



Laboratorio di Reti di Calcolatori

Luca Guarnera, Ph.D.

<https://www.dmi.unict.it/lguarnera/>
luca.guarnera@unict.it

Università di Catania
 Dipartimento di Matematica e Informatica
 Corso di Laurea Triennale in Informatica




 Reti di Calcolatori
 LABORATORIO – 3 CFU

1


Alcune Info ...

- Stanza 311 (secondo piano, Blocco 1)
- Ricevimento:
 - Martedì dalle 11 alle 13
 - Oppure fissiamo un appuntamento via email: luca.guarnera@unict.it



Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori

 Reti di Calcolatori
 LABORATORIO – 3 CFU

2

Laboratorio di Reti di Calcolatori – Programma

- I sistemi virtualizzati: aspetti teorici e implementazioni
- Creazione di una VM linux based
- Configurazione di una VM e collegamento in una LAN privata
- Configurazione di una interfaccia di rete con IPv4 e IPv6.
- Configurazione delle tabelle di routing.
- Configurazione di una rete con LAN differenti connesse da router.
- Uso dei socket in C.
- Esempio di un sistema Client server con UDP
- Esempio di un sistema multiserver con TCP

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori

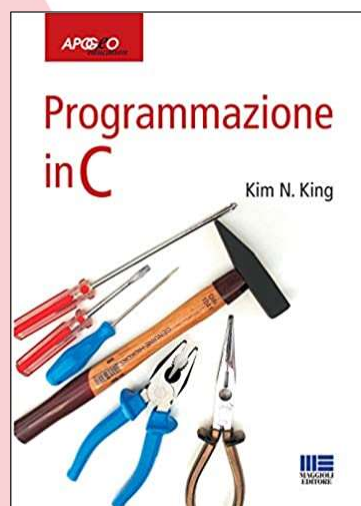


Reti di Calcolatori
LABORATORIO – 3 CFU

3

Notare inoltre che ...

ATTENZIONE: queste slide non sostituiscono il libro di testo o fonti di qualsiasi altro tipo!!



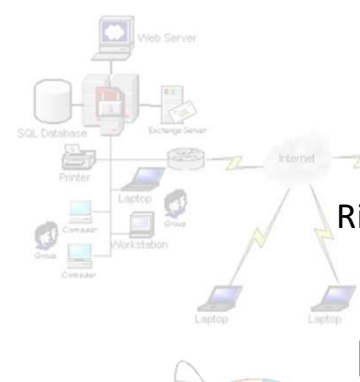
Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

4




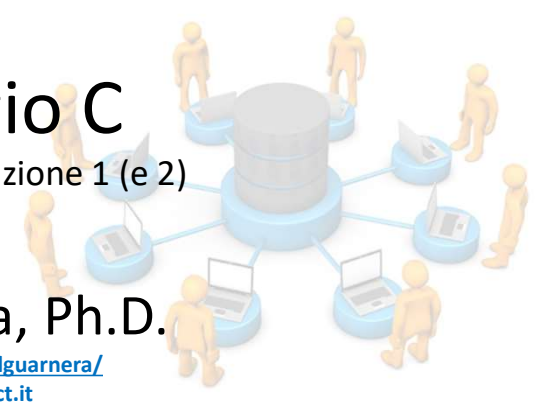
Linguaggio C

Richiami di Programmazione 1 (e 2)

Luca Guarnera, Ph.D.


<https://www.dmi.unict.it/lguarnera/>
luca.guarnera@unict.it

Università di Catania
 Dipartimento di Matematica e Informatica
 Corso di Laurea Triennale in Informatica

Reti di Calcolatori
 LABORATORIO – 3 CFU

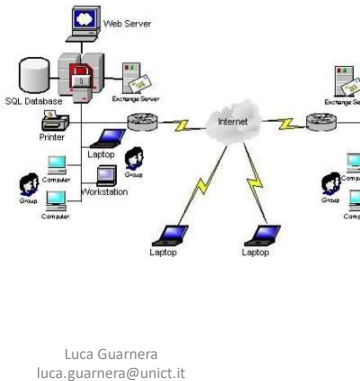


5



Università di Catania

Prima di Iniziare...

C Vs C++

Luca Guarnera
luca.guarnera@unict.it

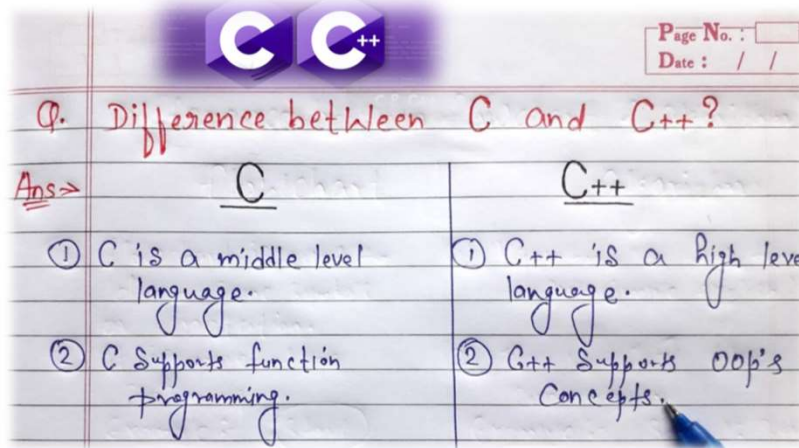
Laboratorio di Reti di Calcolatori

Reti di Calcolatori
 LABORATORIO – 3 CFU

6

C Vs C++

C nacque nel 1978, con la pubblicazione di *The C Programming Language*, di Brian Kernighan e Dennis Ritchie. È un linguaggio general purpose per la programmazione strutturata



Bjarne Stroustrup sviluppò **C++** agli inizi degli anni ottanta come un C con caratteristiche di OOP. In generale, la maggioranza dei programmi C sono anche programmi C++, ma non il contrario.

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

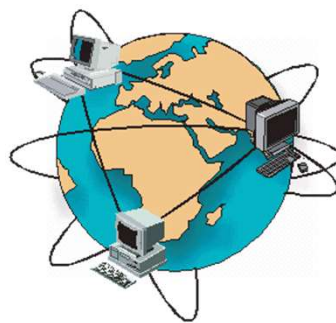
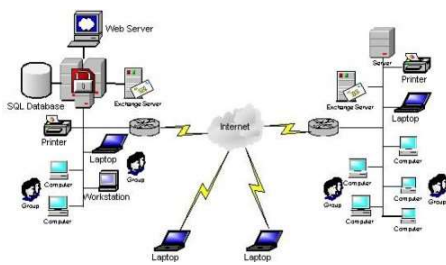
7



Università
di Catania

Il linguaggio C

Richiami Base



Luca Guarnera
luca.guarnera@unicat.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

8

Il primo programma in C

Ritorna il numero di caratteri stampati. La stringa di formato ha 2 tipi di oggetti:

Stampa sullo stdout la lista di argomenti conformemente alla stringa di formato specificata.

```
#include<stdio.h>

int main(int argc, char **argv){
    printf("Hello word!");
    return 0;
}
```

Quanti (*argc*) argomenti abbiamo passato da riga di comando e i relative valori (*argv*)

int printf(char *format, arg list ...)

Estensione
file C
NomeFile.c

FORMATO (%)	TIPO	RISULTATO
c	char	singolo carattere
i,d	int	numero decimale
o	int	numero ottale
x,X	int	numero esadecimale (notazione maiuscola o minuscola)
u	int	intero senza segno
s	char *	stampa una stringa terminata con \0
f	double/float	formato -m.ddd...

COMPILIAMO ED ESEGUIAMO

Con **DevC++**

F9 → COMPILA; F10 → ESEGUE; F11 → COMPILA ed ESEGUE

Da riga di comando

1. gcc programma.c -o output COMPILA
2. ./output ESEGUE

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

9

Printf()

Tra i **doppi apici** il simbolo percentuale **%** specifica che il **carattere che lo segue definisce il formato di stampa della variabile**

```
/* Example for printf() */
#include <stdio.h>

int main(){
    int a = -10;
    unsigned int b = 50;
    char c = 'a';
    printf ("%d, %u, %c \n", a, b, c);

    //printf più dettagliata
    printf ("Intero: %i, Intero senza segno %u, Carattere %c \n", a, b, c);
    return 0;
}
```

OGNI TIPO RICHIEDE IL SUO SPECIFICATORE DI FORMATO IN PRINTF:

- **%c**: caratteri;
- **%d**: interi;
- **%f**: numeri in virgola mobile;
- **%e**: numeri in notazione scientifica (es. 2e-5);
- **%u**: unsigned int;
- **%li**: long int;
- **%lu**: unsigned long int;
- **%Le**: long double in notazione scientifica;
- **%s**: stringhe.

Si possono stampare più variabili con una sola **printf**, indicando prima tra doppi apici il formato in cui si desiderano le visualizzazioni e successivamente i nomi delle rispettive variabili

All'interno dei doppi apici possiamo scrivere anche i commenti

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

10

Tipi di dati

```
#include <stdio.h>
#include <limits.h>
#include <float.h>
int main() {
    printf("TYPE\t\t\tSIZE\t\tMIN\t\t\tMAX\n");
    printf("-----\n");
    printf("char\t\t\t%lu\t%d\t\t\t%d\n", sizeof(char), CHAR_MIN, CHAR_MAX);
    printf("unsigned char\t\t%lu\t%d\t\t\t%d\n", sizeof(unsigned char), 0, UCHAR_MAX);
    printf("-----\n");
    printf("short int\t\t%lu\t%d\t\t\t%u\n", sizeof(short int), SHRT_MIN, SHRT_MAX);
    printf("unsigned short int\t\t%lu\t%d\t\t\t%u\n", sizeof(unsigned short int), 0, USHRT_MAX);
    printf("int\t\t\t%lu\t%d\t\t\t%d\n", sizeof(int), INT_MIN, INT_MAX);
    printf("unsigned int\t\t%lu\t%d\t\t\t%u\n", sizeof(unsigned int), 0, UINT_MAX);
    printf("long int\t\t%lu\t%ld\t\t%ld\n", sizeof(long int), LONG_MIN, LONG_MAX);
    printf("unsigned long int\t\t%lu\t%d\t\t\t%lu\n", sizeof(unsigned long int), 0, ULONG_MAX);
    printf("-----\n");
    printf("float\t\t\t%lu\t%e\t\t\t%e\n", sizeof(float), FLT_MIN, FLT_MAX);
    printf("double\t\t\t%lu\t%e\t\t\t%e\n", sizeof(double), DBL_MIN, DBL_MAX);
    printf("long double\t\t%lu\t%le\t\t\t%le\n", sizeof(long double), LDBL_MIN, LDBL_MAX);
    printf("-----\n");
}
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO - 3 CFU

11

Tipi di dati

TYPE	SIZE	MIN	MAX
char	1	-128	127
unsigned char	1	0	255
short int	2	-32768	32767
unsigned short int	2	0	65535
int	4	-2147483648	2147483647
unsigned int	4	0	4294967295
long int	8	-9223372036854775808	9223372036854775807
unsigned long int	8	0	18446744073709551615
float	4	1.175494e-38	3.402823e+38
double	8	2.225074e-308	1.797693e+308
long double	16	3.362103e-4932	1.189731e+4932

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO - 3 CFU

12

scanf()

- La funzione scanf() permette di gestire l'input da tastiera.
- La dichiarazione della funzione si trova in stdio.h; il suo prototipo è

```
int scanf("stringa di formato", ...);
```

Specificatori di **formato** che indicano al programma come effettuare la conversione dei caratteri inseriti nel buffer di input

Indirizzi delle variabili (ovvero i **puntatori a tali variabili**).

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



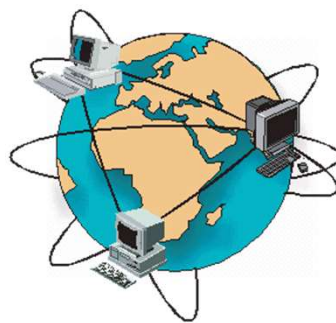
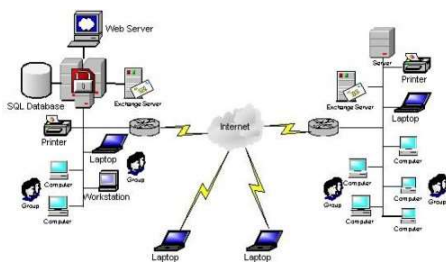
Reti di Calcolatori
LABORATORIO – 3 CFU

13



Università
di Catania

if-then-else, switch-case, espressioni condizionali
While, for, do-while, Break, continue & goto
Array & Matrici



Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

14

If & switch

```
if (cond)
    istr1
else
    istr2
```

```
switch (selettore)
{
    case etichetta_1: istruzione_1; break;
    case etichetta_2: istruzione_2; break;
    ...
    case etichetta_n: istruzione_n; break;
    default: istruzione_default; // opzionale
}
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

15

while, for & do-while

```
while (condizione_ciclo_2)
{
    istruzioni
}
```

```
for (inizializzazione; condizione_ciclo; passo)
{
    istruzioni
}
```

```
do
{
    istruzioni
} while (condizione_ciclo);
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

16

break, continue & goto

istruzioni di salto: break

```
while (test1)
{...
  if(test2)
    break;
  ...
}
```

```
while (test1)
{...
  if(test2)
    continue;
  ...
}
```

```
while (test1)
{...
  if(test2)
    goto L;
  ...
}
...
L: istruzione
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori

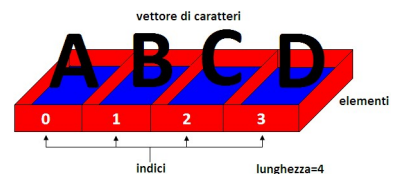
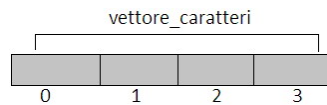


Reti di Calcolatori
LABORATORIO – 3 CFU

17

Array e Matrici

```
char vettore_caratteri[4];
// Inizializzazione dell'array
for(i=0; i<4; i++)
{
  printf("Ins. intero positivo: ");
  scanf("%c", &vettore_caratteri[i]);
}
```



```
char frase[]="Analisi, requisiti";
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	n	a	l	i	s	i	,		r	e	q	u	i	s	i	t	i		\0

(\0 fine stringa)

```
tipo nome[dimensione1][dimensione2]...[dimensioneN];
```

```
int mat[4][3];
```

mat che abbiamo dichiarato contiene 4 righe e 3 colonne

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

18

string.h

Alcune funzioni:

- **strcpy** copia stringa2 su stringa 1:
- **strncpy** copia i primi n caratteri di stringa2 in stringa1:
- **strcat** concatena stringa2 a stringa1:
- **strcmp** serve a confrontare stringa2 con stringa1: se le due stringhe risultano uguali viene restituito 0, se stringa1 è maggiore di stringa2 viene restituito un valore positivo, altrimenti un valore negativo.
-

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

19

struct

```
struct nomeStruttura
{...
    tipoMembro nomeMembro1;
    tipoMembro nomeMembro2;
    ...
    tipoMembro nomeMembroN;
};
```

Gli elementi della struttura sono detti **membri**; essi sono identificati da un **nome**, **nomeMembro**, e da un **tipo**, **tipoMembro**.

Una volta definita una struttura, nomeStruttura diviene un nuovo tipo a tutti gli effetti. Si possono allora definire variabili il cui tipo è nomeStruttura:

struct nomeStruttura nomeVariabile;

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

20

Esempio


```
#include<stdio.h>
struct automobile{
    char marca[30];
    char modello[30];
    int anno;
};
int main(int argc, char **argv){
    struct automobile a[2];
    int i;
    for(i=0; i<2; i++){
        printf("Inserisci la marca dell'automobile\n");
        scanf("%s", &a[i].marca);
        printf("Inserisci il modello\n");
        scanf("%s", &a[i].modello);
        printf("Anno di immatricolazione\n");
        scanf("%d", &a[i].anno);
    }
    for(i=0; i<2; i++){
        printf("[%d] Marca = %s \n", i, a[i].marca);
        printf("[%d] Modello %s \n", i, a[i].modello);
        printf("[%d] Anno di immatricolazione %d \n", i, a[i].anno);
    }
    return 0;
}
```

Luca Guarnera
luca.guarnera@unict.it

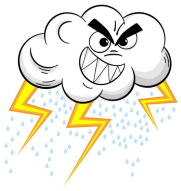
La

alcolatori
ORIO – 3 CFU


21




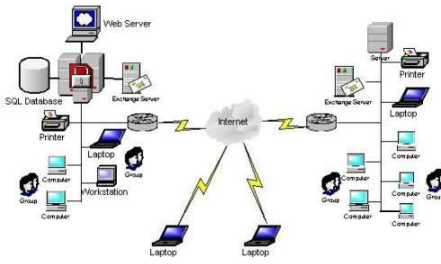


**Università
di Catania**



I Puntatori in C






Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

22

Puntatori

- A ogni variabile corrisponde un nome, una locazione di memoria e l'indirizzo della locazione di memoria.



Viene assegnato il valore costante 5 alla variabile di tipo intero a.

- L'operatore **&** restituisce **l'indirizzo di memoria di una variabile**. Per esempio, l'espressione **&a** è un'espressione il cui valore è l'indirizzo della variabile a.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

23

Puntatori

Un **indirizzo** può essere assegnato solo a una speciale categoria di variabili dette **puntatori**, le quali sono appunto variabili abilitate a contenere un indirizzo.

La sintassi di definizione è

tipoBase *var;

dove varPunt è definita come variabile di tipo “**puntatore a tipoBase**”; in sostanza varPunt è creata per poter mantenere l'indirizzo di variabili di tipo **tipoBase**, che è uno dei **tipi fondamentali**: **char**, **int**, **float** e **double**

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

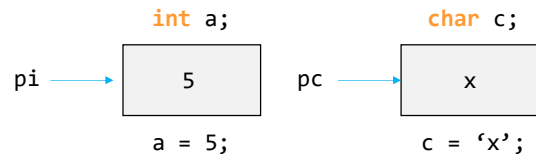
24

Puntatori

```
#include<stdio.h>

int main(int argc, char **argv){
    int a;
    char c;
    int *pi;
    char *pc;
    pi = &a;
    pc = &c;
    return 0;
}
```

Si ha che **pi** è una variabile puntatore a **int**, e **pc** è una variabile di tipo puntatore a **char**. Le variabili **pi** e **pc** sono inizializzate rispettivamente con l'indirizzo di **a** e di **c**.



```
printf("a = %d c = %c", a, c);
printf("a = %d c = %c", *pi, *pc);
```

Output

```
a = 5 c = x
a = 5 c = x
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

25

Altri esempi...

```
#include<stdio.h>

int main(int argc, char **argv){
    char c1, c2;
    char *pc;
    c1 = 'a';
    c2 = 'b';
    printf("c1 = %c c2 = %c", c1, c2);
    pc = &c1; //pc contiene l'indirizzo di c1
    c2 = *pc; //c2 contiene il carattere 'a'
    printf("c1 = %c c2 = %c", c1, c2);
    return 0;
}
```

Dopo l'assegnazione `pc=&c1`; i nomi `c1` e `*pc` sono perfettamente equivalenti (sono alias). L'effetto ottenuto con l'assegnazione `c2=*pc`; si sarebbe ottenuto, in modo equivalente, con l'assegnazione `c2=c1`;

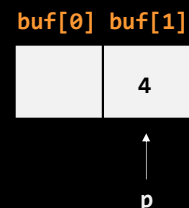
Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori

Con il puntatore a intero `p` e l'operatore `*` si è modificato il contenuto della locazione di memoria `buf[1]`, questa volta preposta a contenere un valore di tipo `int`.

```
#include<stdio.h>

int main(int argc, char **argv){
    int buf[2];
    int *p;
    ...
    p = &buf[1];
    *p = 4;
    return 0;
}
```

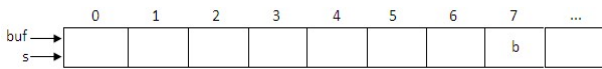


Reti di Calcolatori
LABORATORIO – 3 CFU

26

Array e Puntatori

Gli array e i puntatori in C sono strettamente correlati. Il nome di un array può essere usato come un puntatore al suo primo elemento. Considerando, per esempio:



```
#include<stdio.h>

int main(int argc, char **argv){
    char buf[100];
    char *s;
    s = &buf[0]; /*oppure s = buf; */
    buf[7] = 'a';
    printf("buf[7] = %c\n", buf[7]);
    *(s+7) = 'b';
    printf("buf[7] = %c\n", buf[7]);
    return 0;
}
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

27

Array e Puntatori

```
#include<stdio.h>

int main(int argc, char **argv){
    char buf[2];
    for(int i=0; i<2; i++)
        buf[i] = 'K';
    return 0;
}
```

```
#include<stdio.h>

int main(int argc, char **argv){
    char buf[2];
    char *s;
    s = buf;
    for(int i=0; i<2; i++)
        *s++ = 'K';
    return 0;
}
```

Nell'aritmetica dei puntatori quello che conta è il tipo base. Incrementare di 1 un puntatore significa far saltare il puntatore alla prossima locazione corrispondente a un elemento di memoria il cui tipo coincide con quello base.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

28

Passaggio di parametri per indirizzo

In C **non è possibile passare un array a una funzione**. Eppure esistono molti casi in cui è necessario non solo passare un array ma anche restituire una struttura dati più complessa della semplice variabile char o int.

Per risolvere questo problema bisogna passare per valore l'indirizzo della variabile –array o altro- che si vuol leggere o modificare tramite la funzione.

Passare un indirizzo a una funzione significa renderle nota la locazione dell'oggetto corrispondente all'indirizzo.

In tale maniera le istruzioni all'interno di una funzione possono modificare il contenuto della variabile il cui indirizzo è stato passato alla funzione. Questo meccanismo è noto con il nome di **passaggio di parametri per indirizzo**.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

29

```
#include<stdio.h>
void scambia(int a, int b);
main()
{
    int x=8, y=16;
    printf("Prima dello scambio\n");
    printf("x=%d, y=%d\n", x, y);
    scambia(x, y);
    printf("Dopo lo scambio\n");
    printf("x=%d, y=%d\n", x, y);
}

void scambia(int a, int b)
{
    int tmp;
    tmp=a;
    a=b;
    b=tmp;
}
```

```
Prima dello scambio
x=8, y=16
Dopo lo scambio
x=8, y=16
```

```
#include<stdio.h>
void scambia(int *a, int *b);
main()
{
    int x=8, y=16;
    printf("Prima dello scambio\n");
    printf("x=%d, y=%d\n", x, y);
    scambia(&x, &y);
    printf("Dopo lo scambio\n");
    printf("x=%d, y=%d\n", x, y);
}

void scambia(int *a, int *b)
{
    int tmp;
    tmp=*a;
    *a=*b;
    *b=tmp;
}
```

```
Prima dello scambio
x=8, y=16
Dopo lo scambio
x=16, y=8
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

30

Strutture e Puntatori

```
#include<stdio.h>
struct data{
    int giorno;
    char *mese;
    int anno;
};
int main(int argc, char **argv){
    struct data *pd, oggi;
    pd = &oggi;
    (*pd).giorno = 31;
    (*pd).mese = "Gennaio";
    (*pd).anno = 2023;
    ...
    return 0;
}
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

Attraverso il puntatore **pd** si possono raggiungere i membri della variabile strutturata **oggi**. Le parentesi tonde che circoscrivono ***pd** sono necessarie perchè l'operatore "." ha priorità maggiore rispetto all'operatore "*".

31

Strutture e Puntatori

```
#include<stdio.h>
struct data{
    int giorno;
    char *mese;
    int anno;
};
int main(int argc, char **argv){
    struct data *pd, oggi;
    pd = &oggi;
    pd->giorno = 31;
    pd->mese = "Gennaio";
    pd->anno = 2023;
    ...
    return 0;
}
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

Poichè in C si accede frequentemente a una variabile strutturata tramite puntatore, per evitare costrutti sintattici laboriosi è stato introdotto l'operatore freccia -> per accedere direttamente ai membri di una variabile strutturata puntata da un puntatore.

32

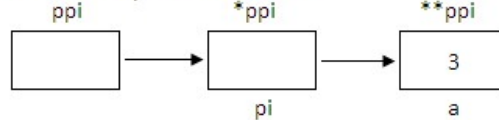
Tipi derivati composti tramite puntatore

Il puntatore può essere abbinato a qualsiasi tipo, compreso se stesso. In effetti il **puntatore a puntatore**, o puntatore di puntatore, è di tipo derivato composto tra i più usati in C. Per esempio: **char **pp**; è la dichiarazione di un puntatore di puntatore a carattere. Un puntatore di puntatore è una variabile abilitata a mantenere l'indirizzo di una variabile puntatore. Per esempio nel programma:

```
#include<stdio.h>
int main(int argc, char **argv){
    int **ppi;
    int a = 3;
    int *pi;
    pi = &a;
    ppi = &pi;
    printf("%d\n", **ppi);
    return 0;
}
```

La variabile **pi** contiene l'indirizzo della variabile intera **a**, e **ppi** contiene l'indirizzo di **pi**. Conseguentemente ***ppi** corrisponde al contenuto di **pi**, cioè all'indirizzo di **a**, e ****ppi** corrisponde al contenuto di **a**. Infatti l'istruzione **printf("%d", **ppi);** visualizza il numero 3:

Puntatore di puntatore a intero



Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



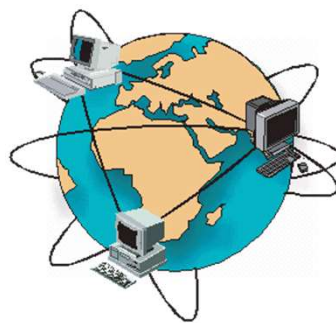
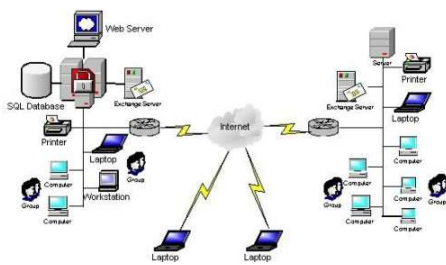
Reti di Calcolatori
LABORATORIO – 3 CFU

33



Università
di Catania

File



Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

34

Gestione Dei File

- Per poter mantenere disponibili i dati tra le diverse esecuzioni di un programma (*persistenza* dei dati) è necessario poterli *archiviare su memoria di massa*.
 - Dischi
 - Nastri
 - cd
 - ...
- I file possono essere manipolati (aperti, letti, scritti...) all'interno di programmi C

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

35

File In C

- Per gestire i file, il C definisce il tipo **FILE**.
- **FILE** è una struttura definita nello header standard **stdio.h**
- Le strutture **FILE** non sono *mai* gestite direttamente dall'utente, ma solo dalle funzioni della libreria standard **stdio**.
- L'utente definisce e usa, nei suoi programmi, solo *puntatori a FILE*.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

36

Come rappresentiamo i dati?

Rappresentazione interna

- Più sintetica
- Non c'è bisogno di effettuare conversioni ad ogni lettura/scrittura
- Si può vedere il contenuto del file solo con un programma che conosce l'organizzazione dei dati

Rappresentazione esterna

- Meno sintetica
- Necessità di conversione ad ogni lettura/scrittura
- Si può verificare il contenuto del file con un semplice editor di testo

FILE BINARI

FILE di TESTO

Luca Guarnera luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori

37

File In C: Apertura

Per aprire un file si usa la funzione:

FILE* fopen(char fname[], char modo[])

Questa funzione apre il file di nome fname nel modo specificato, e restituisce un puntatore a FILE (che punta a una nuova struttura FILE appositamente creata).

- **NB:** il nome del file (in particolare il path) è indicato in maniera diversa nei diversi sistemi operativi (\ nei percorsi oppure /, presenza o assenza di unità, etc). In C per indicare il carattere '\ ' si usa la notazione '\\ '

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

38

File In C: Apertura

Per aprire un file si usa la funzione:

```
FILE* fopen(char fname[], char modo[])
```

modo specifica come aprire il file:

r apertura in lettura (**read**). Se il file non esiste → **fallimento**.

w apertura di un file vuoto in scrittura (**write**). Se il file esiste il suo contenuto viene cancellato.

a apertura in aggiunta (**append**). Crea il file se non esiste.

seguito opzionalmente da:

t apertura in modalità **testo** (default)

b apertura in modalità **binaria**

ed eventualmente da

- **+** apertura con possibilità di *lettura e scrittura*.

FILE IN C: APERTURA

Modi:

- **r+** apertura in lettura e scrittura. Se il file non esiste → **fallimento**.
- **w+** apertura un file vuoto in lettura e scrittura. Se il file esiste il suo contenuto viene distrutto.
- **a+** apertura in lettura e aggiunta. Se il file non esiste viene creato.

FILE IN C: Apertura

- Il *puntatore a FILE* restituito da `fopen()` si deve usare in tutte le successive operazioni sul file.
 - esso assume il valore `NULL` in caso l'apertura sia fallita
 - controllarlo è il solo modo per sapere se il file è stato davvero aperto: non dimenticarlo!
 - se non è stato aperto, il programma non può proseguire → *procedura* `exit()`

```
#include <stdlib.h>
...
FILE *fp;
fp = fopen("esempio.txt", "rt");
if (fp==NULL)
{
    printf("file esempio.txt non trovato");
    exit(-1);
}
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

41

File In C: Chiusura

Per chiudere un file si usa la funzione:

`int fclose(FILE*)`

- Il valore restituito da `fclose()` è un intero
 - 0 se tutto è andato bene
 - EOF in caso di errore.
- Prima della chiusura, tutti i buffer vengono svuotati.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

42

FILE DI TESTO

- Un **file di testo** è un file che contiene **sequenze di caratteri**
- È un caso **estremamente frequente**, con **caratteristiche proprie**:
 - esiste un concetto di **riga** e di **fine riga** ('`\n`')
 - la sequenza di caratteri è chiusa dal **carattere speciale EOF**

43

FILE DI TESTO (segue)

Funzione da console	Funzione da file
int getchar(void);	int fgetc(FILE* f);
int putchar(int c);	int fputc(int c, FILE* f);
char* gets(char* s);	char* fgets(char* s, int n, FILE* f);
int puts(char* s);	int fputs(char* s, FILE* f);
int printf(...);	int fprintf(FILE* f, ...);
int scanf(...);	int fscanf(FILE* f, ...);

- tutte le funzioni da file acquistano una “**f**” davanti nel nome (qualcuna però cambia leggermente nome)
- tutte le funzioni da file hanno un *parametro in più*, che è appunto il puntatore al **FILE** aperto

44

ESERCIZIO

- Si scriva su un file di testo `testo.txt` quello che l'utente inserisce da tastiera parola per parola, finché non inserisce la parola **"FINE"** (usando la `fprintf`).

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

45

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

void scrivi(char testo[50]){
    do{
        printf("Inserisci una parola di massimo 50 caratteri\n");
        scanf("%s",testo);
    }
    while(strlen(testo)>50);
}

int main(int argc, char **argv){

    FILE *f;
    f = fopen("test.txt", "w");

    if(f == NULL){
        printf("Errore nell'apertura del file\n");
        exit(1);
    }

    char testo[50];
    scrivi(testo);

    while(strcmp(testo,"FINE")){
        fprintf(f, "%s\n", testo);
        scrivi(testo);
    }

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

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

46

FUNZIONE feof()

- Durante la fase di accesso ad un file è possibile verificare la presenza del fine file con la funzione di libreria:

```
int feof(FILE *fp);
```

- feof(fp) controlla se è stata raggiunta la fine del file fp nell'operazione di lettura precedente.
- Restituisce il valore
 - 0 se non è stata raggiunta la fine del file,
 - un valore diverso da zero se è stata raggiunta la fine del file

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

47

ESEMPIO

Stampare a video il contenuto di un file di testo **prova.txt** usando **fscanf**.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

48

ESEMPIO

Stampare a video il contenuto di un file di testo **prova.txt** usando **fscanf**.

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

int main(int argc, char **argv){
    FILE *f;
    if((f = fopen("test.txt", "r")) == NULL){
        printf("Errore nella lettura del file\n");
        exit(1);
    }
    while(!feof(f)){
        char t[20];
        fscanf(f, "%s", t);
        printf("%s\n", t);
    }
    fclose(f);
    return 0;
}
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

49

Esempio File Testo

È dato un file di testo **people.txt** le cui righe rappresentano *ciascuna i dati di una persona*, secondo il seguente formato:

- **cognome** (al più 30 caratteri)
- uno o più spazi
- **nome** (al più 30 caratteri)
- uno o più spazi
- **sesso** (un singolo carattere, 'M' o 'F')
- uno o più spazi
- **anno di nascita**

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

50

Esempio File Testo

Si vuole scrivere un programma che

- legga riga per riga i dati dal file
- e ponga i dati in un array di persone
- *(poi svolgeremo elaborazioni su essi)*

Un possibile file
people.txt:

```
Rossi Mario M 1947
Ferretti Paola F 1982
Verdi Marco M 1988
Bolognesi Annarita F 1976
...
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

51

Esempio Completo File Testo

Come organizzarsi?

- 1) Definire una struttura **persona**

Poi, nel main:

- 2) Definire un array di strutture **persona**
- 3) Aprire il file in lettura
- 4) Leggere una riga per volta, e porre i dati di quella persona in una cella dell'array
→ Servirà un indice per indicare la prossima cella libera nell'array.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

52

Esempio Completo File Test

1) Definire una struttura di tipo **persona**

Occorre definire una **struct** adatta a ospitare i dati elencati:

- **cognome** →array di 30+1 caratteri
- **nome** →array di 30+1 caratteri
- **sexso** →array di 1+1 caratteri
- **anno di nascita** →un intero

ricordarsi lo spazio per il terminatore

```
struct persona{
    char cognome[31], nome[31], sexso[2];
    int anno;
};
```

Esempio Completo File Test

Poi, nel main:

- 2) definire un array di **struct persona**
- 3) aprire il file in lettura

```
main() {
    struct persona v[DIM];
    FILE* f = fopen("people.txt", "r");
    if (f==NULL) {
        .../* controllo che il file sia
            effettivamente aperto */
    }
    ...
}
```

Esempio Completo File Testo

Poi, nel main:

- 4) leggere una riga per volta, e porre i dati di quella persona in una cella dell'array

Come organizzare la lettura?

- Dobbiamo leggere delle stringhe separate una dall'altra da spazi
- Sappiamo che ogni singola stringa (cognome, nome, sesso) non contiene spazi

Uso *fscanf*

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

55

Esempio Completo File Testo

Poi, nel main:

- 4) leggere una riga per volta, e porre i dati di quella persona in una cella dell'array

Cosa far leggere a **fscanf**?

- *Tre stringhe* separate una dall'altra da spazi
→ si ripete *tre volte* il formato **%s**
- *Un intero* → si usa il formato **%d**
- **fscanf(f, "%s%s%s%d", ...)**

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

56

Esempio Completo File Test

Poi, nel main:

- 4) leggere una riga per volta, e porre i dati di quella persona in una cella dell'array

Fino a quando si deve leggere?

- Quando il file termina, `fscanf` restituisce `EOF`
→basta controllare il valore restituito
- Si continua fintanto che è diverso da `EOF`

```
while (fscanf(...) != EOF)
```

```
...
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

57

Esempio Completo File Test

Poi, nel main:

- 4) leggere una riga per volta, e porre i dati di quella persona in una cella dell'array

```
main() {
    int k=0; /* indice per array */
    ...
    while(fscanf(f, "%s%s%s%d",
        v[k].cognome, v[k].nome,
        v[k].sesso, &v[k].anno ) != EOF) {
        k++; /* devo incrementare k */
    }
}
```

Ricorda: l'intero richiede l'estrazione esplicita dell'indirizzo della variabile

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

58

fscanf

Notare inoltre che ...

- **fscanf** elimina *automaticamente* gli spazi che separano una stringa dall'altra → non si devono inserire spazi nella stringa di formato
- **fscanf** considera finita una stringa *al primo spazio che trova* → non si può usare questo metodo per leggere stringhe contenenti spazi

Esempio Completo File Test

```
#define DIM 30
#include <stdio.h>
#include <stdlib.h>

struct persona{
    char cognome[31], nome[31], sesso[2];
    int anno;
};

main() {
    struct persona v[DIM]; int k=0; FILE* f;
    if ((f=fopen("people.txt", "r"))==NULL) {
        printf("Il file non esiste!"); exit(1); }
    while(fscanf(f,"%s%s%d", v[k].cognome,
        v[k].nome, v[k].sesso, &v[k].anno ) != EOF)
        k++;
    fclose(f);
}
```

Dichiara la procedura `exit()`

fscanf

Notare inoltre che ...

- `fscanf` elimina *automaticamente* gli spazi che separano una stringa dall'altra → non si devono inserire spazi nella stringa di formato
- `fscanf` considera finita una stringa *al primo spazio che trova* → non si può usare questo metodo per leggere stringhe contenenti spazi

Però ...

- possiamo usare `fscanf` in un'altra modalità, specificando quanti caratteri leggere. Ad esempio:

```
fscanf(f, "%10c", ...)
```

legge esattamente 10 caratteri, spazi inclusi

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

61

ESEMPIO

- È dato un file di testo elenco.txt le cui righe rappresentano ciascuna i dati di una persona, secondo il seguente formato:
- cognome (esattamente 10 caratteri)
- nome (esattamente 10 caratteri)
- sesso (esattamente un carattere)
- anno di nascita
- I primi due possono contenere spazi al loro interno.
- NB: non sono previsti spazi espliciti di separazione

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

62

ESEMPIO COMPLETO

```
#define DIM 30
#include <stdio.h>
#include <stdlib.h>
```

Sappiamo esattamente
la dimensione: 10 + 1

```
typedef struct {
    char cognome[11], nome[11], sesso; int anno;
} persona;
```

```
main() {
```

```
    persona v[DIM]; int k=0; FILE* f;
```

```
    if ((f=fopen("elenco.txt", "r"))==NULL) {
        perror("Il file non esiste!"); exit(1); }
```

```
    while(fscanf(f,"%10c%10c%c%d\n", v[k].cognome,
        v[k].nome, &v[k].sesso, &v[k].anno) != EOF){
        v[k].cognome[10]=v[k].nome[10]='\0'; k++;}
```

```
}
```

Legge esattamente 10
caratteri (spazi inclusi)

Legge 1 carattere
e un intero
(ricordare &)

Ricordare il
terminatore!

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

63

ESERCIZIO

Un file di testo `rubrica.txt` contiene una rubrica del telefono, in cui per ogni persona è memorizzato

- **nome** (stringa di 20 caratteri senza spazi, incluso terminatore)
- **indirizzo** (stringa di 30 caratteri senza spazi, incluso '\0')
- **numero** (stringa di 15 caratteri incluso '\0')

Si scriva un programma C che legge da tastiera un nome, cerca la persona corrispondente nel file `rubrica.txt` e visualizza sullo schermo i dati della persona (se trovata)

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

64

File Di Testo E Console

- In realtà, anche per **leggere da tastiera** e **scrivere su video**, il C usa le **procedure per i file**.
- Ci sono 3 file, detti **canali di I/O standard**, che sono **già aperti**:
 - **stdin** è un file di testo aperto in lettura, di norma agganciato alla tastiera
 - **stdout** è un file di testo aperto in scrittura, di norma agganciato al video
 - **stderr** è un altro file di testo aperto in scrittura, di norma agganciato al video
- Le funzioni di I/O disponibili per i file di testo sono una generalizzazione di quelle già note per i canali di I/O standard.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

65

Lettura Di Stringhe

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

- Trasferisce nella stringa **s** i caratteri letti dal file puntato da fp, fino a quando ha letto n-1 caratteri, oppure ha incontrato un newline, oppure la fine del file.
- Il carattere newline, se letto, e' mantenuto nella stringa **s**
- Restituisce la stringa letta in caso di corretta terminazione; NULL in caso di errore o fine del file.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

66

Scrittura Di Stringhe

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

- Trasferisce la stringa s (terminata da '\0') nel file puntato da fp. Non copia il carattere terminatore '\0' ne` aggiunge un new line finale.
- Restituisce un numero non negativo in caso di terminazione corretta; EOF altrimenti.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

67

Accesso Diretto

Il C consente di gestire i file anche ad **accesso diretto** utilizzando una serie di funzioni della libreria standard.

La funzione **fseek** consente di spostare la testina di lettura su un qualunque **byte** del file

```
int fseek (FILE *f, long offset, int origin)
```

Sposta la testina di **offset byte** a partire dalla posizione **origin** (che vale 0, 1 o 2).

Se lo spostamento ha successo fornisce 0 altrimenti un numero diverso da 0

Attenzione: per file aperti in modalità testo, fseek ha un uso limitato, perché non c'è una corrispondenza tra i caratteri del file e i caratteri del testo (un «a capo» possono essere due caratteri) e quindi quando chiamiamo la fseek con un dato offset possiamo non ottenere la posizione che ci aspetteremmo

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

68

Accesso Diretto

Origine dello spostamento:

costanti definite in **stdio.h**

- | | | |
|---|----------------------------|-----------------|
| 1 | inizio file | SEEK_SET |
| 2 | posizione attuale nel file | SEEK_CUR |
| 3 | fine file | SEEK_END |

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

69

rewind

```
void rewind(FILE *f);
```

Posiziona la testina all'inizio del file

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

70

ftell

```
long ftell(FILE *f);
```

Restituisce la **posizione del byte su cui è posizionata la testina** al momento della chiamata, restituisce -1 in caso di errore.

Il valore restituito dalla `ftell` può essere utilizzato in una chiamata della `fseek`

71

ESEMPIO

Programma che sostituisce tutte le minuscole in maiuscole in un file testo fornito come (unico) dato di ingresso.

Aprire il file e Leggere uno alla volta i caratteri:

- se il carattere è minuscolo
 - spostare la testina indietro di una posizione
 - scrivere il carattere convertito in maiuscolo

72

ESEMPIO

```
#include <stdio.h>
#include <stdlib.h>
main()
{ FILE *fp;
  char nomefile[50], ch;
  printf("Nome file?"); scanf("%s", nomefile);
  if ((fp=fopen(nomefile, "r+"))==NULL)
    exit(-1);
  while (!feof(fp))
  { fscanf(fp, "%c", &ch);
    if ((ch<='z') && (ch>='a'))
    { fseek(fp, ftell(fp)-1, SEEK_SET);
      fprintf(fp, "%c", ch+('A'-'a'));
      fseek(fp, 0, SEEK_CUR);
    }
  }
  fclose(fp);
}
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

73

File Binari

Un file binario è una pura sequenza di byte, senza alcuna strutturazione particolare.

- È un'astrazione di memorizzazione *assolutamente generale*, usabile per memorizzare su file *informazioni di qualsiasi natura*
 - "fotografie" della memoria
 - rappresentazioni interne binarie di numeri
 - immagini, canzoni campionate,
 - ..volendo, *anche caratteri!*
- I file di testo non sono indispensabili: sono semplicemente *comodi!*

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

74

File Binari

Un file binario è una sequenza di byte

- Può essere usato per archiviare su memoria di massa *qualsunque tipo di informazione*
- Input e output avvengono sotto forma di una **sequenza di byte**
- **La fine del file** è rilevata in base *all'esito delle operazioni di lettura*
 - **non c'è EOF**, perché un file binario non è una sequenza di caratteri
 - la lunghezza del file è registrata dal sistema op.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

75

File Binari

- Poiché un file binario è una sequenza di byte, sono fornite due funzioni per *leggere e scrivere* sequenze di byte
 - **fread()** legge una **sequenza di byte**
 - **fwrite()** scrive una **sequenza di byte**
- **Essendo pure sequenze di byte, esse non sono interpretate**: l'interpretazione è "negli occhi di chi guarda".
- **Quindi, possono rappresentare qualunque informazione** (testi, numeri, immagini...)

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

76

OUTPUT BINARIO: fwrite()

Sintassi:

```
int fwrite(addr, int dim, int n, FILE *f);
```

- scrive sul file **n** elementi, ognuno grande **dim** byte (complessivamente, scrive quindi **n × dim** byte)
- gli elementi da scrivere vengono prelevati dalla memoria a partire dall'indirizzo **addr**
- *restituisce il numero di elementi (non di byte!) effettivamente scritti, che possono essere meno di **n**.*

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

77

INPUT BINARIO: fread()

Sintassi:

```
int fread(addr, int dim, int n, FILE *f);
```

- legge dal file **n** elementi, ognuno grande **dim** byte (complessivamente, *tenta di leggere* quindi **n × dim** byte)
- gli elementi da leggere vengono scritti in memoria a partire dall'indirizzo **addr**
- *restituisce il numero di elementi (non di byte!) effettivamente letti, che possono essere meno di **n** se il file finisce prima. Controllare il valore restituito è un modo per sapere cosa è stato letto e, in particolare, per scoprire se il file è finito.*

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

78

ESEMPIO 1

Salvare su un file binario **binary.dat** il contenuto di un array di dieci interi.

79

ESEMPIO 1

Salvare su un file binario **binary.dat** il contenuto di un array di dieci interi.

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char **argv){
    int a[10] = {1,2,3,4, 5,6,7, 8, 9, 10};
    FILE *f;
    if((f = fopen("binary.dad", "wb")) == NULL){
        printf("Errore nell'apertura del file\n");
        exit(0);
    }
    fwrite(a, sizeof(int), 10, f);
    fclose(f);
    return 0;
}
```

In alternativa:

```
fwrite(vet, 10*sizeof(int), 1, fp)
```

80

Esempio 1

- Apriamo il file binario appena creato con un editor di testo ...

```
SOHNULSTXNULETXNULEOTNULENONULACKNULBELNULBSNUL NUL
NUL
```



Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

81

ESEMPIO 2

Leggere da un file binario `bianry.dat` una sequenza di interi, scrivendoli in un array.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

82

ESEMPIO 2

Leggere da un file binario **binary.dat** una sequenza di interi, scrivendoli in un array.

```
#include<stdio.h>
#include<stdlib.h>
int main(int argc, char **argv){
    FILE *f;
    if ((f = fopen("binary.dad", "rb"))== 0){
        printf("Errore nell'apertura del file\n");
        exit(1);
    }
    int a[10];
    //n conterrà il numero di interi letti!
    int n = fread(a, sizeof(int), 10, f);
    for(int i=0; i<n; i++)
        printf("a[%d] = %d\n", i, a[i]);

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

n contiene il numero di
interi effettivamente letti

fread tenta di leggere 10
interi, ma ne legge meno se il
file finisce prima

Luca Guarnera
luca.guarnera@unicit.it

Laboratorio di Reti di Calcolatori

di Calcolatori
LABORATORIO - 3 CFU

83

ESEMPIO 3

Leggere da un file di caratteri **testo.txt** una sequenza di caratteri, ponendoli in una stringa.

```
#include <stdio.h>
#include <stdlib.h>
main()
{ FILE *fp;
  char msg[80], n;
  fp = fopen("testo.txt", "rb");
  if (fp==NULL)
      exit(1); /* Errore di apertura */
  n = fread(msg, 1, 80, fp);
  printf("%s", msg);
  fclose(fp);
}
```

Esperimento: provare a leggere
un file (corto) creato con un
editor qualunque

n contiene il numero di **char**
effettivamente letti (*che non ci
interessa, perché tanto c'è il
terminatore ...*)

Luca Guarnera
luca.guarnera@unicit.it

Laboratorio di Reti di Calcolatori

di Calcolatori
LABORATORIO - 3 CFU

84

ESEMPIO 4

Scrivere su un file di caratteri `testo.txt` una sequenza di caratteri.

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

main()
{ FILE *fp;
  char msg[] = "Ah, l'esame\nsi avvicina!";
  fp = fopen("testo.txt", "wb");
  if (fp==NULL)
    exit(1); /* Errore di apertura */
  fwrite(msg, strlen(msg)+1, 1, fp);
  fclose(fp);
}
```

Dopo averlo creato, provare ad aprire questo file con un editor qualunque (es. blocco note).
(e il terminatore?..)

Un carattere in C ha sempre **size=1**
Sceita: salvare anche il terminatore.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori

Reti di Calcolatori
LABORATORIO - 3 CFU

85

ESEMPIO FILE BINARIO

È dato un file binario `people.dat` i cui record rappresentano *ciascuno i dati di una persona*, secondo il seguente formato:

- **cognome** (al più 30 caratteri)
- **nome** (al più 30 caratteri)
- **sexso** (un singolo carattere, 'M' o 'F')
- **anno di nascita**

Si noti che la creazione del file binario deve essere fatta da programma, mentre per i file di testo può essere fatta con un text editor.

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO - 3 CFU

86

CREAZIONE FILE BINARIO

Per creare un file binario è necessario scrivere un programma che lo crei strutturandolo in modo che ogni record contenga una **struct persona**

```
struct persona
{
    char cognome[31], nome[31], sesso[2];
    int anno;
};
```

I dati di ogni persona da inserire nel file vengono richiesti all'utente tramite la funzione `leggiel()` che non ha parametri e restituisce come valore di ritorno la **struct persona** letta. Quindi il prototipo è:

```
struct persona leggiel();
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

87

CREAZIONE FILE BINARIO

Mentre la definizione è:

```
struct persona leggiel() {
    struct persona e;

    printf("Cognome ? ");
    scanf("%s", e.cognome);
    printf("\n Nome ? ");
    scanf("%s", e.nome);
    printf("\nSesso ? ");
    scanf("%s", e.sesso);
    printf("\nAnno nascita ? ");
    scanf("%d", &e.anno);
    return e;
}
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

88

CREAZIONE FILE BINARIO

```
#include <stdio.h>
#include <stdlib.h>
struct persona
{char cognome[31], nome[31], sesso[2];
  int anno;
};
struct persona leggiel();
main()
{ FILE *f; struct persona e; int fine=0;
  f=fopen("people.dat", "wb");
  while (!fine)
  { e=leggiel();
    fwrite(&e,sizeof(struct persona),1,f);
    printf("\nFine (SI=1, NO=0) ? ");
    scanf("%d", &fine);
  }
  fclose(f);
}
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

89

CREAZIONE FILE BINARIO

L'esecuzione del programma precedente crea il file binario contenente i dati immessi dall'utente. Solo a questo punto il file può essere utilizzato.

Il file **people.dat** non è visualizzabile tramite un text editor: questo è il risultato

```
rossi > ŷŷ @T —8      â3 mario ôÜ_ ôÜ Aw
O F _      DÝ M      nuinH2ô1 ô1 ô1
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

90

ESEMPIO COMPLETO FILE BINARIO

Ora si vuole scrivere un programma che

- legga record per record i dati dal file
- e ponga i dati in un array di persone



91

ESEMPIO COMPLETO FILE BINARIO

```
#define DIM 30
#include <stdio.h>
#include <stdlib.h>

struct persona
{char cognome[31], nome[31], sesso[2];
  int anno;
};

main()
{struct persona v[DIM]; int i=0; FILE* f;
  if ((f=fopen("people.dat", "rb"))==NULL)
    { printf("Il file non esiste!"); exit(1); }
  while(fread(&v[i],sizeof(struct persona),1,f)>0)
    i++;
}
```



92

ESEMPIO COMPLETO FILE BINARIO

```
#define DIM 30
#include <stdio.h>
#include <stdlib.h>

struct persona
{char cognome[31], nome[31], sesso[2];
 int anno;
};

main()
{struct persona v[DIM]; FILE* f;
 if ((f=fopen("people.dat", "rb"))==NULL)
 { printf("Il file non esiste!"); exit(1); }
 fread(v,sizeof(struct persona),DIM,f);
}
```

Dichiara la procedura `exit()`

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

93

ALTRE FUNZIONI RELATIVE AI FILE

`int ferror(FILE *fp)`

- Controlla se è stato commesso un errore nella operazione di lettura o scrittura precedente. Restituisce il valore 0 se non c'è stato errore altrimenti un valore diverso da 0

`void clearerr(FILE *fp)`

- Riporta al valore di default lo stato delle informazioni (eof ed error) di fine file ed errore associate al file

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



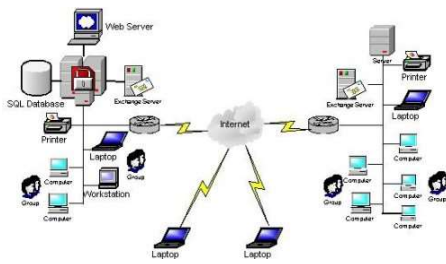
Reti di Calcolatori
LABORATORIO – 3 CFU

94



Università
di Catania

Trattamento dei Bit



Luca Guarnera
luca.guarnera@unicat.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

95

Trattamento dei bit

- Il C fornisce un ricco insieme di operatori, detti **operatori bitwise**, per il trattamento dei bit

Operatori	Funzione
&	Restituisce l'AND bit a bit
	Restituisce l'OR bit a bit
^	Restituisce l'XOR (OR esclusivo) bit a bit
~	Restituisce il NOT bit a bit (complemento a 1)
<<	Restituisce la stringa di binari "shiftata" di n posti verso sinistra
>>	Restituisce la stringa di binari "shiftata" di n posti verso destra

Luca Guarnera
luca.guarnera@unicat.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

96

Trattamento dei bit

a	b	a&b	a b	a^b
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

97

Esercizio

- Supponiamo di avere due int, a e b. Internamente a e b sono rappresentabili in binario:

a = 16 =

b = 24 =

Bit

A programmazione 1 e 2 mi hanno insegnato che ...

TYPE	SIZE	MIN	MAX
char	1	-128	127
unsigned char	1	0	255
short int	2	-32768	32767
unsigned short int	2	0	65535
int	4	-2147483648	2147483647
unsigned int	4	0	4294967295
long int	8	-9223372036854775808	9223372036854775807
unsigned long int	8	0	18446744073709551615
float	4	1.175494e-38	3.402823e+38
double	8	2.225074e-308	1.797693e+308
long double	16	3.362103e-4932	1.189731e+4932

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

98

Esercizio

- Supponiamo di avere due int, a e b. Internamente a e b sono rappresentabili in binario:

Bit

a = 16 = 00000000 00000000 00000000 00010000

b = 24 = 00000000 00000000 00000000 00011000



Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

99

Esercizio

- Supponiamo di avere due int, a e b. Internamente a e b sono rappresentabili in binario:

Bit

a = 16 = 00000000 00000000 00000000 00010000

b = 24 = 00000000 00000000 00000000 00011000

a & b =

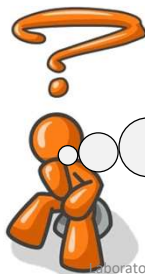
a | b =

a ^ b =

~a =

a << 4 =

a >> 2 =



Nelle precedenti slide abbiamo visto ...

a	b	a&b	a b	a^b
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



Reti di Calcolatori
LABORATORIO – 3 CFU

100

Esercizio

- Supponiamo di avere due int, a e b. Internamente a e b sono rappresentabili in binario:

Bit

a = 16 = 00000000 00000000 00000000 00010000

b = 24 = 00000000 00000000 00000000 00011000

a & b = 00000000 00000000 00000000 00010000 = 16

a | b =

a ^ b =

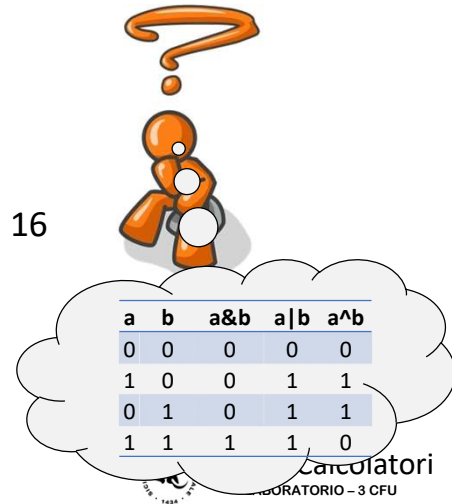
~a =

a << 4 =

a >> 2 =

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



101

Esercizio

- Supponiamo di avere due int, a e b. Internamente a e b sono rappresentabili in binario:

Bit

a = 16 = 00000000 00000000 00000000 00010000

b = 24 = 00000000 00000000 00000000 00011000

a & b = 00000000 00000000 00000000 00010000 = 16

a | b = 00000000 00000000 00000000 00011000 = 24

a ^ b =

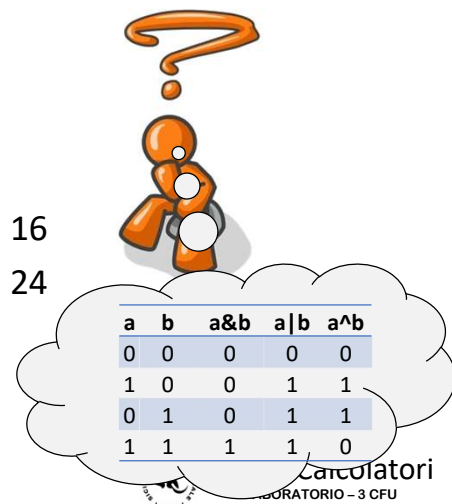
~a =

a << 4 =

a >> 2 =

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



102

Esercizio

- Supponiamo di avere due int, a e b. Internamente a e b sono rappresentabili in binario:

Bit

$a = 16 = 00000000\ 00000000\ 00000000\ 00010000$
 $b = 24 = 00000000\ 00000000\ 00000000\ 00011000$
 $a \& b = 00000000\ 00000000\ 00000000\ 00010000 = 16$
 $a | b = 00000000\ 00000000\ 00000000\ 00011000 = 24$
 $a \wedge b = 00000000\ 00000000\ 00000000\ 00001000 = 8$
 $\sim a =$
 $a \ll 4 =$
 $a \gg 2 =$

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



a	b	a&b	a b	a^b
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

Calcolatori

LABORATORIO - 3 CFU

103

Esercizio

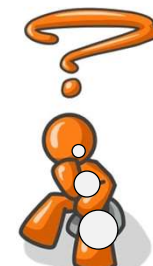
- Supponiamo di avere due int, a e b. Internamente a e b sono rappresentabili in binario:

Bit

$a = 16 = 00000000\ 00000000\ 00000000\ 00010000$
 $b = 24 = 00000000\ 00000000\ 00000000\ 00011000$
 $a \& b = 00000000\ 00000000\ 00000000\ 00010000 = 16$
 $a | b = 00000000\ 00000000\ 00000000\ 00011000 = 24$
 $a \wedge b = 00000000\ 00000000\ 00000000\ 00001000 = 8$
 $\sim a = 11111111\ 11111111\ 11111111\ 11101111$
 $a \ll 4 =$
 $a \gg 2 =$

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



a	b	a&b	a b	a^b
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

Calcolatori

LABORATORIO - 3 CFU

104

Esercizio

- Supponiamo di avere due int, a e b. Internamente a e b sono rappresentabili in binario:

Bit

```

a = 16 = 00000000 00000000 00000000 00010000
b = 24 = 00000000 00000000 00000000 00011000
a & b = 00000000 00000000 00000000 00010000 = 16
a | b = 00000000 00000000 00000000 00011000 = 24
a ^ b = 00000000 00000000 00000000 00001000 = 8
~a    = 11111111 11111111 11111111 11101111
a << 4 = 00000000 00000000 00000001 00000000
a >> 2 = 00000000 00000000 00000000 00001000
  
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



*stringa di binari shiftata
di 4 posti verso sinistra*

Calcolatori

LABORATORIO - 3 CFU

105

Esercizio

- Supponiamo di avere due int, a e b. Internamente a e b sono rappresentabili in binario:

Bit

```

a = 16 = 00000000 00000000 00000000 00010000
b = 24 = 00000000 00000000 00000000 00011000
a & b = 00000000 00000000 00000000 00010000 = 16
a | b = 00000000 00000000 00000000 00011000 = 24
a ^ b = 00000000 00000000 00000000 00001000 = 8
~a    = 11111111 11111111 11111111 11101111
a << 4 = 00000000 00000000 00000001 00000000
a >> 2 = 00000000 00000000 00000000 00001000
  
```

Luca Guarnera
luca.guarnera@unict.it

Laboratorio di Reti di Calcolatori



*stringa di binari shiftata
di 2 posti verso destra*

Calcolatori

LABORATORIO - 3 CFU

106



Laboratorio di Reti di Calcolatori

Luca Guarnera, Ph.D.

<https://www.dmi.unict.it/lguarnera/>
luca.guarnera@unict.it

Università di Catania
Dipartimento di Matematica e Informatica
Corso di Laurea Triennale in Informatica

 **Reti di Calcolatori**
LABORATORIO – 3 CFU

107