Corso di Laurea in *Ingegneria Elettronica, Informatica e delle Telecomunicazioni*

Fondamenti di Programmazione

Matteo Fraschini - DIEE, Edificio M

Presentazione corso

Matteo Fraschini

Ufficio: DIEE, Edificio M, 3° piano

- 📧 matteo.fraschini@unica.it
- 📂 **Informazioni sul corso**: https://web.unica.it/unica/page/it/matteo_fraschini_mat_fondamenti_di_programmazione
- 📰 **Canale Teams**: Fondamenti di Programmazione (t7ee0o4)
- 👥 **Gruppo Telegram**: https://t.me/+U3o4EayKho47ZYl-
- 🖥️ **Youtube channel**: https://www.youtube.com/channel/UCjlOyAkRafPRICvqTEKcJ0A
- 💬 **Ricevimento**: lunedì ore 15 (email)

Cosa vi dovete aspettare da me…

- il vostro apprendimento sarà la mia priorità
- farò il possibile per rispondere nel più breve tempo possibile ai vostri quesiti
- parlerò lentamente
- cercherò di rendere la lezione meno noiosa possibile

Cosa mi aspetto da voi…

- partecipare alle lezioni sarà per voi una priorità
- il massimo rispetto verso i vostri colleghi e verso i docenti
- partecipazione attiva durante le attività del corso

Orario delle lezioni

- Lunedì ore 11
- Martedì ore 12
- Mercoledì ore 11
- Giovedì ore 12
- Venerdì ore 11

Agenda Web: https://unica.easystaff.it/AgendaWeb/index.php?view=home&_lang=it

Corso integrato

Corso integrato di **Sistemi di Elaborazione delle Informazioni**

- `Fondamenti di Programmazione` 6 CFU
- `Calcolatori Elettronici` 6 CFU

📌 Il voto finale corrisponderà alla **media** dei voti dei due moduli

Materiale didattico

Cosa si trova?

- Slides
- Selezione di compiti: testi e soluzioni

> NOTA BENE: Le “soluzioni” proposte in questa sezione vengono fornite con l’esclusiva motivazione di facilitare la preparazione del compito, possono non essere complete e non devono essere considerate definitive o uniche procedure possibili che portino alla soluzione del problema descritto.

Obiettivi.

L’obiettivo principale del corso è quello di fornire allo studente gli strumenti necessari a comprendere i principi fondamentali dell’informatica e della programmazione in C.

Conoscenza e capacità di comprensione.

Lo studente conoscerà i fondamenti dell’informatica come scienza che studia la codifica delle informazioni, gli algoritmi, i principi di funzionamento di sistemi informativi e i fondamenti dei linguaggi di programmazione.

Capacità di applicare conoscenza e comprensione.

Lo studente sarà in grado di comprendere l'organizzazione e la logica del funzionamento dei moderni sistemi informativi, di `sviluppare algoritmi per la soluzione di problemi di media complessità` e di `codificarli in linguaggio C`.

Autonomia di giudizio.

Lo studente sarà in grado di interpretare, valutare ed esprimere giudizi autonomi in relazione a questioni legate ai principi fondamentali dell’informatica e della programmazione in C.

Abilità comunicative.

Lo studente sarà in grado di comunicare informazioni, idee, problemi e soluzioni a interlocutori specialisti e non specialisti.

Capacità di apprendimento.

Lo studente sarà in grado di apprendere metodologie avanzate e nuovi linguaggi di programmazione, applicando con flessibilità i concetti di base forniti nel corso.

Prerequisiti

Conoscenze di base di matematica e algebra. Dimestichezza con l’uso del calcolatore.

Contenuti: teoria

- Rappresentazione delle informazioni.
- Concetto di algoritmo.
- Nozioni di base sull’organizzazione di un calcolatore.
- Nozioni di base sull’organizzazione di un Sistema Operativo.
- Introduzione alla sicurezza informatica.

Contenuti: C prima parte

- Il nucleo del linguaggio C, primi esempi di C.
- Struttura dei programmi in C: dichiarazioni, variabili, costanti, istruzioni.
- Tipi di dato semplici.
- Tipi strutturati: vettori, struct, puntatori.
- Vettori, puntatori, aritmetica dei puntatori.
- Istruzioni di selezione: if-else, switch.
- Istruzioni cicliche: for, do-while.

Contenuti: C seconda parte

- Le funzioni.
- Uso pratico dei sottoprogrammi.
- La gestione della memoria dinamica.
- Operazioni su file in C.

Metodi Didattici

`Lezioni frontali`: 48 ore.

`Esercizi di linguaggio C`: 12 ore.

`Tutorato`: 48 ore.

Verifica dell’apprendimento

- **Organizzazione**: L’esame è articolato in una **prova scritta** e in una **prova orale**. Entrambe le prove sono obbligatorie. La prova scritta è in programma nel giorno e nell’ora indicate su esse3 per l’appello d’esame. La prova orale sarà programmata subito dopo la correzione della prova scritta. *Si sottolinea che potranno sostenere la prova orale solo gli studenti che avranno superato la prova scritta con una valutazione sufficiente (voto non inferiore a 18 trentesimi).*

Verifica dell’apprendimento

- **Valutazione**: Per superare l’esame lo studente dovrà superare entrambe le prove (prova scritta e prova orale) con una valutazione sufficiente (voto non inferiore a 18 trentesimi). Il voto finale (espresso in trentesimi) sarà calcolato - se entrambe le prove risulteranno sufficienti - pesando lo scritto per il 70% e l’orale per il restante 30%.

Verifica dell’apprendimento

- **Contentuo**: La prova scritta riguarda lo svolgimento di esercizi da risolvere attraverso la scrittura di codice C. La prova orale è principalmente orientata a discutere la modalità di svolgimento della prova scritta ma potrà interessare, a discrezione del docente, anche approfondimenti relativi ad altre parti del programma.

Verifica dell’apprendimento

Modalità: Lo studente, dopo aver preso posto in aula, riceverà il testo della prova e dovrà svolgerla in autonomia senza comunicare con nessuno presente in aula o a distanza. Sarà consentito svolgere la prova utilizzando il proprio PC (lo studente si assume la responsabilità di qualsiasi eventuale malfunzionamento del dispositivo), in alternativa lo studente potrà utilizzare carta (fornita dal docente) e penna. L’uso del PC è autorizzato esclusivamente al fine di svolgere l’elaborato. Ogni altro uso è severamente vietato. Prima dello scadere della prova lo studente dovrà consegnare l’elaborato secondo le regole successivamente indicate.

Verifica dell’apprendimento

- **Consegna dell’elaborato**: Lo studente, nel caso abbia svolto la prova al PC, consegnerà l’elaborato tramite l’attività presente nel Team del corso così come illustrato dal docente a lezione. Dovrà essere consegnato esclusivamente il file sorgente (con estensione .c) contenente la soluzione della prova. Lo studente si assume la responsabilità di conoscere la procedura relativa al recupero del file da inviare. Prima dello scadere della prova lo studente dovrà consegnare l’elaborato. Gli studenti che svolgeranno la prova su carta consegneranno l’elaborato al docente sempre entro i tempi prestabiliti.

Verifica dell’apprendimento

Verifica dell’apprendimento

- **Codice d’onore**: Durante la prova lo studente si impegna a non avvalersi dell’aiuto di altre persone, non contattare o tentare di contattare in alcun modo altri allievi, non copiare o osservare le prove di altri allievi e di consegnare il proprio elaborato scritto secondo le modalità previste dal docente. Con la consegna dell’elaborato lo studente si impegna ad accettare il presente codice d’onore. La violazione degli impegni di cui sopra o delle eventuali altre disposizioni indicate dal docente comporta, in ogni caso, l’annullamento della prova.

Verifica dell’apprendimento

- **Prova in itinere, modalità**: Nel rispetto delle date definite dalla Facoltà, si svolgerà una prova in itinere. Tale prova sarà scritta e riguarderà lo svolgimento di esercizi da risolvere attraverso la scrittura di codice C - in relazione solo ed esclusivamente alla parte di programma svolta fino alla data programmata per la prova in itinere.

Verifica dell’apprendimento

- **Prova in itinere, valutazione**: La prova in itinere verrà valutata e allo studente sarà assegnato un punteggio compreso tra 0 e 3. Tale punteggio verrà sommato al voto finale derivante dallo svolgimento della prova scritta e della prova orale (entrambe sempre e comunque obbligatorie). Il punteggio della prova in itinere verrà sommato solo ed esclusivamente nel caso lo studente abbia ottenuto un voto finale sufficiente (voto finale derivante dallo svolgimento della prova scritta e della prova orale non inferiore a 18 trentesimi).

Testi

Bellini, Guidi. **Linguaggio C**. McGraw-Hill

Libri di consultazione:

- Dennis M. Ritchie, Brian W. Kernighan, “Il linguaggio C - Principi di programmazione e manuale di riferimento”, Pearson, 2004 (ISBN: 9788871922003).
- D. Mandrioli et al., “Informatica: arte e mestiere”, Mc. Graw Hill Italia, Milano
- E. Burattini et al., Che C serve? APOGEO
- J. Glenn Brookshear, Fondamenti di Informatica e Programmazione in C. Pearson

Il Tutor

Francesco Aracu: f.aracu@studenti.unica.it

Attività (`56 ore`)

- ricevimento (da definire)
- esercitazioni (da definire)

Prossimi incontri: giovedì 3 (aula B0) e venerdì 4 (aula A) - ore 15-17

Conosciamoci meglio…

Alcune domande… **Mentimeter**

www.menti.com

La mia attività di ricerca

- Brain signal processing
- Connectivity and Network analysis in M/EEG

`Applicazioni avanzate di informatica medica e machine learning` 2 CFU

Cyberchallenge

E’ programma di formazione per i giovani talenti tra i 16 e i 23 anni - ha l’obiettivo di identificare, attrarre, reclutare e collocare la prossima generazione di professionisti della sicurezza informatica

- Presentazione ottobre
- Iscrizioni da novembre

C-hallenge

E’ una competizione di programmazione in C riservata a studenti dell’Università di Cagliari

- Tra Gennaio e Febbraio

Perché dovrei imparare a programmare?

- La tecnologia è ovunque
- Migliora le competenze di logica e di problem-solving
- E’ gratificante
- Mette nelle condizioni di fare qualcosa di nuovo
- Può cambiare la vostra vita

WORLD ECONOMIC FORUM

COMMITTED TO IMPROVING THE STATE OF THE WORLD

The Future of Jobs

Cambiano di conseguenza le competenze e abilità ricercate: nel 2020 il problem solving rimarrà la soft skill più ricercata, ma diventeranno più importanti il pensiero critico e la creatività

Top 10 skills

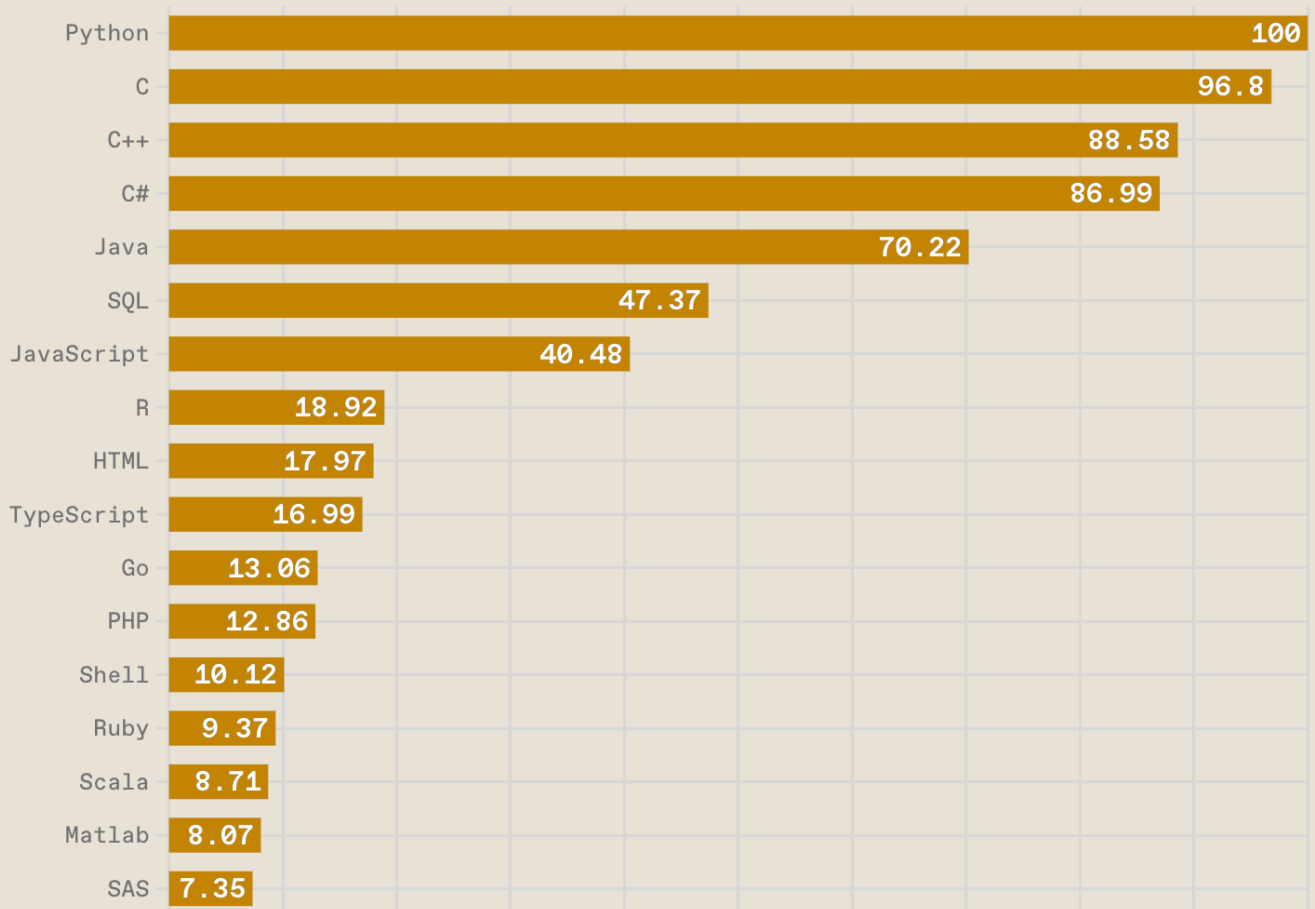
in 2020	in 2015
1. Complex Problem Solving	1. Complex Problem Solving
2. Critical Thinking	2. Coordinating with Others
3. Creativity	3. People Management
4. People Management	4. Critical Thinking
5. Coordinating with Others	5. Negotiation
6. Emotional Intelligence	6. Quality Control
7. Judgment and Decision Making	7. Service Orientation
8. Service Orientation	8. Judgment and Decision Making
9. Negotiation	9. Active Listening
10. Cognitive Flexibility	10. Creativity

Rank	Brand	Brand Value	1-Yr Value Change	Brand Revenue	Industry
1	Apple	$241.2 B	17%	$260.2 B	Technology
2	Google	$207.5 B	24%	$145.6 B	Technology
3	Microsoft	$162.9 B	30%	$125.8 B	Technology
4	Amazon	$135.4 B	40%	$260.5 B	Technology
5	Facebook	$70.3 B	-21%	$49.7 B	Technology
6	Coca-Cola	$64.4 B	9%	$25.2 B	Beverages
7	Disney	$61.3 B	18%	$38.7 B	Leisure
8	Samsung	$50.4 B	-5%	$209.5 B	Technology
9	Louis Vuitton	$47.2 B	20%	$15 B	Luxury
10	McDonald's	$46.1 B	5%	$100.2 B	Restaurants
11	Toyota	$41.5 B	-7%	$187 B	Automotive
12	Intel	$39.5 B	2%	$72 B	Technology

Top Programming Languages 2022

Click a button to see a differently weighted ranking

Spectrum | Jobs | Trending



Sommario degli argomenti

- Introduzione al C
- Le variabili
- Tipo di dati semplici
- Strutture di controllo decisionali
- Strutture di controllo iterative
- Array
- Puntatori
- Funzioni
- Stringhe
- Strutture
- Oggetti dinamici
- File

Introduzione al C

Un po’ di storia…

Nel **1972** Dennis Ritchie sviluppa la prima versione del linguaggio C

Da allora la più significativa estensione è relativa all’introduzione della programmazione orientata agli oggetti (OOP): **C++**

E’ un linguaggio di **alto livello** ma con funzionalità più **tipiche del linguaggio macchina**

Unix/Linux

Un po’ di storia…

1977: presentazione del linguaggio C 1989: pubblicazione dello standard C - C89 1999: pubblicazione dello standard C - C99 2011: pubblicazione dello standard C - C11 2018: pubblicazione dello standard C - C18

IDE – Integrated Development Enviroment

Linux: *gcc*

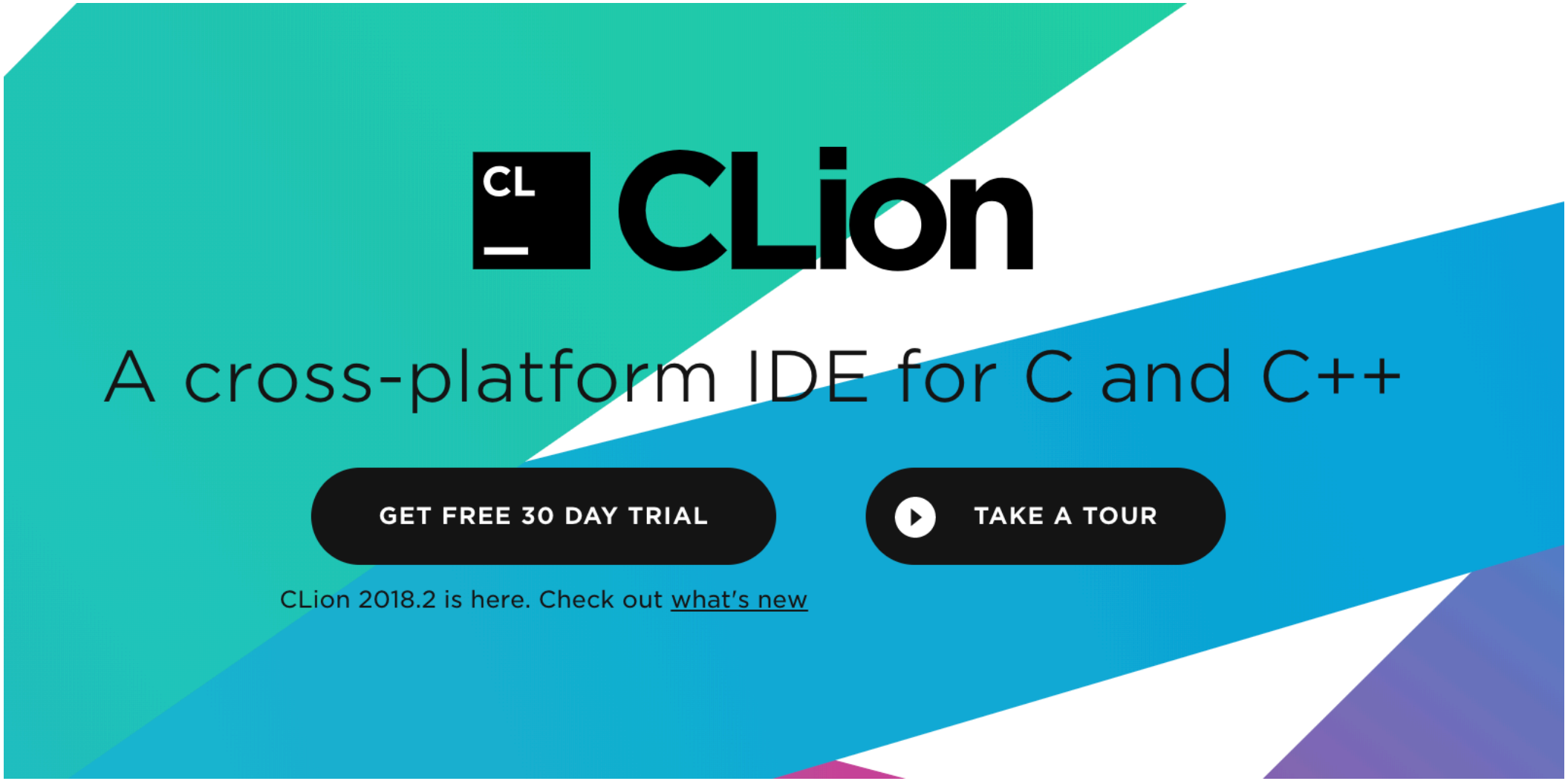
Mac OS: dipendenza da *XCode*

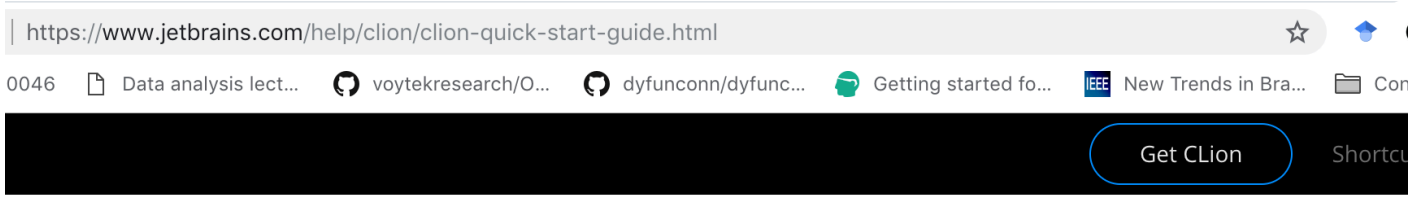
```
gcc -o nome_programma mio_codice.c
```

IDE – Integrated Development Enviroment

Windows (multipiattaforma)

- DEV-C++
- CodeLite
- Code::Blocks
- **Visual Studio Code**: programma, extension, compiler (MinGW per Windows)
- **CLion**: licenza con credenziali istituzionali


CL
CLion
A cross-platform IDE for C and C++
GET FREE 30 DAY TRIAL
TAKE A TOUR
CLion 2018.2 is here. Check out what's new



Which compilers are supported by the IDE?

CLion supports GCC, Clang and Microsoft Visual C++ compilers. This means that on Windows you can select between MinGW (or MinGW-w64), Cygwin and Microsoft Visual Studio tool sets.

What do I need to start with CLion? What are the system requirements?

In general to develop in C/C++ with CLion you need:

- GCC/G++ or Clang, which in case of Windows means using toolchains: MinGW (or MinGW-w64), Cygwin 2.8 (minimum required), or Visual Studio if you are going to use Microsoft Visual C++ compiler instead of GCC/C++ or Clang (refer to our tutorial).

CLion includes bundled GDB (for MinGW on Windows), recent version of LLDB (on Linux and macOS), JDK 1.8 and CMake so you don't need to install them separately. Check the bundled CMake version number in **File | Settings |Build, Execution, Deployment | Toolchains** (or **CLion | Preferences |Build, Execution, Deployment | Toolchains** if you are macOS user).

You can install any of that packages on your system, including custom versions of CMake, compilers and GDB.

The system requirements are:

Windows **macOS** **Linux**

Per smartphone e tablet

- replit.com
- www.onlinegdb.com/online_c_compiler
- vscode.dev

Domande?

www.menti.com

Strumenti utili

- GitHub
- StackOverflow
- ChatGPT

Linguaggi di programmazione

- linguaggi macchina
- linguaggi assembly
- linguaggi di alto livello (compilati o interpretati)

Il linguaggio C

Linguaggio di alto livello

1. Creazione del programma: editor
2. Compilazione: codice oggetto
3. Linking: file eseguibile
4. Loading: caricamento in memoria centrale
5. Esecuzione

Struttura di un programma in C

```c
//eventuali definizioni

int main()
{

//Dichiarazioni

//Istruzioni

return 0;
}
```

Il mio primo programma

```c
//Il mio primo programma in C

#include <stdio.h>

int main()
{
  printf("Hello world");
  return 0;
}
```

Il mio primo programma

```c
/*Il mio primo programma in C*/

#include <stdio.h>

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

Passo passo…

```
/*Il mio primo programma in C*///Il mio primo programma in C
```

rappresentano dei **commenti**

Passo passo...

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

- E' una direttiva al preprocessore.
- Le linee che iniziano con `#` vengono elaborate prima della compilazione del programma.
- Nel programma verrà incluso il contenuto del file `stdio.h`.
- `stdio.h` (standard input-output header) è un file di intestazione che definisce gli stream (flussi) di input e output.

Passo passo...

```
1  int main()
```

- Il `main` è la funzione principale.
- Tutte le funzioni vengono richiamate usando il loro **nome** e tra `()` vengono indicati i parametri.
- `int` indica che la funzione restituisce un intero e le parentesi `{}` definiscono il **corpo** della funzione

Passo passo...

```
1 printf("Hello world\n");
```

- La `printf` è una funzione che scrive sullo standard output (monitor).
- Tra `“”` si può inserire del testo.
- `\n` indica l’inserimento di una nuova linea.
- ogni istruzione deve terminare con un `;`.

Passo passo…

```
1  return 0;
```

- indica che il programma è terminato con successo.

C: variabili e assegnamenti

Dichiarazione di variabili

- E’ il nome (etichetta) di un’area di memoria che il nostro programma potrà sucessivamente utilizzare.
- Dichiarazione delle variabili: definizione delle aree di memoria che saranno utilizzate nel programma.
- Assegnazione di un **nome** (identificatore) e attribuzione del **tipo**.
- Il tipo definisce le caratteristiche che regolano l’uso di una variabile.

```
1 tipo nomevariabile;
```

```
1 tipo nomevariabile1, nomevariabile2;
```

Dichiarazione di variabili

```
tipo nomevariabile1, nomevariabile2;
```

```
int i;
int a,b;
char c;
```

dove `int` e `char` rappresentano due diversi tipi

NB: il C è *case sensitive*! NB: scelta del nome delle variabili

Dichiarazione di costanti

Viene assegnato un valore in maniera permanente

```
1 const float PiGreco = 3.14;
2 const int N = 100;
```

```
1 #define N 100
2
3 //direttiva al preprocessore
4 //non e' scope-controlled
```

Differenze: vantaggi e svantaggi

Istruzione di assegnamento

Viene utilizzato il simbolo **=**

Permette di assegnare ad una variabile un valore o il risultato di una espressione (*o il valore restituito da una funzione*).

identificatore **=** valore

```
1 a=0;
2 c='a';
3 x=x+1; //Come si interpreta?
```

NB: da non confondere con il simbolo **==** (*uguale a*)!

Operatori aritmetici

`+` addizione

`-` sottrazione

`*` moltiplicazione

`/` divisione

`%` modulo (resto della divisione)

Istruzioni di input e output

Standard output: `printf()` Standard input: `scanf()`

Istruzione printf

Stampa su video

```c
printf("Ciao"); //stampa a video: Ciao

printf("Ciao\n"); //stampa a video: Ciao
```

Necessita dell'inclusione della libreria `stdio.h`

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

Istruzione printf

Stampa su video

```c
#include <stdio.h>

int main()
{
  int var=1;
  char c='a';
  printf("%d\n",var);
  //%d formato di stampa: intero sistema decimale
  printf("%c\n",c);
  //%c formato di stampa: carattere
  return 0;
}
```

Istruzione printf

```c
#include <stdio.h>

int main()
{
  int base=2;
  int altezza=4;
  int area;

  printf("La base vale: %d\n",base);
  printf("L'altezza vale: %d\n",altezza);

  area = base * altezza;

  printf("L'area del rettangolo vale: %d\n",area);

  return 0;
}
```

Istruzione printf

```c
#include <stdio.h>

int main()
{
  int base=2;
  int altezza=4;

  printf("Base: %d Altezza: %d\n",base,altezza);
  printf("L'area vale: %d\n",base * altezza);

  return 0;
}
```

Istruzione scanf

Inserimento di valori (da tastiera)

```
1  scanf("%d", &base);
```

! Esercizio 1

Scrivere un programma in C che permetta di calcolare l’area di un rettangolo. La base e l’altezza devono essere letti da tastiera.

✅ Soluzione 1

```c
#include <stdio.h>

int main()
{
  int base, altezza;

  printf("\nInserisci la base: ");
  scanf("%d", &base);

  printf("\nInserisci l'altezza: ");
  scanf("%d", &altezza);

  printf("\nL'area vale: %d",base * altezza);

  return 0;
}
```

Istruzione scanf

Inserimento di valori (da tastiera)

```
1  scanf("%d", &base);
```

Cosa indica il simbolo &? Perchè si usa il simbolo &? Cosa succede se non uso il simbolo &?

Istruzione scanf

Inserimento di valori (da tastiera)

```
1  scanf("%d", &base);
```

- `&` indica l'indirizzo di memoria della variabile `base`
- `&` si usa perché la scanf (il parametro formale è un puntatore) si aspetta un indirizzo (*C: passaggio per valore*)
- se non utilizzo il simbolo `&` il programma si comparta in modo *anomalo*. Perché?

! Esercizio 2

Scrivere un programma in C che esegua la somma di due numeri interi

✅ Soluzione 2

```c
#include <stdio.h>

int main() {
  int n, m, somma;

  printf("\nInserisci un numero: ");
  scanf("%d", &n);

  printf("\nInserisci un altro numero: ");
  scanf("%d", &m);

  //scanf("%d%d", &n,&m);

  somma=n+m;
  printf("\n%d + %d= %d",n,m, somma);

  return 0;
}
```

Espressioni aritmetiche e precedenze

```c
#include <stdio.h>

int main()
{
  int n1=4, n2=2;
  float f1=4,f2=2;

  printf("1 + n1 / 2 * n2 = %d", 1 + n1 / 2 * n2); //5
  printf("\n");
  printf("1 + f1 / 2 * f2 = %f", 1 + f1 / 2 * f2); //5.0
  printf("\n");

  return 0;
}
```

Istruzioni composte

Unione di operatori aritmetici e assegnazione:

`operatore_aritmetico=` (“`+=`”)

```
count += 10;
```

somma l’espressione a destra con l’espressione a sinistra e assegna alla variabile a sinistra

```
count = count + 10;
```

! Esercizio 3

Calcolare diametro, circonferenza e aria di un cerchio a partire dal valore del raggio.

$$\mathbf{r} = \textbf{Raggio}$$

$$\pi = \textbf{Pi Greco}$$

$$\textbf{Circonferenza} = 2 * \pi * \mathbf{r}$$

$$\textbf{Area} = \mathbf{r}^2 * \pi$$

$$\textbf{Area} = \frac{\textbf{Circonferenza} * \textbf{Raggio}}{2}$$

✅ Soluzione 3

```c
#include <stdio.h>

int main()
{
  const float Pi=3.14;
  float raggio, diametro, circ, area;

  printf("Inserisci raggio: ");
  scanf("%f",&raggio);

  diametro=2*raggio;
  circ=diametro*Pi;
  area=Pi*raggio*raggio;

  printf("Diametro: %f ",diametro);
  printf("Circonferenza: %f ",circ);
  printf("Area: %f ",area);

  return 0;
}
```

Conversione

```c
#include <stdio.h>
int main()
{
    float f1=3.7, f2;
    int i1,i2=-10;

    i1=f1; //conversione da float ad intero: 3
    printf("%f -> ad intero produce: %d\n", f1, i1);

    return 0;
}
```

Conversione

```c
#include <stdio.h>
int main()
{
    float f1=3.7, f2;
    int i1,i2=-10;

    f1=i2; // conversione da intero a float: -10.000
    printf("%d -> a float produce: %f\n", i2, f1);

    return 0;
}
```

Conversione

```c
#include <stdio.h>
int main()
{
    float f1=3.7, f2;
    int i1,i2=-10;

    f1=i2/3; // divisione tra interi: -3.000
    printf("%d diviso 3 produce: %f\n", i2, f1);

    return 0;
}
```

Conversione

```c
#include <stdio.h>
int main()
{
    float f1=3.7, f2;
    int i1,i2=-10;

    f2=i2/3.0; // intero diviso float: -3.333
    printf("%d diviso 3.0 produce: %f\n",i2,f2);

    return 0;
}
```

Conversione

```c
#include <stdio.h>
int main()
{
    float f1=3.7, f2;
    int i1,i2=-10;

    f2=(float) i2/3; //cast: -3.333
    printf("(float) %d diviso 3 produce: %f\n",i2,f2);
    return 0;
}
```

Tipi di dato semplici

Tipi di dato

- Il **tipo di dato** specifica un insieme di *valori* e un insieme di possibili *operazioni* che possono essere applicate ad una variabile
- Ogni tipo ha una propria rappresentazione (codifica) in memoria attraverso un'opportuna sequenza di bit
- I linguaggi di programmazione permettono un certo livello di *astrazione* (possiamo ignorare come l'informazione viene rappresentata in memoria)
- Ogni variabile deve avere un tipo (assegnato dalla dichiarazione)

Classificazione dei tipi di dato

- **Tipi semplici**: per rappresentare informazioni semplici
- **Tipi strutturati**: per rappresentare informazioni costituite da diverse componenti

Tipi semplici

- **int** (interi)
- **float** (reali)
- **double** (reali con precisione doppia)
- **char** (caratteri)

Ogni tipo è associato ad un intervallo di valori che è legato alla quantità di memoria allocata (*dipende dalla macchina*)

Tipo int - interi

```
1  int n;
```

La dimensione di una variabile di tipo int (generalmente) è di 4 byte.

```
1  int n = sizeof(int);
```

signed o *unsigned*

short o *long*

`printf("%ld",...)` -> long

`printf("%hd",...)` -> short

Tipo float

```
1  float n;
```

Numeri con parte frazionaria: virgola mobile (*floating point*)

La dimensione di una variabile di tipo float (generalmente) è di 4 byte.

`printf("%f",...)` -> float

`printf("%.3f",...)` -> .000 float

NB: problema dell'approssimazione! 😕

Tipo double

Permette di avere una maggiore precisione

La dimensione di una variabile di tipo double (generalmente) è di 8 byte

Riduce l’effetto degli errori di approssimazione (arrotondamento)

Tipo char: i caratteri

I caratteri possono assumere valori alfanumerici

La dimensione di una variabile di tipo char (generalmente) è di 1 byte.

```
1 char n;
2 n = 'A';
```

`printf("%c",...)` -> char

Altre funzioni standard: `getchar()` e `putchar()`

```
1 x = getchar();
2 putchar(x);
```

Strutture di controllo decisionali

Operatori relazionali

- `==` uguaglianza
- `!=` diversità
- `<` minore di
- `>` maggiore di
- `<=` minore o uguale a
- `>=` maggiore o uguale a

Espressione logica

Espressione che genera un risultato `VERO` o `FALSO`

In C: - 1 `VERO` - 0 `FALSO`

Qualsiasi valore diverso da 0 viene valutato come VERO

Operatori logici

Permettono di concatenare più espressioni logiche

- `!` not
- `&&` and
- `||` or

Tavola di verità

x	y	x && y	x || y	!x
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Alcuni esempi utili

Le espressioni logiche restituiscono un valore numerico!

```
a = i<100;
```

a vale 1 se i<100, vale 0 se i>=100

ATTENZIONE: differenza tra operatore = e == 🚫

if

- Consente di eseguire un'istruzione al verificarsi di una condizione
- Oppure di eseguire **in alternativa** un'altra istruzione

```
if (condizione) {istruzioni} else {istruzioni}
```

> else è opzionale

Se la condizione è seguita da una sola istruzione le parentesi graffe possono essere omesse ⚡

if: esempio

```
1  if(x==0) printf("x vale 0");
```

Esercizio

Scrivere un programma in C che permetta di valutare se un numero intero (letto da tastiera) è pari o dispari

Soluzione

```c
#include <stdio.h>

int main()
{
  int n;

  printf("Inserisci un numero intero: ");
  scanf("%d",&n);

  if(n%2 == 0)
    printf("\n%d e' pari\n",n);
  else
    printf("\n%d e' dispari\n",n);

  return 0;
}
```

Esercizio

Scrivere un programma in C che permetta di valutare se un numero intero (letto da tastiera) è positivo o negativo

Soluzione

```c
#include <stdio.h>

int main()
{
  int n;

  printf("Inserisci un numero intero: ");
  scanf("%d",&n);
  if(n>=0)
    printf("\n%d e' positivo\n",n);
  else
    printf("\n%d e' negativo\n",n);

  return 0;
}
```

Esercizio

Scrivere un programma in C che permetta di verificare se un anno (letto da tastiera) è bisestile.

(divisibile per 4 && non divisibile per 100) || divisibile per 400

Soluzione

```c
#include <stdio.h>

int main()
{
   int anno;

   printf("Inserisci l'anno: ");
   scanf("%d",&anno);

   if ((anno%4 == 0 && anno%100 != 0) || anno%400 == 0)
      printf("\n%d e' bisestile\n", anno);
   else
      printf("\n%d non e' bisestile\n", anno);

   return 0;
}
```

if annidati

```c
//DUMMY EXAMPLE
#include <stdio.h>

int main()
{
   int test1=1,test2=2;
   if(test1==1)
       if(test2==2)
           printf("I test sono verificati\n");
        else
            printf("Test2 non e' verificato\n");
   else
        printf("Test1 non e' verificato\n");

   return 0;
}
```

Ambiguità: `else` è sempre associata all'ultima istruzione `if` senza else

switch

- Il costrutto `if-else` talvolta risulta innaturale.
- Lo `switch` permette di selezionare un'istruzione (tra un insieme di istruzioni possibili) in base al valore assunto una variabile (o da un'espressione)
- Confronto tra risultato espressione (`int` o `char`) e un insime di valori costanti (`int` o `char`).

switch

```
switch (espressione)
{
    case costante1:
            istruzione;
                ...
                break;
        case costante2:
            istruzione;
                ...
                break;
       default:
            istruzione;
}
```

switch

```
switch (espressione)
{
    case costante1:
            istruzione;
                ...
            break;
        case costante2:
        case costante3:
            istruzione;
                ...
            break;
       default:
            istruzione;
}
```

switch

```
switch (espressione)
{
    case costante1:
            istruzione;
                ...
        case costante2:
            istruzione;
                ...
                break;
       default:
            istruzione;
}
```

switch

```c
#include <stdio.h>
int main()
{
  int giudizio;
  printf("Inserisci un giudizio tra 1-3\n");
  scanf("%d",&giudizio);
  switch (giudizio)
  {
    case 1:
        printf("Hai assegnato 1!\n");
        printf("Ci dispiace...\n");
        break;
    case 2:
        printf("Hai assegnato 2!\n");
        printf("Grazie!\n");
        break;
    case 3:
        printf("Hai assegnato 3!\n");
        printf("Grazie mille!.\n");
        break;
    default:
        printf("Giudizio non valido!\n");
  }
  return 0;
}
```

Esercizio

Scrivere un programma in C che permetta di valutare se un numero intero è pari o dispari (si usi switch-case)

Soluzione

```c
#include <stdio.h>
int main()
{
  int n;
  printf("Inserisci un numero intero \n");
  scanf("%d",&n);

  switch (n % 2)
  {
    case 0:
        printf("Il numero %d e' pari\n",n);
        break;
    case 1:
      printf("Il numero %d e' dispari\n",n);
        break;
  }
  return 0;
}
```

Esercizio

Creare una calcolatrice che presi due numeri da tastiera permetta di calcolare in base alla scelta effettuata dall'utente:

- somma
- sottrazione
- moltiplicazione
- divisione

Soluzione

```c
#include <stdio.h>
int main()
{
  int n1,n2;
  char oper;

  printf("Inserisci un primo numero \n");
  scanf("%d",&n1);

  printf("\nDefinisci l'operazione \n");
  printf("+ - * /\n");
  scanf(" %c",&oper);

  printf("\nInserisci un secondo numero \n");
  scanf("%d",&n2);
```

Soluzione

```
  switch (oper)
  {
    case '+':
        printf("%d %c %d: %d\n",n1,oper,n2,n1+n2);
        break;
    case '-':
        printf("%d %c %d: %d\n",n1,oper,n2,n1-n2);
        break;
    case '*':
        printf("%d %c %d: %d\n",n1,oper,n2,n1*n2);
        break;
    case '/':
        printf("%d %c %d: %d\n",n1,oper,n2,n1/n2);
        break;
    default:
        printf("Scelta non valida!\n");
  }
  return 0;
}
```

Esercizio

Creare un programma in C che presi in ingresso il peso (in Kg) e l'altezza (in m) di una persona, calcoli il BMI.

$$BMI = peso/(altezza^2)$$

Il programma dovrà successivamente assegnare una delle seguenti tre classi:

- sottopeso se BMI <= 20
- normopeso se 20<BMI<=30
- sovrappeso se BMI >30

Soluzione

```c
#include <stdio.h>
int main()
{
    float altezza,peso,bmi;
    printf("Inserisci altezza in m:\n");
    scanf("%f",&altezza);
    printf("Inserisci peso in Kg:\n");
    scanf("%f",&peso);
    bmi=peso/(altezza*altezza);
    if(bmi<=20) printf("BMI: %.2f sottopeso\n",bmi);
    else if (bmi>20 && bmi<=30)
          printf("BMI: %.2f Normopeso\n",bmi);
    else printf("BMI: %.2f Sovrappeso\n",bmi);
    return 0;
}
```

Esercizio

Creare un programma in C che permetta di calcolare a scelta l’area di un cerchio, di un rettangolo o di un triangolo.

Soluzione

```c
#include <stdio.h>
int main()
{
  int sel,n,m;
  const float Pi=3.14;

  printf("Calcolo area\n");
  printf("1: cerchio; ");
  printf("2: rettangolo; ");
  printf("3: triangolo.\n");
  scanf("%d",&sel);

```

Soluzione

```
  switch (sel)
  {
    case 1:
      printf("Raggio: \n");
      scanf("%d",&n);
      printf("Area cerchio: %.2f\n",Pi*n*n);
      break;
    case 2:
      printf("Lati: \n");
      scanf("%d%d",&n,&m);
      printf("Area rettangolo: %d\n",n*m);
      break;
    case 3:
      printf("Base e altezza: \n");
      scanf("%d%d",&n,&m);
      printf("Area triangolo: %.2f\n",n*m*.5);
      break;
    default:
      printf("Scelta non valida!\n");
  }
  return 0;
}
```

Strutture di controllo iterative

Iterazione

Permette di sviluppare programmi con processi ripetitivi (**cicli**), evitando di scrivere più volta la/e stessa/e istruzione/i.

Iterazione: cosa non fare!

```c
#include <stdio.h>
int main()
{
  int n,somma=0;

  scanf("%d",&n);
  somma=somma+n;
  scanf("%d",&n);
  somma=somma+n;
  scanf("%d",&n);
  somma=somma+n;
  printf("%d",somma);
}
```

Iterazione: cosa fare!

```c
#include <stdio.h>
int main()
{
  int i,n,somma=0;

  for(i=1;i<=3;i=i+1)
  {
    scanf("%d",&n);
    somma=somma+n;
  }
  printf("%d",somma);
}
```

For

```
for(inizializzazione;condizione;incremento/dec
istruzione;
```

- **inizializzazione**: viene eseguita *solo* la prima volta
- **condizione**: condizione necessaria affinché il ciclo continui
- **incremento/decremento**: ultima espressione del ciclo

For

NB: l’incremento/decremento viene svolto dopo che il corpo del ciclo è stato eseguito!!!

For

Attenzione ai cicli infiniti!

```
for(i=1;i>=1;i=i+1)
```

For

```c
#include <stdio.h>
int main()
{
  int i,n,somma=0;

  for(i=1;i<=3;i=i+1)
  {
    printf("\nInserisci numero %d: ",i);
    scanf("%d",&n);
    somma=somma+n;
  }
  printf("\nLa somma vale: %d\n",somma);
}
```

Operatore: incremento/decremento

`i++;` equivale a `i=i+1;` `i--;` equivale a `i=i-1;`

Attenzione:

`b=2; a=++b;` b=3 e a=3 `b=2; a=b++;` b=3 e a=2

Esercizio

Scivere un programma che fornisca la somma dei primi N numeri naturali. N è definito dall’utente.

Soluzione

```c
#include <stdio.h>
int main()
{
  int i,n,somma=0;

  printf("Quanto vale N? ");
  scanf("%d",&n);
  for(i=1;i<=n;++i)
    somma+=i;
  printf("Somma dei natutali fino a %d: %d\n",n, somma);
}
```

Esercizio

Scrivere un programma che calcoli la media di N numeri. N è definito dall'utente.

Soluzione

```c
#include <stdio.h>
int main()
{
  int i,n,num,somma=0;
  float media;

  printf("Quanti n vuoi inserire? ");
  scanf("%d",&n);
  for(i=1;i<=n;++i)
  {
    printf("\nInserisci numero %d: ",i);
    scanf("%d",&num);
    somma+=num;
  }
  media=(float)somma/n;
  printf("\nLa media vale: %f\n",media);
}
```

Esercizio

Scrivere un programma che calcoli la tabellina del numero N. N è definito dall'utente.

Soluzione

```c
#include <stdio.h>
int main()
{
  int i,n;

  printf("Inserisci un numero ");
  scanf("%d",&n);
  for(i=1;i<=10;++i)
    printf("\n%d * %d = %d",i,n,i*n);
}
```

Esercizio

Scrivere un programma che calcoli le tabelline dei numeri tra l’1 e il 12 e stampi la tabella risultante.

Soluzione

```c
#include <stdio.h>
int main()
{
  int i,j;

  for(i=1;i<=12;++i)
  {
    for(j=1;j<=12;++j)
      printf("%3d ",i*j);
    printf("\n");
  }
}
```

Esercizio

Scrivere un programma che permetta di inserire N voti, calcoli la media e conti il numero di voti sopra una soglia definita dall’utente

Soluzione

```c
#include <stdio.h>
int main()
{
  int n,i,voto,somma=0,soglia,sopra=0;

  printf("Quanti voti?");
  scanf("%d",&n);
  printf("Che soglia?");
  scanf("%d",&soglia);
  for(i=1;i<=n;++i)
  {
    printf("Inserisci voto %d: ",i);
    scanf("%d",&voto);
    somma+=voto;
    if(voto>soglia) ++sopra;
  }
  printf("\nLa media e': %.1f\n",(float)somma/n);
  printf("\n%d voti sopra %d", sopra,soglia);
}
```

While

`while(condizione) {istruzioni;}`

Il ciclo `while` impone la ripetizione del blocco di istruzioni fino a quando la *condizione* non diventa FALSE - valutazione della condizione - se condizione è FALSE -> non viene eseguito il corpo del ciclo - si valuta nuovamente la condizione e così via…

L'inizializzazione della variabile che contralla il ciclo deve essere fatta prima del `while`

L'incremento sarà eseguito come ultima istruzione del corpo del `while`

While

Esempio:

```c
#include <stdio.h>
int main()
{
  int i=1,n,somma=0;

  while(i<=3)
  {
    printf("\nInserisci numero %d: ",i);
    scanf("%d",&n);
    somma=somma+n;
    i++;
  }
  printf("\nLa somma vale: %d\n",somma);
}
```

Esercizio

Scrivere un programma che calcoli la media di N numeri utilizzando `while`

Soluzione

```c
#include <stdio.h>
int main()
{
  int i=1,n,num,somma=0;
  float media;

  printf("Quanti n vuoi inserire? ");
  scanf("%d",&n);
  while(i<=n)
  {
    printf("\nInserisci numero %d: ",i);
    scanf("%d",&num);
    somma+=num;
    ++i;
  }
  media=(float)somma/n;
  printf("\nLa media vale: %f\n",media);
}
```

Esercizio

Scrivere un programma che verifichi se un anno è bisestile usando `while`

(divisibile per 4 `&&` non divisibile per 100) `||` divisibile per 400

Soluzione

```c
#include <stdio.h>

int main()
{
   int anno;

   printf("Inserisci l'anno ");
   printf("(0 per uscire): ");
   scanf("%d",&anno);
   while(anno>0)
   {
     if ((anno%4 == 0 && anno%100 != 0) || anno%400 == 0)
        printf("\n%d e' bisestile\n", anno);
     else
        printf("\n%d non e' bisestile\n", anno);
     printf("Inserisci l'anno ");
     printf("(0 per uscire): ");
     scanf("%d",&anno);
   }
}
```

Esercizio

Scrivere un programma che calcoli diametro, circonferenza e area usando il `while`

Soluzione

```c
#include <stdio.h>
int main()
{
  const float Pi=3.14;
  float raggio, diametro, circ, area;
  char sel='Y';

  while(sel=='Y')
  {
    printf("Inserisci raggio: ");
    scanf("%f",&raggio);
    diametro=2*raggio;
    circ=diametro*Pi;
    area=Pi*raggio*raggio;
    printf("Diametro: %f ",diametro);
    printf("Circonferenza: %f ",circ);
    printf("Area: %f ",area);
    printf("\nY per continuare, N per uscire\n");
    scanf(" %c",&sel);
  }
}
```

Esercizio

Scrivere un programma che calcoli il BMI usando il `while` e verificare i valori per la ripetizione e per l'uscita dal programma

Soluzione

```c
#include <stdio.h>
int main() {
    float altezza,peso,bmi;
    char sel='Y';
    while(sel!='N') {
        printf("Inserisci altezza in m:\n");
        scanf("%f",&altezza);
        printf("Inserisci peso in Kg:\n");
        scanf("%f",&peso);
        bmi=peso/(altezza*altezza);
        if(bmi<=20)
            printf("BMI: %.2f; sottopeso\n",bmi);
            else if (bmi>20 && bmi<=30)
                printf("BMI: %.2f; normopeso\n",bmi);
                else printf("BMI: %.2f; sovrappeso\n",bmi);
        printf("N per uscire, Y per continuare: \n");
        scanf(" %c",&sel);
        while(sel!='N' && sel!='Y')     {
            printf("Scelta non valida \n");
            printf("N per uscire, Y per continuare: \n");
            scanf(" %c",&sel);
        }}}
```

Do While

Le istruzioni `for` e `while` verificano la condizione prima che il ciclo venga eseguito. **Il corpo del ciclo potrebbe non essere eseguito mai!**

Do While

Se vogliamo verificare la condizione alla fine del ciclo possiamo usare il `do-while`.

```
do istruzione; while(espressione);
```

```c
#include <stdio.h>
int main()
{
  int n=0;

  do printf("%d ",n++);
  while(n<=9);
}
```

Do While

Attenzione!

```c
#include <stdio.h>
int main()
{
  int n=0;

  do printf("%d ",n++);
  while(n<=9);
}
```

```c
#include <stdio.h>
int main()
{
  int n=0;

  do printf("%d ",++n);
  while(n<=9);
}
```

Esercitazione

Indovina il numero

Creare un programma che simuli un gioco dove l'utente deve indovinare un numero.

Il programma genera un numero casuale e all'utente viene chiesto di indovinarlo.

L'utente avrà solo 5 tentativi e il programma lo aiuterà dicendo se il numero da indovinare è più grande o più piccolo di quello inserito.

Indovina il numero - prerequisito

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

int main() {
    int n_rand, max = 100, min = 1;

    // srand() usa l'orario attuale...
    srand(time(0));
    // rand() genera pseudo-random numbers, da 0 a RAND_MAX
    // printf("\n%d", RAND_MAX);
    // rand() % (max - min + 1) rende un numero tra 0 e (max - min)
    // printf("\n%d",rand() % (max - min + 1));
    n_rand = (rand() % (max - min + 1)) + min;
    printf("\n%d", n_rand);

    return 0;
}
```

Indovina il numero - soluzione

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

int main() {
    int n_rand, max = 100, min = 1;
    int n_t = 5, i = 1, num, check = 0;

    srand(time(0));
    n_rand = (rand() % (max - min + 1)) + min;
    while(i <= n_t && check == 0){
        printf("\nInserisci un numero tra %d e %d: ", min, max);
        scanf("%d",&num);
        if(num == n_rand) {
            printf("\nBENE! HAI INDOVINATO!");
            check = 1;
        }
        else if(num > n_rand)
            printf("\nHai inserito un numero troppo grande!");
        else printf("\nHai inserito un numero troppo piccolo!");
        i++;
    }
    if(check == 0) printf("\n\nMi dispiace hai perso!");
    return 0;
}
```

Array

Array

E’ un tipo di struttura dati che può memorizzare una raccolta **sequenziale** di elementi dello stesso tipo

Tipo derivato: deve far riferimento ad un tipo *base*

- Quando si usa un array ?
  - Quando abbiamo necessità di trattare un **insieme omogeneo** di dati !

Array

E’ quindi una **variabile strutturata**, i cui elementi sono tutti dello stesso tipo (*tipo del vettore*)

Dichiarazione di un array:

- Indicare nome, tipo e lunghezza 💡

👇

```
1  int vett[10];
```

Array

Accedere agli elementi di un array:

- Tutti gli elementi di un array hanno lo stesso nome: `nome_array`
- Ogni elemento è identificato da un indice: `nome_array[indice]`

Esempio:

`vett[2]` //permette di accedere all'elemento con indice 2

Array

- Il primo elemento di un array ha indice `0` ❗
- L'ultimo elemento di un array ha indice `N-1`, dove `N` indica la dimensione dell'array

```
[0] [1] [2] ... [N-1]
```

Tentare di accedere con un indice diverso (fuori dall'intervallo) può non generare un errore 😕

Array

Assegnamento

```
1 int vett[10], num;
2
3 vett[0]=5;
4
5 num=vett[0]; //assegna 5 a num
```

Array

Permettono di sviluppare programmi concisi ed efficienti - Esempio:

```
1  int voti[10],somma=0,i;
2
3  ...
4
5  for(i=0; i<10; ++i) // <- i parte da 0
6  {
7      printf("Inserisci voto: ");
8          scanf("%d",&voti[i]); // <- conserviamo i voti
9          somma += voti[i];
10 }
```

Array

Considerazione importante!

con `int voti[10];` stiamo dicendo che la lunghezza dell'array sarà 10!

- **Se volessimo inserire 20 elementi?** Modifica noiosa e rischiosa!; Cambiare ogni occorrenza!; Possibili diversi significati del numero 10!

Utilizzo della direttiva `#define` (si usa un nome simbolico)

```
1 #define NVOTI 10
2 int voti[NVOTI];
```

Esempio

```c
#include <stdio.h>
#define NVOTI 10

int main()
{
  int i,n,voti[NVOTI],somma=0;
  float media;
  printf("Quanti voti vuoi inserire? ");
  scanf("%d",&n);
  for(i=0;i<n;i++)  {
    printf("\nInserisci voto %d: ",i+1);
    scanf("%d",&voti[i]);
    somma+=voti[i];
  }
  media=(float)somma/n;
  printf("\nLa media vale: %f\n",media);
}
```

Esempio

```
1  #include <stdio.h>
2  #define NVOTI 10
3
4  ...
5
6    for(i=0;i<n;i++)  {
7      printf("\nInserisci voto %d: ",i+1);
8      scanf("%d",&voti[i]);
9      somma+=voti[i];
10   }
11
12 ...
```

- Vengono caricati `n` voti e non `NVOTI`!
  - Viene utilizzato meno spazio
  - La scelta avviene a cura del programmatore!

Esempio

Controllo dimensione Array

```c
#include <stdio.h>
#define NVOTI 10
int main(){
    int i,n,voti[NVOTI],somma=0;
    float media;

    do {
        printf("Quanti voti vuoi inserire? ");
        scanf("%d",&n);
    } while(n<1 || n > NVOTI);
}
```

Array

Note

- Non è possibile assegnare un array ad un altro array: `vett1=vett2;` NON E' AMMESSA!
- Sono ammesse anche altre inizializzazioni:

```
1  int vett[3]={1,2,3};
```

oppure

```
1  int vett[]={1,2,3};
```

Nel secondo caso la dimensione è pari a 3 👍

Array

Note

- Il C non effettua alcun controllo sui limiti degli array
- In altre parole è possibile inizializzare un array di dimensione N con più di N valori senza avere alcun messaggio di errore in compilazione
- **È compito del programmatore garantire che tutti gli array siano abbastanza grandi da contenere ciò per cui sono stati creati**

Esercizio

Scrivere un programma che esegua la somma di N valori interi precedentemente inseriti in un array

Soluzione

```c
#include <stdio.h>
#define N 10
int main() {
    int i,n,vett[N],somma=0;
    do {
        printf("Lunghezza vettore:");
        scanf("%d",&n);
    } while(n<1 || n>N);
    for(i=0;i<n;++i)  {
        printf("\nInserisci numero %d: ",i+1);
        scanf("%d",&vett[i]);
        somma+=vett[i];
    }
    printf("\nLa somma vale: %d\n",somma);
}
```

Esercizio

Scrivere un programma che esegua la copia di un array in un nuovo array

Soluzione

```c
#include <stdio.h>
#define DIM 100

int main()
{
    int arr1[DIM], arr2[DIM];
    int i, n;

    printf("Numero elementi del primo array:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("elemento - %d : ",i+1);
        scanf("%d",&arr1[i]);
    }
```

Soluzione

```
        for(i=0; i<n; i++)
            arr2[i] = arr1[i];

        printf("\nPrimo array:\n");
        for(i=0; i<n; i++)
            printf("%3d", arr1[i]);
        printf("\n\nSecondo array:\n");
        for(i=0; i<n; i++)
            printf("%3d", arr2[i]);
}
```

Controllo dimensione array 💡

```c
#include <stdio.h>
#define N 10

int main()
{
  int i,n,vett[N],somma=0;

  do  {
    printf("Lunghezza vettore? ");
    scanf("%d",&n);
  } while(n<1 || n>N);

  for(i=0;i<n;++i)  {
    printf("\nInserisci numero %d: ",i+1);
    scanf("%d",&vett[i]);
    somma+=vett[i];
    }
  printf("\nLa somma vale: %d\n",somma);
}
```

Esercizio

Scrivere un programma che memorizzi N voti e determini la media, il voto maggiore e il voto minore

Soluzione

```c
#include <stdio.h>
#define NVOTI 10

int main()
{
  int i,n,voti[NVOTI],somma=0,min,max;
  float media;

  do  {
    printf("Quanti n vuoi inserire? ");
    scanf("%d",&n);
  } while(n<1 || n>NVOTI);

  for(i=0;i<n;++i)  {
    printf("\nInserisci voto %d: ",i+1);
    scanf("%d",&voti[i]);
    somma+=voti[i];
    }
  media=(float)somma/n;
  printf("\nLa media vale: %.1f\n",media);
```

Soluzione

```
1   min=voti[0];
2   max=voti[0];
3   for(i=0;i<n;++i) {
4     if(voti[i]<min) min=voti[i];
5     if(voti[i]>max) max=voti[i];
6   }
7   printf("\nIl voto piu' alto e': %d\n",max);
8   printf("\nIl voto piu' basso e': %d\n",min);
```

Esercizio

Scrivere un programma che esegua la ricerca di un elemento (letto da tastiera) in un array

Soluzione - ricerca sequenziale

```c
#include <stdio.h>
#define DIM 100
int main() {
    int v[DIM], n, i, elem, trovato=0;
    do {
        printf("\nInserisci dimensione array: ");
        scanf("%d", &n);
    } while(n<1 || n>DIM);
    printf("\nInserisci i %d elementi: ",n);
    for(i=0;i<n;i++) {
        printf("\nelemento di indice - %d: ",i);
        scanf("%d",&v[i]);
    }
    printf("\nInserisci elemento da ricercare: ");
    scanf("%d",&elem);
    i=0;
    while(trovato!=1 && i<n) {
        if(elem==v[i]) trovato=1;
        ++i;
    }
    if(trovato==1) printf("\nElemento in pos %d",i);
    else printf("\nElemento non presente");
}
```

Esercizio

Scrivere un programma che ordini un array (in ordine crescente)

Soluzione - bubble sort

```c
#include <stdio.h>
#define DIM 100

int main()
{
    int v[DIM], n, i, j, tmp;

    do  {
      printf("Inserisci dimensione array: \n");
      scanf("%d", &n);
    } while(n<1 || n>DIM);
    printf("Inserisci i %d elementi:\n",n);
    for(i=0;i<n;i++)  {
        printf("elementodi indice - %d : ",i);
        scanf("%d",&v[i]);
    }
```

Soluzione

```
    for(i=0; i<n-1; i++)  {
        for(j=0; j<n-1; j++) {
            if(v[j] > v[j+1]) {
                tmp = v[j];
                v[j] = v[j+1];
                v[j+1] = tmp;
            }
        }
    }
    printf("\nElementi in ordine crescente:\n");
    for(i=0; i<n; i++)
        printf("%d  ", v[i]);
}
```

Matrici

Vettori: array monodimensionali

Matrici: array bidimensionali

- Il C permette di definire array di qualsiasi dimensione

Per accedere ad un elemento di una matrice: `M[i][j]`

- Il primo indice identifica la riga, il secondo la colonna

Dichiarazione: `int M[NR][NC];`

Esempio

```c
#include <stdio.h>
#define DIM 3
int main()
{
    int m[DIM][DIM],i,j;

    printf("Inserisci elementi nella matrice:\n");
    for(i=0;i<DIM;i++) {
        for(j=0;j<DIM;j++) {
            printf("elemento - [%d],[%d] : ",i,j);
            scanf("%d",&m[i][j]);
        }
    }
    printf("\nLa matrice e': \n");
    for(i=0;i<DIM;i++) {
        printf("\n");
        for(j=0;j<DIM;j++)
            printf("%d\t",m[i][j]);
    }
}
```

Esercizio

Scrivere un programma che calcoli la somma di due matrici quadrate di dimensione NxN

Soluzione

```c
#include <stdio.h>
#define DIM 50

int main()
{
    int m1[DIM][DIM],m2[DIM][DIM],sum[DIM][DIM],i,j,n;

    do  {
      printf("Dimensione matrice quadrata: \n");
      scanf("%d", &n);
    } while(n<1 || n>DIM);

    printf("Elementi della prima matrice:\n");
    for(i=0;i<n;i++) {
        for(j=0;j<n;j++) {
            printf("elemento - [%d],[%d] : ",i,j);
            scanf("%d",&m1[i][j]);
        }
    }
```

Soluzione

```
    printf("Elementi della seconda matrice:\n");
    for(i=0;i<n;i++) {
        for(j=0;j<n;j++) {
            printf("elemento - [%d],[%d] : ",i,j);
            scanf("%d",&m2[i][j]);
        }
    }

    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            sum[i][j]=m1[i][j]+m2[i][j];

    printf("\n\nLa somma vale: \n");
    for(i=0;i<n;i++){
        printf("\n");
        for(j=0;j<n;j++)
            printf("%d\t",sum[i][j]);
    }
    printf("\n\n");
}
```

Esercizio

Scrivere un programma che calcoli la somma delle righe e delle colonne di una matrice quadrata di dimensione NxN

Soluzione

```c
#include <stdio.h>
#define DIM 100
int main()
{
    int i,j,k,m1[DIM][DIM],rsum[DIM],csum[DIM],n;

    do  {
      printf("Dimensione matrice quadrata: \n");
      scanf("%d", &n);
    } while(n<1 || n>DIM);

    printf("Inserisci elementi della matrice:\n");
    for(i=0;i<n;i++) {
        for(j=0;j<n;j++) {
            printf("elemento - [%d],[%d] : ",i,j);
            scanf("%d",&m1[i][j]);
        }
    }
```

Soluzione

```
1       for(i=0;i<n;i++) {
2           rsum[i]=0;
3           for(j=0;j<n;j++)
4               rsum[i]=rsum[i]+m1[i][j];
5       }
6
7       for(i=0;i<n;i++) {
8           csum[i]=0;
9           for(j=0;j<n;j++)
10              csum[i]=csum[i]+m1[j][i];
11      }
```

Soluzione

```
    printf("La somma righe e colonne vale:\n");
    for(i=0;i<n;i++) {
        for(j=0;j<n;j++)
            printf("% 4d",m1[i][j]);
        printf("% 8d",rsum[i]);
        printf("\n");
    }
    printf("\n");
    for(j=0;j<n;j++) {
        printf("% 4d",csum[j]);
    }
    printf("\n\n");
}
```

Puntatori

Puntatori

Dichiarazione di variabile: `int n;` - nome - locazione di memoria - indirizzo della locazione di memoria

Puntatori

L'operatore `&` (vedi utilizzo `scanf`) restituisce l'indirizzo di memoria di una variabile

I **puntatori** sono delle variabili alle quali può essere assegnato un indirizzo di memoria

Puntatori: dichiarazione

```
int *int_pointer;
```

Occorre specificare il tipo base (in questo caso `int`) e il nome della variabile (in questo caso `int_pointer`) preceduto dal simbolo `*`

Puntatori: inizializzazione

```
1  int n=10;
2  int *p;
3
4  p=&n; //non assegna a p il valore di n!
```

In questo caso la variabile puntatore ad intero p è inizializzata con l'indirizzo di n

Puntatori: primo utilizzo

```c
#include <stdio.h>

int main()
{
  int n=10;
  int *p;

  p=&n;
  printf("%d %d",n,*p);
}
```

L'operatore $*$ (detto di *indirezione*) applicato ad una variabile puntatore permette di restituire il valore contenuto nella variabile puntata

Puntatori: esempio

```c
#include <stdio.h>

int main()
{
  int n=10,m;
  int *p;

  p=&n;
  m=*p;
  n=20;
  printf("%d %d %d",n,*p,m); //stampa: 20 20 10
}
```

Puntatori: esempio

```c
#include <stdio.h>

int main()
{
  int n=10,m;
  int *p;

  p=&n;
  m=*p;
  n=20;
  printf("%d %d %d",n,*p,m); //stampa: 20 20 10

  *p=30;
  printf("%d %d %d",n,*p,m); //stampa: 30 30 10
}
```

Puntatori: esempio

```c
#include <stdio.h>

int main()
{
  int n=10,m;
  int *p,*q;

  p=&n;
  m=*p;
  n=20;
  printf("%d %d %d",n,*p,m); //stampa: 20 20 10

  *p=30;
  printf("%d %d %d",n,*p,m); //stampa: 30 30 10

  q=p;
  printf("%d %d %d",n,*p,*q); //stampa 30 30 30
}
```

Esercizio.

Scrivere un programma che esegua la somma di due numeri (letti da tastiera) usando i puntatori

Soluzione.

```c
#include <stdio.h>
int main() {

  int n1,n2;
  int *p1,*p2;

  printf("Inserisci un numero: ");
  scanf("%d",&n1);
  printf("Inserisci un altro numero: ");
  scanf("%d",&n2);

  p1=&n1;
  p2=&n2;

  printf("La somma vale %d", *p1 + *p2);
}
```

Esercizio.

Scrivere un programma che definisca il maggiore tra due numeri (letti da tastiera) usando i puntatori

Soluzione.

```c
#include <stdio.h>
int main() {

  int n1,n2,max;
  int *p1,*p2;

  printf("Inserisci un numero: ");
  scanf("%d",&n1);
  printf("Inserisci un altro numero: ");
  scanf("%d",&n2);

  p1=&n1;
  p2=&n2;

  if(*p1>*p2) max=*p1;
  else max=*p2;
  printf("Il valore maggiore e' %d", max);
}
```

Puntatori in profondità

```c
#include <stdio.h>

int main(){
    int v[3] = {1025, 1027, 1024};
    int *ip;
    char *cp;

    ip = &v[0];
    printf("\n%p %d",ip,*ip);
    printf("\n%p %d",ip+1,*(ip+1));
    printf("\n%p %d",ip+2,*(ip+2));

    cp = ip;
    printf("\n%p %d",cp,*cp);
    printf("\n%p %d",cp+1,*(cp+1));
    printf("\n%p %d",cp+1,*(cp+2));
    printf("\n%p %d",cp+1,*(cp+3));
    printf("\n%p %d",cp+1,*(cp+4));
    printf("\n%p %d",cp+1,*(cp+5));
    printf("\n%p %d",cp+1,*(cp+6));
    printf("\n%p %d",cp+1,*(cp+7));

    // 00000000 00000000 00000100 00000001 --> 1025
    // 00000000 00000000 00000100 00000011 --> 1027

    return 0;
}
```

Puntatori e Array

Puntatori e array sono strettamente correlati

E’ possibile definire un puntatore per accedere agli elementi di un array

Puntatori e Array

Il nome di un array rappresenta un puntatore *costante* al suo primo elemento: punta sempre al primo elemento e non è possibile assegnare l'indirizzo di un'altra cella di memoria

```
1 int vett[N];
2 int *vett_ptr;
3
4 vett_ptr = &vett[0];
```

equivale a

```
1 vett_ptr = vett; //non compare &
```

E' quindi possibile accedere agli elementi di un array utilizzando un puntatore.

Puntatori e Array

```c
#include <stdio.h>
int main() {

  int i,vett[3]={1,2,3};
  int *vett_ptr;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]); //1 2 3

  vett_ptr=vett; //vett_ptr punta dove punta vett
  //vett_ptr = &vett[0];
  *(vett_ptr+1)=10;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]); //cosa stampa?
}
```

Puntatori e Array

```c
#include <stdio.h>
int main() {

  int i,vett[3]={1,2,3};
  int *vett_ptr;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]); //1 2 3

  vett_ptr=vett; //vett_ptr punta dove punta vett
  *(vett_ptr+1)=10;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]); //stampa 1 10 3
}
```

Puntatori e Array

Avrei potuto scrivere anche…

```c
#include <stdio.h>
int main() {

  int i,vett[3]={1,2,3};
  int *vett_ptr;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]);

  vett_ptr=vett;
  *(vett_ptr+1)=10;

  for(i=0;i<3;i++)
    printf("%d ",*(vett_ptr+i)); //stampa 1 10 3
}
```

Puntatori e Array

Ma anche...

```c
#include <stdio.h>
int main() {

  int i,vett[3]={1,2,3};
  int *vett_ptr;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]);

  vett_ptr=vett;
  *(vett_ptr+1)=10;

  for(i=0;i<3;i++)
    printf("%d ",*(vett_ptr++));
}
```

Puntatori e Array

Domanda…

Che differenza c’è tra:

```
    for(i=0;i<3;i++)
      printf("%d ",*(vett_ptr++));
}
```

e

```
    for(i=0;i<3;i++)
      printf("%d ",*(vett_ptr+i));
}
```

Puntatori e Array

Risposta

Provate a stampare `*vett_ptr` dopo il primo e dopo il secondo ciclo `for`!

```
    for(i=0;i<3;i++)
      printf("%d ",*(vett_ptr++));
}
```

e

```
    for(i=0;i<3;i++)
      printf("%d ",*(vett_ptr+i));
}
```

Aritmetica dei puntatori

Incremento: permette di passare ad un indirizzo successivo

`p++`, `p+1`, `p+i`

Decremento: permette di passare ad un indirizzo precedente

`p--`, `p-1`, `p-i`

Lo spostamento dipende dal **tipo base**!

Puntatori e Array: ATTENZIONE

Differenza tra `vett_ptr++` e `++vett_ptr`

Puntatori e Array

Ricapitolando...

```c
#include <stdio.h>
int main() {
  int i,vett[3]={1,2,3};
  int *vett_ptr;

  for(i=0;i<3;i++)
    printf("%d ",vett[i]);

  vett_ptr=vett;

  for(i=0;i<3;i++)
    printf("%d ",*(vett+i));

  for(i=0;i<3;i++)
    printf("%d ",*(vett_ptr+i));
}
```

... sono equivalenti!

Puntatori e Array

Ma attenzione…

```c
#include <stdio.h>
int main() {
  int i,vett[3]={1,2,3};

  for(i=0;i<3;i++)
    printf("%d ",*(vett+i));

  for(i=0;i<3;i++)
    printf("%d ",*(vett++)); //ERRORE!!!
}
```

… `vett` è una costante!

Esercizio

Scrivere un programma in C che, utilizzando i puntatori, carichi N valori interi (letti da tastiera) in un array e successivamente stampi i valori inseriti.

Soluzione

Senza puntatori...

```c
#include <stdio.h>
#define DIM 10
int main() {

  int i,n,vett[DIM],*vett_ptr;

  vett_ptr=vett;
  do {
    printf("Inserisci dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("Inserisci i %d elementi:\n",n);
  for(i=0;i<n;i++) {
      printf("elementodi indice - %d : ",i);
      scanf("%d",&vett[i]);
  }
```

Soluzione

Con i puntatori...

```c
#include <stdio.h>
#define DIM 10
int main() {

  int i,n,vett[DIM],*vett_ptr;

  vett_ptr=vett;
  do {
    printf("Inserisci dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("Inserisci i %d elementi:\n",n);
  for(i=0;i<n;i++) {
      printf("elementodi indice - %d : ",i);
      scanf("%d",vett_ptr+i);
  }
```

Soluzione

Ma anche...

```c
#include <stdio.h>
#define DIM 10
int main() {

  int i,n,vett[DIM],*vett_ptr;

  vett_ptr=vett;
  do {
    printf("Inserisci dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("Inserisci i %d elementi:\n",n);
  for(i=0;i<n;i++) {
      printf("elementodi indice - %d : ",i);
      scanf("%d",vett+i); //non sto cambiando vett!
  }
```

Soluzione

Senza puntatori...

```
1    for(i=0;i<n;i++)
2      printf("%d ",vett[i]);
```

Soluzione

Con i puntatori…

```
  for(i=0;i<n;i++)
    printf("%d ",*(vett+i));
}
```

```
  for(i=0;i<n;i++)
    printf("%d ",*(vett_ptr+i));
}
```

```
  for(i=0;i<n;i++)
    printf("%d ",*(vett_ptr++));
}
```

Esercizio

Scrivere un programma in C che calcoli la somma degli elementi di un array usando i puntatori

Soluzione

```c
#include <stdio.h>
#define DIM 10
int main() {

  int i,n,vett[DIM],*vett_ptr,somma=0;

  vett_ptr=vett;
  do {
    printf("Inserisci dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("\nInserisci i %d elementi: \n",n);
  for(i=0;i<n;i++) {
      printf("\nelementodi indice - %d : ",i);
      scanf("%d",vett+i);
  }
```

Soluzione

```
1    printf("\nCalcolo la somma...");
2    for(i=0;i<n;i++) {
3        somma += *vett_ptr;
4        vett_ptr++;
5    }
6    printf("\nLa somma vale: %d\n",somma);
7  }
```

Esempi avanzati sugli array

Esercizio

Inserire un nuovo elemento in un array in una specifica posizione

Soluzione

```c
#include <stdio.h>
#define DIM 5

int main() {
  int v[DIM],n,i;
  int num,pos;

  do  {
    printf("Dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("Inserisci elementi:\n");
  for(i=0;i<n;i++)  {
      printf("elemento di indice - %d : ",i);
      scanf("%d",&v[i]);
  }
```

Soluzione

```
printf("\nInserisci nuovo elemento ");
  scanf("%d",&num);
  printf("\nInserisci la posizione ");
  scanf("%d",&pos);

  if(pos<0 || pos>n || n==DIM) {
    printf("\nInserimento non valido ");
  }
  else {
    //sposta a destra
    for(i=n;i>pos;i--)
      v[i]=v[i-1];
    v[pos]=num;
    n++;
  }
```

Esercizio

Eliminare un elemento da un array in una specifica posizione

Soluzione

```c
#include <stdio.h>
#define DIM 5

int main() {
  int v[DIM],n,i;
  int num,pos;

  do  {
    printf("Dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("Inserisci elementi:\n");
  for(i=0;i<n;i++)  {
      printf("elemento di indice - %d : ",i);
      scanf("%d",&v[i]);
  }
```

Soluzione

```
1  printf("\nIndice elem da cancellare ");
2    scanf("%d",&pos);
3
4    if(pos<0 || pos>n-1) {
5      printf("\nIndice non valido");
6    }
7    else {
8      //sposta a sinistra
9      for(i=pos;i<n;i++)
10       v[i]=v[i+1];
11     n--;
12   }
```

Funzioni

Le funzioni

Le funzioni sono dei **sottoprogrammi** costituiti da un insieme di istruzioni

Il loro uso permette di realizzare programmi più chiari da leggere, da capire e da modificare, infatti… - attraverso l'uso delle funzioni si evita di dover riscrivere più volte del codice - è possibile chiamare una funzione più volte in un programma

Le funzioni sono alla base della *programmazione modulare*

Creazione librerie (scanf, printf) che nascondono i dettagli implementativi

Funzioni: introduzione

`valori in ingresso` -> `FUNZIONE` -> `valore restituito`

Abbiamo già usato le funzioni!

```
1 int main() {
2
3   return 0;
4 }
```

Funzioni: dichiarazione

1. scelta del **nome** della funzione
2. **tipo** del valore restituito
3. **tipo** (e nome) dei parametri in ingresso (*parametri formali*)

```
1  void nome_funzione(void);
```

oppure

```
1  int somma(int, int);
```

Funzioni: definizione

Con la definizione di una funzione *specifichiamo* le istruzioni che caratterizzano la funzione

Esempio:

```
1 int somma(int c,int d) //nome dei parametri formali
2 {
3   int s;
4   s=c+d;
5   return s; //restituiamo s al main (o alla funzione chiamante)
6 }
```

La definizione deve essere inserita subito dopo la chiusura del `main`

Funzioni: chiamata

La **chiamata** permette di *eseguire* la funzione, passare i parametri (*attuali*) ed assegnare il valore restituito

Esempio:

```c
int main() {
  int a=1,b=1,sum;

  sum=somma(a,b); //a e b sono i parametri attuali
  printf("\nLa somma vale %d\n",sum);
}
```

👉 Nella chiamata occorre fare attenzione che il **tipo**, il **numero** e **l'ordine** dei paramtri attuali sia conforme ai parametri formali indicati nella definizione della funzione! ❗

Funzioni: primo esempio

```c
#include <stdio.h>

void messaggio();

int main() {

  messaggio();
}

void messaggio() {

  printf("\nLa mia prima funzione\n");
}
```

Funzioni: secondo esempio

```c
#include <stdio.h>

void messaggio();

int main() {
  int i;

  for(i=1;i<=5;i++)
    messaggio();
}

void messaggio() {

  printf("\nLa mia prima funzione\n");
}
```

Funzioni: altro esempio

```c
#include <stdio.h>

void pari_dispari(int);

int main() {
  int num;

  printf("\nInserisci un numero intero ");
  scanf("%d",&num);
  pari_dispari(num);
}

void pari_dispari(int n) {

  if(n%2==0) printf("\nIl num e' pari\n");
  else printf("\nIl num e' dispari\n");
}
```

Funzioni: più parametri

```c
#include <stdio.h>
void bmi(float,float);

int main() {
  float peso,altezza;

  printf("\nPeso in Kg: ");
  scanf("%f",&peso);
  printf("\nAltezza in m: ");
  scanf("%f",&altezza);
  bmi(peso,altezza);
}

void bmi(float p,float a) {

  printf("\nIl bmi vale %.2f\n",p/(a*a));
}
```

Funzioni: più parametri e return

```c
#include <stdio.h>
float bmi(float,float);

int main() {
  float peso,altezza,mio_bmi;

  printf("\nPeso in Kg: ");
  scanf("%f",&peso);
  printf("\nAltezza in m: ");
  scanf("%f",&altezza);
  mio_bmi=bmi(peso,altezza);
  printf("\nIl bmi vale %.2f\n",mio_bmi);
}

float bmi(float p,float a) {
  float ris;

  ris=p/(a*a);
  return ris;
}
```

Funzioni: più parametri e return

Oppure...

```c
#include <stdio.h>
float bmi(float,float);

int main() {
  float peso,altezza,mio_bmi;

  printf("\nPeso in Kg: ");
  scanf("%f",&peso);
  printf("\nAltezza in m: ");
  scanf("%f",&altezza);
  mio_bmi=bmi(peso,altezza);
  printf("\nIl bmi vale %.2f\n",mio_bmi);
}

float bmi(float p,float a) {

  return p/(a*a);
}
```

Funzioni: passaggio dei parametri

👉 In C avviene sempre e solo **per valore** - *i parametri formali vengono inizializzati con i corrispondenti valori dei parametri attuali*

Una funzione **non modifica** i valori dei parametri attuali 

Funzioni: passaggio dei parametri

La variabile num non viene modificata dalla funzione

```c
#include <stdio.h>
int quadrato(int);

int main() {
  int num,q;

  printf("\nInserisci un numero intero: ");
  scanf("%d",&num);
  q=quadrato(num); //num non viene modificato
  printf("\nIl quadrato di %d vale %d\n",num,q);
}

int quadrato(int n) {
  n=n*n;
  return n;
}
```

Funzioni: passaggio dei parametri

La variabile num non viene modificata dalla funzione

```c
#include <stdio.h>
int quadrato(int);

int main() {
  int num,q;

  printf("\nInserisci un numero intero: ");
  scanf("%d",&num);
  q=quadrato(num);
  printf("\nIl quadrato di %d vale %d\n",num,q);
}

int quadrato(int num) { //neanche in questo caso!!!
  num=num*num;
  return num;
}
```

Funzioni: passaggio dei parametri

Passaggio **esplicito**: utilizzo dei parametri attuali e formali

Passaggio **implicito**: utilizzi di variabili globali (è buona norma evitarle!)

Visibilità (scope)

👉 La dichiarazione di una variabile introduce un nome simbolico **visibile** (*scope*) all'interno di una specifica parte (blocco) del programma

Nome locale – **variabile locale** - visibilità solo all'interno del blocco delle istruzioni della funzione: dal punto della dichiarazione alla fine del blocco

Nome globale – **variabile globale** - definita fuori dalla funzione: dal punto della dichiarazione alla fine del file

👉 I parametri formali sono delle variabili locali!

Visibilità (scope)

Esempio:

```c
#include <stdio.h>

int main() {

  for(int i=0;i<10;i++)
    printf("%d",i);
  printf("%d",i);

}
```

Cosa produce questo programma?

Visibilità (scope)

Esempio:

```c
#include <stdio.h>

int main() {
  int i;

  for(i=0;i<10;i++)
    printf("%d",i);
  printf("%d",i);

}
```

E questo?

Visibilità (scope)

Occorre prestatare particolare attenzione al **mascheramento**

Mascheramento: la dichiarazione di una variabile locale può *nascondere* (e quindi *mascherare*) una variabile con lo **stesso nome** dichiarata in un blocco più esterno (o globale)

Mascheramento

```c
#include <stdio.h>

int main() {
  int i=-1;

  for(int i=0;i<10;i++)
    printf("%d ",i);
  printf("\n%d",i);

}
```

Provate questo codice

Mascheramento

```c
#include <stdio.h>

int quadrato(int);
int n=4; //variabile globale

int main() {
  int n=3; //ATTENZIONE al mascheramento

  printf("\nIl quadrato di %d vale %d\n",n,quadrato(n));
  //Il quadrato di 3 vale 9
}

int quadrato(int n) {
  n=n*n;
  return n;
}
```

E’ possibile che il mascheramento avvenga senza rendersene conto!

Scope: variabile globale

```c
#include <stdio.h>

int quadrato(int);
int n=4; //variabile globale

int main() {

  printf("\nIl quadrato di %d vale %d\n",n,quadrato(n));
  //Il quadrato di 4 vale 16
  printf("Ora n vale: %d\n",n);
  //Ora n vale 4
}

int quadrato(int n) {
  n=n*n;
  return n;
}
```

Scope: variabile globale

```c
#include <stdio.h>

int quadrato();
int n=4; //variabile globale

int main() {

  printf("\nIl quadrato di %d vale %d\n",n,quadrato());
  //Il quadrato di 4 vale 16
  printf("Ora n vale: %d\n",n);
  //Quanto vale n?
}

int quadrato() {
  n=n*n;
  return n;
}
```

Scope: variabile globale

```c
#include <stdio.h>

int quadrato();
int n=4; //variabile globale

int main() {

  printf("\nIl quadrato di %d vale %d\n",n,quadrato());
  //Il quadrato di 4 vale 16
  printf("Ora n vale: %d",n);
  //Ora n vale 16 <-------
}

int quadrato() {
  n=n*n;
  return n;
}
```

Scope: e ancora...

```c
#include <stdio.h>

int quadrato();
int n=4; //variabile globale

int main() {
  printf("\nIl quadrato di %d vale %d\n",n,quadrato());
  //Il quadrato di 4 vale 16
  printf("Ora n vale: %d\n",n); //Ora n vale 16
  int n=5;
  printf("\nIl quadrato di %d vale %d\n",n,quadrato());
  //Il quadrato di ? vale ?
  printf("Ora n vale: %d\n",n); //Ora n vale ?
}

int quadrato() {
  n=n*n;
  return n;
}
```

Scope: e ancora...

```c
#include <stdio.h>

int quadrato();
int n=4; //variabile globale

int main() {

  printf("\nIl quadrato di %d vale %d\n",n,quadrato());
  //Il quadrato di 4 vale 16
  printf("Ora n vale: %d\n",n); //Ora n vale 16
  int n=5;
  printf("\nIl quadrato di %d vale %d\n",n,quadrato());
  //Il quadrato di 5 vale 256
  printf("Ora n vale: %d\n",n); //Ora n vale 5
}

int quadrato() {
  n=n*n;
  return n;
}
```

Scope: ultimo esempio

```c
#include <stdio.h>

int quadrato();
int n=4; //variabile globale

int main() {
  printf("\nIl quadrato di %d vale %d\n",n,quadrato());
  //Il quadrato di 4 vale 16
  printf("Ora n vale: %d\n",n);
  //Ora n vale 16
  n=5;
  printf("\nIl quadrato di %d vale %d\n",n,quadrato());
  //Il quadrato di 5 vale 25
  printf("Ora n vale: %d\n",n);
  //Ora n vale 25
}

int quadrato() {
  n=n*n;
  return n;
}
```

Passaggio dei parametri

Ricapitolando…

- **Parametri formali**: definizione della funzione (tipo, numero e ordine)
- **Parametri attuali**: passati alla funzione quando viene chiamata

Passaggio dei parametri

Ricapitolando…

👉 **In C il passaggio dei parametri avviene per valore, cioè durante la chiamata di una funzione ogni parametro formale viene inizializzato con il valore del corrispondente parametro attuale**

💡 E se volessimo modificare il valore dei parametri attuali all'interno della funzione?

Passaggio dei parametri

💡 E se volessimo modificare il valore dei parametri attuali all'interno della funzione?

🔆 Possiamo passare per valore l'indirizzo di una variabile, cioè inizializzare i parametri formali con l'indirizzo dei parametri attuali.

E quindi usare i **puntatori**!

Esempio: scambia

```c
#include <stdio.h>
void scambia(int, int);

int main() {
  int x=0,y=1;
  printf("\nPrima x vale %d e y vale %d",x,y);
  scambia(x, y);
  printf("\nDopo x vale %d e y vale %d\n",x,y);
}

void scambia(int x, int y) {
  int tmp;
  tmp=x;
  x=y;
  y=tmp;
}
```

Esempio: scambia con i puntatori

```c
#include <stdio.h>
void scambia(int *, int *);

int main() {
  int x=0,y=1;
  printf("\nPrima x vale %d e y vale %d",x,y);
  scambia(&x, &y);
  printf("\nDopo x vale %d e y vale %d\n",x,y);
}

void scambia(int *x, int *y) {
  int tmp;
  tmp=*x;
  *x=*y;
  *y=tmp;
}
```

```
Prima x vale 0 e y vale 1 e Dopo x vale 1 e y vale 0
```

Funzioni e Array

Funzioni e Array

- E’ possibile passare ad una funzione il valore di un elemento di un array? **sì**

```
1 nome_funzione(nome_array[indice]);
```

- E’ possibile passare ad una funzione un array? **sì**, **ma** occorre fare attenzione.

```
1 nome_funzione(nome_array);
```

👉 Sto passando il puntatore al suo primo elemento!

Carica e stampa

```c
#include <stdio.h>
#define N 10

void carica_array(int, int *);
void stampa_array(int, int *);

int main(void) {
    int v[N],n;

    do{
        printf("\nDimensione array: ");
        scanf("%d",&n);
    }while(n<1 || n>N);

    printf("\nIndirizzo di v[0] nel main: %p ",&v[0]);
    carica_array(n,&v[0]); // Equivalente a scrivere carica_array(n,v);
    stampa_array(n,v);

    return 0;
}
```

Carica e stampa

```c
void carica_array(int n, int *x){
    int i;

    printf("\nx e' inizializzata con l'indirizzo di v[0]: %p ", x);
    printf("\nLa variabile x e' locale e si trova in %p",&x);

    for(i=0;i<n;i++){
        printf("\nInserisci elemento: ");
        scanf("%d",x+i); // tramite x carico i valori nell'array v dichiarato nel main
    }
}

void stampa_array(int n, int *v){
    int i;

    printf("\nArray: ");
    for(i=0;i<n;i++){
        printf("%d ",*(v+i));
    }
}
```

Esercizio

Scivere un programma in C che, attraverso l'uso di una funzione, calcoli il minimo di un array di interi

Soluzione

```c
#include <stdio.h>
#define DIM 10

int calcola_minimo(int *, int);

int main() {
  int n,v[DIM],i,min;

  do  {
    printf("Inserisci dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("Inserisci i %d elementi:\n",n);
  for(i=0;i<n;i++)  {
      printf("elemento di indice - %d : ",i);
      scanf("%d",&v[i]);
  }
```

Soluzione

```
  min=calcola_minimo(v, n);
  printf("\nIl minimo vale: %d\n", min);
}

int calcola_minimo(int *v, int n) {
  int i,min=*v;

  for(i=0;i<n;i++) {
    if(*(v+i)<min)
      min=*(v+i);
  }
  return min;
}
```

Esercizio

Scrivere un programma in C che, attraverso l'uso di una funzione, permetta di calcolare la media degli elementi di un vettore

Soluzione

```c
#include <stdio.h>
#define DIM 10

float calcola_media(int *, int);

int main() {
  int n,v[DIM],i;
  float media;

  do  {
    printf("Inserisci dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("Inserisci i %d elementi:\n",n);
  for(i=0;i<n;i++)  {
      printf("elemento di indice - %d : ",i);
      scanf("%d",&v[i]);
  }
```

Soluzione

```
  media=calcola_media(v, n);
  printf("\nLa media vale: %.1f\n", media);
}

float calcola_media(int *v, int n) {
  int i;
  float media=0;

  for(i=0;i<n;i++) {
    media+=*(v+i);
  }
  return media/n;
}
```

Soluzione

Oppure...

```
1  float calcola_media(int *v, int n) {
2    int i;
3    float media=0;
4
5    for(i=0;i<n;i++,v++) {
6      media+=*v;
7    }
8    return media/n;
9  }
```

Domanda di ripasso…

Perché i puntatori hanno un tipo?

Domanda di ripasso…

Perché i puntatori non vengono utilizzati solo per memorizzare l’indirizzo di una variabile!

- Deferenziazione
- Aritmetica dei puntatori

Esempio

```c
#include <stdio.h>

int main(){

    int n = 1025;
    int *ip;
    char *cp;
    ip = &n;
    printf("\n%p %d",ip,*ip);
    printf("\n%p %d",ip+1,*(ip+1));
    cp = (char *)ip;
    printf("\n%p %d",cp,*cp);
    printf("\n%p %d",cp+1,*(cp+1));

    // 00000000 00000000 00000100 00000001

    return 0;
}
```

Stringhe

Stringhe

```
char carattere; //memorizza un solo carattere!
```

Le **stringhe** sono array di caratteri (tipo `char`)

Stringhe

`char c[10];` <- vettore di 10 caratteri

Non è indispensabile specificare il numero di elementi

La dimensione sarà determinata da: - inizializzazione + carattere *null* (`\0`)

```
char stringa[] = "Elementi di Informatica";
```

Stringhe

Il carattere `\0` ci permette di usare una stringa senza conoscerne l'effettiva dimensione!

```c
#include <stdio.h>

int main() {
char stringa[]="Elementi di informatica";
int i=0;

while(stringa[i]!='\0') {
  printf("%c",stringa[i]);
  ++i;
}
}
```

Stringhe

Per visualizzare l'intera stringa si usa il formato `%s`

```c
#include <stdio.h>

int main() {
char stringa[]="Elementi di informatica";

printf("%s",stringa);
}
```

Stringhe

Attenzione!

Se indichiamo la dimensione della stringa occorre considerare lo spazio per il carattere nullo `\0`

```
char stringa[5]="Ciao";
```

Stringhe

Operazioni comuni:

- copiare una stringa
- concatenare stringhe

Stringhe: copia

```c
#include <stdio.h>
#define DIM 100

int main()
{
    char stringa[]="Elementi di Informatica";
    char altra_stringa[DIM];
    int i;

    for(i=0;stringa[i]!='\0';i++)
      altra_stringa[i]=stringa[i];

    altra_stringa[i]='\0';
    printf("Prima: %s - Seconda: %s\n",stringa,
    altra_stringa);
}
```

Stringhe: concatena

```c
//concatena due stringhe
#include <stdio.h>
#define DIM 100

int main()
{
    char stringa1[]="Elementi di ";
    char stringa2[]="Informatica";
    char concat[DIM];
    int i,j;

    for(i=0;stringa1[i]!='\0';++i)
        concat[i]=stringa1[i];

    for(j=0;stringa2[j]!='\0';++j)
        concat[i+j]=stringa2[j];

        concat[i+j]='\0';

    printf("%s",concat);
}
```

Stringhe: input e scanf()

```c
#include <stdio.h>
#define DIM 10

int main()
{
    char stringa[DIM];

    printf("Inserisci una stringa: ");
    scanf("%s",stringa);

    printf("La tua stringa: %s",stringa);
}
```

- Non è necessario usare il simbolo &
- ATTENZIONE: lo spazio interrompe l'immissione

Stringhe: input e getchar()

```c
#include <stdio.h>
#define DIM 100

int main()
{
    char stringa[DIM];
    int i;

  printf("Inserisci testo: ");
  for(i=0;((stringa[i]=getchar())!='\n') && (i<DIM);i++);
  stringa[i]='\0';
  printf("Parola: %s",stringa);
}
```

Con `getchar()` riusciamo a leggere anche lo spazio

Attenzione al `;` alla fine del ciclo `for`

Stringhe: la libreria `<string.h>`

Libreria per operare sulle stringhe:

- `strcpy(str_dest,str_orig)`
- `strncpy(str_dest,str_orig,n_char)`
- `strcat(str1,str2)`
- `strlen(str)` -> rende numero di caratteri
- `strcmp(str1,str2)` -> rende `0` se uguali

Stringhe: conversione

Libreria `<stdlib.h>`

- `atoi(stringa)`: converte una stringa in un `int`
- `atof(stringa)`: converte una stringa in un `double`

Strutture

Strutture

- Quando si usa una struttura ?
  - Quando abbiamo necessità di trattare un **insieme NON omogeneo** di dati !

Strutture

Esempio: memorizzare una data

```
int giorno, mese, anno;
```

Tre variabili per ogni data. Se serve un'altra data, dobbiamo dichiarare altre tre variabili

Le tre variabili sono logicamente collegate

Sarebbe utile poterle raggruppare! Usiamo le strutture

Strutture

```
1 struct data
2 {
3  int giorno;
4  int mese;
5  int anno;
6 };
```

Strutture: definizione

```
1 struct data
2 {
3  int giorno;
4  int mese;
5  int anno;
6 };
```

Questa rappresenta la **DEFINIZIONE** di una struttura.

La definizione permette di creare un **NUOVO TIPO**

Strutture: dichiarazione

```
1 struct data oggi;
2 struct data data_nascita;
```

Con la dichiarazione viene allocato lo spazio in memoria

Strutture: come accedere

Accedere ad una struttura:

`nome_variabile.nome_membro`

```
1 oggi.giorno=6;
2 oggi.mese=11;
3 oggi.anno=2018;
```

Strutture

Esempio:

```c
#include <stdio.h>

 struct data {
 int giorno;
 int mese;
 int anno;
};

int main() {
struct data oggi;

printf("Inserisci la data di oggi: ");
scanf("%d%d%d",&oggi.giorno,&oggi.mese,&oggi.anno);
printf("Oggi: %d/%d/%d",oggi.giorno,oggi.mese,oggi.anno);
}
```

Strutture: inizializzazione

```c
#include <stdio.h>

 struct data {
 int giorno;
 int mese;
 int anno;
};

int main() {
struct data oggi = {6, 11, 2018};

printf("\nOggi e' %d/%d/%d",oggi.giorno,oggi.mese,oggi.anno);
}
```

Strutture: inizializzazione

```c
#include <stdio.h>

 struct data {
 int giorno;
 int mese;
 int anno;
};

int main() {
struct data oggi = {.giorno=6, .mese=11, .anno=2018};

printf("\nOggi e' %d/%d/%d",oggi.giorno,oggi.mese,oggi.anno);
}
```

Strutture: inizializzazione

```c
#include <stdio.h>

 struct data {
 int giorno;
 int mese;
 int anno;
} oggi = {.giorno=6, .mese=11, .anno=2018};

int main() {

printf("\nOggi e' %d/%d/%d",oggi.giorno,oggi.mese,oggi.anno);
}
```

Puntatori a strutture

```c
#include <stdio.h>
#define NOME 50
struct anagr
    {
        int matricola;
        char nome[NOME];
        char cognome[NOME];
    };

int main() {

    struct anagr studente;
    struct anagr *pointer;

    pointer=&studente;
```

Puntatori a strutture

```
1       printf("\nNome studente: ");
2       scanf("%s",studente.nome);
3       printf("\nCognome studente: ");
4       scanf("%s",studente.cognome);
5       printf("\nMatricola: ");
6       scanf("%d",&studente.matricola);
7
8       printf("\n\nDati studente: ");
9       printf("%s %s - matricola %d\n",(*pointer).nome,
10      (*pointer).cognome,(*pointer).matricola);
11  }
```

Puntatori a strutture

In alternativa: `pointer->nome` ...

```
    printf("\nNome studente: ");
    scanf("%s",studente.nome);
    printf("\nCognome studente: ");
    scanf("%s",studente.cognome);
    printf("\nMatricola: ");
    scanf("%d",&studente.matricola);

    printf("\n\nDati studente: ");
    printf("%s %s - matricola %d\n",pointer->nome,
    pointer->cognome,pointer->matricola);
}
```

Puntatori a strutture

E ovviamente...

```
    pointer=&studente;

    printf("\nNome studente: ");
    scanf("%s",pointer->nome);
    printf("\nCognome studente: ");
    scanf("%s",pointer->cognome);
    printf("\nMatricola: ");
    scanf("%d",&pointer->matricola);

    printf("\n\nDati studente: ");
    printf("%s %s - matricola %d\n",studente.nome,
    studente.cognome,studente.matricola);
}
```

Strutture contenenti puntatori

```
1  struct st
2      {
3          int *p1;
4          int *p2;
5      };
6  #include <stdio.h>
7  int main() {
8
9
10     struct st st_pointers;
11     int n1=10, n2;
12
13     st_pointers.p1=&n1;
14     st_pointers.p2=&n2;
15     *st_pointers.p1=20;
16     *st_pointers.p2=*st_pointers.p1 * 2;
17     printf("%d %d",n1,n2);
18 }
```

Cosa stampa a video questo programma?

Struct o typedef?

```c
// Definizione
struct studente
    {
        int matricola;
        char nome[NOME];
        char cognome[NOME];
    };

// Dichiarazione
struct studente mio_studente;
```

```c
// Definizione
typedef struct
    {
        int matricola;
        char nome[NOME];
        char cognome[NOME];
    } studente;

// Dichiarazione
studente mio_studente;
```

Struct o typedef?

Vantaggi nell'uso di `C struct`:

- leggibilità (non maschera)
- compatibilità con vecchi standard
- chiarezza sul fatto che sia definito dall'utente

```
1 typedef int numero;
2
3 numero x = 10;
```

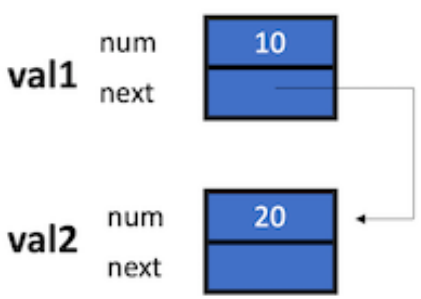
Liste concatenate lineari

Le liste lineari concatenate sono costituite da *serie di elementi omogenei* che occupano in memoria posizioni non adiacenti

Puntatori a strutture e *strutture contenenti puntatori* sono di fondamentale importanza per la creazione e gestione delle liste concatenate lineari

Liste concatenate lineari (statiche)

```c
#include <stdio.h>
struct lista_valori {
    int num;
    struct lista_valori *next;
};
int main() {
    struct lista_valori val1,val2;
    val1.num=10;
    val2.num=20;
    val1.next=&val2;
    printf("%d ",val2.num);
    printf("%d ",(*val1.next).num);
    printf("%d ",val1.next->num);
}
```



Liste concatenate lineari

```c
#include <stdio.h>
struct lista_valori
{
    int num;
    struct lista_valori *next;
};
int main() {
    struct lista_valori val1,val2,val3;
    val1.num=10;
    val2.num=20;
    val3.num=30;

    val1.next=&val2;
    val2.next=&val3;

    printf("%d ",val1.num);
    printf("%d ",val1.next->num);
    printf("%d ",val2.next->num);
}
```

Liste concatenate lineari

Alcune interessanti proprietà:

- eliminare elementi
- inserire elementi

Eliminare `val2` dalla lista: `val1.next = val2.next;`

Inserire nuovo elemento (`val4`) tra `val2` e `val3`:

```
val4.next = val2.next;
val2.next = &val4;
```

Questo è possibile grazie al fatto che gli elementi di una lista non sono memorizzati in sequenza. *Cosa succede negli array?*

Liste concatenate lineari

- Puntatore esterno al primo elemento della lista
  - `struct lista_valori *punt_lista = &val1;`
- Puntatore a NULL (fine lista)
  - `val3.next = NULL;`

Esempio: scorri lista

```c
#include <stdio.h>
    struct lista_valori {
        int num;
        struct lista_valori *next;
    };
int main() {
    struct lista_valori val1,val2,val3;
    struct lista_valori *punt_lista; // Puntatore esterno
    val1.num=10;
    val1.next=&val2;
    val2.num=20;
    val2.next=&val3;
    val3.num=30;
    val3.next=NULL; // Puntatore a NULL
    punt_lista=&val1;

    // non uso più le variabili!
    while(punt_lista != NULL) {
      printf("%d ",punt_lista->num);
      punt_lista=punt_lista->next;
    }
}
```

Liste concatenate lineari: NOTE

```
struct valori *punt_lista=&n1;
```

oppure

```
struct valori *punt_lista;
punt_lista=&n1;
```

Liste concatenate lineari: NOTE

- Liste statiche
- Liste dinamiche

Il programma ha necessità di allocare la memoria per ogni nuovo elemento della lista!

Allocazione dinamica della memoria

Strutture e Array

Strutture e Array

- Strutture contenenti array
- Array di strutture
- Strutture contenenti strutture

Strutture contenenti array

```c
#include <stdio.h>
#define NOME 20
    struct anagr
    {
        int matricola;
        char nome[NOME];
        char cognome[NOME];
    };
int main() {
    struct anagr studente;
    printf("\nNome studente: ");
    scanf("%s",studente.nome);
    printf("\nCognome studente: ");
    scanf("%s",studente.cognome);
    printf("\nMatricola: ");
    scanf("%d",&studente.matricola);

    printf("\n\nLo studente si chiama: ");
    printf("%s %s e la sua matricola e' %d\n",
    studente.nome,studente.cognome,studente.matricola);
}
```

Array di strutture

```c
#include <stdio.h>
#define DIM_NOME 20
#define DIM_STUD 100
    struct anagr {
        int matricola;
        char nome[DIM_NOME];
        char cognome[DIM_NOME];
    };

int main() {
    struct anagr studente[DIM_STUD];
    int n_studenti;
```

Continua...

Array di strutture

```
1  printf("\nQuanti studenti vuoi inserire? ");
2      scanf("%d",&n_studenti);
3
4      printf("Inserisci studenti:\n");
5      for(int i=0;i<n_studenti;++i) {
6          printf("Studente - %d: ", i+1);
7          printf("\nNome studente: ");
8          scanf("%s",studente[i].nome);
9          printf("\nCognome studente: ");
10         scanf("%s",studente[i].cognome);
11         printf("\nMatricola: ");
12         scanf("%d",&studente[i].matricola);
13     }
```

Continua...

Array di strutture

```
1
2       printf("\n\nElenco studenti:\n");
3       for(int i=0;i<n_studenti;++i) {
4           printf("%s %s - matricola: %d\n",
5   studente[i].nome,studente[i].cognome,
6   studente[i].matricola);
7       }
8   }
```

Esercizio

Scrivere un programma che calcoli la media del voto di laurea di N studenti e stampi l'elenco degli studenti con la differenza del loro voto rispetto alla media

Soluzione

```c
#include <stdio.h>
#define DIM_NOME 20
#define DIM_STUD 100
    struct anagr {
        int voto_laurea;
        char nome[DIM_NOME];
        char cognome[DIM_NOME];
    };

int main() {
    struct anagr studente[DIM_STUD];
    int n_studenti,somma=0,i;
    float media;
```

Continua...

Array di strutture

```
1       printf("\nQuanti studenti vuoi inserire? ");
2       scanf("%d",&n_studenti);
3       printf("\nInserisci studenti:\n");
4       for(i=0;i<n_studenti;++i) {
5           printf("Studente - %d: ", i+1);
6           printf("\nNome studente: ");
7           scanf("%s",studente[i].nome);
8           printf("\nCognome studente: ");
9           scanf("%s",studente[i].cognome);
10          printf("\nVoto di laurea: ");
11          scanf("%d",&studente[i].voto_laurea);
12          somma+=studente[i].voto_laurea;
13      }
14      media=(float)somma/n_studenti;
```

Continua...

Array di strutture

```
    printf("\n\nElenco studenti:\n");
    for(i=0;i<n_studenti;++i) {
        printf("%s %s - diff media: %.1f\n",
        studente[i].nome,
        studente[i].cognome,
        studente[i].voto_laurea-media);
    }
}
```

Strutture contenenti strutture

```
1     struct corso {
2         int voto;
3         char nome[DIM_NOME];
4     };
5
6     struct anagr {
7         char nome[DIM_NOME];
8         char cognome[DIM_NOME];
9         struct corso esame[DIM_ESAME];
10    };
```

Esercizio

Creare un programma che permetta di inserire i voti di N esami per M studenti e stampi per ogni studente la media dei voti

Soluzione

```c
#include <stdio.h>
#define DIM_NOME 20
#define DIM_STUD 1000
#define DIM_ESAME 50
    struct corso {
        int voto;
        char nome[DIM_NOME];
    };

    struct anagr {
        char nome[DIM_NOME];
        char cognome[DIM_NOME];
        struct corso esame[DIM_ESAME];
    };
```

Continua...

Soluzione

```c
int main() {
    struct anagr studente[DIM_STUD];
    int n_studenti,n_esami,somma[DIM_STUD]={0};
    float media[DIM_STUD]={0};
```

Continua...

Soluzione

```
1  printf("Qunati studenti vuoi inserire? ");
2      scanf("%d",&n_studenti);
3      printf("\nInserisci studenti.");
4      for(int i=0;i<n_studenti;++i) {
5          printf("\nStudente - %d", i+1);
6          printf("\nNome studente: ");
7          scanf("%s",studente[i].nome);
8          printf("\nCognome studente: ");
9          scanf("%s",studente[i].cognome);
10         printf("\nQuanti esami ha sostenuto?: ");
11         scanf("%d",&n_esami);
12         for(int j=0;j<n_esami;++j) {
13             printf("\nEsame - %d",j+1);
14             printf("\nNome corso: ");
15             scanf("%s",studente[i].esame[j].nome);
16             printf("\nVoto: ");
17             scanf("%d",&studente[i].esame[j].voto);
18             somma[i]+=studente[i].esame[j].voto;
19         }
20         media[i]=(float)somma[i]/n_esami;
21     }
```

Soluzione

```
1       printf("\n\nElenco studenti\n");
2       for(int i=0;i<n_studenti;++i) {
3           printf("%s %s - media: %.1f\n",studente[i].nome,
4           studente[i].cognome,media[i]);
5       }
6   }
```

Oggetti dinamici

Allocazione memoria

Processo utilizzato per assegnare la memoria ad un programma.

Allocazione **statica** (in fase di compilazione - stack).

Allocazione **dinamica** (in fase di esecuzione - heap).

Lo stack è un'area della memoria (*di dimensione fissa*) che conserva le variabili locali, i parametri delle funzioni e le chiamate alla stesse funzioni. Le variabili locali sono attive solo durante la chiamata ad una specifica funzione, dopo vengono "rilasciate".

Allocazione memoria: statica

Stack:

- Allocata / deallocata *automaticamente* dal compilatore
- La dimensione è fissa, definita a “compile-time” e non può “crescere” (problema dello stack overflow)
- LIFO
- Accesso più rapido

Allocazione memoria: dinamica

Heap:

- Allocata / deallocata *manualmente* tramite l'uso di specifiche funzioni
- La dimensione può cambiare, definita a "run-time" ed è meno limitata
- Nessun ordine specifico
- Accesso meno rapido

Oggetti dinamici

I *puntatori* possono essere usati per la creazione e la gestione di oggetti dinamici

Gli oggetti dinamici sono creati durante l'esecuzione del programma. Non è quindi necessaria una dichiarazione esplicita: *non serve il nome e non serve conoscerne il numero* a priori.

La memoria viene allocata attraverso l'utilizzo di specifiche funzioni: per esempio, `malloc()` (restituisce un puntatore)

Allocazione dinamica della memoria

Utilizzo della funzione `malloc()` - serve `stdlib.h`

```
1 p = (int *) malloc (sizeof(int));
```

Questa istruzione permette di allocare (dinamicamente) la memoria necessaria per un intero che sarà accessibile utilizzando il puntatore p.

`(int *)` è un *cast* al tipo specifico.

- Da questo momento possiamo accedere (tramite il puntatore p) a quella locazione di memoria e quindi all'intero senza l'utilizzo di una dichiarazione esplicita.

Allocazione dinamica della memoria

```
1 p = (int *) malloc (sizeof(int));
```

Possiamo modificare il valore puntato da p utilizzando la già nota rappresentazione:

```
1 *p=10;
```

E' necessario deallocare lo spazio per evitare che diventi successivamente inutilizzabile!

```
1 free(p);
```

Esempio stack

```c
void esempio_stack() {
    int a = 10; // 'a' nello stack
    int b = 20; // 'b' nello stack
    printf("a: %d, b: %d\n", a, b);
}
// La memoria è liberata dopo l'uso della funzione.
```

Esempio: stack vs heap

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

int main() {
    int *p; //p è nello stack

    p=(int *)malloc(sizeof(int));
    //malloc restituisce un indirizzo di heap
    *p=10; //modifica di heap

    printf("%p %p %d",&p,p,*p);
    free(p);
}
```

Esempio: step 1

```c
#include <stdio.h>

int main() {
    int n1,n2;

    printf("Inserisci un numero: ");
    scanf("%d",&n1);
    printf("Inserisci un altro numero: ");
    scanf("%d",&n2);

    printf("La somma vale %d", n1 + n2);
    return 0;
}
```

Esempio: step 2

```c
#include <stdio.h>

int main() {
    int n1,n2;
    int *p1,*p2;

    printf("Inserisci un numero: ");
    scanf("%d",&n1);
    printf("Inserisci un altro numero: ");
    scanf("%d",&n2);

    p1=&n1;
    p2=&n2;
    printf("La somma vale %d", *p1 + *p2);
    return 0;
}
```

Esempio: step 3

```c
#include <stdio.h>

int main() {
    int n1,n2;
    int *p1,*p2;

    p1=&n1;
    p2=&n2;

    printf("Inserisci un numero: ");
    scanf("%d",p1);
    printf("Inserisci un altro numero: ");
    scanf("%d",p2);

    printf("La somma vale %d", *p1 + *p2);
    return 0;
}
```

Esempio: step 4

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

int main() {
    int *p1,*p2;

    p1=(int *) malloc(sizeof(int));
    p2=(int *) malloc(sizeof(int));

    printf("Inserisci un numero: ");
    scanf("%d",p1);
    printf("Inserisci un altro numero: ");
    scanf("%d",p2);

    printf("La somma vale %d", *p1 + *p2);
    free(p1);
    free(p2);
    return 0;
}
```

Array dinamico: malloc()

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

int main () {
    int i,n;
    int *ptr;

    printf("Inserisci dimensione array: ");
    scanf("%d",&n);

    ptr=(int*) malloc(n * sizeof(int));
     for(i=0;i<n;i++) {
         printf("\nInserisci valore: ");
         scanf("%d",ptr+i);
     }

    for(i=0;i<n;i++)
        printf("%d ",*(ptr+i));

    free(ptr);
}
```

Array dinamico: calloc() - Contiguous Allocation

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

int main () {
    int i,n;
    int *ptr;

    printf("Inserisci dimensione array: ");
    scanf("%d",&n);

    ptr=(int*) calloc(n, sizeof(int)); //init to 0!
     for(i=0;i<n;i++) {
         printf("\nInserisci valore: ");
         scanf("%d",ptr+i);
     }

    for(i=0;i<n;i++)
        printf("%d ",*(ptr+i));

    free(ptr);
}
```

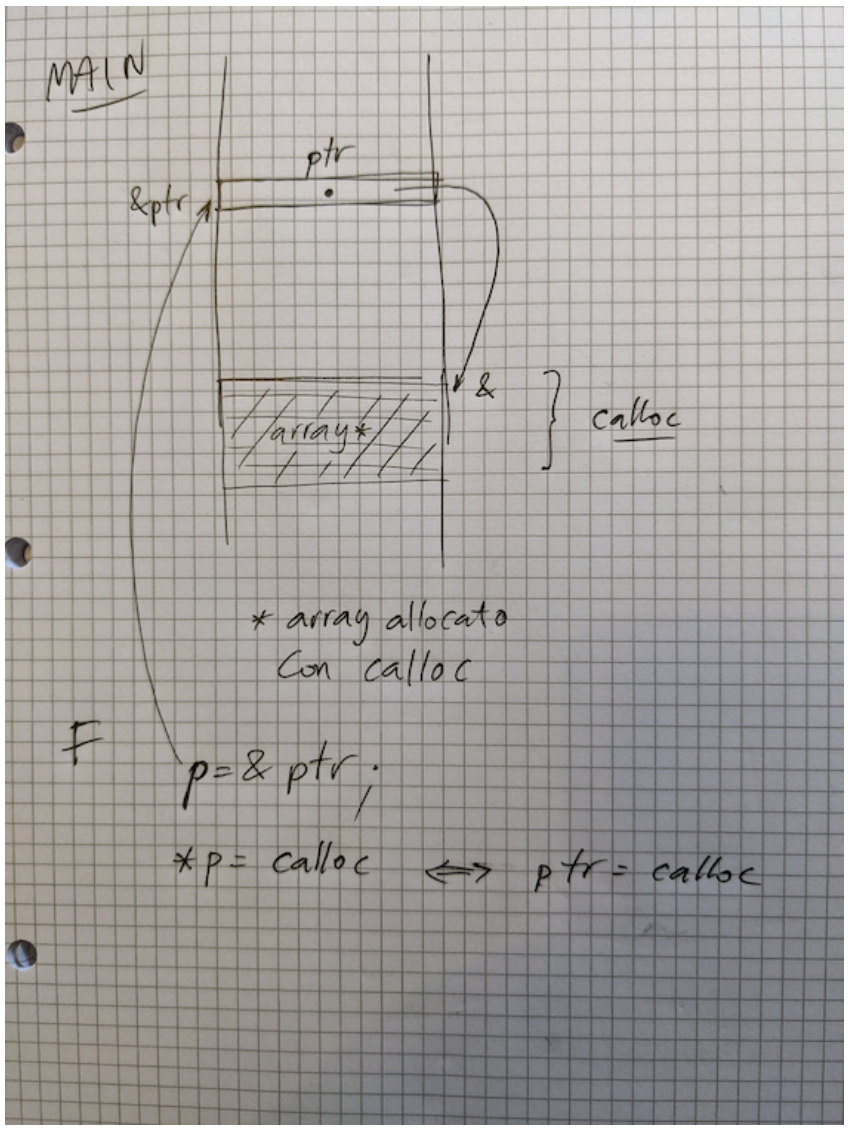
Array dinamico: con funzioni (sol. 1)

```c
#include <stdio.h>
#include <stdlib.h>
int * allocate_array(int size);

int main () {
    int i,n, *ptr;
    printf("\nInserisci dimensione array: ");
    scanf("%d",&n);
    ptr=allocate_array(n);
    printf("ptr: %p\n",ptr);
    for(i=0;i<n;i++) printf("%d ",*(ptr+i));
    free(ptr);
}
int * allocate_array(int size) {
    int i,*p;
    p=(int*) calloc(size, sizeof(int));
    printf("p: %p\n",p);
    for(i=0;i<size;i++)
        scanf("%d",p+i);
    return p;
}
```

Array dinamico: con funzioni (sol. 2)

```c
#include <stdio.h>
#include <stdlib.h>
void allocate_array(int size, int **p);
int main () {
    int i,n, *ptr;
    printf("Inserisci dimensione array: ");
    scanf("%d",&n);
    printf("Indirizzo ptr: %p\n",&ptr);
    allocate_array(n,&ptr);
    printf("Valore di ptr in main: %p\n",ptr);
    for(i=0;i<n;i++) printf("%d ",*(ptr+i));
    free(ptr);
}
void allocate_array(int size, int **p) { //p=&ptr;
    int i;
    printf("Indirizzo ptr nella f: %p\n",p);
    *p=(int*) calloc(size, sizeof(int));
    //*p equivale al valore di ptr del main -> ptr = calloc...
    printf("Contenuto di ptr in f: %p \n",*p);
    for(i=0;i<size;i++) scanf("%d",*p+i); //carico in array dinamico
}
```

Array dinamico



Array dinamico: esempio di realloc()

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

int main() {
    int *p,*q, n=5, m,i;

    p=(int *) malloc(n * sizeof(int));
    for(i=0;i<n;i++) scanf("%d",p+i);
    printf("\n");
    for(i=0;i<n;i++) printf("%d ",*(p+i));
    m=10;
    q=realloc(p,m * sizeof(int));
    for(i=n;i<m;i++) scanf("%d",q+i);
    for(i=0;i<m;i++) printf("%d ",*(q+i));
    free(q);

    return 0;
}
```

Array e liste concatenate

- Array:
  - Occupazione memoria (sovrastima)
  - Velocità (inserimento/eliminazione e spostamento)
- Liste lineari:
  - Insieme di elementi omogenei memorizzati in una posizione qualsiasi (semplicità di inserimento/eliminazione e spostamento)

Primo esempio, crea lista

Creare una lista di N interi e visualizzarla

Primo esempio, crea lista

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

struct lista {
    int num;
    struct lista *next;
};

struct lista *crealista();
void visualizza_lista(struct lista *);
void libera_lista(struct lista *);

int main() {
    struct lista *punt_lista;

    punt_lista=crealista();
    if(punt_lista!=NULL) visualizza_lista(punt_lista);
    if(punt_lista!=NULL) libera_lista(punt_lista);
}
```

Primo esempio, crea lista

```
1  struct lista *crealista() {
2    int n,i;
3    struct lista *p,*paux;
4    printf("Quanti elementi vuoi inserire? ");
5    scanf("%d",&n);
6    if(n==0) p=NULL;
7    else {
8      p=(struct lista *)malloc(sizeof(struct lista));
9      printf("Inserisci valore: ");
10     scanf("%d",&p->num);
11     paux=p;
12     for(i=2;i<=n;i++) {
13       paux->next=(struct lista *)malloc(sizeof(struct lista));
14       paux=paux->next;
15       printf("Inserisci valore: ");
16       scanf("%d",&paux->num);
17     }
18     paux->next=NULL;
19   }
20   return p;
21 }
```

Primo esempio, crea lista

```
1 void visualizza_lista(struct lista *p) {
2   while(p!=NULL) {
3     printf("%d ",p->num);
4     p=p->next;
5   }
6 }
```

Primo esempio, crea lista

```
1  void libera_lista(struct lista *p) {
2      struct lista *paux;
3
4      while(p!=NULL) {
5          paux=p;
6          p=p->next;
7          free(paux);
8      }
9  }
```

Crea lista con puntatore doppio

```c
void crea_lista(struct elemento **);

int main(void) {
    struct elemento *p_lista;

    crea_lista(&p_lista);
    return 0;
}
```

Crea lista con puntatore doppio

```
1  void crea_lista(struct elemento ** p){
2      struct elemento *paux;
3      int n,i;
4
5      printf("\nQuanti elementi vuoi inserire? ");
6      scanf("%d",&n);
7      if(n>0){
8          *p = malloc(sizeof(struct elemento));
9          printf("\nInserisci elemento: ");
10         scanf("%d",&((*p)->val));
11         paux = *p;
12         for(i = 2;i <= n;i++){
13             paux->next = malloc(sizeof(struct elemento));
14             paux = paux->next;
15             printf("\nInserisci elemento: ");
16             scanf("%d",&paux->val);
17         }
18         paux->next=NULL;
19     } else p=NULL;
20 }
```

Esempio: calcola minimo

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

struct lista {
    int num;
    struct lista *next;
};

struct lista *crealista();
void visualizza_lista(struct lista *);
int minimo(struct lista *);

int main() {
    struct lista *punt_lista;
    int min;

    punt_lista=crealista();
    if(punt_lista!=NULL) visualizza_lista(punt_lista);
    if(punt_lista!=NULL) printf("\nIl min vale: %d\n",
    min=minimo(punt_lista));
}
```

Esempio: calcola minimo

```
1  int minimo(struct lista *p) {
2    int min=p->num;
3
4    while(p!=NULL) {
5      if(p->num<min) min=p->num;
6      p=p->next;
7    }
8    return min;
9  }
```

Crea lista, in testa

```
1  struct lista *crea_intesta() {
2    int n,i;
3    struct lista *p,*paux;
4    printf("Quanti elementi vuoi inserire? ");
5    scanf("%d",&n);
6    if(n==0) p=NULL;
7    else {
8      p=(struct lista *)malloc(sizeof(struct lista));
9      printf("Inserisci valore: ");
10     scanf("%d",&p->num);
11     p->next=NULL;
12     for(i=2;i<=n;i++) {
13       paux=(struct lista *)malloc(sizeof(struct lista));
14       printf("Inserisci valore: ");
15       scanf("%d",&paux->num);
16       paux->next=p;
17       p=paux;
18     }
19   }
20   return p;
21 }
```

Inserisci elemento in una lista

La funzione deve **aggiungere** un elemento alla lista ad ogni chiamata. L’inserimento può essere eseguito: - in coda - in testa

Inserisci elemento in coda

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

struct lista {
    int num;
    struct lista *next;
};

struct lista *ins_coda(struct lista *);
void visualizza_lista(struct lista *);
```

Inserisci elemento in coda

```c
int main() {
    struct lista *punt_lista=NULL;
    int sel=-1;
    while(sel!=0) {
      printf("\n1- Inserisci elemento\n");
      printf("\n2- Visualizza elementi\n");
      printf("\n0- Esci\n");
      scanf("%d",&sel);
      switch (sel) {
        case 1:
          punt_lista=ins_coda(punt_lista);
          break;
        case 2:
          if(punt_lista!=NULL) {
            printf("\nLista elementi: ");
            visualizza_lista(punt_lista);
          }
          else printf("\nLista vuota\n");
          break;
        case 0:
          printf("\nProgramma terminato\n");
          break;
        default:
          printf("\nScelta non valida\n");
```

Inserisci elemento in coda

```
1  struct lista *ins_coda(struct lista *p) {
2    struct lista *p1,*paux;
3
4    p1=(struct lista *)malloc(sizeof(struct lista));
5    printf("Inserisci valore: ");
6    scanf("%d",&p1->num);
7    if(p==NULL) {
8      p=p1;
9      p->next=NULL;
10   }
11   else {
12     p1->next=NULL;
13     paux=p;
14     while(paux->next!=NULL) paux=paux->next;
15     paux->next=p1;
16   }
17   return p;
18 }
```

Inserisci elemento in testa

```
1  struct lista *ins_testa(struct lista *p) {
2    struct lista *p1;
3
4    p1=(struct lista *)malloc(sizeof(struct lista));
5    printf("Inserisci valore: ");
6    scanf("%d",&p1->num);
7    if(p==NULL) {
8      p=p1;
9      p->next=NULL;
10   }
11   else {
12     p1->next=p;
13     p=p1;
14   }
15   return p;
16 }
```

I File

I File

Nel C è possibile operare su file solamente in termini di sequenze di byte: **stream**

Non esistono funzioni di alto livello come in altri linguaggi

Usiamo i puntatori. File di **testo** e file **binari**

👉 Attenzione al SO.

Con i file si ha *finalmente* la possibilità di salvare i dati inseriti all'interno del programma

I File: apertura

Prima di poter scrivere o leggere da file è necessario **aprire** il file stesso

Per far questo è necessario includere la libreria `stdio.h` che contiene il tipo derivato `FILE`

👉 Per leggere e scrivere su file si utilizza un puntatore (a tipo derivato `FILE`)

```c
FILE *fileptr; //puntatore al file
```

I File: apertura e chiusura

1. `fopen()` associa un puntatore al file

👉 Oltre al nome del file occorre specificare il tipo di operazione da eseguire

```
1  FILE *fileptr;
2  fileptr = fopen(“my_file”, “r”); //r -> in lettura
```

2. `fclose()` chiude un file e permette di salvare i dati

```
1  fclose(fileptr);
```

I File: operazioni in apertura

I File: funzioni per lettura e scrittura

`fprintf()` e `fscanf()`: simili a `printf()` e `scanf()` ma interagiscono con i file e consentono di agire in modo "formattato"

I File: esempio `fprintf()`

```c
#include <stdio.h>

int main() {
  FILE *fileptr;
  int i;

  fileptr=fopen("file_01.txt","w");
  for(i=1;i<=5;i++)
    fprintf(fileptr,"%d ",i);
  fclose(fileptr);
}
```

I File: esempio `fscanf()`

```c
#include <stdio.h>

int main() {
  FILE *fileptr;
  int i,num;

  fileptr=fopen("file_01.txt","r");
  for(i=1;i<=5;i++) {
    fscanf(fileptr,"%d",&num);
    printf("%d ",num);
  }
  fclose(fileptr);
}
```

I File: fine file...

```c
#include <stdio.h>

int main() {
  FILE *fileptr;
  int num;

  fileptr=fopen("file_01.txt","r");
  while(fscanf(fileptr,"%d",&num)==1) printf("%d ",num);
  fclose(fileptr);
}
```

In C la funzione `feof()` rende un valore diverso da 0 quando arriva a fine file - ma attenzione

I File: altre funzioni per lettura e scrittura

`fgets()`: legge una riga da un file

`fputs()`: scrive una riga su un file

Il fine riga è individuato dal carattere `\n`

I File: fgets()

```c
#include <stdio.h>
#define DIM 100
int main() {
    FILE *fileptr;
    char vettore[DIM];
    fileptr=fopen("file_01.txt","r");
    if(fileptr==NULL) printf("\nFile inesistente\n");
    else {
        fgets(vettore,DIM,fileptr);
        printf("%s",vettore);
        fclose(fileptr);
    }
}
```

I File: `fgets()`

```
s=fgets(vettore,DIM,fileptr);
```

`s` indirizzo di vettore `vettore` vettore che conterrà la riga letta `DIM` dimensione del vettore `fileptr` puntatore al file da leggere

La funzione `fgets()` aggiunge a fine linea il carattere `\0` di fine stringa!

I File: `fgets()` e `stdin`

```c
#include <stdio.h>
#define DIM 100

int main() {
    FILE *fileptr;
    char vettore[DIM];
    fileptr=fopen("file_02.txt","w");
    printf("Inserisci del testo: ");
    fgets(vettore, DIM, stdin);
    fprintf(fileptr, "%s", vettore);
    fclose(fileptr);
    return 0;
}
```

I File: `fgets()` e `stdin`

La funzione `fgets()` può quindi essere sfruttata per leggere una riga da tastiera (superando i limiti della `scanf()`) dirottando l'input dal file alla tastiera (`stdin`)

I File: `fputs()`

La funzione `fputs()` scrive una riga su un file

```
fputs(vettore,fileptr);
```

I File: fputs()

```c
#include <stdio.h>
#define DIM 100
int main() {
    FILE *fileptr;
    char vettore[DIM],invio;
    int n,i;
    fileptr=fopen("file_03.txt","w");
    printf("\nQuante linee vuoi scrivere: ");
    scanf("%d",&n);
    scanf("%c",&invio);
    for(i=1;i<=n;i++) {
        printf("\nInserisci linea %d: ", i);
        fgets(vettore,DIM,stdin);
        fputs(vettore,fileptr);
    }
    fclose(fileptr);
}
```

I File: leggere e scrivere un singolo carattere

`fgetc()` e `fputc()`

- `fgetc(fileptr)`: restituisce un intero. Costante simbolica EOF (-1)
- `fputc(c,fileptr)` - Richiede in ingresso il carattere da scrivere

I File: note

- `fflush(fileptr)`: scarica su disco tutte le scritture contenute nel buffer
- Messagio a video: `fprintf(stdout,"Messaggio per l'utente");`
- La funzione `gets()` non si deve usare!

Esercizio

Scrivere un programma che permetta di caricare N numeri interi in un array e successivamente, tramite apposite funzioni consenta di: - calcolare la media dei valori - scrive su file i valori maggiore della media

Soluzione

```c
#include <stdio.h>
#define DIM 10

float calcola_media(int *, int);
void scrivi_file(int *, int, float);

int main() {
  int n,v[DIM],i;
  float media;

  do  {
    printf("Inserisci dimensione array: \n");
    scanf("%d", &n);
  } while(n<1 || n>DIM);

  printf("Inserisci i %d elementi:\n",n);
  for(i=0;i<n;i++)  {
      printf("elemento di indice - %d : ",i);
      scanf("%d",&v[i]);
  }
```

Soluzione

```
1    media=calcola_media(v, n);
2    printf("\nLa media vale: %.1f",media);
3
4    for(i=0;i<n;i++)
5        printf("\nelemento di indice - %d: %d ",i, v[i]);
6
7    scrivi_file(v, n, media);
8  }
```

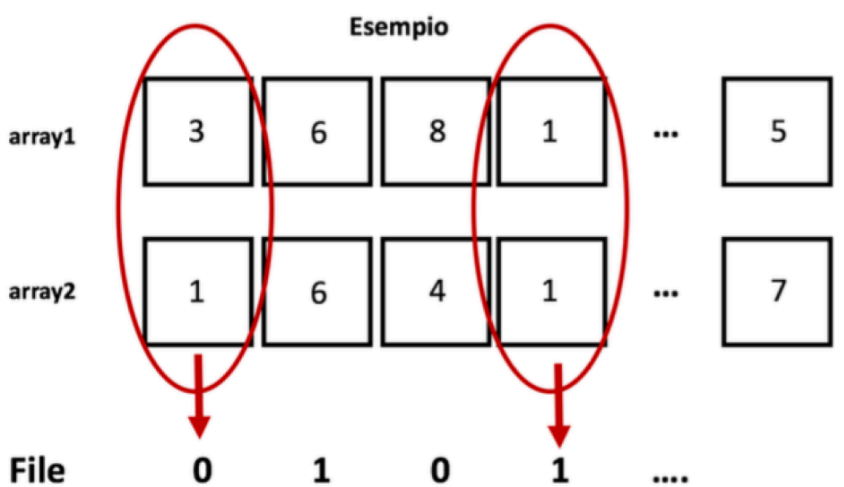
Soluzione

```c
float calcola_media(int *v, int n) {
  int i;
  float media=0;

  for(i=0;i<n;i++)
    media+=*(v+i);
  return media/n;
}

void scrivi_file(int *v, int n, float media) {
  FILE *fileptr;
  int i;

  fileptr=fopen("my_file.txt","w");
  for(i=0;i<n;i++) {
    if(*(v+i)>media)
      fprintf(fileptr,"%d ",*(v+i));
  }
}
```

Esercizio

Scrivere un programma in C che:
- legga M numeri interi da un file e li carichi su un array
- legga N numeri interi da un secondo file e li carichi su un secondo array

M e N non sono noti a priori e comunque entrambi sono minori di 100.
Il programma deve permettere, attraverso apposite **funzioni**, di:
1- calcolare (separatamente per ogni array) la media dei numeri interi contenuti in un determinato array
2- scrivere un nuovo file che contenga una sequenza di 0 e 1 rispettando la seguente regola: nel nuovo file si scriverà 0 quando i due array in posizione ***i*** hanno un valore tra loro diverso, viceversa si scriverà 1 se i due array in posizione *i* hanno un valore uguale. Nello scorrere i due array si presti attenzione al fatto che le loro dimensioni potrebbero essere differenti.



Soluzione

```c
#include <stdio.h>
#define DIM 100

void scrivi_file(int *, int *, int, int);

int main() {
  FILE *fileptr1,*fileptr2;
  int v1[DIM],v2[DIM],i,j;

fileptr1=fopen("file_01.txt","r");
    if(fileptr1==NULL) printf("\nFile inesistente\n");
    else {
        i=0;
        while(fscanf(fileptr1, "%d", &v1[i])==1) i++;
    }
```

Soluzione

```
1   fileptr2=fopen("file_02.txt","r");
2     if(fileptr2==NULL) printf("\nFile inesistente\n");
3     else {
4         j=0;
5         while(fscanf(fileptr2, "%d", &v2[j])==1) j++;
6     }
```

Soluzione

```
  if(fileptr1!=NULL && fileptr2!=NULL)
    scrivi_file(v1, v2, i, j);
}
```

Soluzione

```
1  void scrivi_file(int *v1, int *v2, int d1, int d2) {
2    FILE *fileptr;
3    int i;
4
5    fileptr=fopen("file_03.txt","w");
6    if(d1>d2) d1=d2;
7    for(i=0;i<d1;i++) {
8      if(*(v1+i) == *(v2+i)) fprintf(fileptr,"%d ",1);
9      else fprintf(fileptr,"%d ",0);
10   }
11 }
```

I File: `fread()` - (low-level reading)

`fread()` permette di leggere un file, traferire (stream) su un vettore e restituire il numero di elementi letti

Se è stato letto tutto il contenuto del file `fread()` restituisce `0`

La lettura avviene in sequenza spostando il puntatore del numero di byte necessario.

I File: esempio di lettura

```c
#include <stdio.h>
#define DIM 100

int main() {
  FILE *fileptr;
  char vettore[DIM];
  int size=1; //in byte di 1 elemento
  int n; //num elementi letti
  int i;

  fileptr=fopen("file01","r"); //path!

  if(fileptr==NULL)
    printf("\nFile inesistente\n");
  else {
    n=fread(vettore,size,DIM,fileptr);
      for(i=0;i<n;i++)
        printf("%c",vettore[i]);
    fclose(fileptr);
  }
}
```

I File: `fwrite()` - (low-level writing)

`fwrite()` permette di scrivere su un file, traferire (stream) da un vettore ciò che legge e restituire il numero di elementi scritti

La scrittura avviene in sequenza spostando il puntatore del numero di byte necessario.

I File: esempio di scrittura

```c
#include <stdio.h>
#include <string.h>
#define DIM 100

int main() {

  FILE *fileptr;
  char vettore[DIM];
  int size=1,n,i,len;

  printf("Inserisci del testo: ");
  scanf("%s",vettore);
  len=strlen(vettore);

  fileptr=fopen("file02","w");
  n=fwrite(vettore,size,len,fileptr);
  fclose(fileptr);
}
```

I File: esempio di scrittura

```c
#include <stdio.h>
int main() {
    FILE *fileptr;
    int vettore[5],i;
    fileptr=fopen("file_04","wb");
    if(fileptr==NULL) printf("\nFile inesistente\n");
    else {
        for(i=0;i<5;i++)
            scanf("%d",&vettore[i]);
        fwrite(vettore,sizeof(int),5,fileptr);
        fclose(fileptr);
    }
    fileptr=fopen("file_04","rb");
    if(fileptr==NULL) printf("\nFile inesistente\n");
    else {
        fread(vettore,sizeof(int),5,fileptr);
        printf("\nElementi in array:");
        for(i=0;i<5;i++) printf("\n%d",vettore[i]);
        fclose(fileptr);
    }
}
```

I File: *sequenziale vs random*

Posizionare il puntatore: `fseek()`

Consente di posizionare il puntatore in una qualunque posizione all'interno del file (sia in lettura che in scrittura)

`fseek(fp, n, 0)` fp viene posizionato sul n° byte a partire dall'inizio del file `fseek(fp, n, 1)` fp viene posizionato sul n° byte a partire dalla posizione attuale `fseek(fp, n, 2)` fp viene posizionato sul n° byte a partire dalla fine del file

`n=ftell(fp)` //restituisce la posizione attuale del puntatore

I File: *sequenziale vs random*

```c
#include <stdio.h>
int main() {
  FILE *fileptr;
  int i;
  fileptr=fopen("/Users/matteofraschini/Desktop/file03.txt","w");
  for(i=0;i<=9;i++) {
    printf("%ld ",ftell(fileptr));
    fprintf(fileptr,"%d ",i);
  }
  fclose(fileptr);
  fileptr=fopen("/Users/matteofraschini/Desktop/file03.txt","r+");
  fseek(fileptr, 4, 0);
  fprintf(fileptr,"%d",0);
  fclose(fileptr);
}
```

Approfondimenti

Generazione numeri casuali (0-10)

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int rendi_random(int min, int max);
int main() {
  int i,casuale,n_ripet=20,min=0,max=10;
  srand(time(0));
  for(i=0;i<=n_ripet;i++) {
    casuale=rendi_random(min, max);
    printf("\nNumero casuale: %d",casuale);
  }
  return 0;
}
int rendi_random(int min, int max) {
  int num;

  num = (rand() % (max - min + 1)) + min;
  return num;
}
```

scanf() e spazi

```c
#include <stdio.h>
#define DIM 100

int main() {
  char stringa[DIM];
  scanf("%[^\n]",stringa);
  printf("\n%s",stringa);
  return 0;
}
```

Misurare tempo di calcolo

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

int main() {
  clock_t start,end;
  double elapsed;
  int i;
  start=clock();
  //Esempio di calcolo
  for(i=0;i<1000;i++)
    i*i;
  //Fine esempio
  end=clock();
  elapsed=((double)(end-start))/CLOCKS_PER_SEC;
  printf("\nTempo trascorso: %f",elapsed);
  return 0;
}
```

Ricorsione

Funzioni ricorsive: alternativa alla definzione di un ciclo

Una funzione chiama se stessa…

```c
#include <stdio.h>
int fattoriale(int);

int main() {
  int num,fat;
  printf("\nInserisci un numero intero: ");
  scanf("%d",&num);
  fat=fattoriale(num);
  printf("\n%d!: %d",num,fat);
  return 0;
}

int fattoriale(int num) {
  if(num!=0) return (num*fattoriale(num-1));
  else return 1;
}
```

Ricorsione: nelle liste

```
1 void visualizza_lista(struct lista *p) {
2 while(p!=NULL) {
3 printf("%d ",p->num);
4 p=p->next; }
5 }
```

```
1 void visualizza_lista(struct lista *p) {
2 if(p!=NULL) {
3 printf("%d ",p->num);
4 visualizza_lista(p->next); }
5 }
```