Archivos

Índice

- Tipos de archivos.
- Apertura de archivos.
- Cierre de archivos.
- Lectura de archivos.
- Escritura de archivos.
- Acceso directo en archivos binarios.

Introducción y tipos de archivos

- Los archivos de datos permiten almacenar información de manera permanente y acceder y modificar la misma cuando sea necesario.
- C dispone de un conjunto amplio de funciones de biblioteca para crear y procear archivos de datos.
- Existen dos tipos de archivos de datos: *secuenciales* y *orientados al sistema*.
- Los archivos de datos secuenciales se dividen en dos categorías: archivos de *texto* y archivos *binarios*.

Introducción y tipos de archivos

- Los archivos de texto utilizan caracteres numéricos para representar números y letras para representar texto. Son legibles y se pueden ver y modificar con un editor de texto (p.e. bloc de notas). Tienen sufijo .txt.
- Los archivos binarios se escriben copiando el contenidos de una zona de memoria al disco. No son legibles si se intenta abrir con un editor de texto. Normalmente se usa como sufijo de estos ficheros .dat. Los ficheros binarios ocupan menos memoria que sus equivalentes en texto.

Apertura de archivos: la función fopen

- Antes de usar un archivo en disco es necesario indicarle al SO que lo localice y que reserve un área de memoria para trabajar con él. Esto se realiza con el la función **fopen** (stdio.h) cuyo prototipo es:

FILE *fopen (char *nombre_archivo, char *modo);

nombre_archivo es una cadena de caracteres que contiene el nombre del archivo.

modo es una cadena de caracteres que indica el tipo de operación a realizar con el archivo.

- Devuelve un “puntero a archivo”, que es un puntero a una estructura de datos llamada FILE. En caso de error devuelve NULL.

función fopen: modos de operación

Modo	Descripción
“r”	Abrir un fichero texto para lectura. El archivo debe existir. Se posiciona al principio del archivo.
“r+”	Abrir un fichero texto para lectura y escritura. El archivo debe existir. Se posiciona al principio del archivo.
“w”	Abrir un fichero texto para escritura. Si el archivo existe, borra su contenido y si no existe crea uno nuevo.
“w+”	Abrir un fichero texto para escritura y lectura. Si el archivo existe, borra su contenido y si no existe crea uno nuevo.
“a”	Abrir un fichero texto para añadir. Si el archivo no existe lo crea, sino borra su contenido. Se posiciona al final del archivo.
“xb”	se añade b a cualquiera de los modos anteriores (x) si el archivo es **binario** en lugar de texto.

Cierre de archivos: la función fclose

- Después de usar un archivo en disco hay que cerrarlo. Con esta operación se “desconecta” el archivo del programa y se libera el puntero al archivo.
- Para ello se usa la función **fclose** cuyo prototipo es:

int fclose (FILE *puntero_al_archivo);

puntero_al_archivo es el puntero a archivo devuelto por la función fopen al abrir el archivo que se desea cerrar ahora.

- Devuelve cero si el archivo se cerró con éxito o -1 si ocurrió algún error al cerrarlo.

Lectura en archivos texto: función fscanf

- Para leer de archivos de texto se usa la función **fscanf** cuyo funcionamiento es idéntico al de *scanf*. Su prototipo es:

int fscanf (FILE *punt_archivo, const char *cadena_formato, ….);

- Devuelve el número de variables que se han leído correctamente. Si llega al final final del archivo, el valor devuelto es EOF.

Escritura en archivos texto: función fprintf

- Para escribir en archivos de texto se usa la función **fprintf** cuyo funcionamiento es idéntico al de *printf*. Su prototipo es:

int fprintf (FILE *punt_archivo, const char *cadena_formato, ….);

- Devuelve el número de variables que se han escrito correctamente.

Otras funciones de entrada y salida para archivos de texto

- Hay varias funciones que lee o escriben caracteres o cadenas de caracteres de forma eficiente cuyos prototipos son:

int fgetc (FILE *punt_archivo);

lee el siguiente carácter desde el archivo como un int. Si se llega al final del fichero u ocurre un error devuelve EOF.

int fputc (int carácter, FILE *punt_archivo);

escribe el carácter que se le pasa como argumento en el archivo. El valor devuelto es el carácter escrito o EOF si ocurre un error.

char *fgets (char *cadena, int tam_cad, FILE *punt_archivo);

lee una cadena de caracteres del archivo y lo almacena en la cadena de caracteres cadena. La lectura se acaba cuando se encuentra '\n', EOF o se han leído tam_cad-1 caracteres.

int fputs (const char *cadena, FILE *punt_archivo);

escribe la cadena que se le pasa como argumento en el archivo. El valor devuelto es un número positivo o EOF si ocurre un error.

Lectura/Escritura en archivos binarios

- Los archivos de **texto** sólo contienen caraceters imprimibles (letras, números y signos de puntuación) y se pueden abrir y modificar con un editor de texto (p.e. bloc de notas).
- Los archivos **binarios** almacenen la información como una copia exacta de la memoria del ordenador.
- Los archivos binarios no son fácilmente portables y hay que tener en cuenta diferentes aspectos para adaptar programas que funcionan en varias plataformas.

Lectura en archivos binarios

- Para leer estructuras de datos de un archivo binario se requiere primero abrir el fichero en modo binario y usar fread cuyo prototipo es:

size_t fread (void *estructura, size_t tamano, size_t numero, FILE *archivo);

- Donde size_t es equivalente a **unsigned long int** (número) y **void *** es un puntero genérico válido para cualquier tipo de estructura.
- Devuelve el número de estructuras leídas que será igual al valor solicitado (numero) salvo error o se llegue al final del archivo.

Escritura en archivos binarios

- Para escribir estructuras de datos en un archivo binario se usa fwrite cuyo prototipo es:

size_t fwrite (void *pestructura, size_t tamano, size_t numero, FILE *archivo);

- Donde size_t es equivalente a **unsigned long int** (número) y **void *** es un puntero genérico válido para cualquier tipo de estructura.
- Devuelve el número de estructuras escritas que será igual al valor solicitado (numero) salvo error.

Acceso directo en archivos binarios

- En los archivos binarios de estructuras cada registro ocupa un espacio constante, que es el tamaño de la estructura.
- Esto permite avanzar o retroceder en el archivo para ir a leer un registro concreto, es decir, se permite el acceso directo a los datos.
- La función para moverse en un fichero a una posición concreta es **fseek** que tiene el siguiente prototipo:

size_t fseek(FILE *archivo, long desplazamiento, int origen);

- Donde origen puede ser: SEEK_SET (principio del archivo), SEEK_CUR (posición actual), SEEK_END (final del archivo). desplazamiento mueve el índice de acceso del archivo el número de bytes indicado (puede ser negativo).

Funciones generales feof y rewind

- Hay un par de funciones que pueden resultar muy útiles en el desarrollo de funciones que manejan ficheros.

int feof(FILE *fp);

Devuelve distinto de cero si estamos al final del fichero. En caso contrario, devuelve cero.

void rewind(FILE *fp);

Vuelve al principio del archivo. Equivale a fseek(fp,0,SEEK_SET);

Preprocesador y programas multifichero

Índice

- Directrices del preprocesador.
- Constantes y Macros.
- Inclusión de ficheros.
- Sentencias Condicionales.

Directivas del preprocesador

- Son expandidas en la fase de preprocesado:
  - `#define` : Define una nueva constante o macro del preprocesador.
  - `#include` : Incluye el contenido de otro fichero.
  - `#ifdef #ifndef` : Preprocesamiento condicionado.
  - `#endif` : Fin de bloque condicional.
  - `#error` : Muestra un mensaje de error

Constantes y Macros

- Permite asociar valores constantes a ciertos identificadores expandidos en fase de preprocesamiento:

```
#define variable valor
```

- Define funciones que son expandidas en fase de preprocesamiento:

```
#define macro(args,...) función
```

Constantes y Macros. Ejemplos

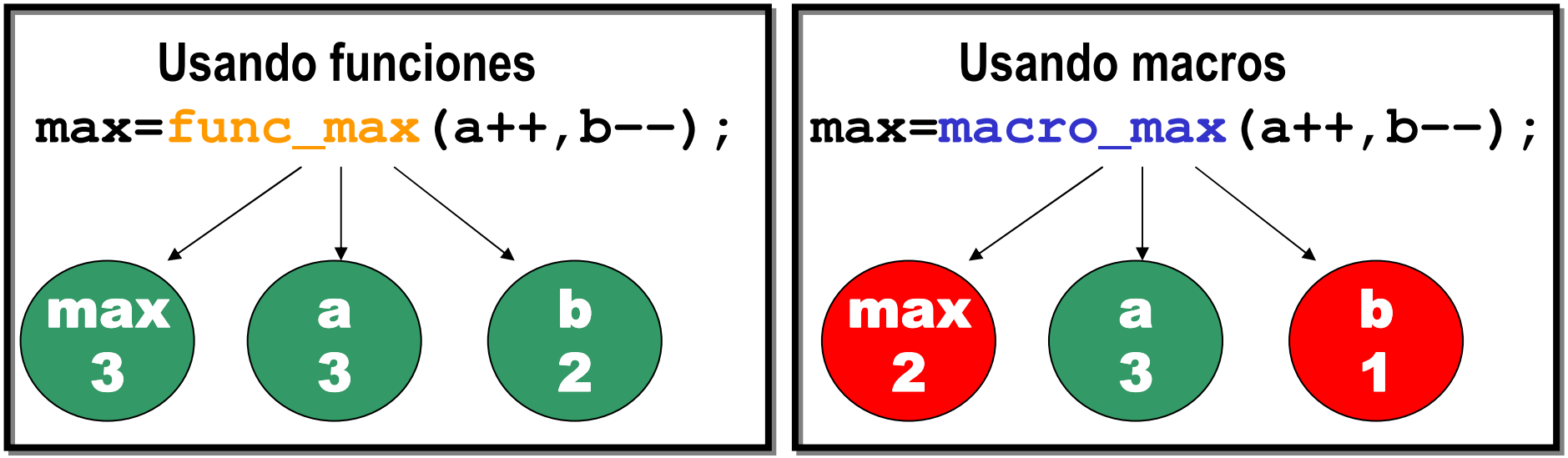
```
#define PI          3.14
#define NUM_ELEM    5
#define AREA(rad)   PI*rad*rad
#define MAX(a,b)    (a>b ? a : b)
int main()
{
  int    i;
  float vec[NUM_ELEM];
  for(i=0;i<NUM_ELEM;i++)
     vec[i]=MAX((float)i*5.2,AREA(i));
}
```

Constantes y Macros. Después del preprocesamiento

```
int main()
{
  int   i;
  float vec[5];
  for(i=0;i<5;i++)
     vec[i]=((float)i*5.2>3.14*i*i ?
             (float)i*5.2 :
             3.14*i*i);
}
```

Macros vs Funciones

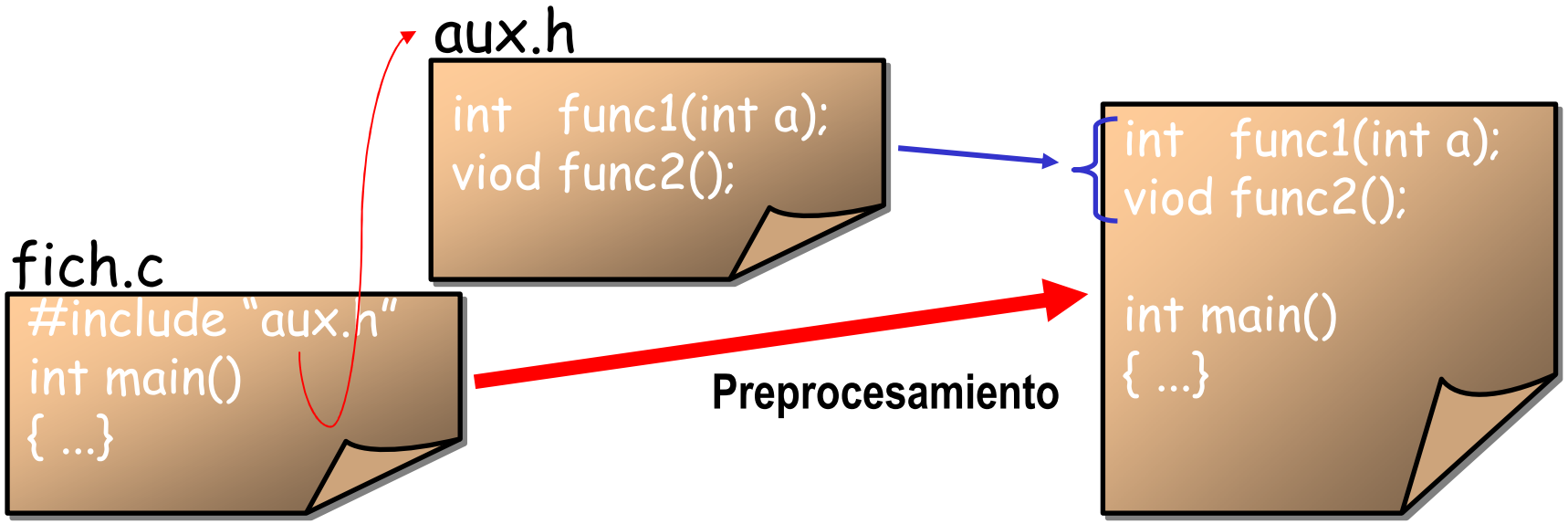
```
#define macro_max(a,b) (a>b ? a : b)
int func_max(int a, int b)
{ return (a>b ? a : b); }
int a=2,b=3,max;
```



Inclusión de Ficheros

- Los prototipos de las funciones usadas por varios ficheros fuente se suelen definir en fichero de cabecera.

```
#include <stdio.h>          Cabeceras del sistema
#include "mis_func.h"       Ficheros de cabecera locales
```



Inclusión de Ficheros

- La inclusión de ficheros esta sujeta a las siguientes recomendaciones:
  - Por lo general los ficheros de cabecera tienen como extensión .h
  - En los ficheros de cabecera no se incluyen implementación de funciones
  - Las variables en un fichero de cabecera son declaradas extern y se encuentran declaradas en algún otro fichero .c

Sentencias Condicionales

- Para incluir código cuya compilación es dependiente de ciertas opciones, se usan los bloques:

```
#ifdef variable
<bloque de sentencias>
...
#endif
#ifndef variable
<bloque de sentencias>
...
#endif
```

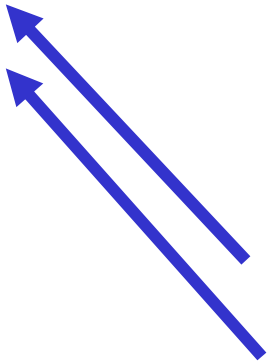
Ejemplo: Depuración

```
#define DEBUG
int main()
{
  int i,acc;
  for(i=0;i<10;i++)
     acc=i*i-1;
#ifdef DEBUG
  printf("Fin bucle acumulador: %d",acc);
#endif
  return 0;
}
```

Ejemplo: Fichero de cabecera

aux.h

```
#ifndef _AUX_H_
#define _AUX_H_
<definiciones>
#endif
```



Evita la redefinicion de funciones y variables

```
#include "aux.h"
#include "aux.h"
int main()
{
...
}
```

Compilación en modo comando

- Visual C se puede usar desde la línea de comandos:

```
> lc -c pp.c
  para compilar fuentes
> lc -o pp.exe pp.obj  pp1.obj
  para enlazar objetos
```