

- Una variable definida como puntero almacena el valor de una dirección de memoria. Esta dirección corresponde a otra variable.

```
int i=10;
int *ptr;
```




```
int i=10;
int *ptr;
ptr=&i;
```



Punteros. Declaración

```
tipo_dato *nombre;
```

El puntero nombre apunta a una variable de tipo tipo_dato.

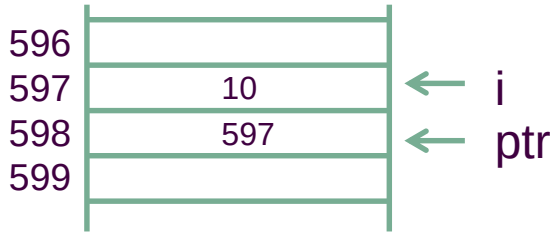
```
int var;
int *ptr;
ptr=&var;
```

```
char var;
int *ptr;
ptr=&var;
```

Punteros. Operaciones básicas

- &i: dirección de la variable i
- *ptr: contenido de la variable a la que apunta ptr

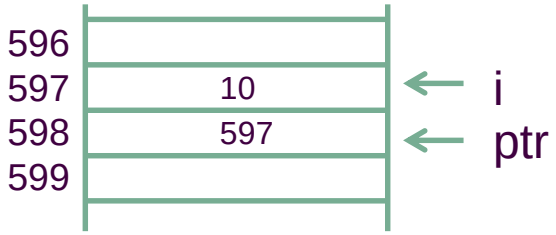
```
int i=10;
int *ptr;
ptr=&i;
```



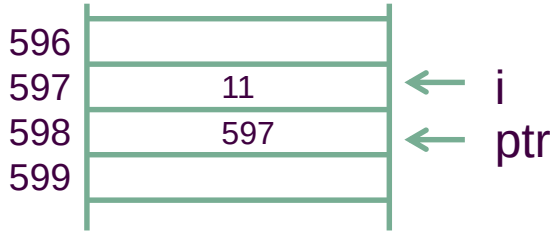
&i=597
*ptr=10

Punteros. Operaciones básicas

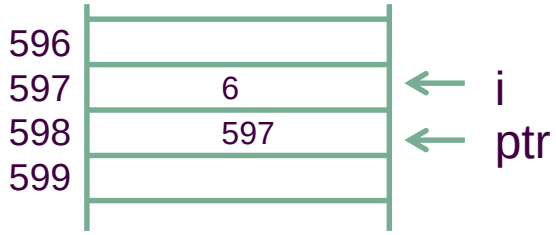
```
int i=10;
int *ptr;
ptr=&i;
```




```
int i=10;
int *ptr;
ptr=&i;
i++;
```




```
int i=10;
int *ptr;
ptr=&i;
i++;
*ptr=6;
```



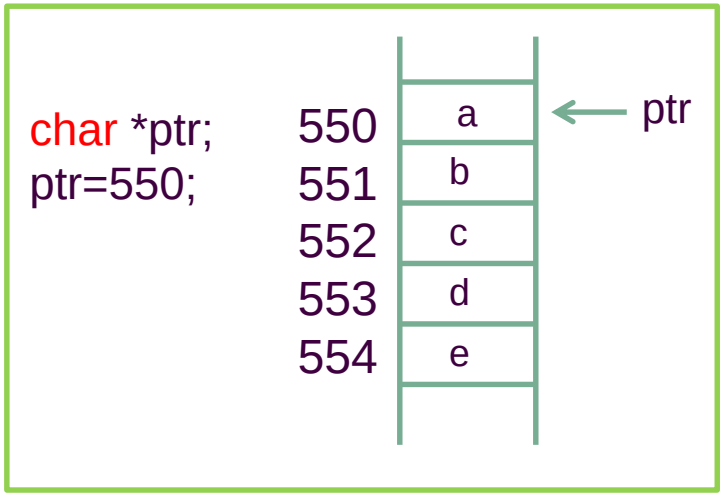
Ejemplo

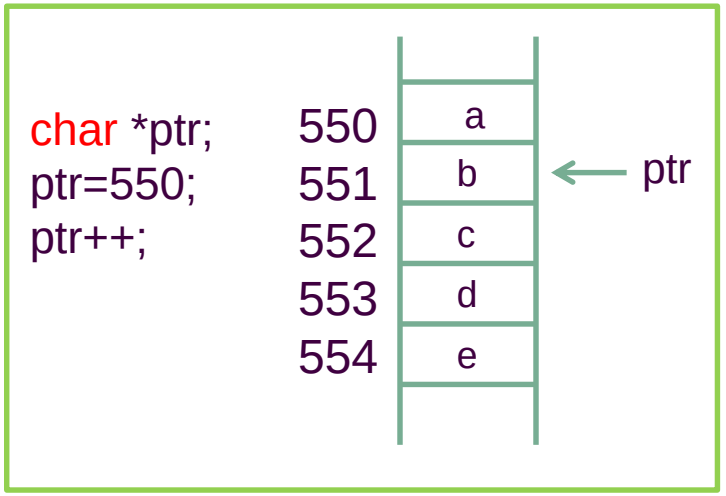
```
#include <stdio.h>
main(){
    int dato=5;
    int *ptDato;

    ptDato = &dato;
    printf("Dato esta almacenada en la direccion %p\n", &dato);
    printf("El valor de la variable ptDato es %p\n, ptDato);

    dato=10;
    printf("El contenido de dato es %d\n", dato);
    printf("Lo apuntado por ptDato es %d\n", *ptDato);

    *ptDato = 5;
    printf("El nuevo contenido de dato es %d\n", dato);
    printf("El nuevo contenido de lo apuntado por ptDato es %d\n, *ptDato);
}
```

```
Dato esta almacenada en la direccion 0022FF54
El valor de la variable ptDato es 0022FF54
El contenido de dato es 10
Lo apuntado por ptDato es 10
El nuevo contenido de dato es 5
El nuevo contenido de lo apuntado por ptDato es 5
```

Punteros. Aritmética

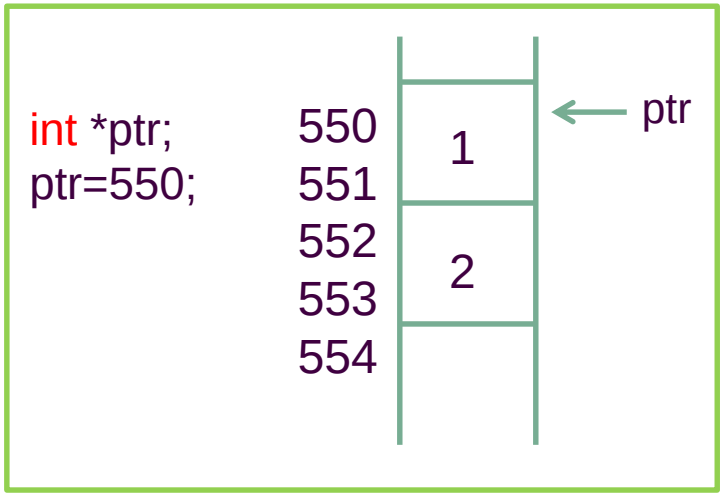
- Sólo se pueden realizar dos operaciones aritméticas sobre los punteros: suma y resta (no se pueden sumar doubles o floats a los punteros, solo valores enteros).
- Cada vez que se incrementa un puntero, apunta a la posición de memoria del siguiente elemento de su tipo base. Cada vez que se decrementa apunta a la posición del elemento anterior.

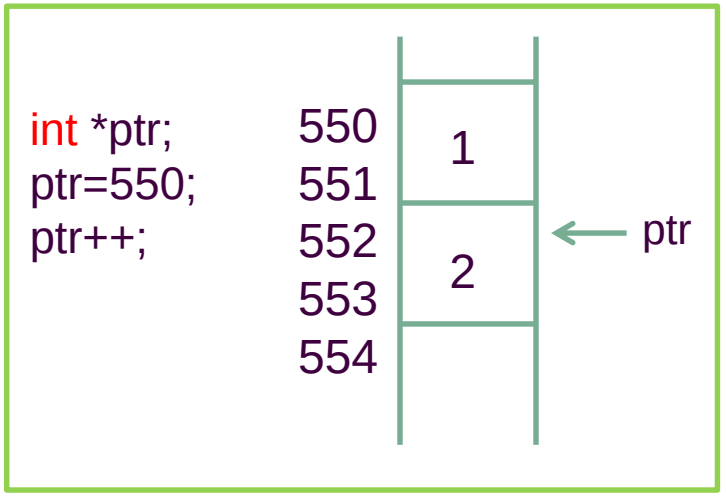




Punteros. Aritmética

- Sólo se pueden realizar dos operaciones aritméticas sobre los punteros: suma y resta (no se pueden sumar doubles o floats a los punteros, solo valores enteros).
- Cada vez que se incrementa un puntero, apunta a la posición de memoria del siguiente elemento de su tipo base. Cada vez que se decrementa apunta a la posición del elemento anterior.





Ejemplo

```
#include <stdio.h>
main(){
    int numero = 10;
    char letra = ‘a’;
    int *ptNum;
    char *ptLetra;

    ptNum = &numero;
    ptLetra = &letra;

    printf(“La direccion almacenada en ptNum es %p\n”, ptNum);
    printf(“La direccion siguiente a la almacenada en ptNum es %p\n”, (ptNum + 1));

    printf(“La direccion almacenada en ptLetra es %p\n”, ptLetra);
    printf(“La direccion siguiente a la almacenada en ptLetra es %p\n”, (ptLetra + 1));
}
```

La direccion almacenada en ptNum es 0022FF54
La direccion siguiente a la almacenada en ptNum es 0022FF58
La direccion almacenada en ptLetra es 0022FF53
La direccion siguiente a la almacenada en ptLetra es 0022FF54

Punteros y arrays

- Cuando declaramos un array se genera un puntero (constante, no se puede modificar) que apunta a la dirección del primer elemento del array.
- Para que un puntero apunte a un array no hace falta usar el operador “dirección” (&).

```
char cad[80], *p1;
p1=cad;
```

- Podemos acceder a los elementos del array con punteros.

```
cad[4] = *(p1+4)
```

Asignación dinámica de memoria

- Durante la ejecución no se pueden añadir variables globales o locales, pero sí lo podemos hacer mediante variables puntero.
- Utilizamos las funciones definidas en la librería stdlib.h:
  - void *malloc(num_bytes) ⟶ reserva memoria
  - void free (void *p) ⟶ libera memoria

- char *p;
  p=(char*)malloc(1000);
- int *p;
  p=(int*)malloc(50*sizeof(int));
  free(p);

Ejemplo sizeof

```
#include <stdio.h>
main(){
    printf(“El tamano del tipo int es %d\n”, sizeof(int));
    printf(“El tamano del tipo long es %d\n”, sizeof(long));
    printf(“El tamano del tipo short es %d\n”, sizeof(short));
    printf(“El tamano del tipo char es %d\n”, sizeof(char));
}
```

El tamano del tipo int es 4
El tamano del tipo long es 4
El tamano del tipo short es 2
El tamano del tipo char es 1

Ejemplo

- Escribe un programa que pida un número entero al usuario. Posteriormente, crea un vector de dicho tamaño y rellena todos sus elementos a 1. Imprímelo por pantalla.
- Escribe un programa que pida al usuario el número de elementos de un vector. Posteriormente crea dicho vector y rellénalo con los números reales que introduzca el usuario. Imprímelo por pantalla.

Solución 1

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

main(){
        int n,i;
        int *vector;
        printf("Introduce el tamano: ");
        scanf("%d",&n);
        vector=(int*)malloc(n*sizeof(int));
        for(i=0;i<n;i++)
                vector[i]=1;
        printf("El vector es: \n");
        for(i=0;i<n;i++)
                printf("%d ",vector[i]);
        printf("\n");
}
```

Solución 2

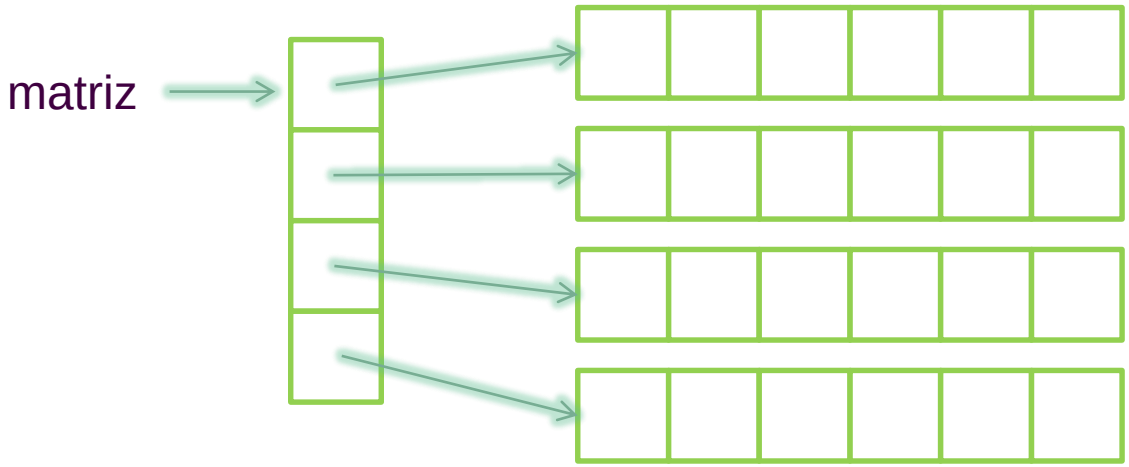
```
#include<stdio.h>
#include<stdlib.h>

main(){
        int n,i;
        float *vector;
        printf("Introduce el tamano: ");
        scanf("%d",&n);
        vector=(float*)malloc(n*sizeof(float));
        printf("Introduce los elementos: ");
        for(i=0;i<n;i++)
                scanf("%.2f",&vector[i]);
        printf("El vector es: \n");
        for(i=0;i<n;i++)
                printf("%f ",vector[i]);
        printf("\n");
}
```

Ejemplo dobles punteros

```
#include <stdio.h>
main(){
    int dato = 5;
    int *ptDato;
    int **ptPtDato;

    ptDato = &dato;
    ptPtDato = &ptDato;

    printf(“La direcccion almacenada en ptDato es %p\n”, ptDato);
    printf(“La direccion almacenaada en ptPtDato es %p\n”, ptPtDato);

    printf(“El valor al que apunta ptDato es %d\n”, *ptDato);
    printf(“El valor al que apunta ptPtDato es %p\n”, *ptPtDato);
    printf(“El valor apuntado doblemente por ptPtDato es %d\n”, **ptPtDato);
}
```

La direccion almacenada en ptDato es 0022FF54
La direccion almacenada en ptPtDato es 0022FF50
El valor al que apunta ptDato es 5
El valor al que apunta ptPtDato es 0022FF54
El valor apuntado doblemente por ptPtDato es 5

Punteros y matrices

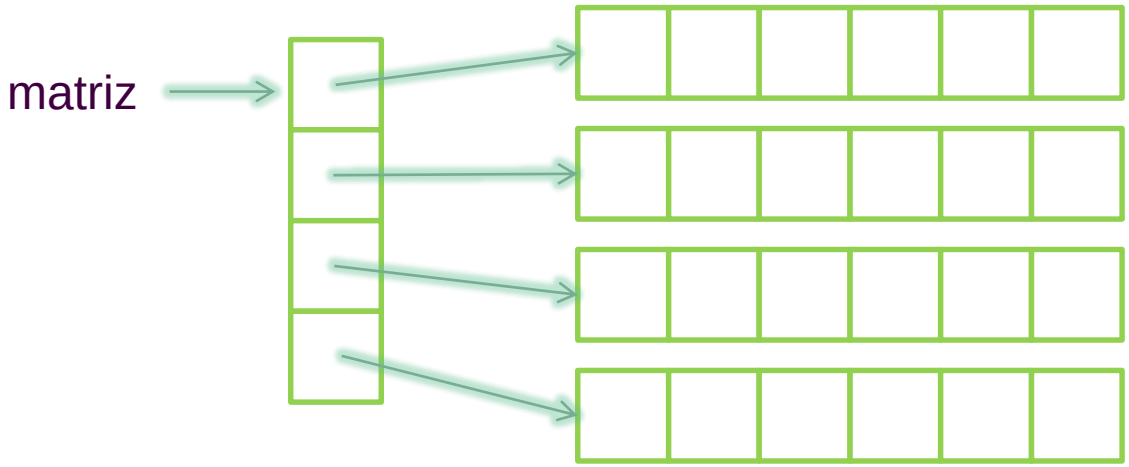


RESERVA DE MEMORIA

```
int **matriz;
matriz=(int**) malloc (4*sizeof(int*));
for (i=0;i<4;i++){
    matriz[i]=(int*) malloc
(6*sizeof(int));
}
```

```
int matriz[4][6];
```

Punteros y matrices



ACCESO A LOS ELEMENTOS

```
for (i=0;i<4;i++){
    for (j=0;j<6;j++){
        printf("%d\t",*(*(matriz+i)+j));
    }
    printf("\n");
}
```

```
for (i=0;i<4;i++){
    for (j=0;j<6;j++){
        printf("%d\t",matriz[i][j]);
    }
    printf("\n");
}
```

Ejemplo

- Escribe un programa que pida al usuario el tamaño de una matriz cuadrada. Posteriormente crea dicha matriz y rellena cada elemento con el número de su columna. Imprímela por pantalla.
- Escribe un programa que pida al usuario el número de filas y columnas de una matriz. Crea dicha matriz y rellena cada elemento con el número de su fila. Imprímela por pantalla.

Solución 1

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

main(){
		int n,i,j;
		int **mat;
		printf("Introduce el tamano de la matriz: ");
		scanf("%d",&n);
		mat=(int**)malloc(n*sizeof(int*));
		for(i=0;i<n;i++)
			mat[i]=(int*)malloc(n*sizeof(int));
		for(i=0;i<n;i++)
			for(j=0;j<n;j++)
				mat[i][j]=j;
		printf("La matriz es: \n");
		for(i=0;i<n;i++){
			for(j=0;j<n;j++)
				printf("%d ",mat[i][j]);
			printf("\n");
		}
}
```

Solución 2

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

main(){
        int fil,col,i,j;
        int **mat;
        printf("Introduce el numero de filas y columnas: ");
        scanf("%d %d",&fil,&col);
        mat=(int**)malloc(fil*sizeof(int*));
        for(i=0;i<fil;i++)
                mat[i]=(int*)malloc(col*sizeof(int));
        for(i=0;i<fil;i++)
                for(j=0;j<col;j++)
                        mat[i][j]=i;
        printf("La matriz es: \n");
        for(i=0;i<fil;i++){
                for(j=0;j<col;j++)
                        printf("%d ",mat[i][j]);
                printf("\n");
        }
}
```

Estructuras

- Es una colección de variables que se referencia bajo un único nombre.
- Permite mantener junta información que está relacionada.
- Una definición de estructura forma una plantilla que puede utilizarse para crear variables de estructura

```
typedef struct persona{
    inf edad;
    float peso;
    int altura;
}persona;

persona persona1;
persona1.edad=20;
```

Ejemplo

- Escribe un programa que pida al usuario información sobre dos cajas. Dicha información incluye su altura (número entero), su anchura (número entero) y su identificación (cadena de caracteres). Para almacenar la información de cada caja, define y utiliza una estructura. Posteriormente debe imprimir esta información por pantalla.

```
Introduce los datos de la caja 1: 10 20 caja1
Introduce los datos de la caja 2: 50 50 caja2
Estas son las cajas:
caja1: 10 cm (alto) x 20 cm (ancho)
caja2: 50 cm (alto) x 50 cm (ancho)
```

Solución

```
#include<stdio.h>

typedef struct caja{
	int alt;
	int anch;
	char id[8];
}caja;

main(){
int i;
	caja miscajas[2];
	for(i=0;i<2;i++){
		printf("Introduce los datos de la caja %d: ",i+1);
		scanf("%d %d %s", &miscajas[i].alt, &miscajas[i].anch, miscajas[i].id);
	}
	printf("Estas son las cajas:\n");
	for(i=0;i<2;i++)
		printf("%s: %d cm (alto) x %d cm (ancho)\n",miscajas[i].id, miscajas[i].alt,
miscajas[i].anch);
}
```

Estructuras y punteros

- Para acceder a un campo de la estructura, mediante una variable puntero, utilizamos el operador ->

```
typedef struct persona{
    inf edad;
    float peso;
    int altura;
}persona;
```

```
persona varpersona, *ptrpersona;
ptrpersona=&varpersona;
varpersona.edad=30;
varpersona.peso=60.5;
varpersona.altura=165;
printf("Los datos son: altura %d peso %.2f y edad
        %d\n", (*ptrpersona).altura, (*ptrpersona).peso,
        (*ptrpersona).edad);
printf("Los datos son: altura %d peso %.2f y edad
        %d\n", ptrpersona->altura, ptrpersona->peso,
        ptrpersona->edad);
```

Ejemplo

- Escribe un programa que pida al usuario información sobre dos cajas. Dicha información incluye su altura (número entero), su anchura (número entero) y su identificación (cadena de caracteres). Para almacenar la información de cada caja, define y utiliza una estructura. Utiliza una variable de tipo puntero para almacenar la información de cada caja. Posteriormente debe imprimir esta información por pantalla.

```
Introduce los datos de la caja 1: 10 20 caja1
Introduce los datos de la caja 2: 50 50 caja2
Estas son las cajas:
caja1: 10 cm (alto) x 20 cm (ancho)
caja2: 50 cm (alto) x 50 cm (ancho)
```

Solución

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

typedef struct caja{
		int alt;
		int anch;
		char id[8];
}caja;

main(){
		caja *caja1, *caja2;
		caja1=(caja*)malloc(sizeof(caja));
		caja2=(caja*)malloc(sizeof(caja));
		printf("Introduce los datos de la caja 1: ");
		scanf("%d %d %s",&caja1->alt, &caja1->anch, caja1->id);
		printf("Introduce los datos de la caja 2: ");
		scanf("%d %d %s",&caja2->alt, &caja2->anch, caja2->id);
		printf("Estas son las cajas:\n");
		printf("%s: %d cm (alto) x %d cm (ancho)\n",caja1->id, caja1->alt, caja1->anch);
		printf("%s: %d cm (alto) x %d cm (ancho)\n",caja2->id, caja2->alt, caja2->anch);
}
```

Funciones

```
tipo_de_datos nombre_de_la_funcion (tipo y nombre de argumentos)
{
    acciones;
}
```

- tipo_de_datos: tipo del dato que devuelve la función. Si no devuelve ningún dato utilizamos void.
- nombre_de_la_funcion: identificador de la función.
- tipo y nombre de argumentos: parámetros que recibe la función. Son variables locales dentro de la función. Siguen la forma: tipo1 variable1, tipo2 variable2, …
- acciones: sentencias que ejecutará la función. La instrucción return devuelve el valor de la variable que lo acompaña. Si no devuelve ningún valor, la sentencia return es optativa.

No se puede devolver un vector o una matriz, aunque sí un puntero a una estructura.

Funciones. Declaración

- Toda función debe ser declarada o implementada antes de ser utilizada.
- La declaración mediante prototipo se hace al comienzo del fichero, después de los #define y los #include.
- El prototipo es de la forma

tipo_de_datos nombre_funcion (tipo y nombre de argumentos)

- Coincide con la primera línea de la definición salvo:
  - Termina con el carácter “;”.
  - El nombre de los argumentos es optativo (el tipo es obligatorio)

Funciones. Paso de argumentos

- Paso por valor
  - Las variables dentro de la función son copias de las variables con que son llamadas.
  - Los cambios realizados dentro de la función no se pasan al programa que los ha llamado.
- Paso por referencia
  - Los argumentos son las direcciones de memoria de las variables.
  - Las modificaciones dentro de las funciones afectan al programa que las llamó.
  - Los arrays (vectores, matrices y cadenas de caracteres) se pasan siempre por referencia.

Funciones. Paso de argumentos

Paso por valor

```
#include <stdio.h>
void suma_valor (int x);
main(){
    int x=10;
    suma_valor(x);
    printf("P. prin: x vale %d",x);
}
void suma_valor (int x){
    x+=5;
    printf("Funcion: x vale %d",x);
    return;
}
```

Funcion: x vale 15
P. prin: x vale 10

Paso por referencia

```
#include <stdio.h>
void suma_referencia (int *x);
main(){
    int x=10;
    suma_referencia(&x);
    printf("P. prin: x vale %d",x);
}
void suma_referencia (int *x){
    *x=*x+5;
    printf("Funcion: x vale %d",*x);
    return;
}
```

Funcion: x vale 15
P. prin: x vale 15

Funciones. Paso de argumentos

```
#include <stdio.h>
void a_mayusculas (char frase[]);
main(){
    char cadena[]="esto es una prueba";
    a_mayusculas (cadena);
    printf("Después de ejecutar la funcion:\n%s\n",cadena);
}
void a_mayusculas (char frase[]){
    int i;
    for (i=0;i<strlen(frase);i++){
        if (frase[i]>=97 && frase[i]<=122)
            frase[i]-=32;
    }
    return;
}
```

Después de ejecutar la funcion:
ESTO ES UNA PRUEBA

Ejemplo

- Completa el siguiente programa y añade la llamada y la implementación de la función perfecto (devuelve 1 si un entero es perfecto – igual a la suma de sus divisores - y 0 si no lo es).

```
#include <stdio.h>

main(){
    int num;
    int resul;
    printf(“Introduce un numero entero: “);
    scanf(“%d”,&num);
    if (resul==1)
        printf(“El numero %d es perfecto\n”,num);
    else
        printf(“El numero %d no es perfecto\n”,num);
}
```

Solución

```
resul=perfecto(num);

int perfecto (int num){
        int suma=0;
        int i;
        for(i=1;i<=num/2;i++){
                if(num%i==0)
                        suma=suma+i;
        }
        if(num==suma)
                return 1;
        else
                return 0;
}
```

Ejemplo

- Completa el siguiente programa y añade la llamada y la implementación de la función cambiar (cambia el valor de dos variables entre sí).

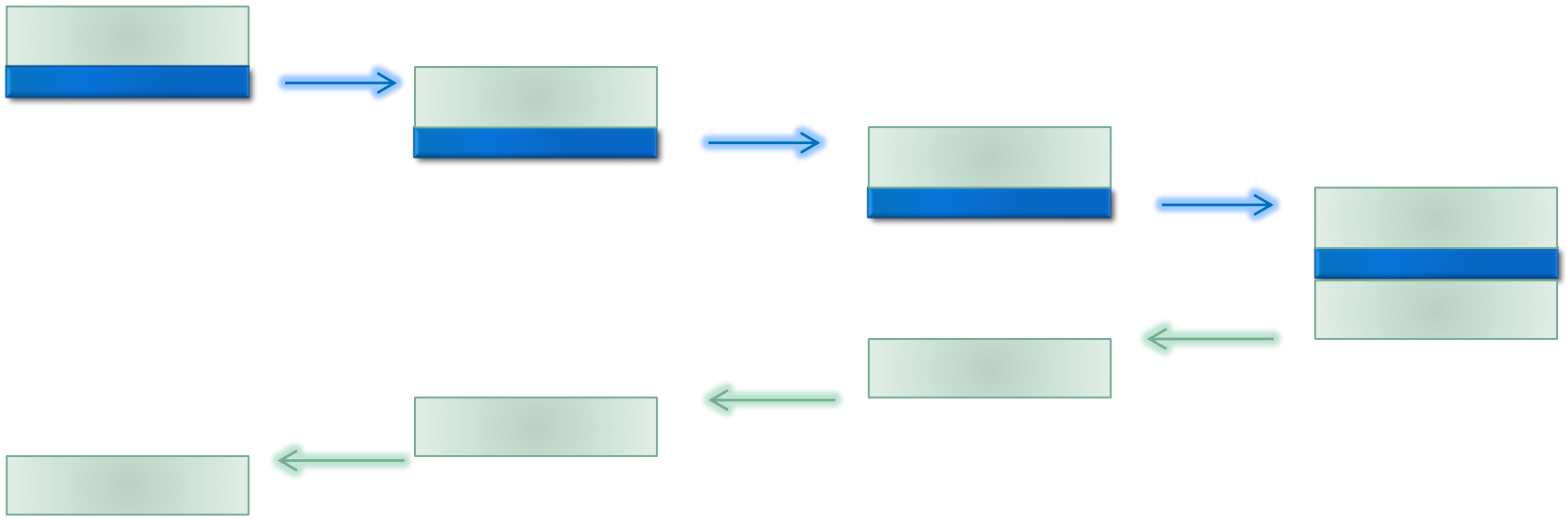
```
#include <stdio.h>

main(){
    int a,b;
    printf(“Introduce dos numeros enteros: “);
    scanf(“%d %d”,&a, &b);
    printf(“Ahora a vale %d y b vale %d”,a,b);
}
```

Solución

```
cambiar(&a,&b);


void cambiar (int *a, int *b){
        int aux;
        aux=*a;
        *a=*b;
        *b=aux;
}
```

Funciones recursivas

- Una función se llama a sí misma varias veces
- Es necesario un caso base
- La solución siempre debe combinar soluciones de problemas más pequeños



Ejemplo

- Factorial de un número

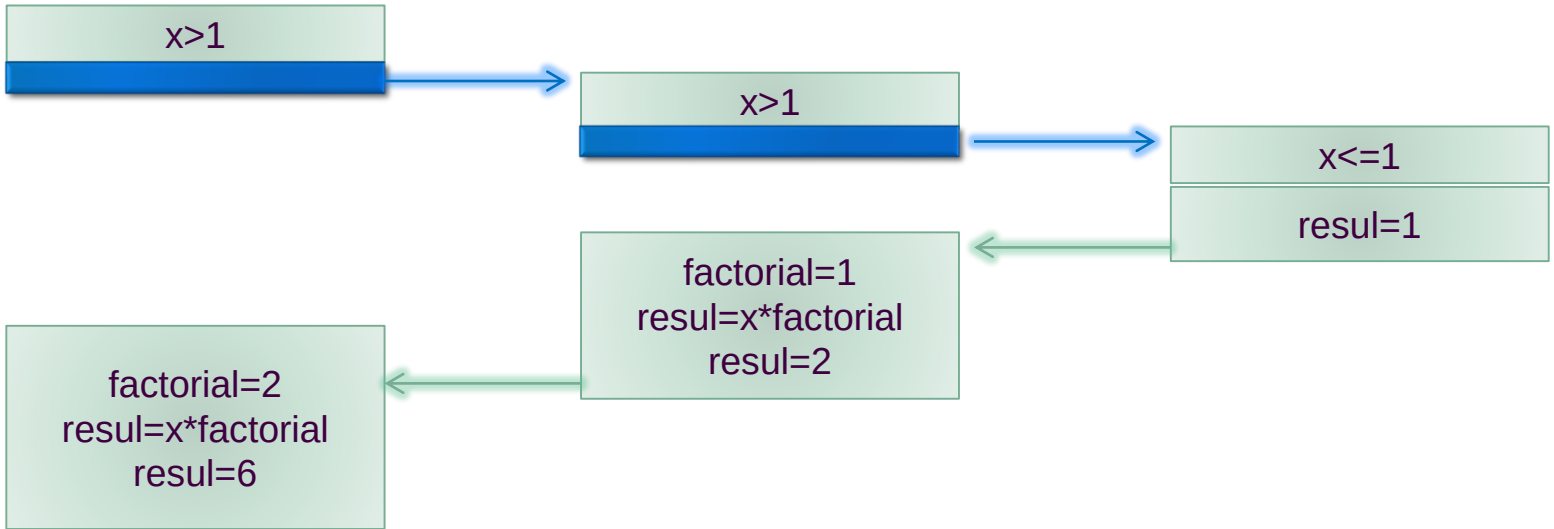
$$n! = \begin{cases} 1 & si \quad n = 1 \\ n * (n-1)! & en\ otro\ caso \end{cases}$$

  - La solución general utiliza la solución del mismo problema pero en un tamaño menor (n-1)!
  - El caso base es cuando el número es 1

Ejemplo

- Factorial de un número

```
int factorial (int x){
    int resul;
    if (x>1)
        resul=x*factorial(x-1);
    else
        resul=1;
    return resul;
}
```



Ejemplo

- Escribir una función recursiva espar que determine si un número entero dado es par o no, conociendo sólo que:
  - 1 es impar
  - si un número es impar, su antecesor es par, y viceversa

- ¿Cuál es el caso base?
- ¿Cuál es el subproblema, de tamaño menor, en el que baso mi solución?
- ¿Cómo complemento esa solución parcial?

Solución

- ¿Cuál es el caso base?
  - Si n = 1, entonces espar=0
- ¿Cuál es el subproblema, de tamaño menor, en el que baso mi solución?
  - espar(n-1)
- ¿Cómo complemento esa solución parcial?
  - si espar(n-1)=0 entonces espar(n)=1
  - si espar(n-1)=1 entonces espar(n)=0

Solución

```
int espar(int n){
        if(n==1)
                return 0;
        else
                if(espar(n-1)==0)
                        return 1;
                else
                        return 0;
}
```

Ejemplo

- Escribe un programa que sume los elementos de un vector de forma recursiva

```
Introduce el numero de elementos: 6
Introduce los valores:
4 5 3 0 1 6
La suma de los elementos es 19
```

Solución

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

int suma(int *t, int i, int j);

main(){
		int n,i,sum;
		int *v;
		printf("Introduce el numero de elementos: ");
		scanf("%d",&n);
		v=(int*)malloc(n*sizeof(int));
		printf("Introduce los valores:\n");
		for(i=0;i<n;i++)
				scanf("%d",&v[i]);
		sum=suma(v,0,n-1);
		printf("La suma de los elementos es %d\n",sum);
}

int suma(int *t, int i, int j){
		int sol;
		if(i==j)
				sol=t[i];
		else
				sol=t[i]+suma(t,i+1,j);
		return sol;
}
```