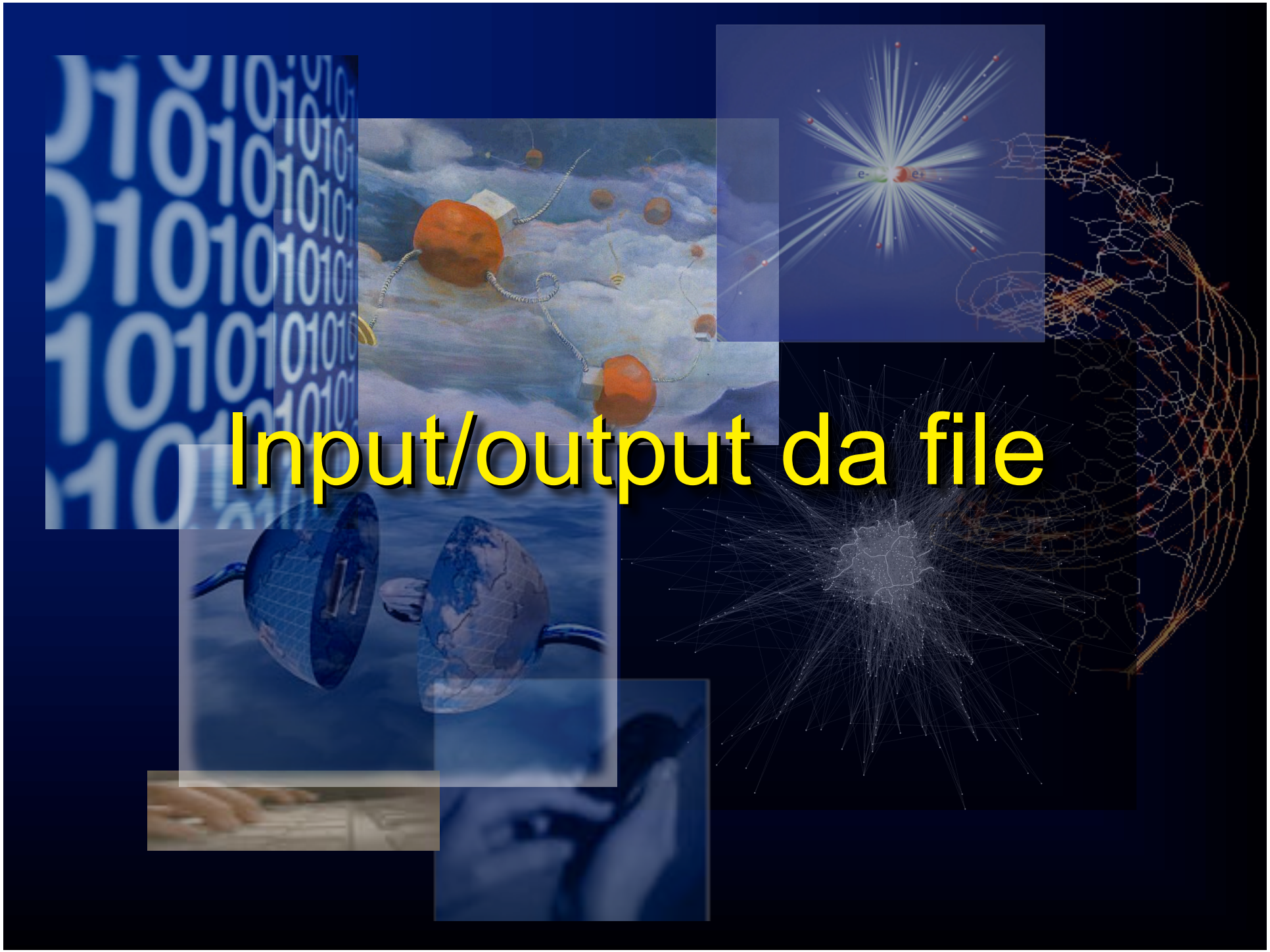
Corso di Informatica A.A. 2009-2010

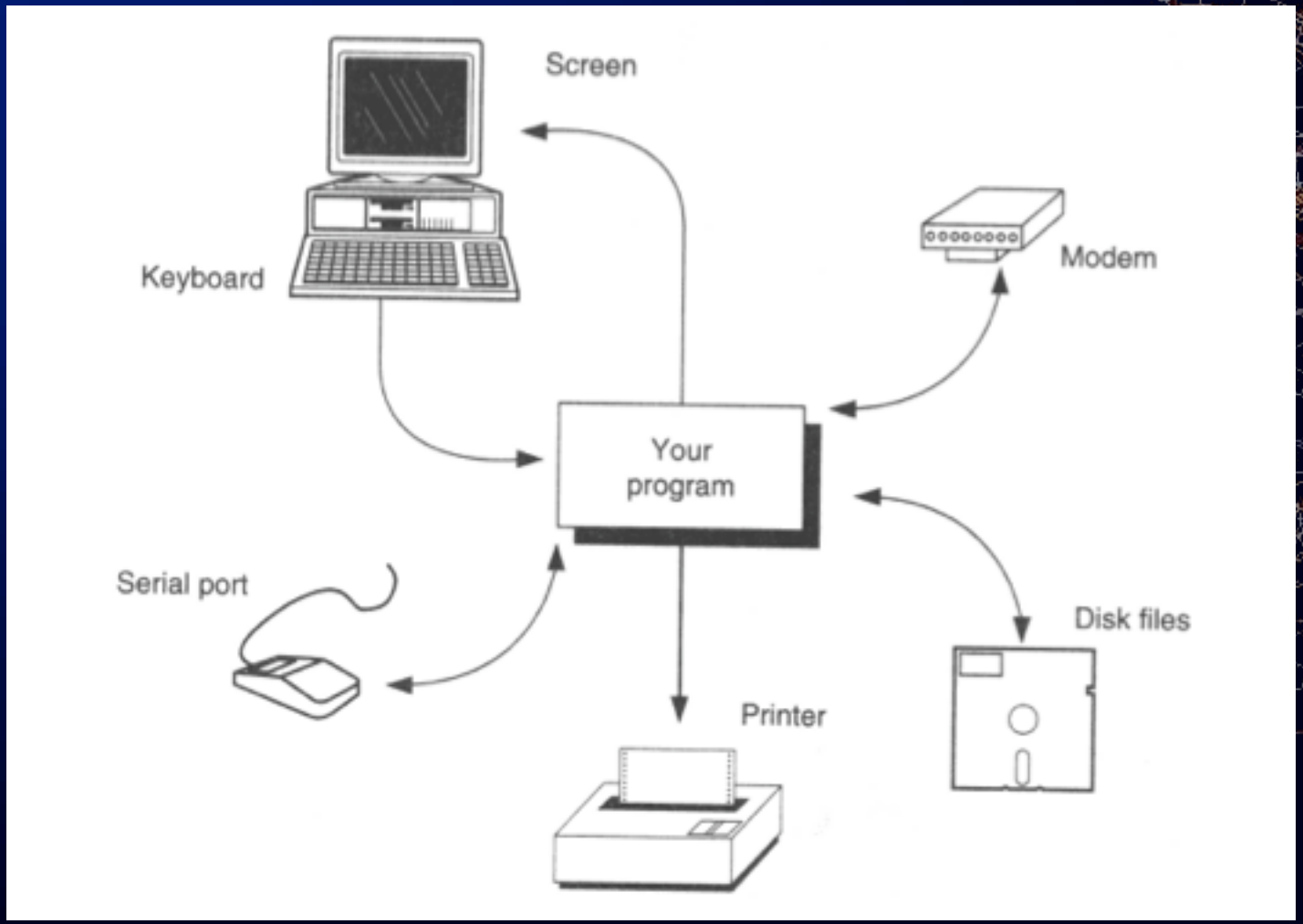
Lezione 13


Input/output da file

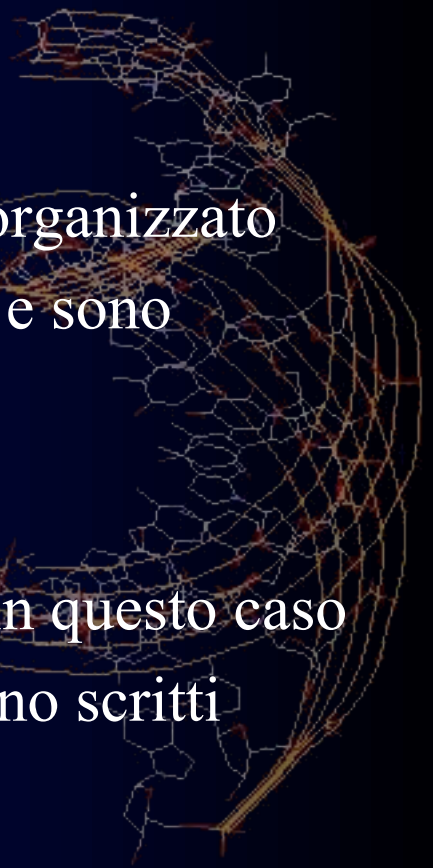
Stream

Si definisce stream una sequenza di bytes di dati che può essere in ingresso (input stream) o in uscita (output stream) dal programma

Ogni stream in C è connesso ad un *file*. Per *file* si intende in questo caso un *intermediario* tra lo stream esaminato dal programma e il dispositivo reale in uso per l'operazione di input o output


Screen
Keyboard
Modem
Your program
Serial port
Disk files
Printer

Streams di testo e streams binari



Uno stream di testo è costituito solo da caratteri ed è organizzato in linee che possono essere lunghe sino a 255 caratteri e sono terminate da un carattere di newline

Uno stream binario può trattare qualsiasi tipo di dato, in questo caso i bytes di dati non sono interpretati in alcun modo e sono scritti e/o letti così come sono

Streams predefiniti

In C ci sono tre streams predefiniti che sono aperti automaticamente all'inizio dell'esecuzione di ogni programma: `stdin,stdout,stderr`

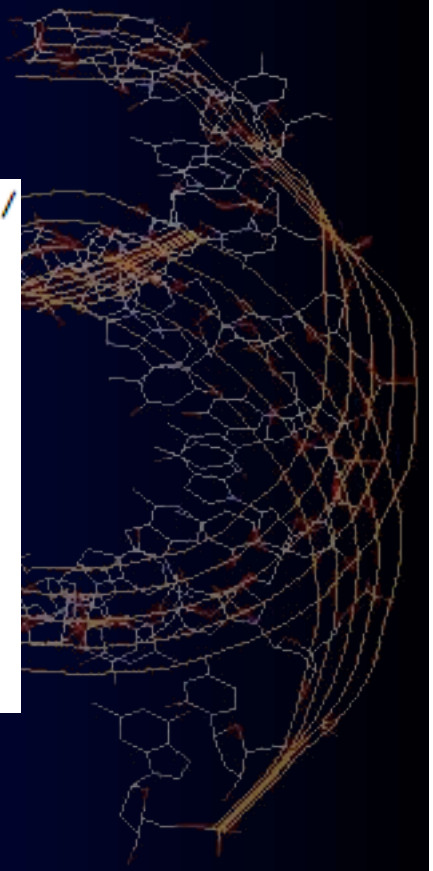
Quando sono state utilizzate le funzioni `printf()` o `puts()` per visualizzare del testo sullo schermo è stato utilizzato lo stream `stdout.` Mentre `gets()` e `scanf()` hanno utilizzato lo stream `stdin` per leggere del testo dalla tastiera

Funzioni standard per lo stream I/O

`printf()`	output formattato
`puts()`	output di una stringa
`putchar()`	output di un carattere
`scanf()`	input formattato
`gets()`	input di una stringa
`getchar()`	input di un carattere

Esempio

```
/* Demonstrates the equivalence of stream input and output. */
#include <stdio.h>

main()
{
   char buffer[256];

   /* Input a line, then immediately output it. */

   puts(gets(buffer));

   return 0;
}
```



Input da tastiera

- Input di un carattere
- Input di una linea
- Input formattato

Input di un carattere

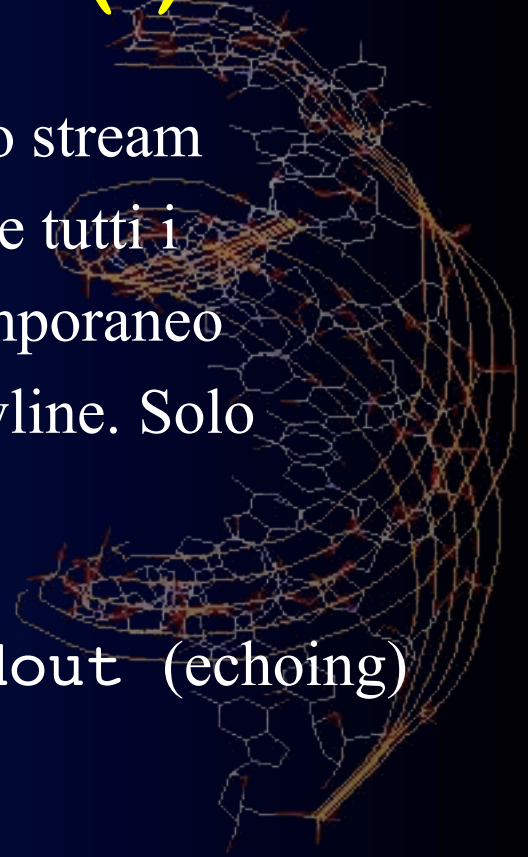
Le funzioni di input di carattere leggono un carattere alla volta dallo stream in input. Quando sono chiamate restituiscono il carattere successivo nello stream o `EOF` se è stata raggiunta la fine del file o vi è stato un errore

`EOF` è una costante definita pari a `-1` in `stdio.h`

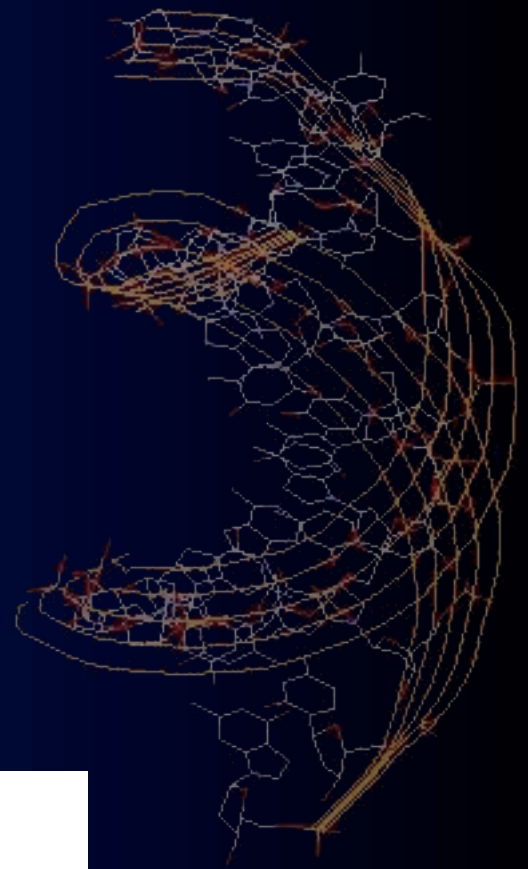
La funzione `getchar()`

La funzione `getchar()` legge un carattere dallo stream `stdin`. La funzione utilizza il buffer nel senso che tutti i caratteri ricevuti sono immagazzinati in un file temporaneo sino a quando l'utente non invia il carattere di newline. Solo a questo punto i caratteri sono inviati allo `stdin`

Ogni carattere è inviato automaticamente allo `stdout` (echoing)



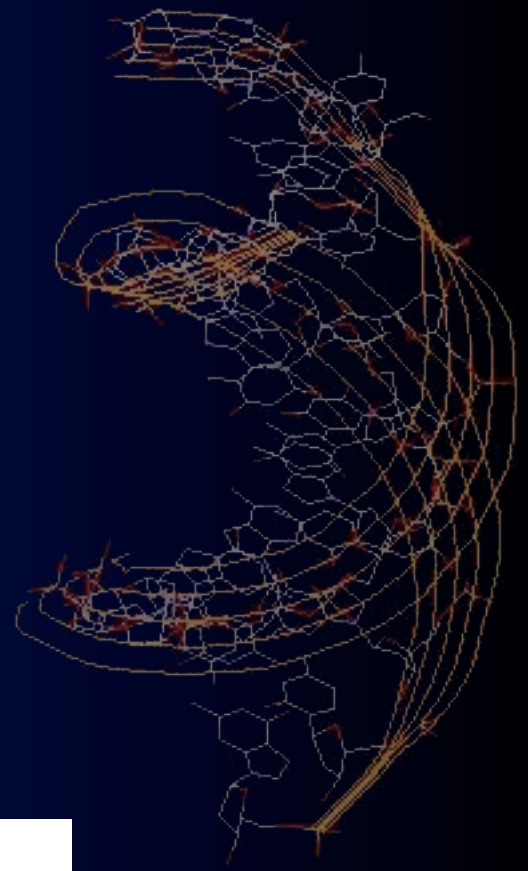
```c
 /* Demonstrates the getchar() function. */

#include <stdio.h>

main()
{
    int ch;

    while ((ch = getchar()) != '\n')
         putchar(ch);

    return 0;
}
```



INPUT/OUTPUT:

```
This is what's typed in.
This is what's typed in.
```

```
/* Using getchar() to input strings. */

#include <stdio.h>

#define MAX 80

main()
{
     char ch, buffer[MAX+1];
:    int x = 0;
:
:    while ((ch = getchar()) != '\n' && x < MAX)
:        buffer[x++] = ch;
:
    buffer[x] = '\0';

    printf("%s\n", buffer);

    return 0;
}
```



OUTPUT:

```
This is a string
This is a string
```

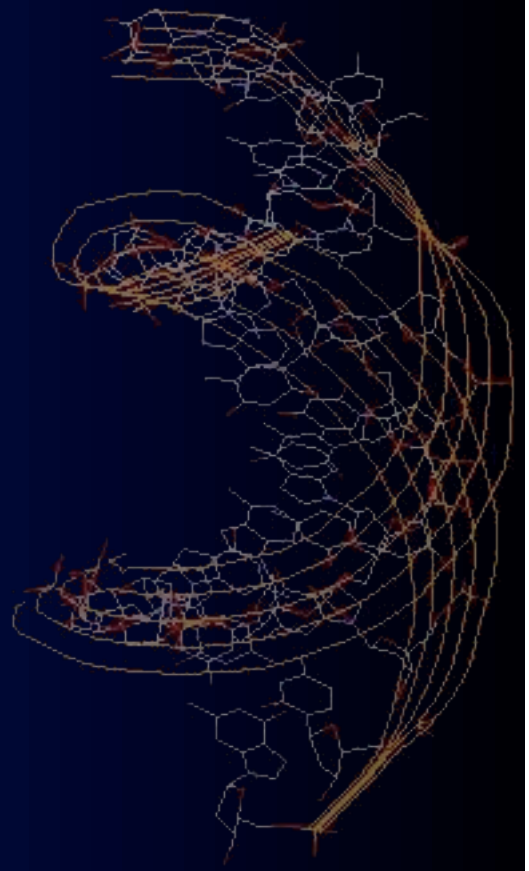
Input da tastiera

Input di una linea

```
char *gets(char *str);
```

Input formattato

La funzione `scanf()`



Caratteri extra

La funzione `scanf()` utilizza il buffer. Nessun carattere è realmente ricevuto dallo `stdin` sino all'invio di un Enter da parte dell'utente.

L'esecuzione ritorna dalla `scanf()` solo quando è stato ricevuto sufficiente input da soddisfare le specifiche contenute nella stringa di formato.

La funzione `scanf()` processa solo il numero di caratteri necessario per soddisfare la sua stringa di formato.

Se vi sono caratteri extra non necessari essi restano in attesa nello `stdin`

Caratteri extra

Esaminiamo questa situazione:

```
scanf("%d %d", &x, &y);
```

Tre possibili casi:

1. L'utente digita 12 14 e invia Enter
2. L'utente digita 12 seguito da un Enter. In questo caso la `scanf()` attende l'input mancante
3. L'utente digita 12 14 16 e invia Enter. La `scanf()` legge 12 e 1 e ritorna lasciando i caratteri 1 e 6 in attesa nello `stdin`

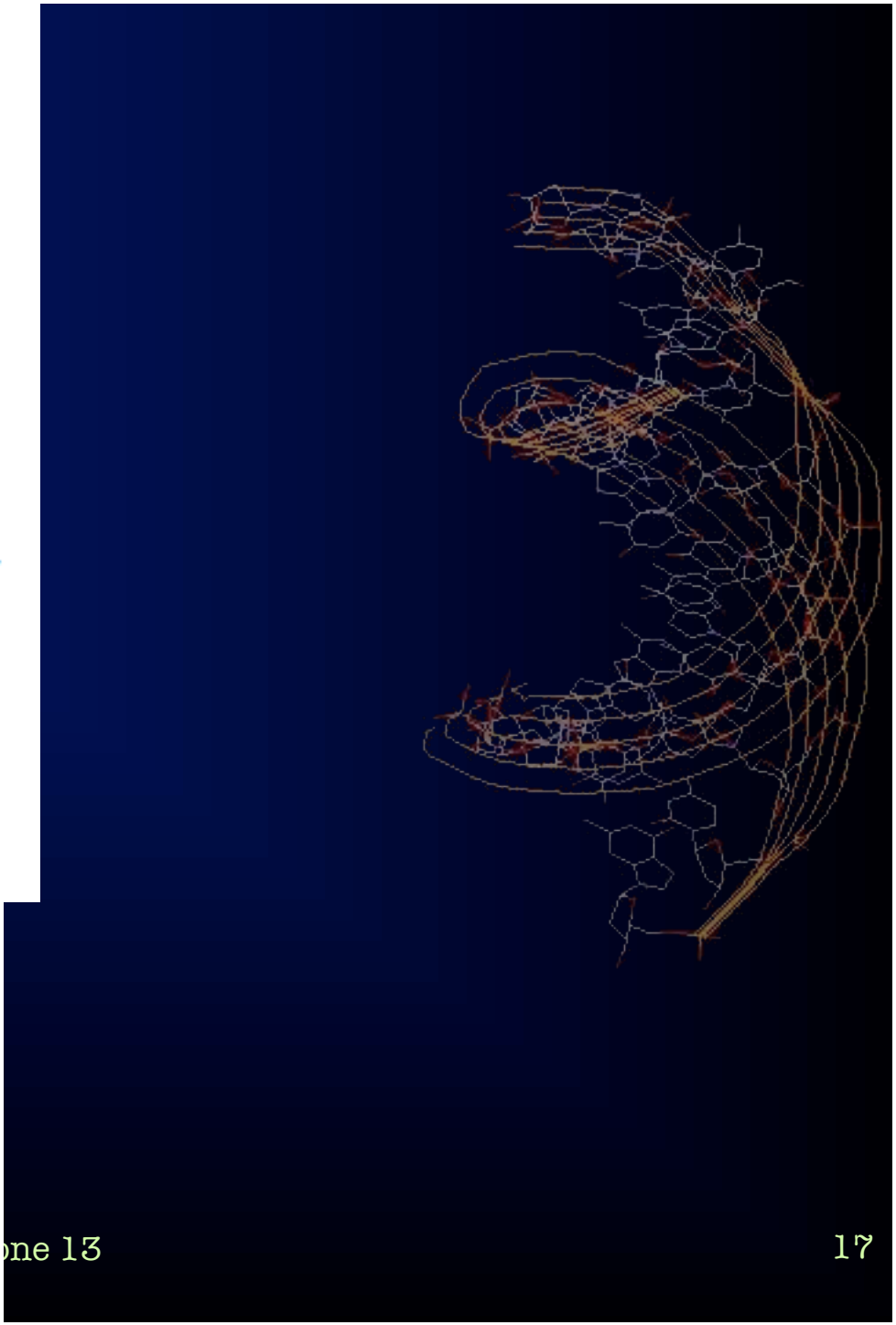
PROBLEMA

Caratteri extra

Come evitare che si verifichi questo problema ?
È sufficiente che gli utenti del programma non commettano mai errori quando forniscono delle informazioni !! :-)

Esiste però una soluzione anche per chi ha scarsa fiducia nel prossimo

```c
/* Clearing stdin of extra characters. */

#include <stdio.h>

void clear_kb(void);

main()
{
    int age;
     char name[20];

     /* Prompt for user's age. */

     puts("Enter your age.");
     scanf("%d", &age);

     /* Clear stdin of any extra characters. */

     clear_kb();

     /* Now prompt for user's name. */

     puts("Enter your first name.");
     scanf("%s", name);
     /* Display the data. */
     printf("Your age is %d.\n", age);
     printf("Your name is %s.\n", name);

     return 0;
}

void clear_kb(void)

/* Clears stdin of any waiting characters. */
{
     char junk[80];
     gets(junk);
}
```



Output su schermo

- Output di un carattere
- Output di una linea
- Output formattato

Output di un carattere

Le funzioni di output di carattere inviano un carattere alla volta allo stream in output.

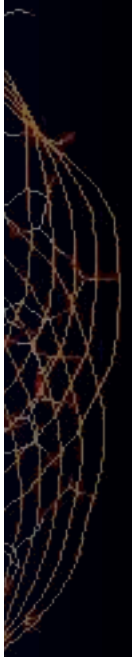
La funzione `putchar()`

```
int putchar(int c);
```

La funzione `putchar()` scrive il carattere immagazzinato nella variabile `c` nello stream `stdout` . La funzione ritorna il carattere stesso o `EOF` in caso di errore.

```
/* Demonstrates putchar(). */

#include <stdio.h>
main()
{
     int count;

     for (count = 14; count < 128; )
          putchar(count++);
:
:    return 0;
: }
```

```c
/* Using putchar() to display strings. */

#include <stdio.h>

#define MAXSTRING 80

char message[] = "Displayed with putchar().";
main()
{
     int count;

     for (count = 0; count < MAXSTRING; count++)
     {

          /* Look for the end of the string. When it's found, */
          /* write a newline character and exit the loop. */

          if (message[count] == '\0')
          {
               putchar('\n');
               break;
          }
          else

          /* If end of string not found, write the next character. */

               putchar(message[count]);
     }
     return 0;
 }
```



Output di stringhe

```
int puts(char *cp);
```

La funzione `puts()` visualizza l'intera stringa su `stdout` e ritorna un intero positivo in caso di successo o altrimenti `EOF`

```
/* Demonstrates puts(). */

#include <stdio.h>

/* Declare and initialize an array of pointers. */

char *messages[5] = { "This", "is", "a", "short", "message." };

main()
 {
     int x;

     for (x=0; x<5; x++)
         puts(messages[x]);

     puts("And this is the end!");

     return 0;
 }
```

Output formattato

Per l'output formattato su `stdout` si utilizza la funzione `printf()`

I/O su file

Bisogna creare uno stream associato ad uno specifico file

Streams di testo sono associati a file in modalità testo.
I file in modalità di testo consistono in una sequenza di linee.
Ogni linea contiene 0 o più caratteri e termina con uno o più caratteri che indicano la fine della linea. La massima lunghezza è di 255 caratteri.

Una linea di un file in modalità testo non equivale ad una stringa in C.
Non vi è alcun carattere di terminazione (\0).

Streams binari sono associati a file in modalità binaria. Tutti i dati sono scritti e letti senza alcuna modifica e senza alcuna separazione in Linee.

Gestione dei files

Il C permette di gestire i files ad "alto livello", tramite puntatori a strutture di tipo `FILE`.

Per poter accedere ad un file da un programma C la prima operazione da effettuare è l'associazione di uno stream al file stesso.

Tale operazione equivale all'apertura del file. La funzione di libreria `fopen()`, definita in `stdio.h` accetta in ingresso un nome file, esegue le necessarie trattative col sistema operativo e restituisce un puntatore a una struttura di tipo `FILE` che deve essere utilizzata nelle successive operazioni sul file stesso. In questo modo si ha accesso allo stream di bytes rappresentato dal file stesso.

```
FILE *fopen(char *name, char *mode);
```

La funzione `fopen()`

```
FILE *fopen(char *name, char *mode);
```

L'argomento `name` rappresenta il nome del file da aprire, può essere una stringa letterale racchiusa tra virgolette o un puntatore a stringa

`r`	Opens the file for reading. If the file doesn't exist, fopen() returns NULL.
`w`	Opens the file for writing. If a file of the specified name doesn't exist, it is created. If a file of the specified name does exist, it is deleted without warning, and a new, empty file is created.
`a`	Opens the file for appending. If a file of the specified name doesn't exist, it is created. If the file does exist, new data is appeneded to the end of the file.
`r+`	Opens the file for reading and writing. If a file of the specified name doesn't exist, it is created. If the file does exist, new data is added to the beginning of the file, overwriting existing data.
`w+`	Opens the file for reading and writing. If a file of the specified name doesn't exist, it is created. If the file does exist, it is overwritten.
`a+`	Opens a file for reading and appending. If a file of the specified name doesn't exist, it is created. If the file does exist, new data is appended to the end of the file.

Apertura dei files

Se il file che stro cercando di aprire in lettura non esiste (ad es. ho sbagliato a scrivere il nome) o non ho il permesso di aprire in scrittura il file di output la chiamata ad fopen restituisce un puntatore a NULL. In tal caso dovrei segnalare un errore ed uscire. Posso farlo ad es. ridefinendo una opportuna funzione MyFopen in analogia con quanto fatto per MyMalloc:

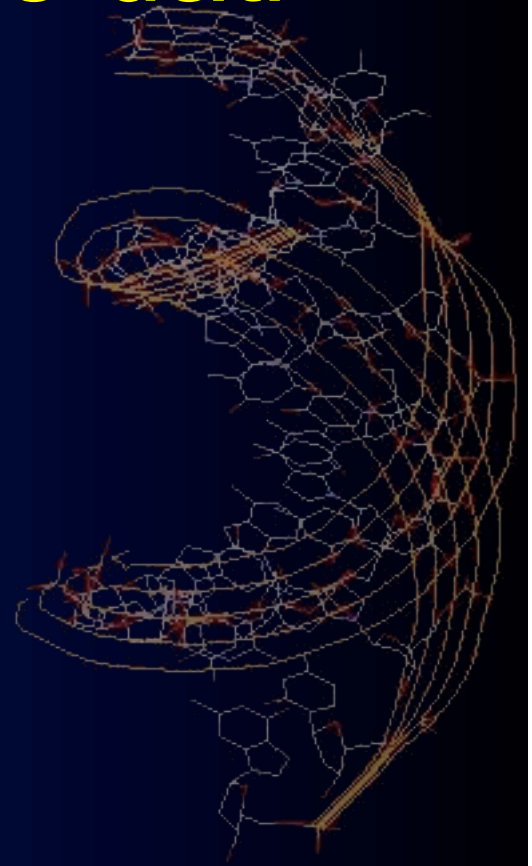
```
FILE *MyFopen(char *name, char *mode) {

FILE *f;
f = fopen(name,mode);
if(f==NULL) {
   fprintf(stderr,"Error opening file %s",name);
   exit(1);
}
return f;
}
```

Esempio

```
#include <stdio.h>

int main(){
 FILE *infile, *outfile, *logfile, *tempfile;

 infile   = MyFopen("dati.dat","r");
 /* infile è il puntatore ad un file di nome dati.dat
aperto in sola lettura ("r")*/
 outfile  = MyFopen("risultati.txt","w");
/* w sta per write (scrittura) */
 logfile  = MyFopen("registro.log","a");
/* a sta per append (aggiunta in coda)*/
 tempfile = MyFopen("tmpfile.dat", "w+");
/* w+ (o r+) sta per lettura-scrittura */
…
return 0;
}
```

Lettura e scrittura di file dati

1. I/O formattato (modalità testo)
2. I/O di carattere (modalità testo)
3. I/O diretto (modalità binaria)



fprintf e fscanf

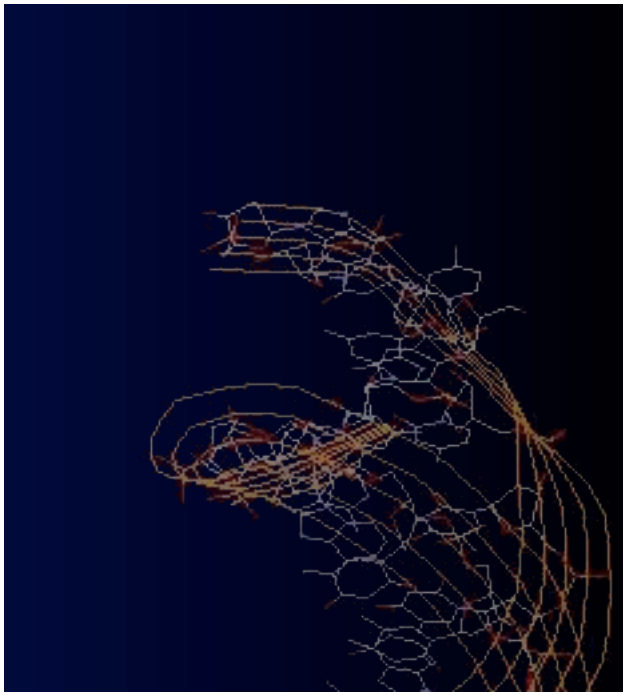
Le funzioni principali per l'I/O formattato da file sono :

```
int fprintf(FILE * fp, const char  *format,…);
int fscanf(FILE * fp, const char  *format,…);
```

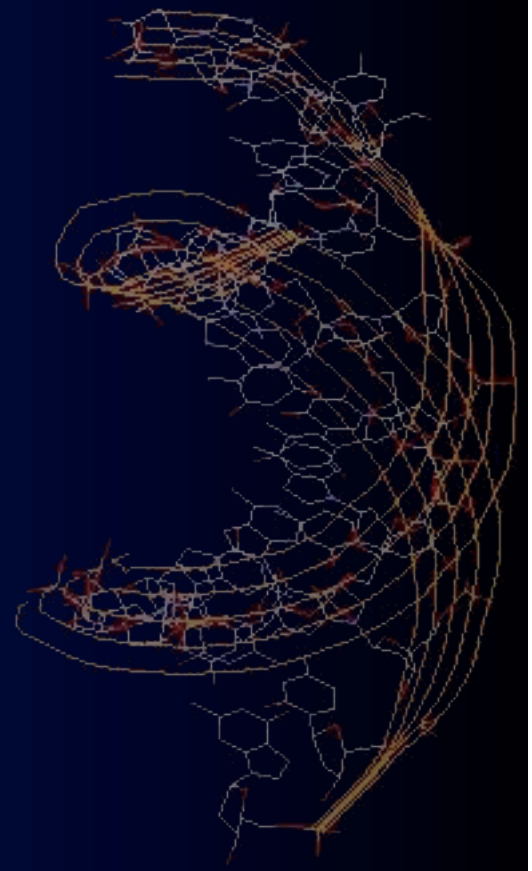
Il primo argomento è un puntatore a FILE (ad es. il risultato di una chiamata a fopen). Gli altri argomenti (la stringa di formato e la sequenza di argomenti opzionali) seguono le stesse identiche convenzioni usate per printf e scanf. In effetti il C mette a disposizione tre puntatori *stdin*, *stdout*, *stderr* per i file standard di I/O. Si ha allora :

```
printf("Hello");     ⇔   fprintf(stdout,"Hello");
scanf("%d",&i);      ⇔   fscanf(stdin,"%d",&i);
```

```c
/* Demonstrates the fprintf() function. */
#include <stdlib.h>
#include <stdio.h>

void clear_kb(void);

int main()
{
    FILE *fp;
     float data[5];
     int count;
     char filename[20]

     puts("Enter 5 floating-point numerical values.");

     for (count = 0; count < 5; count++)
         scanf("%f", &data[count]);

     /* Get the filename and open the file. First clear stdin */
     /* of any extra characters. */

     clear_kb();
     puts("Enter a name for the file.");
     gets(filename);

     if ( (fp = fopen(filename, "w")) == NULL)
     {
         fprintf(stderr, "Error opening file %s.", filename);
         exit(1);
     }

     /* Write the numerical data to the file and to stdout. */

     for (count = 0; count < 5; count++)
     {
         fprintf(fp, "\ndata[%d] = %f", count, data[count]);
         fprintf(stdout, "\ndata[%d] = %f", count, data[count]);
     }
     fclose(fp);
     printf("\n");
     return(0);
}
```



```c
/* Reading formatted file data with fscanf(). */
#include <stdlib.h>

#include <stdio.h>

int main()
{
    float f1, f2, f3, f4, f5;
    FILE *fp;

    if ( (fp = fopen("INPUT.TXT", "r")) == NULL)
    {
        fprintf(stderr, "Error opening file.\n");
        exit(1);
    }

    fscanf(fp, "%f %f %f %f %f", &f1, &f2, &f3, &f4, &f5);
    printf("The values are %f, %f, %f, %f, and %f\n.",
            f1, f2, f3, f4, f5);

    fclose(fp);
    return(0);
}
```



```
123.45      87.001
100.02
0.00456     1.0005
```

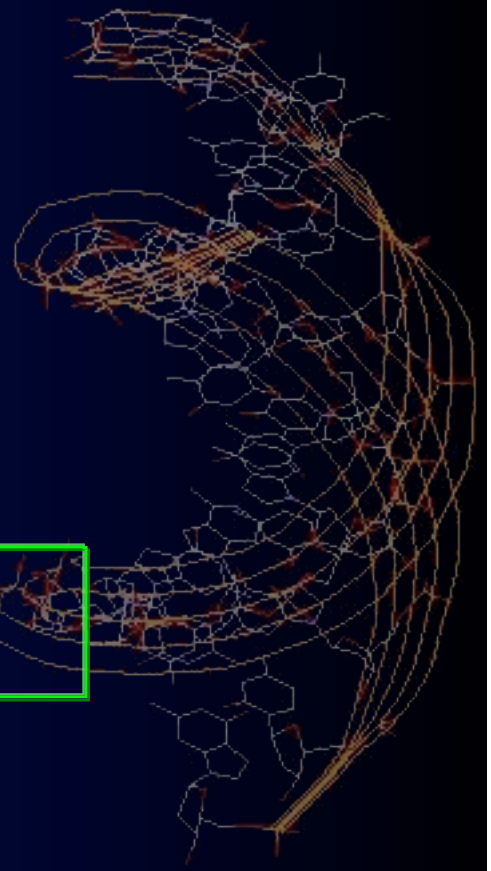
Altre funzioni di I/O su file

Funzioni per l'I/O di singoli caratteri:

```
int fgetc(FILE * fp);
int fputc(int c, FILE * fp);
```

Funzioni per l'I/O di stringhe:

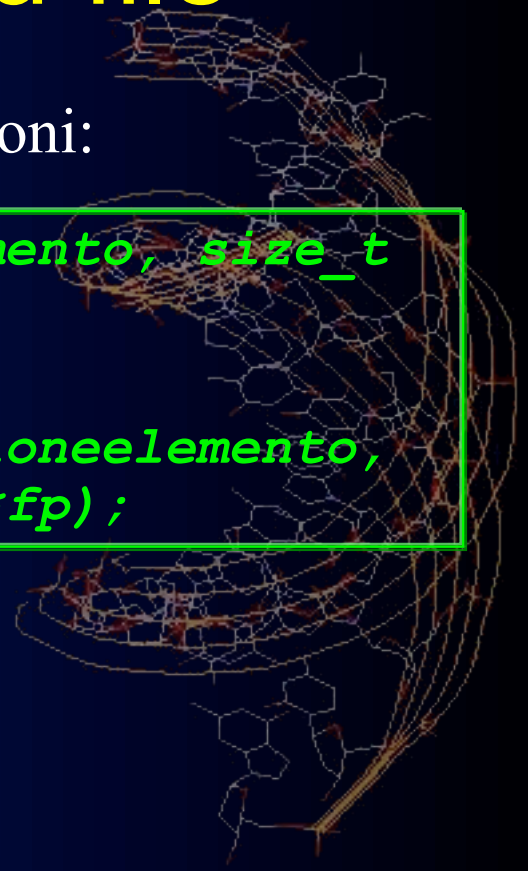
```
char *fgets(char *str, int n, FILE *fp);
char fputs(char *str, FILE *fp);
```



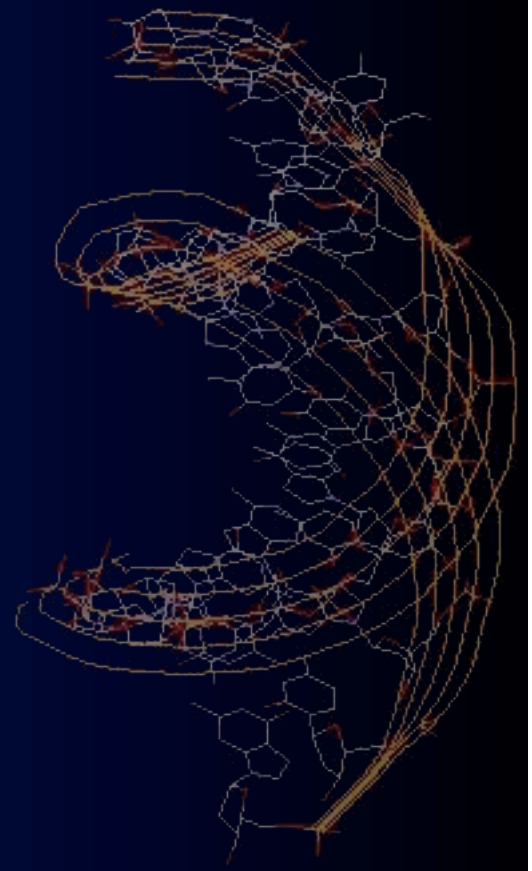
Altre funzioni di I/O su file

Funzioni per l'I/O diretto di bytes senza conversioni:

```
size_t fread(void *pt,size_t dimensioneelemento, size_t
            numeroelementi, FILE *fp);

size_t fwrite(const void *pt,size_t dimensioneelemento,
             size_t numeroelementi, FILE *fp);
```



```c
/* Direct file I/O with fwrite() and fread(). */
#include <stdlib.h>
#include <stdio.h>

#define SIZE 20

int main()
{
    int count, array1[SIZE], array2[SIZE];
    FILE *fp;

    /* Initialize array1[]. */

    for (count = 0; count < SIZE; count++)
        array1[count] = 2 * count;

    /* Open a binary mode file. */

    if ( (fp = fopen("direct.txt", "wb") ) == NULL)
    {
        fprintf(stderr, "Error opening file.");
```



```
        exit(1);
    }
    /* Save array1[] to the file. */

    if (fwrite(array1, sizeof(int), SIZE, fp) != SIZE)
    {
        fprintf(stderr, "Error writing to file.");
        exit(1);
    }

    fclose(fp);

    /* Now open the same file for reading in binary mode. */

    if ( (fp = fopen("direct.txt", "rb")) == NULL)
    {
        fprintf(stderr, "Error opening file.");
        exit(1);
    }

    /* Read the data into array2[]. */

    if (fread(array2, sizeof(int), SIZE, fp) != SIZE)
    {
        fprintf(stderr, "Error reading file.");
        exit(1);
    }

    fclose(fp);

    /* Now display both arrays to show they're the same. */

    for (count = 0; count < SIZE; count++)
        printf("%d\t%d\n", array1[count], array2[count]);
    return(0);
}
```