

Laboratorio di Reti di Calcolatori

Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma

Luca Iezzi Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Reti di Calcolatori A.A. 2022/23

Prof.ssa Chiara Petrioli Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma

Laboratorio di Reti di Calcolatori

2

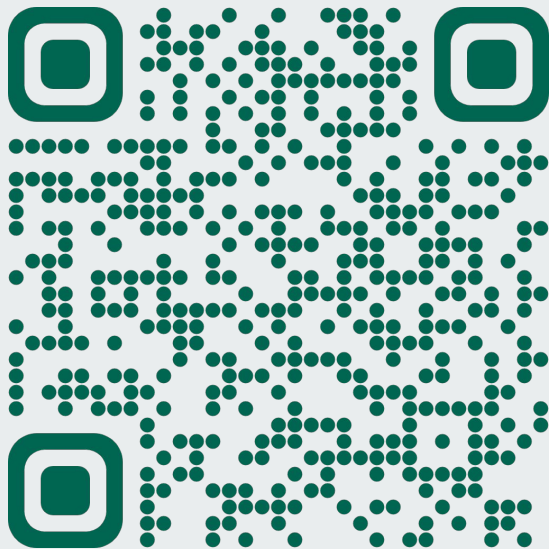
- Reti di Calcolatori: composte da numerosi apparati → **capacità computazionali** eterogenee
 - Calcolatori (ad es. PC, Workstation, Smartphone), anche chiamati **host, end systems**
 - Componenti di rete (ad es. Router, Switch, Wireless Access Point)
 - Dispositivi embedded (ad es. Sensori, Smart Objects)
- Sistemi embedded
 - Tipicamente low power e poche risorse computazionali
- Linguaggio C
 - Poco overhead di esecuzione
 - Controllo esplicito della memoria

Obiettivo del corso: potenziare la programmazione in C anche in ottica sistemi embedded

Laboratorio di Reti di Calcolatori

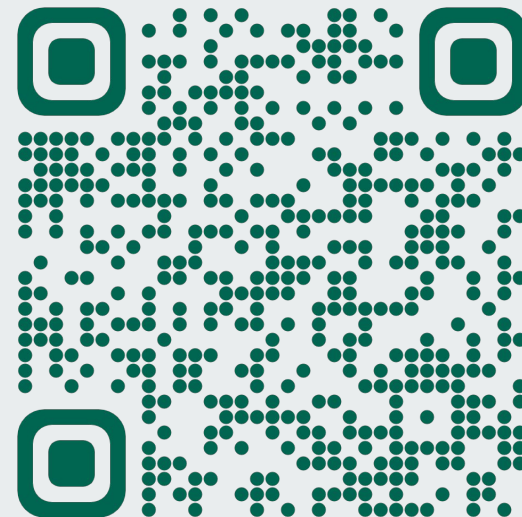
3

Sito del Laboratorio di Reti



Argomenti lezioni e soluzioni esercizi

Repository GitHub



Slide lezioni e materiale aggiuntivo

Modalità di esame

4

Obbligatorio

- Scritto relativo ai contenuti del laboratorio
- Svolto unitamente alla parte di teoria in sede di esoneri ed appelli

Opzionale

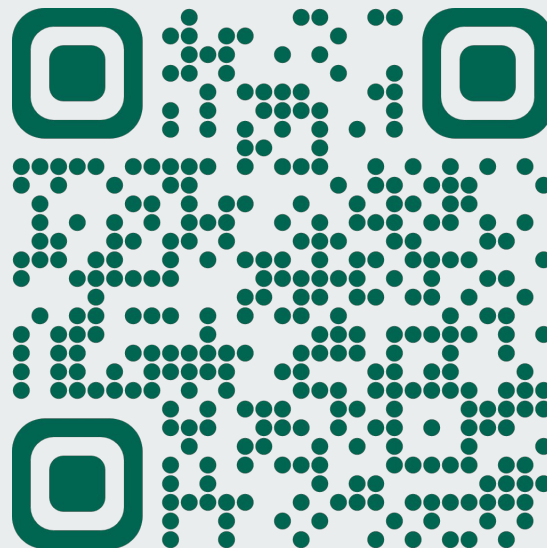
- Progetto sui temi mostrati a lezione di laboratorio
- Fino a +3 punti sul voto finale ottenuto tramite esoneri ed appelli

1. Verifica preliminare del linguaggio C

Verifica preliminare del linguaggio C

6

- Quiz anonimo
- Formato: vero/falso e risposta multipla (una sola corretta)
- Contenuto: conoscenza livello **base** e **intermedia** del linguaggio C
- **Scopo:** adeguare il contenuto del corso



2. Discussione dei risultati

3. Basi del linguaggio C

Basi del linguaggio C

9

Caratteristiche del linguaggio C

- Linguaggio compilato
- Tipizzazione statica
- Paradigma imperativo e procedurale

Compilatori

Windows

Qualsiasi IDE¹ per C.
Esempi: *Dev-C++*, *Visual Studio*,
CodeLite

Utilizzare l'interfaccia grafica per
compilare ed eseguire il programma.

Mac

Installare GCC / Clang e scrivere
in *Xcode*, *Visual Studio Code*, ecc.

```
gcc -o esempio_exec esempio.c  
clang esempio.c -o esempio_exec
```

Linux

GCC è pre-installato nella
maggioranza delle distro Linux.

```
gcc -o esempio_exec esempio.c
```

¹: *Integrated Development Environment*, ambiente di sviluppo integrato

Basi del linguaggio C

10

Source file: `example.c`Eseguibile: `example_exec`

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 // This program sums two integers.
5 int main(int argc, char *argv[]){
6     if(argc != 3){
7         printf("ERROR: invalid arguments provided.\n");
8         printf("Usage: ./example_exec Integer1 Integer2\n");
9         return 1;
10    }
11
12    int int1 = atoi(argv[1]);
13    int int2 = atoi(argv[2]);
14    int sum = int1 + int2;
15    printf("Sum of %d and %d is equal to %d.\n", int1, int2, sum);
16
17    return 0;
18 }
```

Direttive al preprocessore

Commento singola linea

Valore di ritorno che indica un errore

Definizione variabili con assegnazione di
valori convertiti ad interi da stringheValore di ritorno che indica
corretta esecuzione

Header file inclusi:

- **stdio.h**
Funzionalità di I/O (`printf`)
- **stdlib.h**
Funzionalità di utilità generale (`atoi`)

Funzione **main**:

- Prima funzione invocata durante l'esecuzione, può invocare altre funzioni
- Tipo di ritorno: **int**
- Argomenti **argc** e **argv**: parametri da riga di comando

Istruzione **if**:

- Esegue un blocco di codice a seconda del valore di un'espressione booleana: ovvero solo se **true**

Funzione **printf**:

- Stringa formattata mostrata su **stdout**
- **%d** per variabili intere (es. `sum`)

Parole chiave riservate in C

11

Gli identificatori in C (per variabili, funzioni, ecc.) non possono sovrascrivere le seguenti **parole chiave riservate**:

<code>alignas</code> (C23)	<code>extern</code>	<code>sizeof</code>	<code>_Alignas</code> (C11)
<code>alignof</code> (C23)	<code>false</code> (C23)	<code>static</code>	<code>_Alignof</code> (C11)
<code>auto</code>	<code>float</code>	<code>static_assert</code> (C23)	<code>_Atomic</code> (C11)
<code>bool</code> (C23)	<code>for</code>	<code>struct</code>	<code>_BitInt</code> (C23)
<code>break</code>	<code>goto</code>	<code>switch</code>	<code>_Bool</code> (C99)
<code>case</code>	<code>if</code>	<code>thread_local</code> (C23)	<code>_Complex</code> (C99)
<code>char</code>	<code>inline</code> (C99)	<code>true</code> (C23)	<code>_Decimal128</code> (C23)
<code>const</code>	<code>int</code>	<code>typedef</code>	<code>_Decimal32</code> (C23)
<code>constexpr</code> (C23)	<code>long</code>	<code>typeof</code> (C23)	<code>_Decimal64</code> (C23)
<code>continue</code>	<code>nullptr</code> (C23)	<code>typeof_unqual</code> (C23)	<code>_Generic</code> (C11)
<code>default</code>	<code>register</code>	<code>union</code>	<code>_Imaginary</code> (C99)
<code>do</code>	<code>restrict</code> (C99)	<code>unsigned</code>	<code>_Noreturn</code> (C11)
<code>double</code>	<code>return</code>	<code>void</code>	<code>_Static_assert</code> (C11)
<code>else</code>	<code>short</code>	<code>volatile</code>	<code>_Thread_local</code> (C11)
<code>enum</code>	<code>signed</code>	<code>while</code>	

keyword	used as	defined in
<code>_Alignas</code> (C11)	<code>alignas</code> (removed in C23)	<code>stdalign.h</code>
<code>_Alignof</code> (C11)	<code>alignof</code> (removed in C23)	<code>stdalign.h</code>
<code>_Atomic</code> (C11)	<code>atomic_bool</code> , <code>atomic_int</code> , ...	<code>stdatomic.h</code>
<code>_BitInt</code> (C23)	(no macro)	
<code>_Bool</code> (C99)	<code>bool</code> (removed in C23)	<code>stdbool.h</code>
<code>_Complex</code> (C99)	<code>complex</code>	<code>complex.h</code>
<code>_Decimal128</code> (C23)	(no macro)	
<code>_Decimal32</code> (C23)	(no macro)	
<code>_Decimal64</code> (C23)	(no macro)	
<code>_Generic</code> (C11)	(no macro)	
<code>_Imaginary</code> (C99)	<code>imaginary</code>	<code>complex.h</code>
<code>_Noreturn</code> (C11)	<code>noreturn</code>	<code>stdnoreturn.h</code>
<code>_Static_assert</code> (C11)	<code>static_assert</code> (removed in C23)	<code>assert.h</code>
<code>_Thread_local</code> (C11)	<code>thread_local</code> (removed in C23)	<code>threads.h</code>

Nel contesto delle direttive al preprocessore, anche le seguenti parole chiave sono **riservate**:

<code>if</code>	<code>ifdef</code>	<code>include</code>	<code>defined</code>
<code>elif</code>	<code>ifndef</code>	<code>embed</code> (C23)	<code>__has_include</code> (C23)
<code>else</code>	<code>elifdef</code> (C23)	<code>line</code>	<code>__has_embed</code> (C23)
<code>endif</code>	<code>elifndef</code> (C23)	<code>error</code>	<code>__has_c_attribute</code> (C23)
	<code>define</code>	<code>warning</code> (C23)	
	<code>undef</code>	<code>pragma</code>	

Aggiuntivamente, il preprocessore riconosce le seguenti parole chiave al di fuori del contesto delle direttive e sono quindi **riservate**:

`_Pragma` (C99)

Dichiarazione e definizione di variabili

12

Dichiarazione

```
1 int x;           Variabile intera.  
2 int arr[5];      Variabile array di interi con dimensione 5.  
3 int *hugeArr;    Variabile puntatore ad intero.
```

Per ogni variabile viene specificato solo tipo ed identificatore: generalmente contengono valori spuri, ovvero ciò che è contenuto in memoria.

Definizione

```
1 int x = 42;       Inizializzazione con integer literal 42.  
2 int arr[5] = {1,2,3,4,5}; Dimensione array opzionale.  
3 int *hugeArr = arr; Indirizzo del primo elemento di arr.
```

Contestualmente alla dichiarazione, per ogni variabile avviene anche un'operazione di assegnazione. Il valore viene esplicitamente fornito tramite inizializzazione.

Buona abitudine: inizializzare sempre ogni variabile.

Tipi primitivi in C

13

Il tipo di una variabile determina la dimensione della porzione di memoria allocata ed il range di valori ammesso.

Classificazione	Tipo	Dimensione ²	Range valori	Format string
Caratteri	char	Almeno 8 bit	± 127	%hhd
Numeri interi	short	Almeno 16 bit	$\pm 32\,767$	%hd
	int	Almeno 16 bit	$\pm 2\,147\,483\,647$	%d
	long	Almeno 32 bit	$\pm 2\,147\,483\,647$	%ld
	long long	Almeno 64 bit	$\pm 2^{63}$	%lld
Numeri decimali	float	>32 bit, di cui >6 significant	Esponente [-37, 38]	%f
	double	>64 bits, di cui >15 significant	Esponente [-307, 308]	%lf
	long double	>64 bits, di cui 15/18/33 significant (dipende dim.)	Esponente [-4931, 4932]	%Lf

2: L'effettiva dimensione di ogni tipo è determinata dal compilatore usato; il linguaggio C si limita ad imporre una dimensione minima

signed e unsigned

14

Per default, i tipi per i numeri interi sono **signed**: in questo modo è possibile rappresentare valori sia positivi che negativi. Qualora non si avesse necessità di rappresentare valori negativi all'interno di una variabile, questa può essere esplicitamente dichiarata di un tipo **unsigned**.

Classificazione	Tipo	Dimensione ²	Range valori	Format string
Caratteri	unsigned char	Almeno 8 bit	[0, 255]	%hhu
Numeri interi	unsigned short	Almeno 16 bit	[0, 65 535]	%hu
	unsigned int	Almeno 16 bit	[0, 4 294 967 295]	%u
	unsigned long	Almeno 32 bit	[0, 4 294 967 295]	%lu
	unsigned long long	Almeno 64 bit	$\pm 2^{64} - 1$	%llu

Svariati errori di tipo vengono intercettati nella compilazione ed altri durante l'esecuzione, ma alcuni **errori silenziosi** possono comunque avvenire:

```
1 unsigned short x = 0;  
2 x--;
```

2: L'effettiva dimensione di ogni tipo è determinata dal compilatore usato; il linguaggio C si limita ad imporre una dimensione minima

Operatori comuni

Le variabili possono essere manipolate con numerosi operatori.

Assegna il valore dell'espressione a destra alla variabile a sinistra.

Combinano l'assegnazione con un'operazione aritmetica. La variabile di sinistra è allo stesso tempo sia il **primo operando** che la variabile in cui sarà contenuto il risultato dell'espressione.

Common operators						
assignment	increment decrement	arithmetic	logical	comparison	member access	other
<pre>a = b a += b a -= b a *= b a /= b a %= b a &= b a = b a ^= b a <=< b a >=> b</pre>	<pre>++a --a a++ a--</pre>	<pre>+a -a a + b a - b a * b a / b a % b ~a a & b a b a ^ b a << b a >> b</pre>	<pre>!a a && b a b</pre>	<pre>a == b a != b a < b a > b a <= b a >= b</pre>	<pre>a[b] *a &a a->b a.b</pre>	<pre>a(...) a, b (type) a a ? b : c sizeof Alignof (since C11)</pre>

Pre-incremento/decremento (++a / --a):
valore aggiornato (di 1) **prima della**
valutazione dell'intera espressione.

Post-incremento/decremento (a++ / a--):
valore aggiornato (di 1) **dopo la valutazione**
dell'intera espressione.

Shift: il valore della variabile a
viene *spostato* (rappresentazione
in bit) di b posizioni a sinistra (<<,
moltiplicazione) o destra (>>,
divisione).

Operatori **bit-wise**

Particolare attenzione

- Operatore **dereference** (*a)
Ottiene il valore di una *variabile*
puntatore a
- Operatore **indirizzo** (&a)
Ottiene l'*indirizzo della locazione di*
memoria che è stata allocata dalla
variabile a

Cast esplicito: conversione della
variabile a nel tipo type.
Se type è **void**, l'espressione
viene valutata per i suoi **side**
effects ed il valore di ritorno è
scartato.

Input e output

16

Output

Come visto in precedenza, la funzione `printf` viene usata per stampare contenuti a video (**standard output**).

- Ogni tipo di dato ha una **format string** associata (ad es. `%d` per gli interi)
- Istruzioni simili permettono di scrivere contenuti su file stream (`fprintf`) o buffer (`sprintf`)

Stringhe C → Array di caratteri terminati da `'\0'`: format string `%s` oppure funzioni dedicate `puts` e `fputs`

Input

In modo analogo a `printf`: funzione `scanf`

È possibile leggere stringhe C in questo modo?

```
1 char str[5];  
2 printf("Enter a string:\n");  
3 scanf("%s", str);
```

`scanf` accetta **variabili**
puntatori come argomenti

```
1 int x = 0, y = 0;  
2 float z = 0.f;  
3 printf("Enter x, y, z:\n");  
4 scanf("%d%d%f", &x, &y, &z);
```

Valori **multipli**
possono essere letti
dallo standard input
in una sola istruzione

Input e output: stringhe C

17

```
1 char str[5];  
2 printf("Enter a string:\n");  
3 scanf("%s", str);
```

- Possibile? Sì
- Sicuro? **No!**



Un input utente con lunghezza superiore a 5 manderebbe in crash il programma

Soluzione

Funzione **fgets** → Legge al più un numero esplicito di caratteri oppure fino al primo '`\0`' incontrato

```
1 char str[5];  
2 printf("Enter a string:\n");  
3 fgets(str, 5, stdin);
```

Il numero dei caratteri
letti sarà **in realtà 4**:
l'ultimo carattere è il
null-termination '`\0`'

`fgets` supporta ogni
tipo di stream, anche
file, perciò lo
standard input deve
essere reso **esplicito**

Input e output: buffer input sporco

18

L'uso alternato di `scanf` e `fgets` può incappare in alcuni problemi:

```
1 int age = 0;
2 char name[50];
3 printf("Enter age and name:\n");
4 scanf("%d", &age);
5 fgets(name, 50, stdin);
```



Output

Enter age and name:
42

...Program finished with exit code 0

Soluzione

`scanf` lascia il carattere **new line** ' `\n` ' nel **buffer di input**, che viene quindi letto dalla seguente `fgets`.
Il suggerimento è quello di usare sempre `fgets` ed effettuare il parsing della stringa tramite `sscanf`.

```
1 int age = 0;
2 char name[50];
3 printf("Enter age and name:\n");
4 fgets(name, 50, stdin);
5 sscanf(name, "%d", &age);
6 fgets(name, 50, stdin);
```



Output

Enter age and name:
42
Mark

...Program finished with exit code 0



4. Esercizi

Esercizi

20

1. Scrivere un programma che legge tre valori da standard input e ne stampi la media aritmetica.
Hint: utilizzare cast esplicito
Esempio: 10, 20, 50 \rightarrow \sim 26.667
2. Riprodurre Esercizio 1 senza utilizzare cast espliciti.
3. Scrivere un programma che legge una data nel formato `gg/mm/aaaa` e la memorizzi in un'apposita `struct Data`, **solo** tramite puntatore
4. Scrivere un programma che risolva il problema di `scanf` visto a lezione, **senza** utilizzare `fgets`.
5. Scrivere un programma che legge due date nel formato `gg/mm/aaaa`, le memorizzi apposite variabili `struct Data`, e restituisca una data intermedia.
Hint: arrotondare per eccesso
Esempio: 10/05/2023, 20/03/1900 \rightarrow 15/04/1962

Extra

6. Scrivere un programma che legge due date nel formato `gg/mm/aaaa` e restituisca se la prima è antecedente (-1), uguale (0) o successiva alla seconda (1).
Esempio: 05/06/2007, 07/07/2023 \rightarrow -1

Riferimenti

21

- <https://en.cppreference.com/w/c/io>
- <https://en.cppreference.com/w/c/program>
- <https://en.cppreference.com/w/c/keyword>
- <https://en.cppreference.com/w/c/language/declarations>
- <https://en.cppreference.com/w/c/language/expressions>
- <https://en.cppreference.com/w/c/language/cast>
- <https://en.cppreference.com/w/c/language/conversion>
- <https://en.cppreference.com/w/c/io/fprintf>
- <https://en.cppreference.com/w/c/io/puts>
- <https://en.cppreference.com/w/c/io/fputs>
- <https://en.cppreference.com/w/c/io/fscanf>
- <https://en.cppreference.com/w/c/io/fgets>