

# **METODOLOGÍA DE LA PROGRAMACIÓN**

**PRÁCTICAS**

# OBJETIVOS de la Práctica 1

---

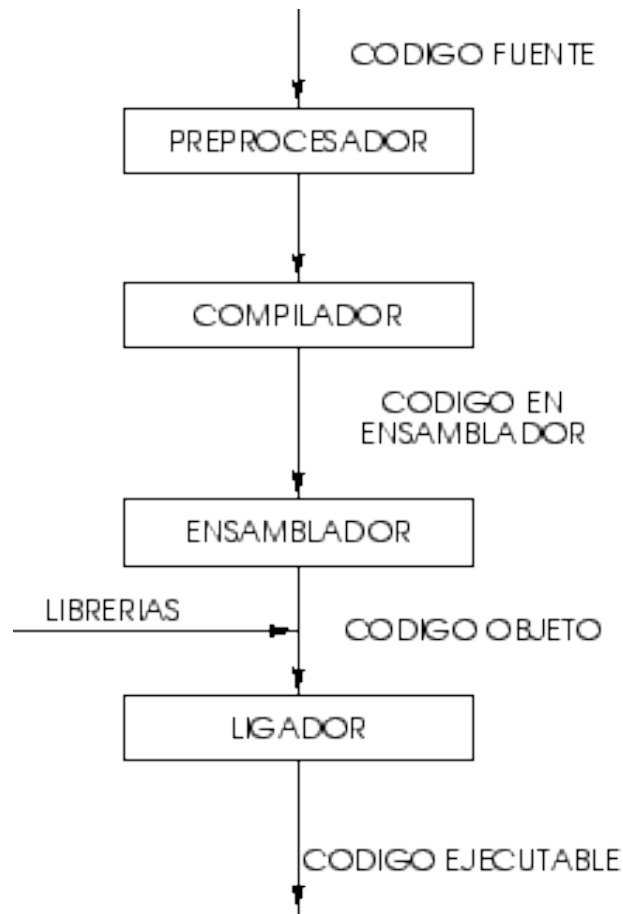
Repaso de:

- Proceso de Compilación de C
- Directivas del Preprocesador

Nuevos temas en C:

- Asignación **Dinámica** de Memoria
- Tratamiento de **Ficheros de Texto**

# Proceso de Compilación en C



El preprocesador acepta el código fuente como entrada y es responsable de:

- quitar los comentarios
- interpretar las **directivas del preprocesador** las cuales comienzan por #.

# Directivas del Preprocesador

- Una **directiva** es una sentencia especial que se ejecuta al principio del proceso de compilación.
  - ▢ Inclusión de ficheros: **#include**
  - ▢ Constantes o macros: **#define    #undef**
  - ▢ Para compilaciones condicionadas:  
**#if , #elseif, #else, #endif**

# Directiva `#include`

- **`#include`** : Inclusión de ficheros cabecera

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

**`#include "funciones.h"`**

# Directivas `#define` `#undef`



**#define** para definir **constantes**

**#define FALSO 0**

**#undef** para eliminar una definición

**#undef FALSO**

# Directivas #define #undef

## **#define** para definir **macros**

Una macro equivale a una expresión, una sentencia o un grupo de sentencias. (son parecidas a las funciones pero en el proceso de compilación se tratan de forma diferente)

```
#define bienvenida printf("Bienvenido al programa",  
    \n);
```

```
#define MAX(A,B) ( (A > B) ? (A) : (B) )
```

```
#define cuadrado(x) (x)*(x)
```

## **#undef** para eliminar una macro

```
#undef cuadrado
```

# Características de las Macros

- Pueden contener varias líneas escribiendo al final de cada línea una barra invertida (\) menos en la última línea.
- Puede sustituir a una función pero ...
  - ▮ Aumenta el código objeto porque el Preprocesador sustituye cada aparición del nombre de la macro por su definición completa.
  - ▮ No hace comprobación de tipos de datos por lo que existe más riesgo de errores cuando tienen argumentos.
  - ▮ Hay un mayor riesgo de efectos laterales.



# Directivas #if #elseif #else #endif

- Selección: en **general, la selección es como la de C**

```
#if expr
...
#elif expr
...
#elif expr
...
#else
...
#endif
```

Donde expr puede ser

- **defined** const
- **! defined** const
- **const == valor**

# Directivas: operador **defined**

- **#if defined**  $\equiv$  **#ifdef**
- **#if !defined**  $\equiv$  **#ifndef**
- Lo usaremos para programar los ficheros de cabecera y evitar que se dupliquen las inclusiones

```
/*Este es un fichero fichero.h*  
#ifndef_FICHERO_H_  
#define_FICHERO_H_  
    /* Aqui el código */  
#endif
```

# Uso en nuestros programas

```
#ifndef _LOGICO_  
#define _LOGICO_  
    enum logico{falso,verdad};  
#endif
```

# Ejemplo 1. Uso de Directivas

```
#include <stdio.h>
#include <stdlib.h>
#define NUM 5
#ifndef BIENVENIDA
#define BIENVENIDA printf("\n ***** Bienvenidos a las prácticas de
    MP"); \
                printf(" *****\n\n");
#endif

#define CUADRADO(x) x * x
int main() //ejemplo de uso de directivas
{
    BIENVENIDA;
    printf("\n el cuadrado de %d es %d \n", NUM, CUADRADO(NUM));
    printf("\n efecto colateral: el cuadrado %d es %d \n", NUM+2,
        CUADRADO(NUM+2));
    system("PAUSE");
    return 0;
```

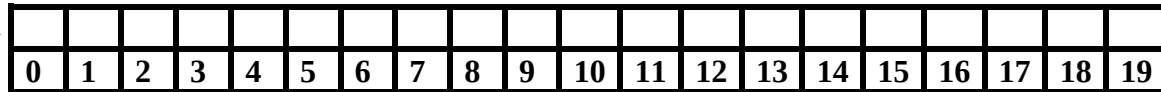
# Asignación de Memoria

- De forma **estática**: en tiempo de compilación
  - **Declarando variables y reservando un espacio fijo en memoria**
- De forma **dinámica**: en tiempo de ejecución
  - **Malloc**
  - **Calloc**
  - **Realloc**
  - **Free**

# Asignación Estática de Memoria

```
#define MAX 20
```

```
int valores[MAX];
```



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

```
int num_elem;
```

```
...
```

```
// Tabla de multiplicar del 3
```

```
num_elem= ?? // asignamos un valor dado
```

```
(for i=0; i<num_elem; i++)
```

```
    valores[i]=i*3;
```

# Asignación Estática de Memoria

## Inconvenientes:

- Puede estar **reservando más espacio del necesario**

num\_elem = 2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

- Puede **necesitar más espacio que el asignado**

num\_elem = 50    A partir de i=20 ¡¡ Error !!

# Asignación Dinámica de Memoria

- **Malloc** (**M**emory **alloc**ate) se utiliza para reservar memoria para una determinada variable. Dicha variable debe ser un puntero (Asignación de memoria sin inicializar: contiene basura)

prototipo: **void \*malloc(size\_t tam)**

uso: puntero = (tipobase\*) malloc ( n<sup>o</sup>\_elementos \*sizeof(tipobase))

Ejemplo 2: reserva memoria 6 posiciones para caracteres

```
→ char *pc;  
pc = (char *) malloc(6*sizeof(char))
```

Comprueba que la reserva de espacio se ha realizado correctamente

```
if (pc==NULL)  
    fprintf(stderr, "Error de asignación de memoria");  
else ....
```

pc → @ s r z q &



# Asignación Dinámica de Memoria

- **Calloc** Asigna memoria e **inicializa** todas las posiciones reservadas para que no contengan basura

prototipo: **void \*calloc(size\_t nelem, size\_t tam)**

uso: puntero = (tipobase\*) calloc ( nº\_elementos, sizeof(tipobase))

`int *pi;`

`pi = (int *) calloc(3, sizeof(int));`

pi →

# Asignación Dinámica de Memoria

**Realloc:** permite reasignar zonas de memoria previamente reservadas con malloc o calloc. Libera la región de memoria ocupada por 'puntero' y le asigna otra nueva, en la que copia todo lo que había en la región anterior.

```
void * realloc( void *p, size_t tam)
```

```
int *pi;
```

```
pi = (int *) calloc(3, sizeof(int));
```

pi →

```
....
```

```
pi = (int *) realloc(pi,5*sizeof(int));
```

pi →

& %

# Asignación Dinámica de Memoria

**free:** libera memoria asignada dinámicamente

**void \*free(void \*p)**

free(pi);

free(pc);

# Ejemplo 2

## 1. Crear funciones para:

- ▢ Leer datos de tipo entero desde la entrada estándar (teclado) y almacenarlos en un vector
- ▢ Imprimir el contenido de un vector en pantalla
- ▢ Calcular el cuadrado de los elementos de un vector

# Ejemplo 2

## **Función para introducir los valores del vector desde teclado**

```
void leer_datos (int *v, int n)
{
    int i;
    printf("Introduzca los elementos del vector:\n");
    for(i=0;i<n;i++)
    {
        printf("elemento [ %d] = \n ", i+1);
        scanf("%d", &v[i]);
    }
}
```

# Ejemplo 2

## Función para mostrar en pantalla los valores del vector

```
void imprimir (int *v, int n)
{
    int i;

    printf("Los valores de los elementos del vector son: \n");
    for(i=0;i<n;i++)
        printf("v[ %d] = %d  ", v[i+1]);
}
```

# Ejemplo 2

## **Función para elevar al cuadrado los valores del vector**

```
void vector_cuadrado (int *v, int n)
{
    int i;

    for(i=0;i<n;i++)
        v[i] = cuadrado(v[i]);
}
```

# Ejemplo 2

1. Reservar memoria para un vector de enteros, comprobando que no hay errores en la operación
2. Introducir datos en el vector desde el teclado llamando a la función creada anteriormente
3. Elevar al cuadrado cada elemento del vector
4. Comprobar que se han almacenado los datos introducidos (imprimiendo el contenido del vector)
5. Aumentar el tamaño del vector en X posiciones (el valor X se pide por teclado)
6. Mostrar el contenido del vector ampliado.
7. Liberar la memoria asignada y acaba el programa



# Ejemplo 2

```
int main()
{
    int *v, nelem, n;

    do{
        printf("\nIntroduzca la dimensión del vector: \n");
        scanf("%d",&nelem);
    }while (nelem<0);

    v= (int *) calloc (nelem, sizeof(int)); /* Reserva memoria con calloc */

    if ( v==NULL)
    {
        fprintf(stderr,"Error de asignación de memoria");
        exit(1);
    }
```

# Ejemplo 2

```
leer_datos(v, nelem); /* Introduce datos desde el teclado en el vector */
vector_cuadrado(v, nelem);
imprimir(v, nelem); /* Muestra su contenido */

printf("Introduzca el número de elementos que desea ampliar el vector: \n");
scanf("%d", &n);
v=(int *) realloc(v, (nelem +n)*sizeof(int)); /* Aumenta el tamaño del vector */

if (v==NULL)
{
    fprintf(stderr,"Error de asignación de memoria");
    exit(1);
}
imprimir(v, nelem+n); /* Muestra el contenido del vector ampliado */

free(v); /* Libera el espacio reservado para ese vector */
}
```

# GESTIÓN DE ARCHIVOS DE TEXTO

- Los archivos permiten mantener los datos o la información de forma permanente en un periférico auxiliar de almacenamiento (discos, pendrives, etc.)
- Los ficheros de texto constan de una secuencia de caracteres ASCII que finaliza con el **carácter de fin de fichero** (representado por la macro **EOF**) y se organizan en líneas terminadas por un carácter de **fin de línea** ('\n').

# GESTIÓN DE ARCHIVOS DE TEXTO

- **#include <stdio.h>.**
- El fichero de cabecera `stdio.h` contiene las definiciones de constantes y tipos y las declaraciones de las funciones necesarias para manejar ficheros.
- Pasos básicos:
  1. Declaración de una variable de tipo fichero
  2. Apertura del fichero
  3. Lectura / escritura de información
  4. Cierre del fichero

# 1. Declaración

1. Declaración de una variable para almacenar la información del periférico a la memoria principal:

□ **FILE** \*var\_fichero;

/\* var\_fichero es una variable de tipo fichero \*/

## 2. Apertura

**FILE \*fopen** (const char \*nombre\_fichero, const char \*modo);

**var\_fichero=fopen**("miarchivo.txt","r");

- **nombre fichero** : puntero a una cadena de caracteres que contiene el nombre del fichero (puede incluir el nombre del directorio)
- **modo**: puntero a una cadena de caracteres que indica cómo se debe abrir el fichero de texto:
  - r    Abre un fichero existente para lectura
  - w    Crea un nuevo fichero para escritura (sobrescribe si existe)
  - a    Abre un fichero existente para añadir (crea si no existe)
  - r+   Abre un fichero existente para lectura/escritura
  - w+   Crea un fichero nuevo para lectura/escritura (sobrescribe si existe)
  - a+   Abre para añadir o crea un fichero para lectura/escritura

# 3. Lectura de información de un fichero

**Lectura de un carácter** en la posición actual y avanza a la siguiente posición o devuelve EOF si es final de fichero.

- **int fgetc (FILE \*f);**

- **int getc (FILE \*f);**

char c;

c=fgetc(var\_fich);    c=getc(var\_fich);

**Lectura de cadenas de caracteres** , los copia hasta llegar a un carácter de fin de línea ('\n'), un EOF o hasta leer tam-1 caracteres. Después pone un carácter nulo ('\0') al final de la cadena. También devuelve un puntero a la cadena leída.

- **char \*fgets (char \*s, int tam, FILE \*f);**

Funciona exactamente igual que scanf() excepto que lee los datos del fichero f en lugar de hacerlo de stdin (entrada estándar, normalmente el teclado).

- **int fscanf (FILE \*f, const char \*formato, ...);**

# 3. Escritura de información en un fichero

**Escritura de un carácter** en la posición actual y avanza a la siguiente posición.

- ▮ **int fputc (int c, FILE \*f);**
- ▮ **int putc (int c, FILE \*f); //cuidado suelen ser macros**

**Escritura de cadenas de caracteres** , copia la cadena s

- ▮ **char \*fputs (char \*s, FILE \*f);**

Funciona exactamente igual que printf() excepto que escribe los datos del fichero f en lugar de hacerlo de stdout (entrada estándar, normalmente la pantalla).

- ▮ **int fprintf (FILE \*f, const char \*formato, ...);**



## 4. Cierre

---

**int fclose(FILE \*f)**

fclose(var\_fichero);

```
#include <stdio.h>
#include <stdlib.h>
#define TAM 10
int main() {
```

```
/* Lee el contenido de un fichero de texto
carácter a carácter y lo imprime en pantalla */
```

```
    FILE *fichero;
    char c, s[TAM];
    //Prueba leyendo carácter a carácter
    fichero=fopen("mensajes.txt","r");
    if (fichero==NULL)
        printf("\n mensaje de error por no haber podido abrir el archivo con fgetc o
        getc\n");
    else{
        //c=fgetc(fichero);
        c=getc(fichero);
        while (c!=EOF){
            printf("%c",c);
            //c=fgetc(fichero);
            c=getc(fichero); }
        fclose(fichero);
    }
}
```

```
/* Lee el contenido de un fichero de texto  
MAX_P caracteres simultáneamente  
y lo imprime en pantalla */
```

```
/* Prueba leyendo cadenas */  
printf("Prueba con la instrucción fgets\n");  
fichero=fopen("mensajes.txt","r");  
if (fichero==NULL)  
printf("\n mensaje de error por no haber podido abrir el archivo  
con fgets \n");  
else{  
    while (fgets(s,TAM,fichero)!=NULL){  
        printf("%s",s);  
        system("pause");  
    }  
    fclose(fichero);  
}
```

# Ejercicio

- En el guión de prácticas aparece una descripción del tratamiento de ficheros de texto.
- Escribe un programa que lea el texto contenido en un archivo utilizando asignación dinámica de memoria.

# Asignación dinámica

```
#define MAX_P 25 // constante que establece el nº máximo de  
caracteres a leer
```

```
...
```

```
FILE * fichero;
```

```
int i;
```

```
char *txt, *aux, ;
```

```
fichero = fopen("prueba.txt", "r");
```

```
if (fichero != NULL)
```

```
{ // 1. Reserva MEMORIA
```

```
// 2. Lee primera cadena desde el fichero
```

```
i=1;
```

```
while(!feof(fichero)){ // feof se ha de usar después de leer algo del fichero
```

```
if (i>1)
```

```
    strcat(txt,aux); // 3. Concatena el nuevo texto al ya leído anteriormente
```

```
    i++;
```

```
// 4. Lee otra cadena desde el fichero
```

```
//5. Aumenta el tamaño
```

```
}
```

# Asignación dinámica

```
#define MAX_P 25 // constante que establece el nº máximo de  
caracteres a leer
```

```
...
```

```
FILE * fichero;
```

```
int i;
```

```
char *txt, *aux;
```

```
fichero = fopen("prueba.txt", "r");
```

```
if (fichero != NULL)
```

```
{
```

```
txt=(char *) malloc(MAX_P); // 1. Reserva MEMORIA
```

```
aux=(char *) malloc(MAX_P);
```

```
fgets(txt,MAX_P,fichero); // 2. Lee primera cadena desde el fichero
```

```
i=1;
```

```
while(!feof(fichero)){
```

```
    if (i>1)
```

```
        strcat(txt,aux); // 3. Concatena el nuevo texto al ya leído anteriormente
```

```
    i++;
```

```
    fgets(aux,MAX_P,fichero); // 4. Lee otra cadena desde el fichero
```

```
    txt=(char *) realloc(txt, i*MAX_P); //5. Aumenta el tamaño
```

# DISEÑO MODULAR


- **Módulo:** Componente del algoritmo destinado a resolver un subproblema del problema total planteado.
- En C el programa completo constituirá un proyecto y los módulos corresponderán a Ficheros que contienen parte de un programa (trabajo por proyectos)
  - ▢ Programa completo -> Proyecto .dev
  - ▢ Módulos del programa-> Ficheros .c y .h
- Cada módulo puede contener definiciones de tipos, constantes, variables y funciones.
- Puede haber elementos públicos o/y privados (secciones de importación y exportación).

# DISEÑO MODULAR

- **Sección de importación** ↔ **directivas #include** : conjunto de elementos utilizados en un módulo y declarados o definidos en otros módulos.
- **Sección de exportación** ↔ **NombreModulo.h** conjunto de elementos que un módulo hace públicos para ser utilizados en otros módulos.
  - **NombreModulo.h** incluye las **declaraciones de todos los elementos públicos** o exportables.
  - En C no es posible exportar constantes y tipos de datos ocultando su implementación. Por tanto, debemos incluir también en este fichero las **definiciones de constantes y tipos** que se quieran hacer **públicos**.
- **Sección de Implementación** ↔ **NombreModulo.c**
  - **definición o implementación de todas las funciones, tanto públicas como privadas.**
    - **Static nombreFunción** para hacer que una función sea privada, o sea, de uso exclusivo dentro del módulo en que está definida
    - Por defecto son públicas



# DISEÑO MODULAR

- Sección de Implementación  **NombreModulo.c**
  - definición o implementación de todas las funciones, tanto públicas como privadas.
- Es necesario añadir la directiva **#include** “**NombreModulo.h**” al principio del fichero de código **NombreModulo.c** porque:
  - Constantes y tipos públicos se definen en el fichero de cabecera (.h)
  - Tienen que ser utilizados en el fichero de código (.c)

# Directiva `#include`

- **`#include`** : Inclusión de ficheros cabecera

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

**`#include "funciones.h"`**

# Directivas: operador **defined**

- **#if defined**  $\equiv$  **#ifdef**
- **#if !defined**  $\equiv$  **#ifndef**
- Lo usaremos para programar los ficheros de cabecera y evitar que se dupliquen las inclusiones

```
/*Este es un fichero fichero.h*  
#ifndef_FICHERO_H_  
#define_FICHERO_H_  
    /* Aqui el código */  
#endif
```

# Estructura del programa del Anagrama

- **Main.c** : lee texto de un archivo y llama a la función que cuenta los anagramas.
- **Permutaciones.c**: lee palabras y cuenta el número de permutaciones de un texto.
- **Texto.c**: extrae las palabras de un texto.
- **Palabras.c**: determina cuándo una palabra es permutación de otra.

# Palabras.h

```
#define MAX_P 20
```

```
#define FIN_PAL '\0'
```

```
typedef enum {falso , verdadero} logico;
```

```
typedef char palabra[MAX_P+1];
```

```
logico palabra_es_permut(palabra pal1, palabra pal2);
```

# Palabras.c

```
#include <string.h>
#include <stdio.h>
#include "palabras.h"
//FUNCIONES PÚBLICAS:
logico palabra_es_permut(palabra pal1, palabra pal2)
{
    logico permut;
    .....
    return permut;
}
//Funciones NO exportables
static int longitud(palabra pal)
{ .....}
static int pos_caracter(palabra pal, char c)
{ .....}
static void borrar(palabra pal, int pos)
{.....}
```

# texto.h

//Fichero de cabecera texto

```
#ifndef _LOGICO_
#define _LOGICO_
```

```
enum logico{falso,verdad};
```

```
#endif
```

```
#define MAX_T 300
```

```
/* typedef char texto[MAX_T+1];
*/
```

```
typedef char *texto;
```

```
void leer_fichero(texto txt);
```

```
void leer_primera_palabra(texto txt, int *cursor, palabra pal);
```

```
void leer_siguiete_palabra(texto txt, int *cursor, palabra pal);
```

```
logico fin_de_texto(texto txt, int cursor);
```

# texto.c

```
#include <string.h>
#include <stdio.h>
#include "palabras.h"
#include "texto.h"

// implementación de las funciones públicas

void leer_primera_palabra(texto txt, int *cursor,
    palabra pal)
{.....}

void leer_siguiente_palabra(texto txt, int *cursor,
    palabra pal)
{.....}

logico fin_de_texto(texto txt, int cursor)
{.....}
```



# Permutaciones.h



```
#ifndef _PERMUTACIONES_  
#define _PERMUTACIONES_  
  
int permut_pal(texto txt);  
  
#endif
```

# Permutaciones.c

```
#include <stdio.h>
#include "palabras.h"
#include "texto.h"
#include "permutaciones.h"
```

```
int permut_pal(texto txt)
{
    .....
}
```

# Anagrama.c

```
#include <stdio.h>
#include <stdlib.h>
#include "Palabras.h"
#include "Texto.h"
#include "Permutaciones.h"
int main(){
.....
}
```

