Sortida estàndard

- La instrucció que permet fer la sortida de dades per la sortida estàndard és:

```
printf( format, arguments);
```

- **`format`** és una cadena que indica com sortiran les dades per pantalla.
- **`arguments`** és una llista d'expressions que es volen escriure per la sortida estàndard.

Formats i tipus d'argument

format	tipus arg.	resultat
`%d`, `%i`	enter	enter decimal amb signe
`%u`	enter	enter decimal sense signe
`%x`, `%X`	enter	enter hexadec. sense signe
`%ld`, `%lx`	enter	enter llarg
`%f`	real	real amb punt i signe
`%e`, `%E`	real	notació exponencial
`%g`	real	notació segons el valor
`%c`	caràcter	un caràcter
`%s`	cadena	cadena de caràcters

Seqüències d'escapament

Seqüència	Efecte
`'\n'`	Salta línia
`'\t'`	Tabulador
`'\"'`	Cometes
`'\\'`	backslash
`'\?'`	Signe d'interrogació

Exemples de sortida estàndard

```
printf("Hola␣\n");
printf("El␣numero␣12␣es␣%d", 12);
printf("Escriure␣%c\t%d\t%g\n",'a', 23*k, 2.5e+5);
```

Entrada stàndard

- Permet llegir dades de l'entrada stàndard.
- La instrucció en C de llegir dades per l'entrada stàndard és:

```
scanf(format, arguments);
```

- Retorna sempre el valor de nombre de dades que s'han llegit correctament.
- **`format`** és semblant a la instrucció **`printf`**, encara que en alguns casos pot comportar-se una mica diferent.
- **`arguments`** és una llista d'adreces de variables que es volen llegir per l'entrada estàndard.

Exemples d'entrada estàndard

```
scanf("%f", &a); /*a es una variable real */

scanf("%c", &lletra); /*lletra es una variable caracter */

scanf("%d␣%d", &x, &y); /*x i y son variables enteres */

scanf("%d::%d", &x, &y); /*x i y son variables enteres */
```

Fitxers en C

- La instrucció per obrir un fitxer en C és:

```
descriptor_fitxer = fopen( nomfitxer, mode);
```

- El descriptor de fitxer és l'identificador de l'arxiu i es declara com:

```
FILE *descriptor_fitxer;
```

- Si el descriptor és `NULL` vol dir que no s'ha pogut obrir el fitxer.
- El mode és la manera d'obrir el fitxer.
- Per tancar un fitxer en C, s'usa la instrucció:

```
fclose(descriptor_fitxer);
```

Modes d'obertura d'un fitxer

mode	Què vol dir?
`r`	Fitxer de lectura que ha d'existir
`w`	Crea un nou fitxer per escriure Si existeix, es borra el seu contingut Si no existeix, es crea
`a`	Afegeix al final en un fitxer ja existent

Lectura i escriptura en fitxer

- La instrucció per lectura en un fitxer prèviament obert:

  ```
  fscanf(descriptor_fitxer, format, arguments);
  ```

  si retorna el valor **`EOF`** indica que s'han acabat les dades
- I per escriptura:

  ```
  fprintf(descriptor_fitxer, format, arguments);
  ```

- El format i els arguments tenen la mateixa sintaxi que les instruccions **`scanf(..)`** i **`printf(..)`** vistes prèviament.

Tot programa té definits 3 descriptors **estàndar** per defecte:

- **`stdin`** entrada;
  `scanf(fmt,args)` equival a **`fscanf(stdin,fmt,args)`**
- **`stdout`** sortida;
  `printf(fmt,args)` equival a **`fprintf(stdout,fmt,args)`**
- **`stderr`** sortida d'errors, també associada a pantalla

Exercici

Feu un programa en C que faci una còpia d'una imatge de tipus **ppm** simple, fent un "reverse" del colors.
El format d'una imatge d'aquest tipus es composa de:

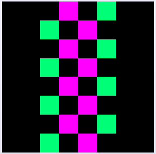
- El "*valor màgic*" **P3**
- mida en pixels en x i en y, expressats com a enters decimals
- enter positiu **maxC** (usualment $2^k - 1$), seguit de tantes tripletes d'enters positius ($\leq$**maxC**) com **midaX*midaY**.
  (**0**,**0**,**0**) negre (**maxC**,**maxC**,**maxC**) blanc.

```
P3
MidaX
MidaY
255
R G B
R G B
...
```

```
P3
MidaX MidaY
15
R G B R G B .....
R G B R G B .....
R G B R G B .....
...
```

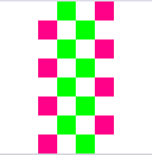
Exemple

```
P3
8 8
15
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0
```



Exemple

```
P3
8 8
15
 15 15 15  15 15 15  15 15 15   0 15  0
 15 15 15  15  0  8  15 15 15  15 15 15
 15 15 15  15 15 15  15  0  8  15 15 15
  0 15  0  15 15 15  15 15 15  15 15 15
 15 15 15  15 15 15  15 15 15   0 15  0
 15 15 15  15  0  8  15 15 15  15 15 15
 15 15 15  15 15 15  15  0  8  15 15 15
  0 15  0  15 15 15  15 15 15  15 15 15
 15 15 15  15 15 15  15 15 15   0 15  0
 15 15 15  15  0  8  15 15 15  15 15 15
 15 15 15  15 15 15  15  0  8  15 15 15
  0 15  0  15 15 15  15 15 15  15 15 15
 15 15 15  15 15 15  15 15 15   0 15  0
 15 15 15  15  0  8  15 15 15  15 15 15
 15 15 15  15 15 15  15  0  8  15 15 15
  0 15  0  15 15 15  15 15 15  15 15 15
```



Solució

```
#include<stdio.h>

int main (void) {
  FILE *inp, *out;
  char nomE[80], nomS[80], magic[]={'␣','␣','\0'};
  unsigned int  i,r,g,b, mX, mY,maxC;

  scanf("␣%s", nomE);
  inp = fopen(nomE,"r");
  if ( inp == NULL ) {
    printf("Error␣obrint␣fitxer␣d'entrada\n");
    return 0;
  }
```

Solució ...

```
scanf("␣%s", nomS);
out = fopen(nomS,"w");
if ( out == NULL ) {
  fclose(inp);
  printf("Error␣obrint␣fitxer␣de␣sortida\n");
  return 0;
}
```

Solució ...

```
fscanf(inp,"%c%c",&magic[0],&magic[1]);
if ( magic[0] != 'P' || magic[1]!='3') {
  fclose(inp);
  fclose(out);
  printf("Error: magic number incorrect\n");
  return 0;
}
fprintf(out,"%s\n",magic);
fscanf(inp, " %u %u", &mX, &mY);
fprintf(out, "%u %u\n", mX, mY);
fscanf(inp, " %u", &maxC);
fprintf(out, "%u\n", maxC);
```

Solució ...

```
  for (i=0; i< mX*mY; i++) {
    if ( fscanf(inp, "␣%u␣%u␣%u", &r, &g, &b) == EOF ) {
      fclose(inp); fclose(out);
      printf("Error:␣falten␣tripletes\n");
      return 0;
    }
    if ( r>maxC || g>maxC || b>maxC ) {
      fclose(inp); fclose(out);
      printf("Error:␣tripleta␣erronia␣%u␣%u␣%u\n",r,g,b);
      return 0;
    }
    /* reverse dels colors */
    fprintf(out, "%u␣%u␣%u\n", maxC-r, maxC-g, maxC-b);
  }
  if ( fscanf(inp, "␣%u␣%u␣%u", &r, &g, &b) != EOF )
    printf("Error:␣massa␣tripletes,␣sortida␣truncada\n");
    fclose(inp); fclose(out);
    return 0;
}
```

Exemple: còpia de fitxers

Programa en C que faci una còpia d'un arxiu de texte en un altre.

- **`char *fgets(char *string, int n, FILE *stream);`**
  - Llegeix, del fitxer especificat per l'apuntador **`stream`**, una línia, i la posa en la cadena `string`.
  - S'atura si: o bé s'han llegit `n-1` caràcters, o bé quan hi ha el caràcter **`'\n'`**, o bé quan s'acaba el contingut del fitxer.
  - Si s'ha pogut llegir una línia, retorna el mateix paràmetre `string`.
  - Si s'ha acabat el fitxer i no s'ha llegit cap caràcter, llavors el contingut de `string` no canvia i retorna l'apuntador **`NULL`**.
- **`int fputs(char *string, FILE *stream);`**
  - Escriu al fitxer especificat per l'apuntador **`stream`**, la tira de caràcters `string` sense el **`'\0'`** final.
  - Retorna un valor no negatiu si tot va bé i **`EOF`** si falla

Codi

```
#include <stdio.h>
#define LONG 81
int main (void) {
  char linia[LONG], origen[30], copia[30];
  FILE *pfl, *pfe;
  printf("fitxer␣original␣?␣(<30␣caracters)\n");
  scanf("%s", origen);
  printf("nom␣de␣la␣copia␣?␣(<30␣caracters)\n");
  scanf("%s", copia);
```

Codi (cont.)

```
  pfl = fopen(origen, "r");
  if (pfl == NULL) {
    printf("Error␣obertura␣origen\n"); return 1;
  }
  pfe = fopen(copia, "w");
  if (pfe == NULL) {
    printf("Error␣obertura␣copia\n"); return -1;
  }

  while (fgets (linia, LONG, pfl) != NULL) {
    fputs(linia, pfe);
  }
  fclose(pfl);
  fclose(pfe);
  return 0;
}
```

Exercici

Programa en C que trobi la longitud de la línia més llarga d'un fitxer, llegint caràcter a caràcter, per a saber on acaben les línies cal usar que el final de línia és `'\n'`
Per a la lectura s'usarà la funció

- `int getc( FILE *stream )`
  - llegeix el següent caràcter del fitxer apuntat per **stream**
  - avança l'indicador de posició dins del fitxer (la propera crida llegirà el següent caràcter)
  - retorna el caràcter llegit (convertir a `int`), o bé **EOF** (si no ha pogut llegir)

Arguments en la línia de comandes

- Quan s'executa un programa es poden passar arguments a la funció **`main`** des de la línia de comandes: **`./nom_programa arg1 arg2 arg3 ... argn`**
- El prototipus de la funció **`main`** és llavors:

```
int main (int argc, char *argv[])
```

- **`argc`** indica el nombre d'arguments inclòs el nom del programa
- **`argv[0]`** és el nom del programa executat (**`./nom_programa`**)
- **`argv[i]`** és el **`i`**-èssim paràmetre, com a tira de **`char`** que caldrà convertir

Refem el programa de copiar `ppm` per a usar-ho

Solució

```
#include<stdio.h>
int main (int argc, char *argv[]) {
  FILE *inp, *out;
  char magic[]={'␣','␣','\0'};
  unsigned int  i,r,g,b, mX, mY,maxC;
  if (argc != 3 ) {
    printf("Error:␣Us␣%s␣fileInput␣fileOutput\n",argv[0]);
    return 0;
  }
  inp = fopen(argv[1],"r");
  if ( inp == NULL ) {
    printf("Error␣obrint␣fitxer␣d'entrada\n");
    return 0;
  }
  out = fopen(argv[2],"w");
  if ( out == NULL ) {
    fclose(inp);
    printf("Error␣obrint␣fitxer␣de␣sortida\n");
    return 0;
  }
```

Arguments numèrics

Els arguments sempre es reben com a tires de caràcters, són útils les funcions:

- `int atoi(const char *p);` converteix a `int`
- `long atol(const char *p);` converteix a `long`
- `double atof(const char *p);` converteix a `double`

Atenció: aquestes funcions no detecten errors

Exemple:

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

int main(int argc, char *argv[]) {
  int val;
  double dval;
  if (argc < 2) {
    printf("Us: %s strInt [strDouble]\n", argv[0]);
    return 0;
  }
  val = atoi(argv[1]);
  printf("atoi() returned %d\n", val);
  if (argc == 3 ) {
    dval = atof(argv[2]);
    printf("atof() returned %g\n", dval);
  }
  return 0;
}
```

Exemple:(execució)

- **`./conver -123 +1.57e-7`**

```
atoi() returned -123
atof() returned 1.57e-07
```

- **`./conver -12e3 +1.57f-7`**

```
atoi() returned -12
atof() returned 1.57
```