Laboratorio di Reti di Calcolatori

Reti di Calcolatori A.A. 2023/24

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

Michele Mastrogiovanni - Dipartimento di Ingegneria Informatica

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Laboratorio di Reti di Calcolatori

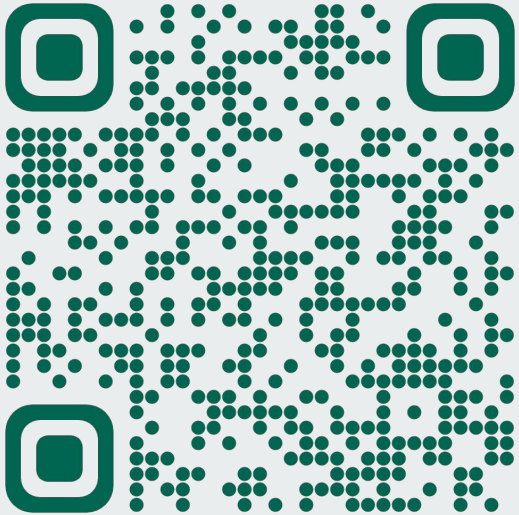
- Reti di Calcolatori: composte da numerosi apparati → capacità computazionali eterogenee
  - Calcolatori (ad es. PC, Workstation, Smartphone), anche chiamati host, end systems
  - Componenti di rete (ad es. Router, Switch, Wireless Access Point)
  - Dispositivi embedded (ad es. Sensori, Smart Objects)
- 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

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Laboratorio di Reti di Calcolatori

Repository GitHub



Slide lezioni e materiale aggiuntivo

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Modalità di esame

Obbligatorio

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

Opzionale

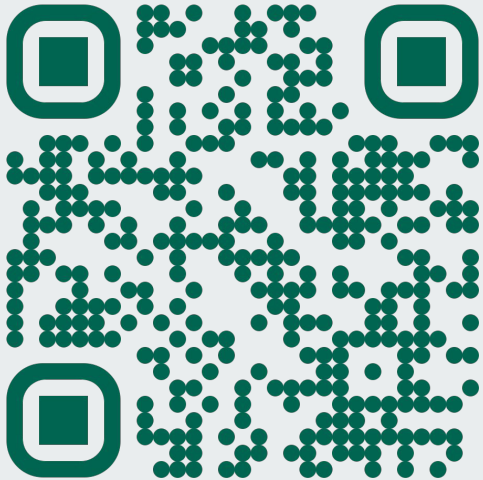
- Progetto sui temi mostrati a lezione di laboratorio
- Fino a +3 punti sul voto finale ottenuto tramite esoneri ed appelli

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

1. Verifica preliminare del linguaggio C

Verifica preliminare del linguaggio C

- Quiz anonimo conoscitivo
- Scopo: adeguare la comunicazione del corso



Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

2. Discussione dei risultati

3. Basi del linguaggio C

Basi del linguaggio C

Caratteristiche del linguaggio C

- Linguaggio compilato
- Tipizzazione statica
- Paradigma imperativo e procedurale

Compilatori

Windows

Qualsiasi IDE[1] 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

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Basi del linguaggio C

Source file: `example.c`

Eseguibile: `example_exec`

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

// This program sums two integers.
int main(int argc, char *argv[]){
  if(argc != 3){
    printf("ERROR: invalid arguments provided.\n");
    printf("Usage: ./example_exec Integer1 Integer2\n");
    return 1;
  }

  int int1 = atoi(argv[1]);
  int int2 = atoi(argv[2]);
  int sum = int1 + int2;
  printf("Sum of %d and %d is equal to %d.\n", int1, int2, sum);

  return 0;
}
```

Direttive al preprocessore

Commento singola linea

Valore di ritorno che indica un errore

Definizione variabili con assegnazione di valori convertiti ad interi da stringhe

Valore 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`)

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Parole chiave riservate in C

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

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

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

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

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

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

_Pragma (C99)

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Dichiarazione e definizione di variabili

Dichiarazione

```
1 int x;
2 int arr[5];
3 int *hugeArr;
```

Variabile intera.
Variabile array di interi con dimensione 5. 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;
2 int arr[5] = {1,2,3,4,5};
3 int *hugeArr = arr;
```

Inizializzazione con integer literal `42`. Dimensione array opzionale.
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.

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Tipi primitivi in C

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

Classificazione	Tipo	Dimensione[2]	Range valori	Format string
Caratteri	`char`	Almeno 8 bit	± 127	`%hhd`
Numeri interi	`short`	Almeno 16 bit	± 32 767	`%hd`
	`int`	Almeno 16 bit	± 2 147 483 647	`%d`
	`long`	Almeno 32 bit	± 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

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

`signed` e `unsigned`

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:

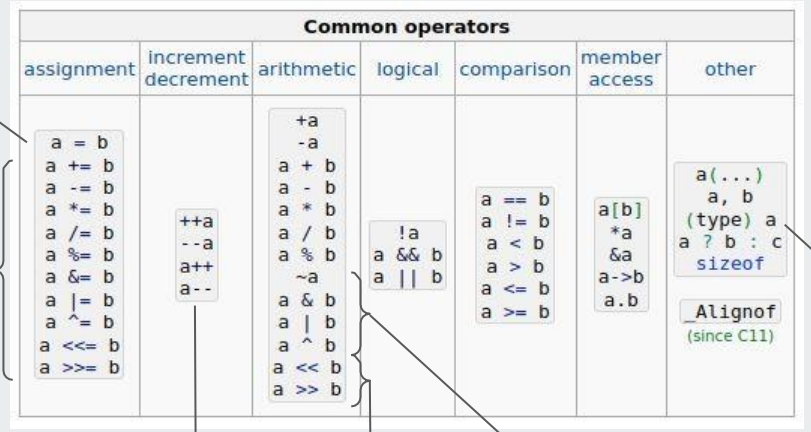
```
1 unsigned short x = 0;
2 x--;
```

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

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

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.

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.

Operatori bit-wise

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

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.

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Input e output

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`

```
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

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

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Input e output: stringhe C


```
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

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Input e output: buffer input sporco

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

```c
int age = 0;
char name[50];
printf("Enter age and name:\n");
scanf("%d", &age);
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`.

```c
int age = 0;
char name[50];
printf("Enter age and name:\n");
fgets(name, 50, stdin);
sscanf(name, "%d", &age);
fgets(name, 50, stdin);
```

Output

```
Enter age and name:
42
Mark

...Program finished with exit code 0
```

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - Luca Iezzi Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

4. Esercizi

Esercizi

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 →
   ~26.667
2. Riprodurre Esercizio 1 senza utilizzare cast espliciti.
3. Scrivere un programma che risolva il problema di `scanf` visto a lezione, senza utilizzare `fgets`.
4. 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 → 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 → -1

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma

Riferimenti

- 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

Ringraziamenti: - Emanuele Giona Dipartimento di Informatica, Sapienza Università di Roma - **Luca Iezzi** Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma