

Operaciones de direcciones

* Lee o modifica el valor apuntado por la expresión. Se corresponde con un puntero y el resultado es del tipo apuntado.

• Devuelve un puntero al objeto utilizado como operando, que debe ser un dato (variable dotada de una dirección de memoria). El resultado es un puntero de tipo idéntico al del operando. Permite acceder a un miembro de un dato agregado (unión, estructura).

→ Accede a un miembro de un dato agregado (unión, estructura) apuntado por el operando de la izquierda.

Operador ()

• Es el operador de llamada a funciones. Sirve para encerrar los argumentos de una función, efectuar conversiones explícitas de tipos, indicar en el seno de una declaración que un identificador corresponde a una función, resolver los conflictos de prioridad entre operadores.

Operador []

• Sirve para dimensionar los arrays y designar un elemento de un array.

Operador sizeof

Con frecuencia se necesita conocer el tamaño en bytes de un tipo de dato o variable. C proporciona el operador sizeof, que toma un argumento, bien un tipo de dato o bien el nombre de una variable (escalar, array, registro, etc.).

string.h

unsigned strlen(const char *cad);

char *strcpy(char *cad1, const char *cad2);

copia cad2 en cad1

char *strncpy(char *cad1, const char *cad2, unsigned n);

no modifica
el contenido
de cod1

copia maximo n caracteres
de cod2 en cod1.

char *strcat(char *cad1, const char *cad2);

concatena cod2 al final de cod1
devuelve cod1.

char *strncat(char *cad1, const char *cad2, unsigned n);

concatena maximo n caracteres
de cod2 al final de cod1
devuelve cod1.

int strcmp(const char *cad1, const char *cad2);

compara cod1 con cod2
devuelve 0 son iguales
 <0 si cod1 < cod2
 >0 si cod1 > cod2

int strncmp(const char *cad1, const char *cad2, unsigned n);

compara maximo n caracteres
de cod1 y cod2 devuelve
0 son iguales
 <0 si cod1 < cod2
 >0 si cod1 > cod2

char *strchr(const char *cad, int c);

Localiza la primera instancia del caracter c en cad.
Devuelve un puntero a c o NULL si no lo encuentra.

`char *strchr(const char *cad1, const char *cad2);`
Localiza la primera aparición de `cad2`
en la cadena `cad1`.
Devuelve la dirección de comienzo de
`cad2` en `cad1` o `NULL` si no lo encuentra.

`char *strrchr(const char *cad, int c);`
Localiza la última aparición del carácter
`c` en `cad`.
Devuelve un puntero a `c` o `NULL` si no
lo encuentra.

stdlib.h → Conversión de cadenas.

`double atof(const char *cad);`
Devuelve la cadena `cad` convertida a double.

`int atoi(const char *cad);`
Devuelve la cadena `cad` convertida a int.

`long atoll(const char *cad);`
Devuelve la cadena `cad` convertida a long.

`char *itoa(int n, char *cad, unsigned b);`
Convierte el entero `n` en una cadena de caracteres,
que la almacena en `cad` en base `b` (2-36).

ctype.h → Devuelve `cad`.

`int isdigit(int c);`
Devuelve 1 verdadero si `c` es un dígito y 0 en caso contrario.

`int isalpha(int c);`
Devuelve 1 verdadero si `c` es una letra y 0 en
caso contrario.

`int isalnum(int c)`

Devuelve verdadero si c es un dígito o una letra
y 0 en caso contrario.

`int islower(int c)`

Devuelve verdadero si c es una letra minúscula
y 0 en caso contrario.

`int isupper(int c)`

Devuelve verdadero si c es una letra mayúscula
y 0 en caso contrario.

`int ispace(int c)`

Devuelve verdadero si c es un carácter de espacio
(salto de línea '\n', espacio en blanco ' ', siguiente
línea '\f', retorno de carro '\r', tabulador horizontal
'\t' o tabulador vertical '\v')
y cero en caso contrario.

`int tolower(int c)`

Si c es una letra mayúscula la convierte a minúscula,
si no, la devuelve sin modificar.

`int toupper(int c)`

Si c es una letra minúscula la convierte a mayúscula,
si no, la devuelve sin modificar.

Funciones estándar de I/O stdio.h

`int getchar(void)`

Devuelve el siguiente carácter de la entrada estándar (Teclado normalmente). Lo recoge después de pulsar Intro.

Si hay error de lectura o se alcanza el final del fichero devuelve EOF.

`char *gets(char *cad)`

Lee caracteres de la entrada estándar hasta encontrar un salto de linea o el fin del fichero y los almacena en la cadena cad. Añade '\0' al final.

Devuelve cad si no hay error en la lectura y NULL en caso contrario.

`int putchar(int c)`

Imprime el carácter c en la salida estándar (pantalla normalmente).

`int puts(const char *cad)`

Imprime la cadena de caracteres cad en la salida estándar seguida de un salto de linea.

Si tiene éxito devuelve un '\n' y EOF en caso contrario.

Funciones
I/O de
consola
conio.h

`int getch(void)`

Devuelve el carácter leído de teclado sin esperar que se pulse Intro. No hace eco en la pantalla.

`int getche(void)`

Devuelve el carácter leído de teclado sin esperar a que se pulse Intro. Hace eco en la pantalla.

Funciones
estándar de
I/O
stdio.h

`int putchar(int c)`

Imprime el carácter c en la pantalla.

Devuelve el carácter impreso o EOF en caso de error.

Funciones de manejo de memoria stdlib.h

void *memcpy (void *cod1, const void *cod2, unsigned n)

Copia n bytes de cod2 en cod1.

Devuelve cod1.

No controla el resultado, podría haber sobrescrito.

void *memmove (void *cod1, const void *cod2, unsigned n)

Copia n bytes desde cod2 a cod1.

Devuelve cod1.

Usa una zona auxiliar para la copia.

void *memcmp (const void *cod1, const void *cod2, unsigned n)

Compara los n primeros caracteres de cod1 y cod2.

Devuelve

0 si son iguales

<0 si cod1 < cod2

>0 si cod1 > cod2

Práctica 8

i bytes ?
de memoria

Manejo de uno hasta entero de enteros

& Falta hacer

(fscanf) devuelve el numero
de elementos leidos

nº elementos → ... 3,2,1 → =fscanf
dato se lee ... dato incorrecto → 0 → =fscanf

Ficheros • Comprobar que existe fichero

antes de leer. Un fichero es una secuencia de bytes almacenada en el disco que puede manejarse de dos maneras:

Texto: Los datos se interpretan como caracteres organizados en líneas, con un carácter especial para fin de linea (\n).

Binario: Los datos se manejan como bytes en bruto, tal avol están almacenados en el disco.

Los programas acceden a los ficheros a través de un puntero de tipo FILE *, que es una estructura definida en la biblioteca estandar <stdio.h> y contiene información sobre el fichero y su buffer asociado.

Apertura de un fichero

El fichero se asocia al programa mediante la función fopen, que recibe el nombre del fichero y el modo de apertura (lectura, escritura, etc.)

FILE *f;

f = fopen("archivo.txt", "r");

"r" lectura

"w" escritura

"a" añadir al final (lo crea si no existe)

"rb"

"wb"

{ Para binario.

"ab"

Closure libera los recursos asociados al fichero y graba los datos pendientes en disco si el fichero estaba en modo escritura.

Lectura y escritura en ficheros

Se puede usar `fprintf`

$EOF = -1$

```
FILE f* = fopen("salida.txt", "w");
fprintf(f, "Hola mundo\n");
fclose(f);
```

Se puede usar `fgets` para leer líneas completas o `scanf` para leer datos formateados:

Por alguna razón en los dispositivos pone int.

```
FILE *f = fopen("entrada.txt", "r");
char linea[100];
while (fgets(linea, 100, f) != NULL) {
    printf("Leído: %s", linea);
}
fclose(f);
```

• `void rewind(FILE *f)`

• Coloca el puntero de archivo al principio del fichero asociado a f.

• No desasocia el fichero ni cambia su estado de apertura.

• No devuelve ningún valor.

- La función `fwrite` se usa para escribir estructuras completas o datos en bruto:

```
FILE *f = fopen("datos.dat", "wb");
int numeros[] = {1, 2, 3, 4};
fwrite(&numeros, sizeof(int), 4, f);
fclose(f);
```

`int remove(const char *)`

• Borra el fichero

• Devuelve 0 si la operación tiene éxito.

• Devuelve un valor diferente de 0 si hay error, creando el archivo no existente (-1).

- La función `fread` se usa para leer datos binarios del fichero:

```
FILE *f = fopen("datos.dat", "rb");
int buffer[4];
fread(buffer, sizeof(int), 4, f);
fclose(f);
```

Tema 8 Estructuras.

struct FEC {

Global

Si — También se puede definir fecha hoy `ago.`

Almohadones struct FEC fecha_hoy;

Con `typedef` se define un nuevo tipo de dato, lo que permite:

typedef FEC f

};

FEC fecha_hoy;

Esto se usa en estructuras dinámicas.

Acceso a los campos

vble. campo

tipo
estructura / no puntero

puntero → campo

↑
puntero
a una estructura

(*puntero).campo

} Equivalente

Se pueden asignar estructuras. Cuando asignas una estructura a otro, se copian campo a campo. Cada miembro de la estructura de destino recibe el valor correspondiente del miembro de la estructura de origen.

`struct libro novela,historia;`

...
`novela = historia;`

Comparación de estructuras. No puedes usar operadores de comparación (`==`, `>`, etc.) para comparar estructuras completas.

Si quieres comparar dos estructuras debes hacerlo campo por campo, utilizando las funciones apropiadas para cada tipo de dato (por ejemplo `strcmp()` para cadenas, y operadores estándar para tipos primitivos como `int` o `float`).

Paso a función

Por valor:

```
void modificarFecha(struct fecha f) {  
    f.dia = 1; // Esto no modifica la estructura original.  
}
```

Por referencia:

```
int main() {  
    struct fecha fechahoy = {19, 11, 2024};
```

```
    modificarFecha(&fechahoy);
```

```
    void modificarFecha(struct fecha *f) {
```

```
        (*f).dia = 1; // Esto si modifica la estructura original.  
    }
```

Las estructuras pueden ser devueltas por las funciones, devolviendo una copia que puede ser asignada como se necesite.

↑
~~struct EMP *scn-EMP();~~
~~struct EMP e;~~
~~return &e;~~ ← La solución es
No funciona
convertis esto en
un pointer.

Pro quequieres un pointer a un struct

```
struct EMP *scn-EMP()  
{  
    struct EMP e;  
    return &e;  
}
```

Problema
El free se hace
en el main.

`int fputc(int c, FILE *f);` → igual a `putc`.

• Escribe un único carácter `c` en el fichero `f`.

- Si tiene éxito devuelve el carácter escrito.
- Si falla devuelve EOF

`int fputs(const char *cad, FILE *f);`

• Escribe una cadena de caracteres (terminada en `\0`) en un fichero.

- Devuelve valor no negativo si tiene éxito, y si falla devuelve EOF.
- No escribe el carácter `\0` al fichero.

`int fgetc(FILE *f);` → igual a `getc`.

• Lee el siguiente carácter del fichero asociado a `f`.

- Devuelve el carácter leído como int.
- Si llega al final del fichero o hay un error, devuelve EOF.

`char * fgets(char *cad, int n, FILE *f);`

• Lee una linea del fichero y la almacena en `cad`.

- Lee como máximo $n-1$ caracteres, dejando espacio para `\0` al final.

• Detiene la lectura si encuentra un salto de linea o el final del fichero.

- Devuelve un puntero a `cad` si tiene éxito o NULL si ocurre un error o se llega al final del fichero sin leer nada.

Uso del buffer

Cuando se trabaja con ficheros en C:

- Los datos no se leen/escriben directamente del disco. En su lugar se utiliza un buffer.
- Operaciones como fwrite escriben primero al buffer; solo cuando este lleno, los datos se envian al disco.
- Se puede vaciar el buffer manualmente con fflush

Fin de fichero y detección de errores.

- feof(FILE *) Devuelve un valor distinto de cero si alcanzo el final del fichero.
- perror(FILE *) Comprueba si hubo errores en las operaciones de lectura/escritura.

returna se devuelve
un valor y se sigue
desde exit() donde se invoca.
un valor que es error
solo de todo. numeros != 0
break corta en la
estructura solo
de otra q continua

Argumentos en main.

- Permiten pasar parámetros al programa desde la linea de comandos.

tipo main (int argc, char * argv[]);

normalmente int (ejecución normal, !=0 algun error).

- argc: Número de argumentos (incluido el nombre del programa).
- argv: Arreglo de cadenas con los argumentos.

Manejo de bits

1 XOR

1 OR

& AND

~ NOT

Relleno {>> Desplazamiento dcha.
con ceros } << Desplazamiento izda.

Unión

La cantidad de memoria para la unión es igual a la anchura de la variable más grande.

La diferencia que tiene con las struct es que en la unión se comparte la memoria para las variables que contiene, por lo que es útil cuando las variables no se utilizan a la vez.

Para referirse a los miembros de la unión se utiliza el operador(.) o bien (\rightarrow) $((*)\cdot)$

Ejercicio examen

Leer de un fichero y escribir en otro siempre que se encuentre una palabra clave en dicha linea.