

Laboratorio II

Corso A

Lezione 1

Descrizione del corso

Contatto con C

Esercizi

Introduzione al corso

- Laboratorio II
 - Programmazione C
 - IO, compilazione e librerie, chiamate di sistema, multithreading.
 - Scripting in bash
 - Verilog & Assembly
 - Supporto ai corsi di A & SO, Ricerca Operativa, Paradigmi, Calcolo Numerico, Statistica
 - Lezioni ed esercitazioni

Docenti

Alina Sirbu alina.sirbu@unipi.it

Massimo Torquati massimo.torquati@unipi.it

- Ricevimenti di gruppo - nelle prossime settimane.
- Ricevimento docenti: sui loro siti.
- Sirbu: ricevimento su appuntamento.

Valutazione

Track 1: Prove in itinere + mini-progetto + orale

Prove in itinere: compiti in classe e a casa.

Mini-progetto: compito a casa più consistente.

Orale: discussione del mini-progetto.

||

Track 2: Progetto + orale.

Progetto: copre tutti gli argomenti del corso.

Orale: discussione del progetto e domande aggiuntive.

Piattaforme

- Materiale: Microsoft Teams
 - Link disponibile su esami.unipi.it
 - Slide, codice, registrazioni delle lezioni, news!!!
- Esercizi:
 - <https://evo.di.unipi.it/student/courses/15>
 - Esercizi per ogni lezione - non valutati
 - Compiti - valutazione in itinere
- Sviluppo:
 - Esercizi in classe - portate i vostri laptop
 - Macchina virtuale - laboratorio2.di.unipi.it

Macchina virtuale remota

- Host:
 - laboratorio2.di.unipi.it
- Username & password:
 - credenziali di Alice
- Come connettersi:
 - `ssh` - secure shell - apre una linea di comando in remoto
 - `ftp/sftp` - secure file transfer protocol - trasferimento di file
 - dalla linea di comando o tramite clienti `ssh/ftp/sftp` (PuTTY, FileZilla, VSCode)

DEMO

```
ssh utente@laboratorio2.di.unipi.it
```

Perché C?

Sviluppato dai Bell Labs negli anni 70 per programmare i sistemi UNIX

Dennis Ritchie

Ken Thompson

Brian Kernighan

E' alla base di tutti i sistemi che utilizziamo adesso

Offre accesso a basso livello alle risorse



Efficiente nell'utilizzo delle risorse

Più facile ottimizzare il codice



Perché C?

Ancora un linguaggio molto utilizzato(TIOBE index
Sett 2022, <https://www.tiobe.com/tiobe-index/>)

Sep 2022	Sep 2021	Change	Programming Language		Ratings	Change
1	2	▲		Python	15.74%	+4.07%
2	1	▼		C	13.96%	+2.13%
3	3			Java	11.72%	+0.60%
4	4			C++	9.76%	+2.63%
5	5			C#	4.88%	-0.89%
6	6			Visual Basic	4.39%	-0.22%
7	7			JavaScript	2.82%	+0.27%

Struttura di un programma C

Paradigma imperativo

Strutturato a blocchi - collezione di funzioni

Funzione `main` - l'inizio dell'esecuzione del programma

Struttura

Direttive pre-processore

Dichiarazioni globali

Funzioni



```
#include <stdio.h>

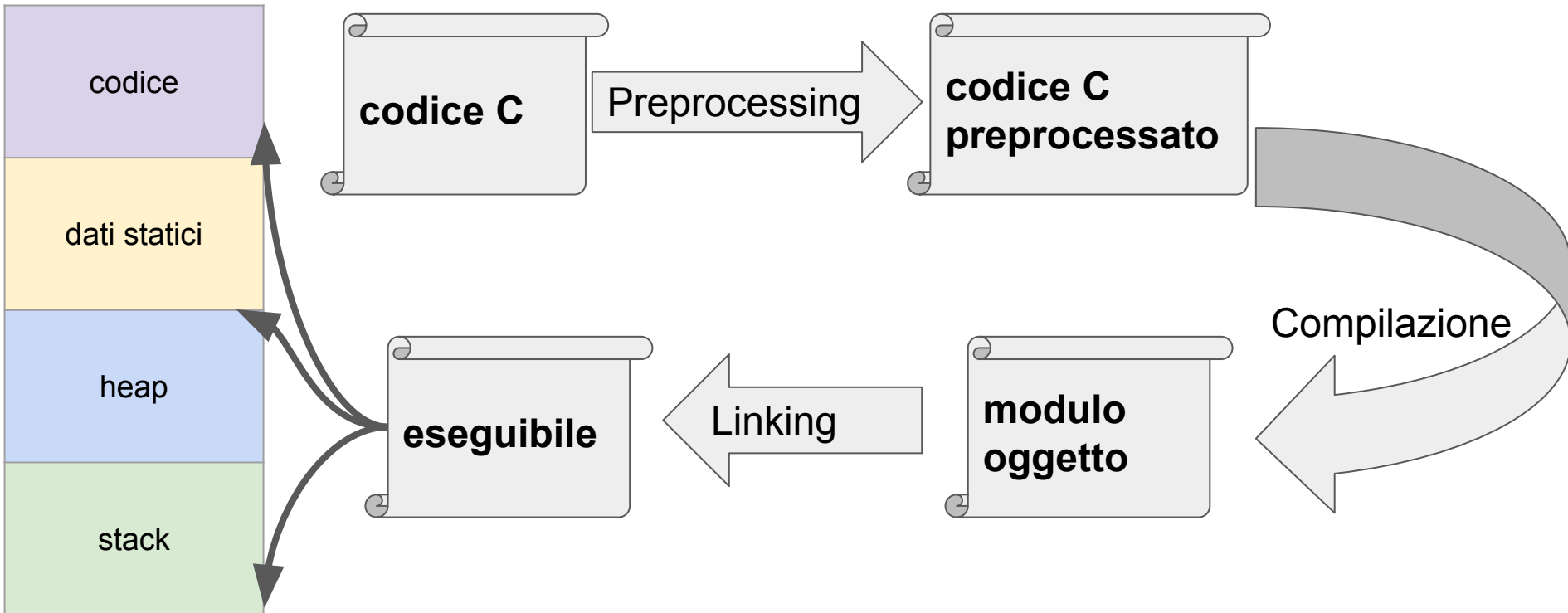
int globalA;

int main(void) {

    printf("Hello world\n");
    return 0;
}
```

The diagram illustrates the structure of a C program. On the left, three labels are listed: 'Direttive pre-processore', 'Dichiarazioni globali', and 'Funzioni'. Arrows point from these labels to the corresponding parts of a C code snippet on the right. 'Direttive pre-processore' points to the `#include <stdio.h>` line. 'Dichiarazioni globali' points to the `int globalA;` line. 'Funzioni' points to the `int main(void) {` line, which marks the start of the main function block.

Dal codice sorgente al eseguibile



Dal codice all'eseguibile

- Il programma è un file di testo
- Per essere eseguito:
 - Pre-processor
 - Eseguite le direttive pre-processor
 - Vengono sostituite alcune informazioni simboliche con contenuti specificate nelle direttive
 - Inclusione di file (e.g. librerie <stdio.h>, altri file "my_lib.c")
 - Definizione di macro (e.g. costanti, comandi semplici)
 - Compilato
 - Il codice sorgente .c viene trasformato in codice oggetto .o
 - Collegato (linking)
 - Collega il file oggetto in un eseguibile.
 - Caricato in memoria

```
#include <stdio.h>

#define CONST_MESSAGE "Hello world\n"

int main(void) {

    printf(CONST_MESSAGE);
    return 0;
}
```

Dal codice oggetto al eseguibile : GCC

- GCC: Gnu Compiler Collection - compilatore mainstream
(<https://alibabatech.medium.com/gcc-vs-clang-llvm-an-in-depth-comparison-of-c-c-compilers-899ede2be378>)

- Esegue preprocessing, compilazione e linking in un solo comando

```
gcc codice.c -o eseguibile.out
```

- Per eseguire l'eseguibile:

```
./eseguibile.out
```

- Documentazione gcc (opzioni -Wall, -pedantic , -g):

```
man gcc
```

<https://gcc.gnu.org/onlinedocs/gcc-8.4.0/gcc/Invoking-GCC.html#Invoking-GCC>

DEMO

Dal codice oggetto al eseguibile : GCC

- Possiamo anche separare le 3 fasi
- Preprocessing

```
gcc -E codice.c -o codiceP.c
```

- Compilazione

```
gcc -c codice.c -o codice.o
```

```
gcc -c codiceP.c -o codice.o
```

- Linking

```
gcc codice.o -o eseguibile.out
```

Tipi di dato

- C è un linguaggio strongly typed
- E' buona prassi definire e inizializzare le variabili all'inizio della funzione/blocco
- Pochi tipi di dato primitivi:
 - char: caratteri 'a', 'A', '1'
 - int: numeri interi 123, -45 (boolean)
 - float: numeri reali 0.45, 23
 - double: numeri reali in precisione doppia
- Qualifier:
 - short, long per int
 - signed, unsigned per char e int
 - long per double
- Lunghezza della rappresentazione dipende dalla macchina
 - operatore sizeof

```
#include <stdio.h>

int main(void) {

    int a = sizeof(int);

    printf("%d\n", a);

    return 0;
}
```

Conversione tra tipi di dato

- Implicita:
 - quando utilizziamo delle espressioni che coinvolgono tipi diversi
 - le variabili di tipo più 'corto' vengono trasformate in variabili di tipo più lungo
 - char,short ->int -> long -> float -> double -> long double
 - quando assegnamo un valore di una variabile ad un'altra variabile di tipo diverso
 - 'vince' il tipo della variabile a sinistra
 - perdita di informazione se a sinistra il tipo è più 'corto'
- Esplicita
 - utilizziamo l'operatore di casting
- Ci sono tipi di dato non compatibili: non possiamo trasformare una stringa in un intero
 - il compilatore ce lo dice : warning o errore

Conversione tra tipi di dato

```
#include <stdio.h>

int main(void) {
    int a=5;
    float b=a;
    printf("a=%d\nb=%f\n", a,b);
    return 0;
}
```

```
#include <stdio.h>

int main(void) {
    int a=5;
    float b=a/2;
    printf("a=%d\nb=%f\n", a,b);
    return 0;
}
```

Conversione tra tipi di dato

```
#include <stdio.h>

int main(void) {
    int a=5;
    float b=a;
    printf("a=%d\nb=%f\n", a,b);
    return 0;
}
```

```
#include <stdio.h>

int main(void) {
    int a=5;
    float b=a/2;
    printf("a=%d\nb=%f\n", a,b);
    return 0;
}
```

a=5
b=2.000000

Conversione tra tipi di dato

```
#include <stdio.h>

int main(void) {
    int a=5;
    float b=a;
    printf("a=%d\nb=%f\n", a,b);
    return 0;
}
```

```
#include <stdio.h>

int main(void) {
    int a=5;
    float b=a/2;
    printf("a=%d\nb=%f\n", a,b);
    return 0;
}
```

a=5
b=2.000000

```
int main(void) {
    int a=5;
    float b=a/2.0;
    printf("a=%d\nb=%f\n", a,b);
    return 0;
}
```

```
int main(void) {
    int a=5;
    float b=(float)a/2;
    printf("a=%d\nb=%f\n", a,b);
    return 0;
}
```

Conversione tra tipi di dato

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

```
int main(void) {  
    int a=5;  
    float b=a;  
    printf("a=%d\nb=%f\n", a,b);  
    return 0;  
}
```

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

```
int main(void) {  
    int a=5;  
    float b=a/2;  
    printf("a=%d\nb=%f\n", a,b);  
    return 0;  
}
```

a=5

b=2.000000

```
int main(void) {
```

```
    int a=5;  
    float b=a/2.0;  
    printf("a=%d\nb=%f\n", a,b);  
    return 0;  
}
```

a=5

b=2.500000

```
int main(void) {
```

```
    int a=5;  
    float b=(float)a/2;  
    printf("a=%d\nb=%f\n", a,b);  
    return 0;  
}
```

a=5

b=2.500000

Comandi condizionali

If-else:

```
if (espressione)
    comando
else
    comando
```

Switch:

```
switch (espressione){
    case espr_costante:
        comando
        break;
    case espr_costante:
        comando
        break;
    default: comando
}
```

Comandi iterativi

WHILE

```
while  
(espressione)
```

commando

DO- WHILE

```
do
```

commando

```
while (espressione)
```

FOR

```
for (espressione1; espressione2; espressione3)
```

commando

```
#include <stdio.h>

int main(void) {

    int n= 12345;

    while (n>0){
        printf("%d\n",n%10);
        n/=10;
    }

    return 0;
}
```

```
#include <stdio.h>

int main(void) {

    int n= 12345;

    for(;n>0;n/=10){
        printf("%d\n",n%10);
    }

    return 0;
}
```

Array

Collezione di valori

Struttura di dati omogenea

Gli elementi hanno lo stesso tipo

Per definire un array bisogna conoscere la sua dimensione (a meno che non usiamo i puntatori, non oggi !)

```
int interi[8];
```

```
#define N 100
```

```
double reali[N];
```

```
float reali1[n]; //funziona solo se n è inizializzato
```

Per accedere ad un elemento utilizziamo l'operatore [] : `a[i]`

Alla dichiarazione viene allocato uno spazio contiguo di dimensione giusta in memoria (sullo stack)

```
#include <stdio.h>

int main(void) {

    int n= 12345;
    int cifre[10]; //non molto bello :)
    int dim=0;

    for(;n!=0;dim++,n/=10){
        cifre[dim]=n%10;
    }

    for(int i=0;i<dim;i++){
        printf("%d\n",cifre[i]);
    }

    return 0;
}
```

Tipi di dato custom - struct

Possiamo definire strutture di dati non omogenee

Parola chiave `struct` - definisce tipi di dati complessi con vari membri

L'ordine dei membri definisce l'ordine in memoria - spazio contiguo.

Per accedere ad un membro utilizziamo la sintassi `variabile.membro`

```
#include <stdio.h>

#define LEN 20

struct Studente {
    char nome[LEN];
    int eta;
};

int main(void) {

    struct Studente s= { "Antonio",15};

    printf("Nome: %s, Età: %d\n",s.nome, s.eta);
    printf("Dimensione in memoria: %lu\n", sizeof(s));

    return 0;
}
```


Tipi di dato custom - struct

Possiamo definire strutture di dati non omogenee

Parola chiave `struct` - definisce tipi di dati complessi con vari membri

L'ordine dei membri definisce l'ordine in memoria - spazio contiguo.

Per accedere ad un membro utilizziamo la sintassi `variabile.membro`

```
#include <stdio.h>

#define LEN 20

struct Studente {
    char nome[LEN];
    int eta;
};

int main(void) {

    struct Studente s= { "Antonio",15};

    printf("Nome: %s, Età: %d\n",s.nome, s.eta);
    printf("Dimensione in memoria: %lu\n", sizeof(s));

    return 0;
}
```

```
> clang-7 -pthread -lm -o main main.c
> ./main
Nome: Antonio, Età: 15
Dimensione in memoria: 24
> []
```

Tipi di dato custom - struct

```
#include <stdio.h>

#define LEN 20

struct Studente {
    char nome[LEN];
    int eta;
};

int main(void) {

    struct Studente s= { 15, "Antonio"};

    printf("Nome: %s, Età: %d\n",s.nome, s.eta);
    printf("Dimensione in memoria: %lu\n", sizeof(s));

    return 0;
}
```

Tipi di dato custom - struct

```
> clang-7 -pthread -lm -o main main.c
main.c:12:28: warning: incompatible pointer to integer conversion
      initializing 'char' with an expression of type 'char [8]'
      [-Wint-conversion]
    struct Studente s= { 15, "Antonio"};
                          ~~~~~
1 warning generated.
> ./main
Nome: T, Età: 0
Dimensione in memoria: 24
> []
```

```
#include <stdio.h>

#define LEN 20

struct Studente {
    char nome[LEN];
    int eta;
};

int main(void) {

    struct Studente s= { 15, "Antonio"};

    printf("Nome: %s, Età: %d\n",s.nome, s.eta);
    printf("Dimensione in memoria: %lu\n", sizeof(s));

    return 0;
}
```

Tipi di dato custom - union

- `struct` definisce un tipo di dato con vari membri, concatenati in memoria, tutti presenti allo stesso tempo
 - funzionalità simile agli oggetti
- A volte necessario avere una variabile che prende valori di tipo diverso, ma solo uno alla volta può essere definito
 - `union`
- una variabile di questo tipo occupa sempre lo stesso spazio in memoria (il membro più lungo)

```
#include <stdio.h>

#define LEN 20

union Studente {
    char nome[LEN];
    int matricola;
};

int main(void) {

    union Studente s={"Antonio"}; //solo primo campo

    printf("Nome: %s, Matricola: %d\n", s.nome,
        s.matricola);
    printf("Dimensione in memoria: %lu\n", sizeof(s));

    s.matricola=1111;
    printf("Nome: %s, Matricola: %d\n", s.nome,
        s.matricola);
    printf("Dimensione in memoria: %lu\n", sizeof(s));

    return 0;
}
```

Tipi di dato custom - union

- `struct` definisce un tipo di dato con vari membri, concatenati in memoria, tutti presenti allo stesso tempo
 - funzionalità simile agli oggetti
- A volte necessario avere una variabile che prende valori di tipo diverso, ma solo uno alla volta può essere definito
 - `union`
- una variabile di questo tipo occupa sempre lo stesso spazio in memoria (il membro più lungo)

```
#include <stdio.h>

#define LEN 20

union Studente {
    char nome[LEN];
    int matricola;
};

int main(void) {

    union Studente s={"Antonio"}; //solo primo campo

    printf("Nome: %s, Matricola: %d\n", s.nome,
        s.matricola);
    printf("Dimensione in memoria: %lu\n", sizeof(s));

    s.matricola=1111;
    printf("Nome: %s, Matricola: %d\n", s.nome,
        s.matricola);
    printf("Dimensione in memoria: %lu\n", sizeof(s));

    return 0;
}
```

```
> clang-7 -pthread -lm -o main main.c
> ./main
Nome: Antonio, Matricola: 1869901377
Dimensione in memoria: 20
Nome: W, Matricola: 1111
Dimensione in memoria: 20
> □
```

Tipi di dato custom - typedef

Le keyword `struct` e `union` vanno sempre ripetute quando si dichiara una nuova variabile

E' comodo dare un nome più corto al nuovo tipo di dato:

```
typedef struct {...} Studente;
```

`typedef` si può usare anche per rinominare dei tipi esistenti.

```
typedef char[] String;
```

Domande?

