Linguaggio C:
Stringhe di caratteri e File

ESEMPIO: Lettura da tastiera e stampa a video di una sequenza di caratteri

```
#include <stdio.h>
main()
{
   int c;
   printf("Inserisci dei caratteri");
   printf("(CTRL+D per finire)");
   while ((c=getchar()) != EOF)
         putchar(c);
}
```

ESEMPIO: Visualizzazione dei caratteri ASCII

```
#include <stdio.h>
main()
{
      char c=' ';
      while (c <= '}')
      {
            putchar(c);
            c++;
      }
}
```

I/O DI CARATTERI: getchar e putchar

Le funzioni **getchar** e **putchar**, definite nell'header file *stdio.h*, servono per l'input e l'output di caratteri

int getchar()

restituisce un intero corrispondente al codice del tasto della tastiera premuto (o il valore predefinito EOF in caso di segnalazione di fine file)

int putchar(char c)

invia al video il carattere c e restituisce il corrispondente codice del tasto della tastiera se l'operazione di output ha avuto successo

Da tastiera il valore predefinito EOF può essere inserito con CTRL+D o con CTRL+Z, a seconda del calcolatore

ESEMPIO: Lettura da tastiera e visualizzazione del testo trasformato in maiuscolo

```
#include <stdio.h>
#include <ctype.h>
main()
{
   int ci, co;
   printf("Inserisci dei caratteri");
   printf("(CTRL+D per finire)");
   while ((ci=getchar()) != EOF)
   {
        if (isalpha(ci) && islower(ci))
             co=toupper(ci);
        else
             co=ci;
        putchar(co);
   }
}
```

MANIPOLAZIONE DI CARATTERI

Funzioni definite nell'header file *ctype.h* per operare sui caratteri prescindendo dalla loro codifica (fatta salva la consecutività dei caratteri numerici '0', '1', ... e dei caratteri minuscoli e maiuscoli 'a', 'b', ..., 'A', 'B', ...):

isalpha verifica se l'argomento è una lettera dell'alfabeto
isdigit " " " è una cifra
islower " " " è una lettera minuscola
isupper " " " è una lettera maiuscola
ispunct " " " è un carattere di punteggiatura
isspace " " " è un carattere di separazione
tolower restituisce la lettera minuscola equivalente al carattere maiuscolo passato come argomento
toupper restituisce la lettera maiuscola equivalente al carattere minuscolo passato come argomento

Sono tutte funzioni di tipo intero (simulazione del tipo logico), con argomento di tipo carattere

STRINGHE

Una variabile di tipo char può contenere un solo carattere; per trattare sequenze di caratteri il linguaggio C introduce le **stringhe**, intese come array di caratteri in cui è specificato il carattere di terminazione della stringa ('\0').

DICHIARAZIONE E ASSEGNAZIONE DI UNA STRINGA:

```
char giorno[7]="sabato";
```

's'	'a'	'b'	'a'	't'	'o'	'\0'

N.B.:

`char c='a'`; indica un unico carattere memorizzato in un'unica locazione di memoria

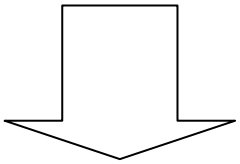
`char c[2]="a"`; indica una stringa, memorizzata in due locazioni di memoria consecutive contenenti i caratteri `'a'` e `'\0'`

LETTURA DI UNA STRINGA:

```
char giorno[100];
scanf("%s",giorno);
```

N.B. giorno è un array, per cui il secondo argomento passato alla scanf (**`giorno`**) è l'indirizzo del primo elemento dell'array giorno (**`&giorno[0]`**)

In **`giorno`** verrà memorizzata una sequenza di caratteri immessi in input racchiusa fra caratteri di separazione (spazio, tabulazione, andata a capo, ...)



- una stringa non può contenere degli spazi
- l'array che deve contenere la stringa deve essere opportunamente dimensionato

ESEMPIO: Calcolo della lunghezza di una stringa (n. di caratteri escluso '\0')

```
#include <stdio.h>
#define MAXL 100
int lungh_string(char stringa[]);
main()
{
   char str1[MAXL];
   printf("Inserisci una stringa\n");
   scanf("%s",str1);
   printf("Lunghezza stringa = %d",
          lungh_string(str1));
}
int lungh_string(char stringa[])
{
   int indice=0;
   while (stringa[indice])
        indice++;
   return(indice);
}
```

ESEMPIO: Copia di una stringa in un'altra

```
#include <stdio.h>
#define MAXL 100
int lungh_string(char s[]);
void copy_string(char s1[], char s2[]);
main()
{
   char str1[MAXL], str2[MAXL];
   printf("Inserisci una stringa");
   scanf("%s",str1);
   copy_string(str1,str2);
   printf("Stringa copiata:%s\n",str2);
}
void copy_string(char s1[], char s2[])
{
   int  i;
   for (i=0; i<=lungh_string(s1); i++)
        s2[i]=s1[i];
}
```

ESEMPIO: Ricerca di una stringa in un'altra

```
#include <stdio.h>
#define MAXL 100
int lungh_string(char s[]);
int search_str(char s1[], char s2[])
{
   int trovato=0, l1, l2, i, j;
   l1=lungh_string(s1);
   l2=lungh_string(s2);
   if (l1>=l2)
   {     i=0;
         while((!trovato) && (i<=l1-l2))
         {     trovato=1;
               for (j=0;j<l2;j++)
                     if (s2[j] != s1[i+j])
                           trovato=0;
               i++;}
   }
   return(trovato);
}
```

ESEMPIO: Ricerca di una stringa in un'altra (cont.)

```
main()
{
   char str1[MAXL], str2[MAXL];
   int  trovato;

   printf("Inserisci una stringa");
   scanf("%s",str1);
   printf("Inserisci una chiave");
   scanf("%s",str2);
   trovato=search_str(str1,str2);
   if (trovato)
         printf("Chiave trovata\n");
   else
         printf("Chiave non trovata\n");
}
```

MANIPOLAZIONE DI STRINGHE

Il linguaggio C standard prevede varie funzioni per la manipolazione di stringhe, contenute nell'header file *string.h*

strlen(s) restituisce la lunghezza della stringa s

strcpy(s1,s2) copia s2 in s1

strcmp(s1,s2) confronta le due stringhe e restituisce un intero negativo se s1 < s2, zero se s1 = s2, un intero positivo se s1 > s2

strcat(s1,s2) concatena s1 con s2, copiando i caratteri che costituiscono s2 in coda a quelli di s1, e restituendo un puntatore a s1

strstr(s1,s2) ricerca la stringa s2 nella stringa s1, restituendo un puntatore al punto di s1 dove inizia s2 (oppure NULL se s2 non è presente)

atof(s) converte la stringa s in un numero reale a doppia precisione, restituendo il risultato in un double

atoi(s) converte la stringa s in un numero intero

ESEMPIO: Stampa di dati su un file testo

```
#include <stdio.h>
main()
{
  FILE *fp;
  int i;

  if ((fp=fopen("file.dat","w")) == NULL)
    printf("Errore apertura file.dat\n");
  else
  {
    for (i=1; i<=10; i++)
      fprintf(fp, "%d\n",i);
    fclose(fp);
  }
}
```

ESEMPIO: Lettura di dati da un file testo

```
#include <stdio.h>
main()
{
  FILE *fp;
  int i, stato;

  if ((fp=fopen("file.dat","r")) == NULL)
    printf("Errore apertura file.dat\n");
  else
  {
    for (stato=fscanf(fp, "%d", &i);
         stato != EOF;
         stato=fscanf(fp, "%d", &i))
      printf("%d\n",i);
    fclose(fp);
  }
}
```

I FILE

Un *file testo* è una sequenza di componenti tutti dello stesso tipo caratterizzata dall'accesso sequenziale.

Per operare su un file testo, questo deve essere "aperto":

```
fp=fopen("nome_file","tipo_apertura");
```

con:

- `fp` puntatore a file: `FILE *fp;`
- `"nome_file"` stringa contenente il nome del file, incluso il percorso nel file system
- `"tipo_apertura"` stringa che specifica il tipo di operazioni da compiere sul file ("**r**"=solo lettura; "**w**"=scrittura; "**a**"=appendi; "**r+**"=lettura e scrittura)

Una volta terminato di operare su un file, questo deve essere "chiuso":

```
fclose(puntatore_a_file);
```

Il C standard prevede alcuni file testo standard:

- **`stdin`** = *standard input* (tastiera)
- **`stdout`** = *standard output* (video)
- **`stderr`** = *standard error* (video)

Questi sono sempre predisposti, per cui
non devono essere né aperti né chiusi

Per la lettura e la scrittura su file testo sono disponibili funzioni analoghe a quelle per i file testo standard:

```
fscanf(puntatore_a_file,"%d",&intero);
fprintf(puntatore_a_file,"%d",intero);
carattere=getc(puntatore_a_file);
putc(carattere, puntatore_a_file);
```

I file considerati sinora sono detti ***file testo***:
le informazioni contenute nel file, rappresentate in binario nel calcolatore, vengono convertite nel formato del loro tipo di dato e così memorizzate

Nei ***file binari***, invece, le informazioni contenute nel file sono memorizzate direttamente in binario.

File testo vs. **file binari**:

- i **file testo** possono essere facilmente manipolati mediante un qualsiasi *text editor*
- i **file testo** sono trasportabili
- i **file binari** occupano meno spazio
- i **file binari** sono più veloci da leggere e a scrivere

ESEMPIO: Stampa di dati su un file binario

```
#include <stdio.h>
main()
{
  FILE *fp;
  int i;

  if ((fp=fopen("file.dat","wb")) == NULL)
    printf("Errore apertura file.dat\n");
  else
  {
    for (i=1; i<=10; i++)
      fwrite(&i, sizeof(int), 1, fp);
    fclose(fp);
  }
}
```

ESEMPIO: Lettura di dati da un file binario

```
#include <stdio.h>
main()
{
  FILE *fp;
  int i, stato;

  if ((fp=fopen("file.dat","rb")) == NULL)
    printf("Errore apertura file.dat\n");
  else
  {
    for (stato=fread(&i, sizeof(int), 1, fp);
         stato != EOF;
         stato=fread(&i, sizeof(int), 1, fp))
      printf("%d\n",i);
    fclose(fp);
  }
}
```

Per operare su un file binario, questo deve essere "aperto" in modo binario:

```
fp=fopen("nome_file","tipo_apertura");
```

con `"tipo_apertura"`:

- `"rb"` = solo lettura
- "wb" = scrittura
- "ab" = appendi

Per la lettura e la scrittura su file binario sono disponibili le funzioni:

```
fread(ind_dati, sizeof(dati), num_dati, puntatore_a_file);

fwrite(ind_dati, sizeof(dati), num_dati, puntatore_a_file);
```

FINE