# Lezione 7: Funzioni e gestione dei File

Laboratorio di Elementi di Architettura e Sistemi Operativi

18 Aprile 2012

## Le funzioni

### Funzioni

- Un programma C consiste di una o più funzioni
  - Almeno `main()`
- Funzionamento identico ai metodi di Java
- Definizione delle funzioni
  - Prima della definizione di `main()`
  - Dopo la definizione di `main()` $\Rightarrow$ necessario premettere in testa al file il *prototipo* della funzione:
    - Nome
    - Argomenti
    - Tipo del valore di ritornato (`void` se la funzione non ritorna valori)
  - `return` per terminare e ritornare un valore
  - I prototipi si possono raccogliere in header file da includere con la direttiva `#include`

### Funzioni e prototipi: esempio

```
double f(int x)
{
  ...
}

int main ()
{
  ...
  z = f(y);
  ...
}
```

```
double f(int);

int main ()
{
  ...
  z = f(y);
  ...
}

double f(int x);
{
  ...
}
```

Dichiarazione

Definizione

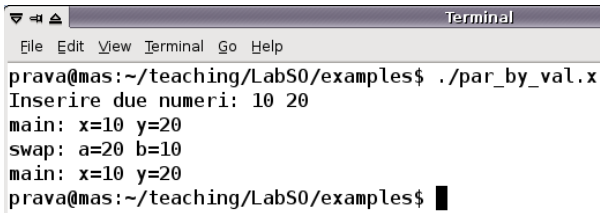
## Passaggio dei parametri

- In C, il passaggio dei parametri avviene per *valore*
  - **Significato:** Il valore dei parametri attuali viene copiato in variabili locali della funzione
- Implicazione
  - I parametri attuali non vengono MAI modificati dalle istruzioni della funzione
- *Eccezione:*
  - Stringhe e vettori sono passati per *indirizzo*:
    - possono quindi essere modificati dalle istruzioni della funzione.

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

void swap(int a,int b) {
  int tmp;
  tmp = a;
  a = b;
  b = tmp;
  printf("swap: a=%d
          b=%d\n",a,b);}
```

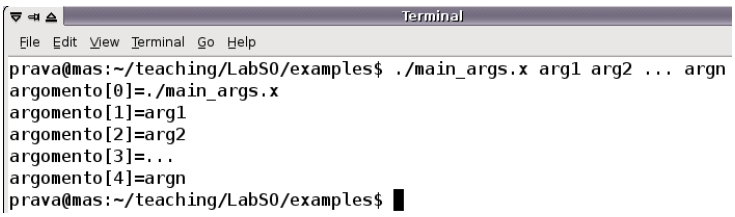
```
int main()
{
  int x,y;
  printf("Inserire due
          numeri: ");
  scanf("%d %d",&x,&y);
  printf("main: x=%d
          y=%d\n",x,y);
  swap(x,y);
  /* x e y NON VENGONO
     MODIFICATI */
  printf("main: x=%d
          y=%d\n",x,y);}
```




## Parametri del `main`

- Come in JAVA è possibile passare dei parametri direttamente dalla linea di comando
- Sintassi:
  - Numero di parametri: `int argc`
  - Vettore di parametri: `char *argv[]`
  - `argv[0]` = *nome del programma*
- il `main` può ritornare un valore `int` come exit status del programma
- Esempio:

```
int main(int argc, char *argv[]){
  int t;
  for(i = 0; i < argc; i++)
    printf("argomento[%d]=%s\n",i,argv[i]);
  return 0;
}
```




## Funzioni di libreria

- Il C prevede numerose funzioni predefinite per scopi diversi
- Particolarmente utili sono
  - Funzioni matematiche
  - Funzioni di utilità
- Definite in specifiche *librerie*
- Tutte descritte nel `man`

## Funzioni matematiche

- Utilizzabili con `#include <math.h>`

| *funzione* | *definizione* |
|---|---|
| `double sin (double x)` | seno |
| `double cos (double x)` | coseno |
| `double tan (double x)` | tangente |
| `double asin (double x)` | arcoseno |
| `double acos (double x)` | arcoseno |
| `double atan (double x)` | arcotangente |
| `double atan2 (double y, double x)` | atan ( y / x ) |
| `double sinh (double x)` | seno iperbolico |
| `double cosh (double x)` | coseno iperbolico |
| `double tanh (double x)` | tang. iperbolica |

| *funzione* | *definizione* |
|---|---|
| `double pow (double x, double y)` | $x^y$ |
| `double sqrt (double x)` | radice quadrata |
| `double log (double x)` | logaritmo naturale |
| `double log10 (double x)` | logaritmo decimale |
| `double exp (double x)` | $e^x$ |
| `double ceil (double x)` | ceiling(x) |
| `double floor (double x)` | floor(x) |
| `double fabs (double x)` | valore assoluto |
| `double fmod (double x, double y)` | modulo |

### Funzioni di utilità

- Classificazione caratteri: `#include <ctype.h>`

| *funzione* | *definizione* |
|---|---|
| `int isalnum (char c)` | Se c è lettera o cifra |
| `int isalpha (char c)` | Se c è lettera |
| `int isascii(char c)` | Se c è lettera o cifra |
| `int islower(char c)` | Se c è minuscola |
| `int isdigit (char c)` | Se c è cifra |
| `int isupper (char c)` | Se c è maiuscola |
| `int isspace(char c)` | Se c è spazio,tab,\n |
| `int iscntrl(char c)` | Se c è di controllo |
| `int isgraph(char c)` | Se c è stampabile, non spazio |
| `int isprint(char c)` | Se c è stampabile |
| `int ispunct(char c)` | Se c è di interpunzione |

- Funzioni matematiche intere: `#include <stdlib>`

| *funzione* | *definizione* |
|---|---|
| `int abs (int n)` | valore assoluto |
| `long labs (long n)` | valore assoluto |
| `div_t div (int numer, int denom)` | quoto e resto della divisione intera |
| `ldiv_t ldiv(long numer, long denom)` | quoto e resto della divisione intera |

- Stringhe: `#include <string.h>`

| *funzione* | *definizione* |
|---|---|
| `char* strcat (char* s1, char* s2);` | concatenazione di s1 e s2 |
| `char* strchr (char* s, int c);` | trova c dentro s |
| `int strcmp (char* s1, char* s2);` | confronto |
| `char* strcpy (char* s1, char* s2);` | copia s2 in s1 |
| `int strlen (char* s);` | lunghezza di s |
| `char* strncat (char* s1,char* s2,int n);` | concat. n car. max |
| `char* strncpy (char* s1,char* s2,int n);` | copia n car. max |
| `int strncmp(char* dest,char* src,int n);` | cfr. n car. max |

## I file

## File sequenziali

- Accesso tramite variabile di tipo *stream* (flusso di caratteri)
- Definiti in `stdio.h`
- **Definizione:** (fare attenzione al maiuscolo!)

  `FILE *<identificatore>;`

- Al momento dell'attivazione di un programma vengono automaticamente attivati (aperti) tre stream:

  `stdin` standard input (tastiera)
  `stdout` standard output (schermo)
  `stderr` standard error (schermo)

  sono direttamente utilizzabili nelle istruzioni per l'accesso ai file

## Apertura di un file

Per accedere ad un file è necessario aprirlo:

- `FILE* fopen(char* nomefile, char* modo);`

  collega un *file fisico* (su disco) ad un *file logico* (stream)

- `nomefile`: nome del file fisico (eventualmente con percorso assoluto o relativo rispetto alla cartella di lancio del programma)
- `modo`: il tipo di accesso al file

`"r"` sola lettura
`"w"` sola scrittura (cancella il file se esiste)
`"a"` append (aggiunge in coda ad un file o crea file se non esiste)
`"r+"` lettura/scrittura su file esistente
`"w+"` lettura/scrittura (cancella il file se esiste o crea file se non esiste)
`"a+"` lettura/scrittura-in-coda (crea file se non esiste)

- ritorna il puntatore allo stream in caso di successo, altrimenti ritorna `NULL`

## Chiusura di un file

Quando l'utilizzo di un file è terminato, è consigliabile chiuderlo:

- `int fclose(FILE* stream);`

  cancellazione della connessione di un file fisico con uno stream

- `stream`: uno stream aperto in precedenza con `fopen()`
- Valore di ritorno

`0` se la chiusura avviene correttamente
`EOF` in caso di errore

## Riapertura di un file

Chiusura e riapertura di uno stream:

- `FILE *freopen(char* nomefile, char* modo, FILE* stream);`

  chiude uno stream e lo riapre assegnandolo ad un file diverso

- `stream`: uno stream aperto in precedenza con `fopen()`
- `nomefile`, `modo` e valore di ritorno come in `fopen()`
- equivalente a `fclose(stream); stream = fope(nomefile, modo)`
- obbligatorio per riassegnare `stdin`, `stdout` e `stderr`, per i quali non si può usare `fopen`

## Esempio di apertura e chiusura di un file

```
...
FILE *fp; /* variabile di tipo stream */
...

fp = fopen("testo.dat","r");
/* apro 'testo.dat' in lettura*/
if (fp == NULL)
printf("Errore nell"apertura\n");
else {
  /* qui posso accedere a 'testo.dat' usando fp */

  ...

  fclose(fp);
}
...
```

## Lettura/scrittura a caratteri

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

- Legge un carattere alla volta dallo `stream`
- Restituisce il carattere letto o `EOF` in caso di fine file o errore
- *NOTA:* `getchar()` equivale a `getc(stdin)` e `fgetc(stdin)`

```
int putc (int c, FILE* stream);
int fputc (int c, FILE* stream);
```

- Scrive un carattere alla volta sullo `stream`
- Restituisce il carattere scritto o `EOF` in caso di errore
- *NOTA:* `putchar(c)` equivale a `putc(c, stdout)` e `fputc(c, stdout)`

### Lettura/scrittura a righe

```
char* fgets(char* s,int n, FILE* stream);
```

- `n` è la lunghezza del vettore
- Legge una riga dallo `stream` fermandosi al più dopo `n-1` caratteri
- L'eventuale `'\n'` NON viene eliminato (diverso da `gets`!)
- Restituisce il puntatore alla stringa o `NULL` in caso di fine file o errore

```
int fputs(char* s, FILE* stream);
```

- Scrive la stringa `s` sullo `stream` senza aggiungere `'\n'` (diverso da `puts`!)
- Restituisce l'ultimo carattere scritto, oppure `EOF` in caso di errore

### Lettura/scrittura formattata

```
int fscanf(FILE* stream, char *formato, ... );
```

- Come `scanf()`, con un parametro addizionale che rappresenta uno `stream`
- Restituisce il numero di campi convertiti, oppure `EOF` in caso di fine file

```
int fprintf(FILE* stream, char *formato, ... );
```

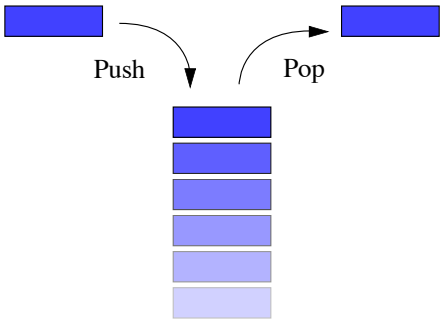
- Come `printf()`, con un parametro addizionale che rappresenta uno `stream`
- Restituisce il numero di byte scritti, oppure `EOF` in caso di errore

## Esercizio 3

1. Realizzare un insieme di funzioni per gestire una pila (stack) di valori floating point. In particolare, si implementino le operazioni di inserimento (`pop`) ed estrazione (`push`) di un valore nella pila. Si assuma che la pila possa contenere al massimo 10 valori.
2. Utilizzare le funzioni di gestione della pila per implementare una calcolatrice che esegua le quattro operazioni aritmetiche (`+`, `-`, `*`, `/`) usando la notazione polacca inversa (RPN). La calcolatrice riceve le operazioni da eseguire dalla tastiera, leggendo un operatore o operando per ogni riga. Dopo aver letto un numero o un operatore (ed aver eseguito l'operazione) mostra i valori contenuti nello stack. Il programma termina quando l'utente inserisce `q`.

## Le pile

- Una pila (o stack) è un tipo di dato astratto, in cui i dati vengono estratti in ordine rigorosamente inverso rispetto a quello in cui sono stati inseriti.
- Una pila si manipola tramite due operazioni:
  - `push`: inserimento di un nuovo dato in cima alla pila
  - `pop`: estrazione del dato in cima alla pila corrente




- *Implementazione:* un vettore per contenere gli elementi, ed un indice all'ultimo elemento inserito.

## La notazione Polacca inversa

$((a + b) * c)/(d - e)$ ← Notazione infissa

$a\ b\ +\ c\ *\ d\ e\ -\ /$ ← Notazione postfissa o Polacca inversa

- ogni operatore segue i suoi operandi
- non necessita di parentesi
- non c'è precedenza tra gli operatori
- chiamata così per analogia con la *notazione polacca* (dove gli operatori precedono gli operandi), inventata dal matematico polacco Jan Łukasiewicz.

## Implementazione della notazione polacca inversa

Leggo l'operazione da sinistra verso destra:

- quando trovo un operando lo pongo in cima allo stack
- quando trovo un operatore, estraggo il numero corretto di operandi dallo stack
- applico l'operatore
- il risultato viene posto sulla cima dello stack

**Esempio:**
Operazione: `a b + c * d e - /`
Evoluzione del contenuto dello stack:

```
a
a b
(a+b)
(a+b) c
(a+b)*c
(a+b)*c d
(a+b)*c d e
(a+b)*c (d-e)
(a+b)*c/(d-e)
```