# Aprenda C en un par de horas

Sintaxis del lenguaje C para
implementar un programa imperativo

# Programa “Hola, muy buenas...”

```
/* Programa simple */
/* Los ficheros cabecera incluyen la declaración de
   funciones como stdio.h y definición de macros y ctes */
#include <stdio.h> /*Incluye el contenido de stdio.h*/
main( ) {
  printf("Hola, muy buenas...");
}
```

# Variables

```
#include <stdio.h>
main( ) {
  int entero;  /* entero con signo */
  char caracter; /* carácter ASCII */
  float real;  /* real simple precisión */
  entero = 2+2;
  caracter = 'a';
  real=6.023E23;
  printf("\nResultado: %d\t'%c' ",entero, caracter);
  printf("\treal %f", real);
}
```

# Constantes

```
#include <stdio.h>
#define MAX 50
main( ) {
  const int entero=3;
  const float PI=3.1415926;
  printf("\nResultado: %d, otro %d",entero, 66);
  printf("\treal %f", real);
}
```

# Constantes alfanuméricas

```
char l, a='b';    /* reserva dos espacios en memoria */
                  /* para guardar en cada uno un */
                  /* carácter alfanumérico */
l= a;             /* la var. l tiene almacenada la letra b */
l= 'a';           /* la var. l tiene almacenada la letra a */
```

# Dualidad carácter/valor ASCII

```
#include <stdio.h>
main( ) {
  char a='C',b='f', c='3';
  int x;
  x=a-'A';
  printf(“\nDistancia: %d”,x);
  printf(“\nValor numérico: %d”, c-'0'); /* 51-48 */
  a=a+('a'-'A'); /* +32 pasa a minúsculas*/
  b=b-32;   /* Pasa a mayúsculas */
  printf("\n%c \t%c", a,b); /* resultado */
}
```

# Operadores matemáticos

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

**`a=c/b;`** **Si son enteros, sólo da el cociente de la división. Si uno de ellos (por lo menos) es real, da la división con todos los decimales posibles**

**`a=c%b;`** **Sólo se puede usar con enteros, y da el resto de la división entera**

# Abreviaturas

**`a=a+1;` -> `a++;` o también `++a;` (Hay diferencia)**

**`b=b-1;` -> `b--;` o también `--b;` (Hay diferencia)**

```
b = b + c; -> b += c;
b = b - c; -> b -= c;
b = b * c; -> b *= c;
b = b / c; -> b /= c;
```

**¡¡Cuidado!!**

```
c=3;
```

**`b=c+1;` -> b tiene 4 y c tiene 3 (`d=c+1; b=d;`)**

**`b=c++;` -> b tiene 3 y c tiene 4 (`d=c; c=c+1; b=d;`)**

**`b=++c;` -> b tiene 4 y c tiene 4 (`c=c+1; d=c; b=d;`)**

# Entrada/salida

```
#include <stdio.h>
void main(){
  int num;
  char car, nombre[10]; /* Cadena de caracteres */
  printf("Introduce un numero entero");
  scanf("%d", &num);
  printf(" la variable \"car\": ");
  fflush(stdin); /* Vacía el búfer del teclado */
  scanf("%c", &car);
  fflush(stdin);
  printf("\nIntroduce un nombre");
  scanf("%s", nombre);
  printf("\n\nEl número es %d, \t y el ", num);
  printf("carácter %c.\n", car);
  printf("La cadena es %s", nombre);
}
```

# Entrada/salida

```
int i1, i2;
char s1[30], s2[30];
scanf("%d", &i1);
scanf("%d", &i2);
fgets(s1,30,stdin);
fgets(s2,30,stdin);
```

Supongamos que el usuario introduce “20\n30\npablo\n”, la secuencia de lectura sería la siguiente:

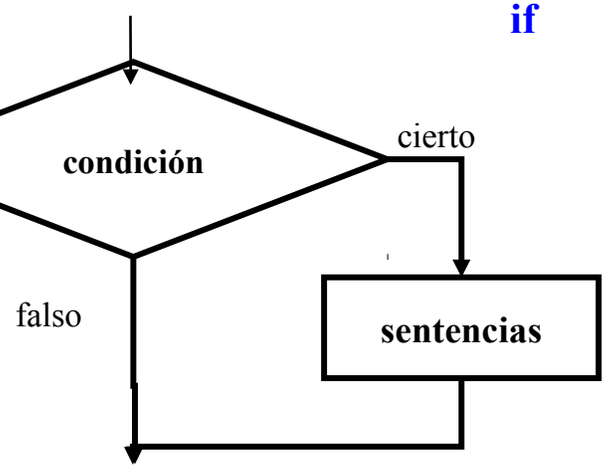
| Entrada | Buffer antes | Instrucción | Buffer después |
|---|---|---|---|
| 20\n | 20\n | scanf("%d", &i1); | \n |
| 30\n | \n30\n | scanf("%d", &i2); | \n |
| \n | \n | fgets(s1,30,stdin); | |
| pablo\n | pablo\n | fgets(s2,30,stdin); | |

# Entrada/salida

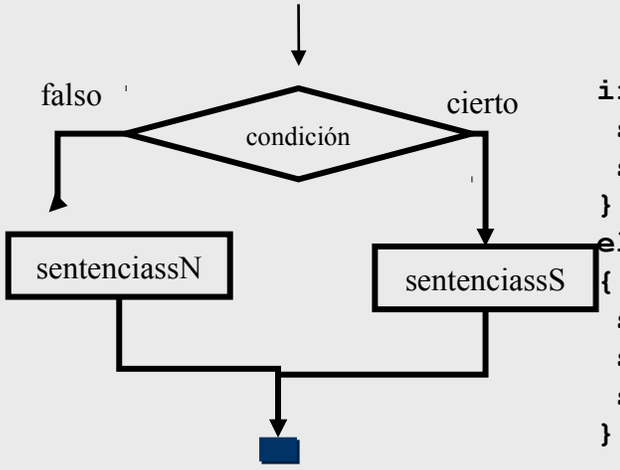
**El primer scanf tiene que leer del buffer hasta que encuentra un número (%d). En cuanto encuentra el 20 termina de leer, y deja en el buffer un '\n'. El segundo scanf, tras la entrada del usuario, se encuentra con \n30\n, y tiene que realizar la misma tarea que el anterior, leer un número. Se salta el primer '\n', y lee 30, dejando de nuevo un '\n' en el buffer. La función gets es más simple, y lo único que hace es leer todo lo que haya en el buffer hasta que encuentre un '\n', y lo copia en la variable correspondiente. Así, el primer gets se encuentra un \n, lo consume pero no copia nada en s1. El segundo gets se encuentra pablo\n, así que lee todo lo que hay en el buffer y lo guarda en s2. Para permitir que pablo se copie en s1, una posibilidad es incluir una llamada a getchar() antes de emplear gets para leer s1. Esta función lee un carácter del buffer, consumiendo así el '\n' que impedía rellenar s1 con pablo.**

# Instrucciones condicionales: `if`

```
if (condición)
{
 sentencia1;
 sentencia2;
 sentencia3;
}
```

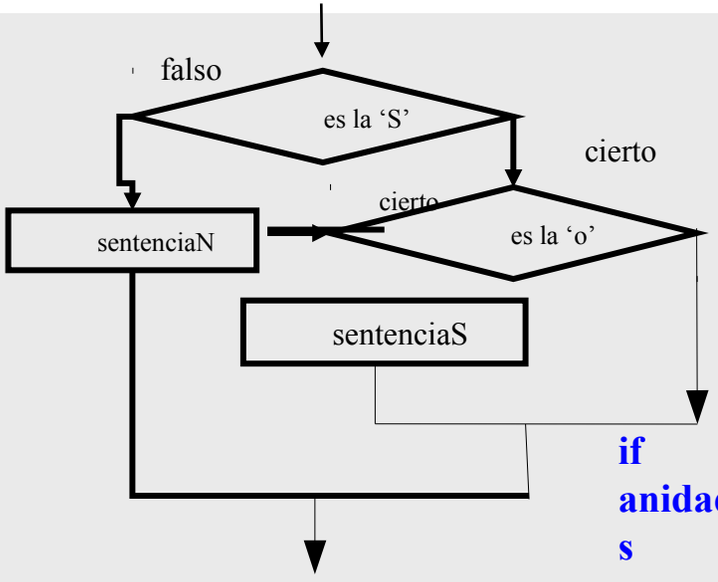



**if**




```
if(condición){
 sentenciaS1;
 sentenciaS2;
}
else
{
 sentenciaN1;
 sentenciaN2;
 sentenciaN3;
}
```

**if-else**




```
if(c1=='S'){
 if(c2=='o')
  sentenciaS;
}
else
{
 sentenciaN;
}
```

**if anidados**

# Operaciones condicionales

< menor que

<= menor o igual que

== igual (dos iguales)

!= distinto de

> mayor que

>= mayor o igual que

|| uno u otro, o los dos (*or* lógico)

&& uno y otro (*and* logico)

!( expresión) no es cierto, no ocurre ese algo (*not* lógico)

# if

```
/* if anidados adecuados para switch.C */
#include <stdio.h>
int nota;
void main(){
   printf("Dame tu nota ");
   scanf("%d", &nota);
   if(nota==0||nota==1||nota==2||  nota==3||nota==4)
   {
   printf("\nLo siento, has suspendido \n");
   printf("Si intentas otra, apruebas\n\n");
   }
   else if (nota==5 || nota==6)
   printf("\nUn aprobado \n");
   else if (nota==7 || nota==8)
   printf("\nUn notable, muy bien \n");
   else if (nota==9)
   printf("\nSobresaliente \n");
   else if (nota==10) printf("\nFelicidades, un 10 \n");
   else if (nota==11) {
   printf("\n Menos lobos... \n");
   printf("\n¿Qué nota es ésa? \n");
   } else printf("\n¿Qué nota es ésa? \n");
   getch(); /* para el programa hasta pulsar una tecla*/
}
```

# switch

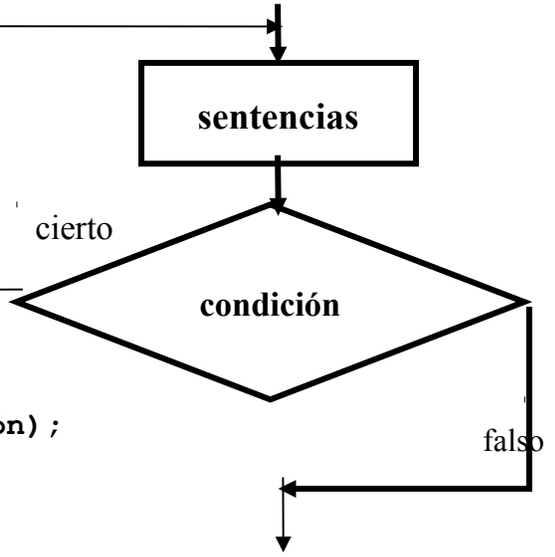
```
#include <stdio.h>
int nota;
void main(){
   printf("Dame tu nota "); scanf("%d", &nota);
   switch(nota){
    case 0: case 1: case 2: case 3:case 4:
    printf("\nLo siento, has suspendido \n");
    printf("Si intentas otra vez, apruebas\n\n");
    break;
    case 5:case 6: printf("\nUn aprobado \n"); break;
    case 7:
    case 8:
    printf("\nUn notable, muy bien \n");
    break;
    case 10:
    printf("\nFelicidades, un 10 \n");
    case 9: printf("\nSobresaliente \n"); break;
    case 11:
    printf("\n Menos lobos... \n");
    default:
    printf("\n¿Qué nota es ésa? \n");
   } /* fin switch */
   getch(); /* para el programa hasta pulsar tecla*/
}
```

# Instr. iterativas: ciclos

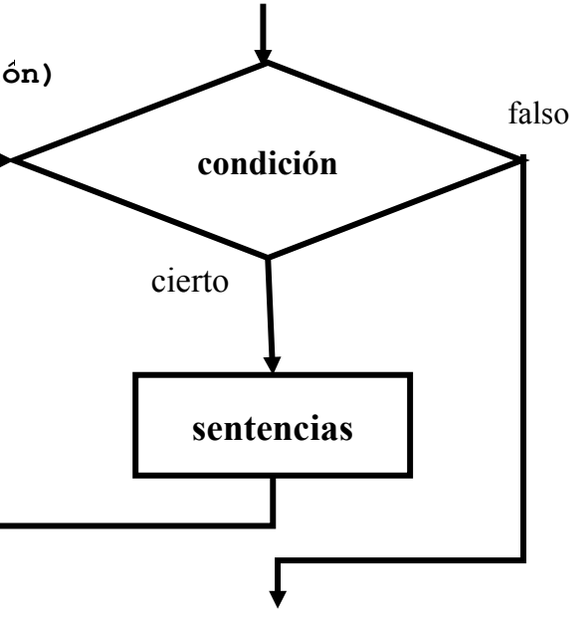
```
#include <stdio.h>
void main() {
   char sn;
   int n=10;
   do{
    printf("\n¿seguimos?(S/N)");
    fflush(stdin);
    scanf("%c", &sn);
   } while (sn=='s' || sn=='S');

   while (n>0) {
    printf("\t%d,",n);
    n--;
   }
   printf("\t%d.",n);
}
```

**do while**

```
do
{
 sentencia1;
 sentencia2;
 sentencia3;
}
while(condición);
```




```
while (condición)
{
 sentencia1;
 sentencia2;
 sentencia3;
}
```




**while**

# Instr. iterativas: ciclos

```
for (cont=1; cont<=10; cont=cont+1)
   printf("\n¡Hola!");
```

**Para imprimir los múltiplos de 7 menores de 500:**

```
for (cont=7; cont<500; cont=cont+7)
   printf("\n%d",cont);
```

**Y si queremos una cuenta atrás:**

```
for (cont=10; cont>0; cont=cont-1)
   printf("\n%d",cont);
```

**O también:**

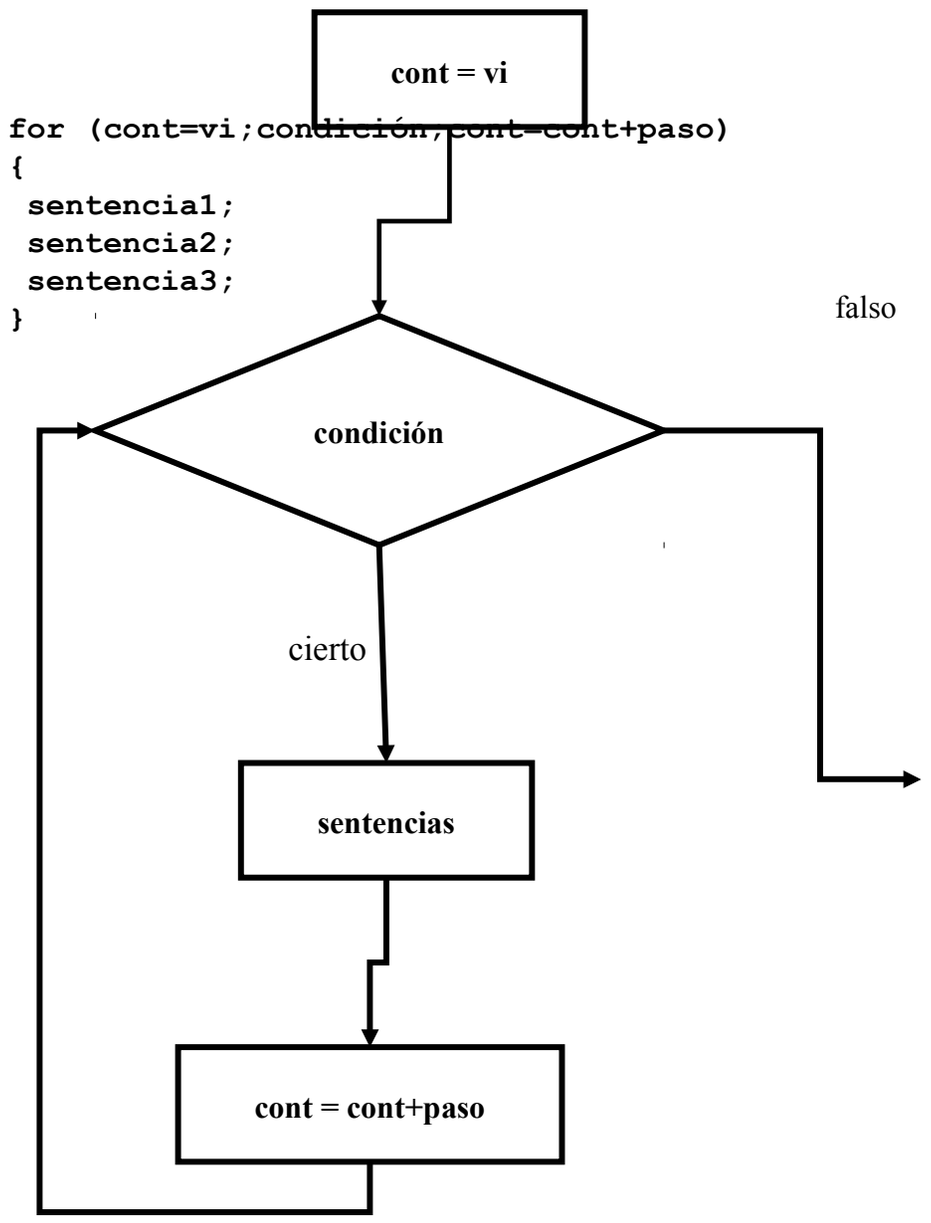
```
for (cont=10; cont>0; cont--)
   printf("\n%d",cont);
```

**O también:**

```
cont=10;
while (cont>0)
   printf("\n%d",cont--);
```

**El factorial en una línea:**

```
for(i=1,f=1; i<=x; f*=i, i++);
```




for

# Tablas, arrays, vectores, matrices, arreglos

| posición | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| valor | 5 | 786 | 783 | 784 | 785 | 456 | 88 | 458 | 98 | 810 |

```
#include <stdio.h>
#define N 10
main( ) {
  int i, v[N], aux;
  for (i=0; i<N; i++) {
   printf("\nDame el %d° valor: ",i+1);
   scanf("%d", &v[i]);
  }
  aux=v[0];
  for (i=0; i<N-1; i++) v[i]=v[i+1];
  v[N-1]=aux;
  printf("\nDesplazada a la izquierda: \n");
  for (i=0; i<N; i++) printf("\t%d ", v[i]);
}
```

# Matrices multidimensionales

```
float t[2][5]= {{1,2,3,4,5},{6,7,8,9,0}};
...
printf ("%d",t[1][3]);
```
**imprimirá 9**

```
printf ("%d",t[3][1]);
```
**no es correcto**

| | c0 | c1 | c2 | c3 | c4 |
|---|---|---|---|---|---|
| fila 0 | 1 | 2 | 3 | 4 | 5 |
| fila 1 | 6 | 7 | 8 | 9 | 0 |

# Cadenas de caracteres

| pos | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | | 99 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| valor | ‘B’ | ‘l’ | ‘a’ | ‘d’ | ‘e’ | ‘ ’ | ‘R’ | ‘u’ | ‘n’ | ‘n’ | ‘e’ | ‘r’ | ‘\0’ | ... | |

```
#include <stdio.h>
#include <string.h>
#define N 255
void main() {
  char cad[N],cad2[N],cop[]="Cadena copiada"; /*cop 15 pos.*/
  int i=0, longitud;
  printf("\n\nDame una cadena que la copio en otra: \n");
  fgets(cad,255,stdin); /*lee hasta EOL o EOF o 255 incluye
  '\0'*/
  longitud=strlen(cad);/*N° caracteres sin contar el /0*/
  while (cad[i]!='\0'){
  cad2[i]=cad[i]; /* Copiamos si cad[i]!='\0' */
  printf("\ncad[%d]=%c",i,cad2[i]);
  i=i+1;
  } /* Equivalente a strcpy(cad2,cad); */
  cad2[i]='\0'; /* Añadimos el fin de cadena*/
  printf("\n\n%s: \n%s", cop, cad2);
}
```

En la condición del `while` podríamos haber puesto `(i<longitud)` y el programa sería equivalente. Si ponemos `(i<=longitud)` también copiaría el carácter de fin de cadena, lo mismo que si usáramos un `do while`.

# Subprogramas

```
#include <stdio.h>
long fact(int n);
main( ) {
  printf("\nEl factorial de 14 es %ld", fact(14));
}
long fact(int n) { /* iterativo */
  long r=1;
  int i;
  for (i=1; i<=n; i++)
   r*= (long) i;
  return r;
}
```

La *función* devuelve un `long` a partir del dato de entrada `int`

# Función recursiva

```
#include <stdio.h>
long fact(int n) { /* recursivo */
  if (n==1)
   return 1;
  else
   return (long)n*fact(n-1);
}
main( ) {
  printf("\nEl factorial de 14 es %ld", fact(14));
}
```

# if ( es_negativo(x) )

```
#include <stdio.h>
#define VERD 1
#define FALSO 0
int es_negativo(int n);
void main( ) {
int num;scanf(“%d”,&num);
if (es_negativo(num)) {printf("%d es negativo",num);}
}
int es_negativo(int n) {
  if (n<0)
  return VERD;
  else
  return FALSO;
}
```

# if (es_primo(x))

```
int es_primo (int n) {
/* devolvemos 0 (falso) o 1 (verdadero es cualquier n ≠ 0) */
  int i;
  if (n<=1) return 0;
  for(i=1; i<=n; i=i+1)
   if(n%i==0)     /* i es divisor de n */
      cont=cont+1;  /* tenemos otro divisor de n */
  if (cont==2)
   return 1;  /*tiene dos divisores: el 1, y él mismo*/
  else
/* else opcional, el return rompe la ejecución, sale de la función */
   return 0;
/* tiene más de dos divisores, aparte del 1 y de él mismo */
}
```

# Función sobre tablas

```
#include <stdio.h>
#define N 100
int media(int t[], int nelem) {
  int i, s=0;
  for (i=0; i<nelem; i++) s+=t[i];
  return s/nelem;
}
main( ) {
  int tabla[N],i;
  for (i=0; i<N; i++) {
  printf("\nDame el %d° valor: ",i+1);
  scanf("%d", &tabla[i]);
  }
  printf("\nLa media de la tabla es ");
  printf("%d", media(tabla,N));
}
```

# Procedimientos

```
#include <stdio.h>
#define N 10
void pedir_tabla(int t[]) {
   int i;
   for (i=0; i<N; i++)
   scanf("%d", &t[i]);
}
void izq(int v[]) {
   int i, aux;
   aux=v[0]; for (i=0; i<N-1; i++) v[i]=v[i+1]; v[N-1]=aux;
}
void mostrar_tabla(int t[]) {
   int i;
   for (i=0; i<N; i++) printf("\t%d ", t[i]);
}
main( ) {
   int v[N];
   pedir_tabla(v);
   izq(v);
   printf("\nDesplazada a la iquierda: \n");
   mostrar_tabla(v);
}
```

# Procedimientos

```
#include <stdio.h>
#include <string.h>
#define N 255
void copiar_cadena(char cad[], char cad2[]) {
   int i=0, longitud;
   longitud=strlen(cad);
   while (i<longitud){
    cad2[i]=cad[i]; /* Copiamos si cad[i]!='\0' */
    i=i+1;
   }
   cad2[i]='\0'; /* Añadimos el fin de cadena*/
}
void main() {
   char cad[N],cad2[N],cop[]="Cadena copiada"; /*cop 15 pos.*/
   printf("\n\nDame una cadena que la copio en otra: \n");
   fgets(cad,N,stdin); /* Pido una cadena con espacios */
   copiar_cadena(cad,cad2);
   printf("\n\n%s: \n%s", cop, cad2);
}
```

# Punteros, apuntadores (pointers)

```
int *p;                  /* p es un puntero a enteros */
```

Es decir, contiene una dirección de memoria, y el lenguaje sólo permite que apunte a enteros

```
int x=10, y=0;


...
  p=&x;      /* p contiene la dirección de memoria */
      /* donde se almacena x */
  y=*p;      /* El valor de lo apuntado por p */
      /* se asigna a y */
```

# Punteros, apuntadores (pointers)

```
#include <stdio.h>

int main()

{int nu[]={100,800,200};

char ca[]={'a','b','c'};

int *p;char *q;

//Para saber si funciona correctamente &var+1

    ó usando tanto si var es caracter o numero

printf("La dirección es:%p y el contenido es:
%d\n",&nu[0]+2,*(&nu[0]+2));

p=nu;

p=p+2;

printf("\nLa dirección es:%p y el contenido es:
%d\n",p,*p);
```

# Punteros, apuntadores (pointers)

```
printf("La dirección es:%p y el contenido es:
    %c\n",&ca[0]+2,*(&ca[0]+2));

q=ca;

q=q+2;

printf("\nLa dirección es:%p y el contenido es:
    %c\n",q,*q);

//Cual es el salto

printf("El tamaño de int es de %d bytes y el de
    char es %d",sizeof(int),sizeof(char));

printf("\nLa dirección base en array integer es
%p y después de suma 1 es de:%p\n",nu,nu+1);

printf("\nLa dirección base en array caracteres
es %p y después de suma 1 es de:
%p\n",ca,ca+1);

return 0;

}
```

# Punteros, apuntadores (pointers)

La dirección es:0xbfef9704 y el contenido es:200

La dirección es:0xbfef9704 y el contenido es:200

La dirección es:0xbfef9713 y el contenido es:c

El tamaño de int es de 4 bytes y el de char es 1

La dirección base en array integer es 0xbfef96fc y después de suma 1 es de:0xbfef9700

La dirección base en array caracteres es 0xbfef9711 y después de suma 1 es de:0xbfef9712

# Cadenas con punteros

```
#include <stdio.h>
#include <string.h>
char * nombremes(int n);
main() {
  char *s;
  s=nombremes(4);
  printf("%s",s);
}
char * nombremes(int n){
  static char * nmes[]={"err","ene","feb","mar","abr",
     "may","jun","jul","ago","sep","oct","nov","dic"};
  return ( (n<1||n>12) ? nmes[0] : nmes[n]); /* if */
}
```

# Cadenas con punteros

```
int cont_car(char car, char s[]) {/*Con índice*/
  int cont=0,i;
  for (i=0;s[i]!='\0';i++)
   if (s[i]==car) cont++;
  return cont;
}
int cont_car(char car, char s[]) { /*Con puntero*/
  int cont=0;
  char *p;
  for (p=s;*p!='\0';p++)
   if (*p==car)
      cont++;
  return cont;
}
```

# Operaciones lógicas con bits

```
#include <stdio.h>
void main(){int i,cont=0;
char dato='a',mascara=0x01; //'a'->97->1100001
for (i=0;i<8;i++)//mascara=00000001
  {if (dato & mascara)          //true si !=0
       {cont++;}                //1100001&0000001=1
   mascara=mascara<<1;          //mascara=00000010
  }  //1100001&0100000=32
printf("\nEn %c hay %d unos", dato,cont);
}
```

**& and >> Desplazamiento a la derecha**

**| or << Desplazamiento a la izquierda**

**~ not Lo mismo con iguales &= |= ^=**

**^ xor <<= >>= (Idem que `f*=i o f=f*i`)**

# struct

```
struct dir {
char nombre[30];
char calle[40];
int codigo;
};
//Para declarar una variable de tipo estructura dir
struct dir info_dir,info_dir1;
//Asignar valor a la variable:
info_dir.codigo=48004;
//Mostrar:
printf("%d",info_dir.codigo);
//Arrays de estructuras
struct dir info_dir[100];
info_dir[3].codigo=48004;
```

# struct

```
struct st{
  struct {int a1, a2, a3;}a;
  struct {int a1, a2, a3;}b;
} *puntero, estructura;

puntero = &estructura;
estructura.a.a1=1;
puntero->b.a1=2; ≡ ((struct st)(*puntero)).b.a1=2;
```

# Ficheros

**Formato de declaración de un fichero: FILE *fd (en la declaración no se indica que tipo de datos contiene)**

**A la hora de abrir se le dice si queremos tratar como texto o binario. Los primeros sirven para almacenar caracteres y los segundos, cualquier tipo de datos.**

**Las operaciones obligatorias son:**

**-apertura**

**-cierre**

**Las operaciones opcionales son:**

**-lectura**

**-escritura**

# Ficheros

- **fichero=fopen(nombre-fichero,modo);**

**Ejm: if ((entrada= fopen("fich.en","r"))==NULL)**
**{printf(”error en apertura”);}**

**devuelve:un puntero a fichero o NULL (error)**

**1° param: “/home/euiti/lana.txt”**

**2° param: r (sólo lectura, si el fichero no existe error); w (crea sino existe o trunca si existe); a (añade al final del fichero, sino existe lo crea); r+ (r and w, pero no trunca, ni permite crear sino existe); w+ (w and r, permite truncar y crear ); idem para wb,rb,ab,wb+,rb+... es para en binario sin conversiones.(*truncar: vaciar el fichero completamente)**

# Ficheros

```
int fclose(FILE *fp);

Ejm:fclose(entrada);
```

# Ficheros:Lectura y escritura

Hay diferentes maneras de realizar las lecturas y escrituras:

a) de caracteres:fgetc y fputc

b) de cadenas:fgets y fputs

c) formateadas de texto:fscanf y fprintf

d) por bloques (binario):fread y fwrite

# Ficheros de caracteres:fgetc y fputc

fgetc() lee el siguiente caracter del flujo y

retorna el unsigned char leido convertido en

entero o el valor de EOF o un error.

```
int fgetc(FILE *stream);
```

Ejm:c=fgetc(entrada);

1° param: descriptor del fichero

fputc() escribe en el flujo el caracter c,

convirtiendolo antes en un unsigned char

```
int fputc(int c, FILE *flujo);
```

Ejm:fputc(cs,salida

1° param:entero a introducir.

2° param:descriptor de fichero.

# Ficheros de caracteres:fgetc y fputc

```
#include <stdio.h>
#define EOF (-1)
void main() {
  FILE *entrada, *salida;
  int i, c, cs;
  entrada= fopen("fich.en","r");
  salida = fopen("fich.sa","w");
  c=fgetc(entrada);
  while(c!=EOF) {
  cs = (c>='A' && c<='Z' ? c+32: c);
  fputc(cs,salida);
  c=fgetc(entrada);
  }
  fclose(entrada);fclose(salida);
}
```

# Ficheros de caracteres: fgetc y fputc

2 maneras de recorrer el fichero de inicio a fin (de manera secuencial):

a)while (!feof(entrada))

{c=fgetc(entrada);

printf(“%c”,c)}

b)

while((c=fgetc(entrada))!=EOF)

{printf(“%c”,c);}

feof retorna un valor distinto de cero(true) cuando lee el indicador de final de fichero.

fgetc() lee el siguiente caracter del flujo y retorna el caracter leido convertido en entero o el valor de EOF o un error.

# Agenda

```
typedef struct
{
char nombre[TAMANOSTRING];
char apellido1[TAMANOSTRING];
char telefono[10];
} tipo_amigo;
int main(void)
{
    tipo_amigo amigos[TAMANO];
    int cont=0,
    cont=cargar(amigos,FICHERO);
    guardar(cont,amigos,FICHERO);
    return 0;
}
```

# Agenda. Cadenas: fgets y fputs

fputs() escribe la cadena de caracteres s en flujo, sin su terminador '\0'

int **fputs**(const char cadena[], FILE *fichero);

-Retorna un número no negativo si acaban bien, o EOF/NULL en caso de error.

char * **fgets**(char cadena[], int n, FILE *fichero);

-Retorna un puntero a la cadena pasada o NULL si final de fichero o error

-n será el tamaño de la cadena a leer, sizeof(cadena)

# Agenda. Cadenas: fgets y fputs

```
int cargar(tipo_amigo amigos[], char nombrefichero[])
{ int i; FILE *fdagenda;
  fdagenda= fopen(nombrefichero, "r");
  if (!fdagenda)  { printf("El archivo %s no existe\n", nombrefichero);
                    return 1;}
  i = 0;
  while (!feof(fdagenda))
          {  fgets(amigos[i].nombre, sizeof(amigos[i].nombre), fdagenda);
             fgets(amigos[i].apellido1, sizeof(amigos[i].apellido1), fdagenda);
             fgets(amigos[i].telefono, sizeof(amigos[i].telefono), fdagenda);
             i ++;   }
  fclose (fdagenda);
  return i-1;
}
```

# Agenda. Cadenas: fgets y fputs

```
int guardar(int cont,tipo_amigo amigos[],char nombrefichero[])
{
int i;
FILE *fdagenda;
fdagenda= fopen(nombrefichero, "w");
if (!fdagenda)
{ printf("el archivo no se puede crear");
  return FALSE;
}
i = 0;
while (i<cont)
{fputs (amigos[i].nombre, fdagenda);
fputs (amigos[i].apellido1, fdagenda);
fputs (amigos[i].telefono, fdagenda);
i=i+1;
}
fclose (fdagenda);
return TRUE;
}
```

# Agenda. Cadenas: fprintf y fscanf

int fprintf(FILE *stream, const char *format, …): fprintf escribe en el fichero las varibeles con formato

-Retorna: En caso de éxito, estas funciones devuelven el número de caracteres escritos en el fichero. Si se encuentra un error de salida, se devuelve un valor negativo.

int fscanf(FILE *stream, const char *format, …): fscanf lee su entrada según un formato del puntero a FILE flujo

- Estas funciones devuelven el número de elementos de la entrada asignados, que pueden ser menores que los formatos suministrados para conversión, o incluso cero, en el caso de un fallo de concordancia. Cero indica que, mientras había caracteres disponibles en la entrada, no ocurrió ninguna asignación; normalmente esto es debido a un carácter de entrada inválido, como un carácter alfabético para una conversión `%d'. Se devuelve el valor EOF si ha habido un fallo de entrada antes de ninguna conversión, como cuando se llega al final de la entrada. Si ocurre un error de lectura o se llega al final de la entrada después de que se haya hecho alguna conversión al menos, se devuelve el número de conversiones completadas hasta ese punto con éxito.

# Agenda. Con formato: fprintf y fscanf

```
int cargar(tipo_amigo amigos[], char nombrefichero[])
{ int i; FILE *fdagenda;
  fdagenda= fopen(nombrefichero, "r");
  if (!fdagenda)  { printf("El archivo %s no existe\n", nombrefichero);
                    return 1;}
  i = 0;
  while (!feof(fdagenda))
            {fscanf(fdagenda, “%s”, amigos[i].nombre);
             fscanf(fdagenda, “%s”, amigos[i].apellido1);
             fscanf(fdagenda, “%s”, amigos[i].telefono);
             i ++;}
  fclose (fdagenda);
  return i-1;
}
```

# Agenda. Con formato: fprintf y fscanf

```
int guardar(int cont,tipo_amigo amigos[],char nombrefichero[])
{
int i;
FILE *fdagenda;
fdagenda= fopen(nombrefichero, "w");
if (!fdagenda)
{ printf("el archivo no se puede crear");
  return FALSE;
}
i = 0;
while (i<cont)
{fprintf(fdagenda, “%s\n”, amigos[i].nombre);
 fprintf(fdagenda, “%s\n”, amigos[i].apellido1);
 fprintf(fdagenda, “%s\n”, amigos[i].telefono);
 i=i+1;
}
fclose (fdagenda);
return TRUE;
}
```

# Agenda. Por bloques (binario): fread y fwrite

```
typedef struct
{
char nombre[TAMANOSTRING];
char apellido1[TAMANOSTRING];
char apellido2[TAMANOSTRING];
char telefonomovil[10];
char correo[TAMANOSTRING];
} tipo_amigo;
int main(void)
{
    tipo_amigo amigos[TAMANO];
    int cont=0,
    cont=cargar(amigos,FICHERO);
    guardar(cont,amigos,FICHERO);
    return 0;
}
```

# Agenda. Por bloques (binario):fread y fwrite

```
int cargar(tipo_amigo amigos[],char nombrefichero[])
{
int i,cont;
FILE *fdagenda;
fdagenda= fopen(nombrefichero, "rb");
if (!fdagenda)
{ printf("El archivo %s no existe\n",nombrefichero);
  return FALSE;
}
i = 0;
while (!feof(fdagenda))
{
fread (&amigos[i], sizeof(tipo_amigo), 1, fdagenda);
i ++;
}
cont=i-1;
fclose (fdagenda);
return cont;
}
```

# Agenda. Por bloques (binario):fread y fwrite

```
int guardar(int cont,tipo_amigo amigos[],char nombrefichero[])
{
int i;
FILE *fdagenda;
fdagenda= fopen(nombrefichero, "wb");
if (!fdagenda)
{ printf("el archivo no se puede crear");
  return FALSE;
}
i = 0;
while (i<cont)
{
fwrite (&amigos[i], sizeof(tipo_amigo), 1, fdagenda);
i=i+1;
}
fclose (fdagenda);
return TRUE;
}
```

# Argumentos del programa

```
#include <stdio.h>
int main(int argc, char *argv[]) {
  int i;
  printf("\nPrograma: %s\n",argv[0]);
  for(i=1;i<argc;i++)
   printf("argumento %d: %s\n",i,argv[i]);
  return 0;
}
```

# Varios

**En stdlib.h:**

```
//exit:terminación normal del programa,y devolución del estado
  al proceso padre.
void exit (int status)
exit(EXIT_SUCCESS); //#define EXIT_SUCCESS=0
exit(EXIT_FAILURE); //#define EXIT_FAILURE=1
int atoi (char *); convierte un cadena en entero
```

**En string.h:**

```
int strlen(char *);
char * strcpy(char *dest,const char * orig);
char * strcat(char *dest,const char * orig);
```

**Conversión en memoria:**

```
sprintf(cadena, "formato", lista de expresiones);
Ejm:sprintf(cadena, "%03d", numero); convierte un 3 en “003/0”
  introduciéndolo en cadena.
```