

Laboratorio di Programmazione

Introduzione alla programmazione in C

Dott.ssa Veronica Rossano
rossano@di.uniba.it
<http://www.di.uniba.it/~rossano>

Testi adottati

- A. Kelley, I. Pohl
C Didattica e Programmazione
Pearson Education/Addison-Wesley
- B.W. Kernighan, D.M. Ritchie
Linguaggio C (ANSI C), II ed.
Pearson Education/Addison-Wesley

Testi consigliati

- H.M. Deitel, P.J. Deitel, *Corso Completo di programmazione (Terza Edizione)*, APOGEO EDUCATION
- Hanly J.R., Koffman E.B.
Problem Solving and Program Design in C
Pearson Education/Addison-Wesley

Dispense

- Le dispense del laboratorio di programmazione sono disponibili sulla piattaforma di e-learning all'indirizzo:
<http://multimedialab.di.uniba.it/>

Ambiente di sviluppo...

- Eclipse IDE for C/C++

Eclipse is an open source community whose projects are focused on building an extensible development platform, runtimes and application frameworks for building, deploying and managing software across the entire software lifecycle. Many people know us, and hopefully love us, as a Java IDE but Eclipse is much more than a Java IDE.

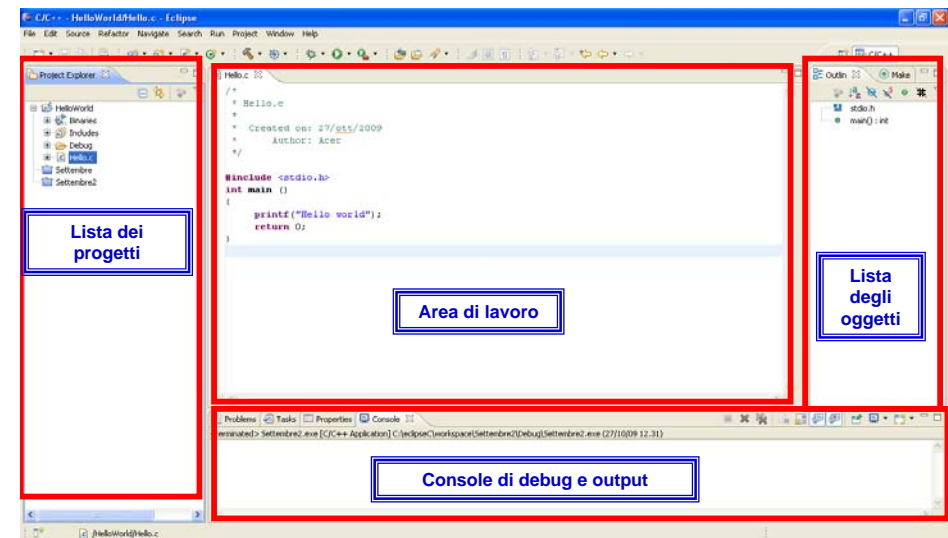
...Ambiente di sviluppo...

- Per scaricare l'IDE di Eclipse
<http://www.eclipse.org/downloads/>
- Oltre all'ambiente di sviluppo è necessario installare anche il compilatore C
- Le istruzioni di installazione dell'IDE di Eclipse e del compilatore **MinGW** sono reperibili all'indirizzo
http://maresca.dis.unina.it/wiki/index.php/Installazione_di_Eclipse

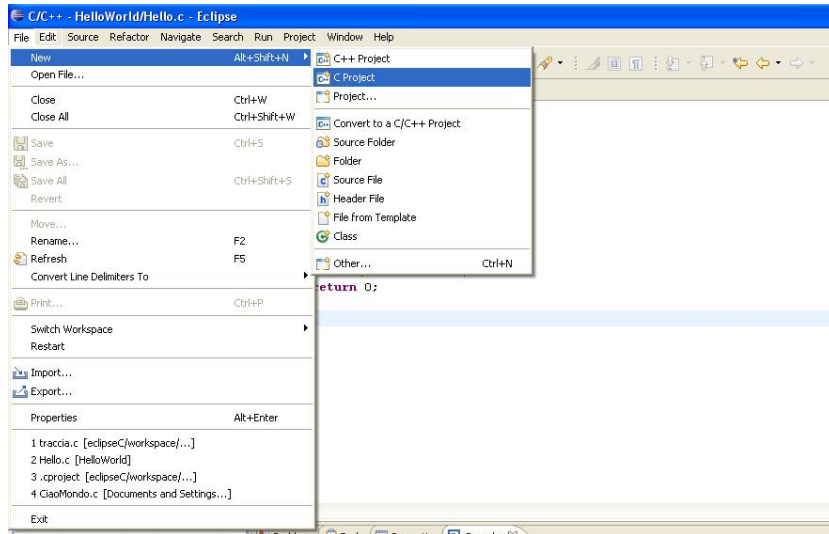
...Ambiente di sviluppo (in alternativa)

- Dev-C++ ambiente integrato di sviluppo per linguaggi di programmazione C/C++
- Dev-C++ è un Free Software distribuito con licenza GNU General Public License (GPL)
- L'ambiente è facilmente reperibile sulla rete, ed inoltre è scaricabile dalla piattaforma di e-learning

Il workbench di ECLIPSE



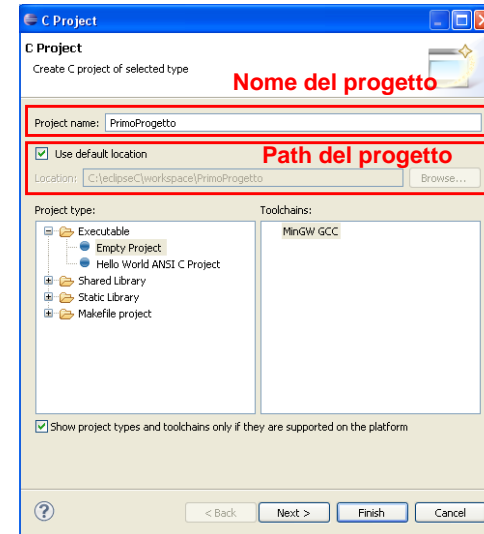
Eclipse: creare un progetto...



Corso laurea in Informatica e Comunicazione Digitale (sede di Taranto) – a.a. 2009-10
Laboratorio di Programmazione - Veronica Rossano

9/43

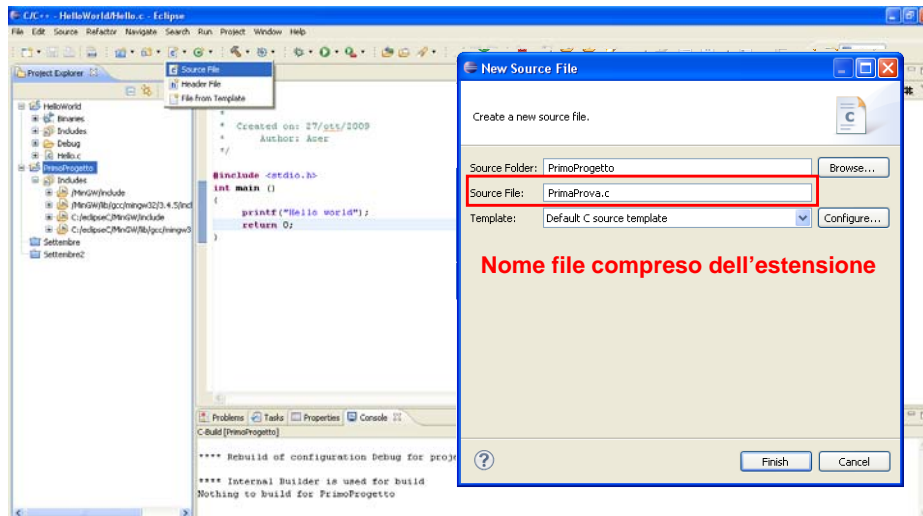
...Eclipse: creare un progetto



Corso laurea in Informatica e Comunicazione Digitale (sede di Taranto) – a.a. 2009-10
Laboratorio di Programmazione - Veronica Rossano

10/43

...Eclipse: creare un file C



Il primo programma in C...

```
#include <stdio.h>

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

- È un semplice programma che stampa a video la stringa Hello World!

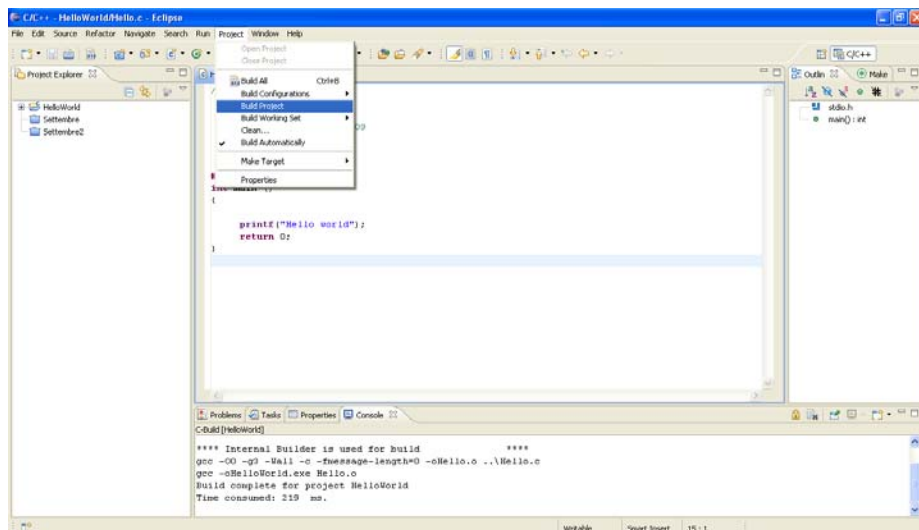
...il primo programma in C...

- L'**include** un comando che consente di richiamare ed utilizzare le librerie
 - ▢ **Stdio.h** è una libreria standard di C che consente di utilizzare il comando **printf** per la stampa a video
- La parte principale di un programma in C è il **main**
 - ▢ Il **main** è indispensabile ed unico
 - ▢ **Int** prima del main consente al programma di restituire un valore
- Le **parentesi graffe** delimitano l'inizio e la fine di una sequenza di istruzioni
- Il **punto e virgola** chiude un'istruzione
- L'istruzione **return 0;** indica che il programma è terminato con successo

...il primo programma in C

- I programmi C vengono salvati in file con estensione **.c**
- Per essere eseguiti devono essere prima compilati
- La compilazione è il processo di traduzione da linguaggio sorgente (il C nel nostro caso) a linguaggio oggetto (comprensibile alla macchina)
- Il risultato del processo di compilazione è un file eseguibile (**.exe**)

Compilare un programma...



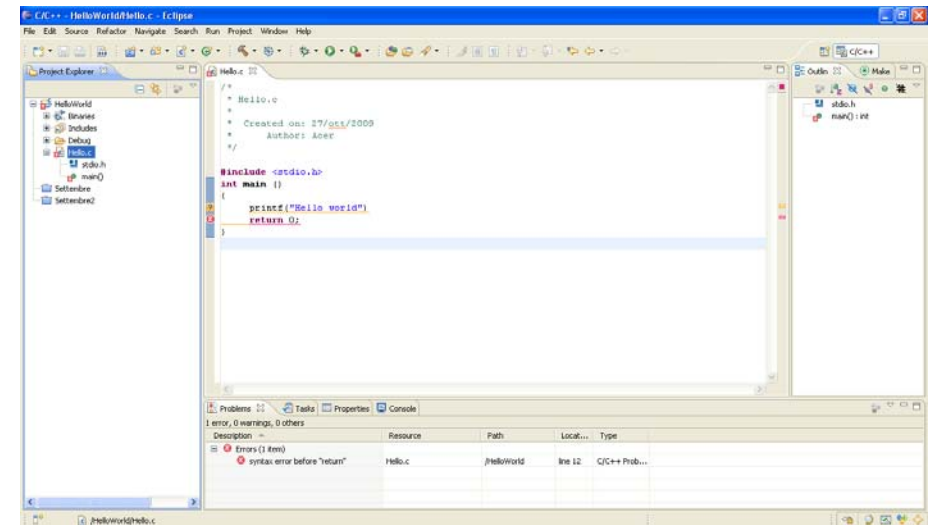
...Compilare un programma...

- Le fasi di compilazione di un programma C
 - ▢ Il codice sorgente viene controllato dal preprocessore che:
 - rimuove eventuali commenti presenti nel sorgente
 - interpreta speciali direttive per il preprocessore denotate da **"#"** (come ad esempio **#include**)
 - rileva eventuali errori sintattici
 - ▢ Il risultato del preprocessore sarà un nuovo codice sorgente "espanso" pronto per essere tradotto dal compilatore C in codice assembly

...Compilare un programma...

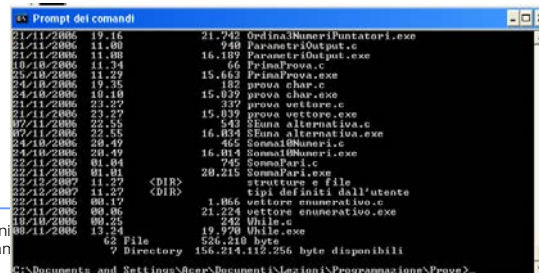
- L'assembler crea il codice oggetto salvandolo in un file oggetto (.o in Unix/Linux e .obj in Dos/Windows)
- Il Link editor ha il compito di collegare tutti i file oggetto risolvendo eventuali dipendenze e creando un unico programma eseguibile
- La compilazione individua errori di sintassi ma non può rilevare errori logici (come ad esempio un ciclo che non termina)
- In alcuni casi possono essere segnalati dei **Warning** che non costituiscono errore, ma che segnalano parti di codice strane e quindi sulle quali porre attenzione per eventuali errori logici

Errori di compilazione

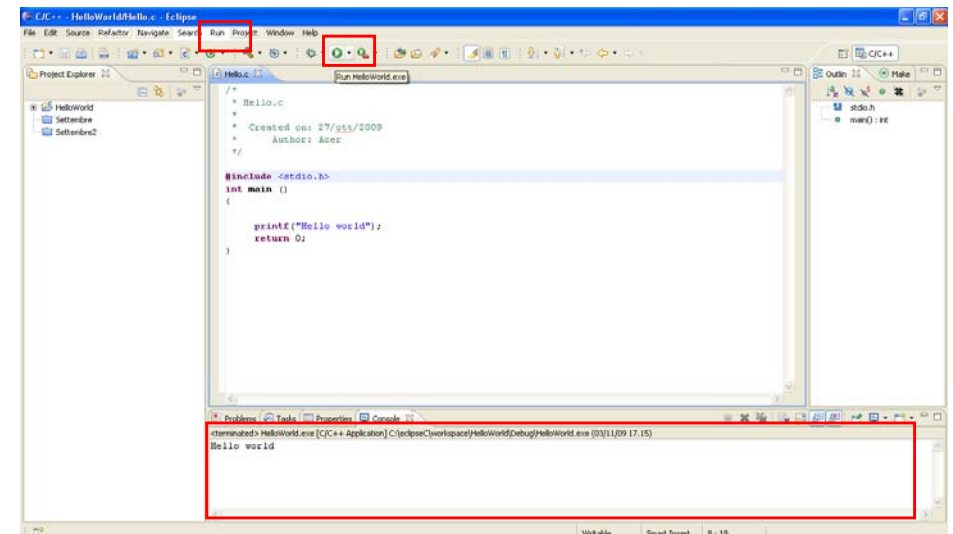


L'esecuzione di un programma...

- Un programma può essere mandato in esecuzione:
 - ▣ Direttamente dall'ambiente di programmazione cliccando sul comando Run
 - ▣ Cliccando sul file eseguibile creato in fase di compilazione
 - ▣ Dal prompt dei comandi



...L'esecuzione di un programma



Struttura di un programma C

```

Direttive per il preprocessore (es. #include)
Dichiarazione variabili globali
int main()
{
    Variabili locali al main;
    Istruzione1;
    Istruzione2;
    Istruzione3;
    ...
    IstruzioneN;
    return 0;
}

```

Corso di laurea in Informatica e Comunicazione Digitale (sede di Taranto) – a.a. 2009-10
Laboratorio di Programmazione - Veronica Rossano

21/43

Un secondo esempio

```

#include <stdio.h>
int main ()
{
    int a, b, media; /*definizione delle variabili
                    locali al main*/

    a=6;
    b=10;
    media=(a+b)/2;
    printf("Media tra 6 e 10: %d", media);
    return 0;
}

```

- Programma che calcola la media tra 6 e 10

Corso di laurea in Informatica e Comunicazione Digitale (sede di Taranto) – a.a. 2009-10
Laboratorio di Programmazione - Veronica Rossano

Una variabile è...

- ...un nome simbolico (identificatore) che denota un'area di memoria e, tramite essa, il valore contenuto
 - Contiene una rappresentazione di un oggetto su cui l'algoritmo opera
- Nel linguaggio C la definizione di una variabile è obbligatoria e deve precedere le istruzioni che la utilizzano
- La dichiarazione segue la sintassi

Tipo_variabile Identificatore_variabile

- Int a;
- Char risposta;

Corso di laurea in Informatica e Comunicazione Digitale (sede di Taranto) – a.a. 2009-10
Laboratorio di Programmazione - Veronica Rossano

23/43

Tipi di variabili in C...

- La definizione (o dichiarazione) del tipo di variabile è necessaria per informare l'esecutore su
 - dominio della variabile
 - insieme di operazioni effettuabili su di essa
 - modo attraverso cui ci si può riferire ad essa
- Il linguaggio C offre i seguenti tipi di dati predefiniti:
 - Int - Numero intero
 - Float - Numero reale (32 bit)
 - Double - Numero reale "lungo" (64 bit)
 - Char - Tipo Carattere

Corso di laurea in Informatica e Comunicazione Digitale (sede di Taranto) – a.a. 2009-10
Laboratorio di Programmazione - Veronica Rossano

24/43

Il tipo int(eger)

- È utilizzato in C per poter rappresentare i numeri interi
- [-32768, 32767] rappresenta il range di dati di tipo int
- Le operazioni possibili sono le classiche operazioni aritmetiche (+, -, *, /) e operazioni logiche di confronto
- Esempi di valori integer sono:
 - -10500 256 +36 32767

I tipi float e double (real)...

- Sono utilizzati in C per rappresentare un sottoinsieme dei numeri reali
 - alcuni numeri reali non possono essere rappresentati in nessuna notazione perché troppo grandi o troppo piccoli
- C utilizza la notazione in virgola mobile per la rappresentazione dei numeri reali
 - 2.345e2 rappresenta 234.5 (cioè $2.345 \cdot 10^2$)

...I tipi float e double (real)

- Le operazioni possibili sono le classiche operazioni aritmetiche (+, -, *, /) e operazioni logiche di confronto
- Esempi di valori float e double sono:
 - -15.0e-04 12e+5 256.36
- Esempi di valori non corretti sono:
 - 150 (manca il punto decimale) 32,56 (la virgola non è ammessa)
 - -15e-0.3 (0.3 non è un esponente valido)

Tipo Char (carattere)

- È utilizzato per rappresentare tutti i singoli caratteri alfanumerici
- Un valore di tipo char all'interno di una istruzione si racchiude tra apici
 - Iniziale = 'A'
- Le operazioni possibili sono le operazioni logiche di confronto
- Esempi di valori char sono:
 - 'A' '3' '*' ' ' ''

Gli identificatori...

- Un identificatore, dal punto di vista sintattico, è una sequenza di caratteri alfabetici e/o numerici
 - In C un identificatore DEVE iniziare con una **lettera** o con il simbolo `_` (underscore)
- Un identificatore **standard** è un nome che ha già un significato preciso (ad es. `printf`, `scanf`)

...Gli identificatori

- Un identificatore definito dall'utente può essere utilizzato per:
 - una variabile
 - una operazione definita dall'utente (una funzione)
- ATTENZIONE:
 - Non si possono usare le parole riservate (es. `printf`)
 - Gli identificatori **MEDIA**, **media** e **Media** sono diversi
 - Gli identificatori non possono contenere spazi

Le istruzioni di assegnazione

- Un'istruzione di assegnazione memorizza un valore o un risultato di una computazione in una variabile
- La sintassi in C:

Variabile = Espressione

- `a=6;`
- `b=secondo_valore;`
- `Media = (a+b)/2;`

dove *Espressione* può essere:

- un valore costante
- una variabile
- il risultato di un'espressione aritmetica

Le istruzioni di ingresso

- Istruzioni di ingresso consentono di acquisire informazioni dall'esterno
- L'operatore **scanf** è una funzione definita nella libreria `<stdio.h>`
- La sintassi:

```
scanf (formato_input, lista_input);
■ scanf ("%d", &elemento);
■ scanf ("%c", &carattere);
```

- Dove **formato_input** indica il tipo di dato che l'esecutore deve aspettarsi in input. È indicato usando un **placeholder**

Placeholder (I/O)

Placeholder

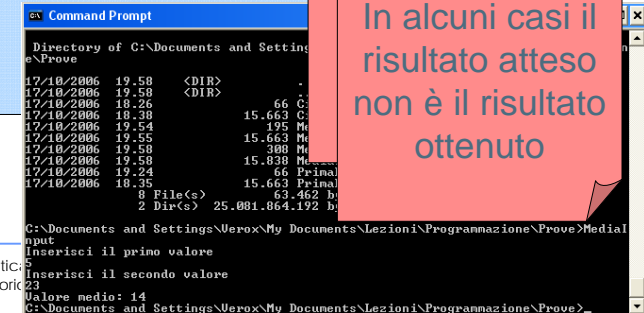
- %c
- %d
- %f
- %lf

Tipo variabile

- Char
- Int
- Float
- Double

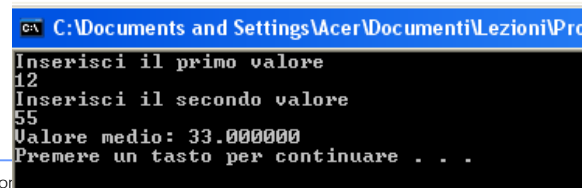
Un esempio più "generale"

```
#include <stdio.h>
int main ()
{
    int a, b, media;
    /*definizione delle variabili locali al main*/
    printf("Inserisci il primo valore \n");
    scanf("%d", &a);
    printf("Inserisci il secondo valore \n");
    scanf("%d", &b);
    media=(a+b)/2;
    printf("Valore medio: %d ", media);
    return 0;
}
```



L'esempio "generale"

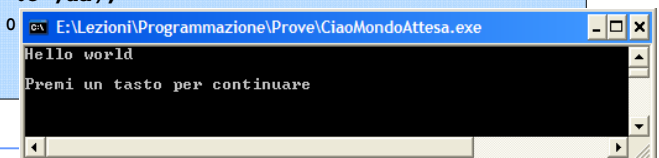
```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int a, b; /*definizione delle variabili locali al main*/
    float media;
    printf("Inserisci il primo valore \n");
    scanf("%d", &a);
    printf("Inserisci il secondo valore \n");
    scanf("%d", &b);
    media=(a+b)/2;
    printf("Valore medio: %f \n", media);
    system ("pause");
    return 0;
}
```



Inserire una pausa nell'esecuzione...

- Per imporre delle pause nell'esecuzione è possibile
 - chiedere un input all'utente

```
#include <stdio.h>
int main ()
{
    char a;
    printf("Hello world \n\n");
    printf("Premi un tasto per continuare... ");
    scanf("%c", &a);
    return 0;
}
```

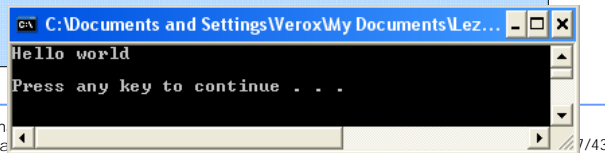


...Inserire una pausa nell'esecuzione

- Utilizzare una funzione standard di libreria
 - `system("pause")` che è incluso nella libreria `stdlib.h`

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

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



Corso di laurea in Informatica e Comunicazione Digitale (sede di Taranto) – a.a. 2009-10
Laboratorio di Programmazione - Veronica Rossano

7/43

Un esercizio

- Creare l'algoritmo e poi il programma che consenta di calcolare l'indice di massa corporea (BMI) sapendo che BMI è uguale al peso (espresso in kg) diviso l'altezza (espressa in m) al quadrato

Corso di laurea in Informatica e Comunicazione Digitale (sede di Taranto) – a.a. 2009-10
Laboratorio di Programmazione - Veronica Rossano

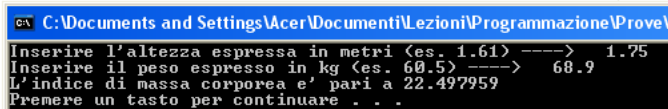
38/43

Calcolo dell'BMI

```
/*
Name: Calcola l'Indice di Massa Corporea
Copyright:
Author:
Date: 24/10/06 18.52
Description: Questo programma consente di calcolare l'indice di massa corporea (bmi)
*/
```

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

int main ()
{
    float altezza, peso, bmi;
    printf("Inserire l'altezza espressa in metri (es. 1.61) ----> ");
    scanf("%f", &altezza);
    printf("Inserire il peso espresso in kg (es. 60.5) ----> ");
    scanf("%f", &peso);
    /* calcolo dell'indice di massa corporea */
    bmi = peso / (altezza * altezza);
    printf("L'indice di massa corporea e' pari a %f \n", bmi);
    system("pause");
    return 0;
}
```



Commenti...

- Sono parti del programma che però sono ignorate dal compilatore e non sono tradotte in linguaggio macchina.
- Sono utili perché:
 - rendono più facile la comprensione di un programma
 - per descrivere lo scopo del programma,
 - per descrivere il significato degli identificatori e/o lo scopo di ciascun passo del programma

Corso di laurea in Informatica e Comunicazione Digitale (sede di Taranto) – a.a. 2009-10
Laboratorio di Programmazione - Veronica Rossano

40/43

...Commenti...

- La sintassi per un commento:

```
/*
    commento
*/
```

- L'intestazione tipica di un programma inserita automaticamente con DEV-C++

```
/*
Name:
Copyright:
Author:
Date:
Description:
*/
```

Corso di laurea in Informatica e Comunicazione Digitale (sede di Taranto) – a.a. 2009-10
Laboratorio di Programmazione - Veronica Rossano

41/43

Un esercizio

- Creare l'algoritmo e poi il programma che consenta di convertire una distanza espressa in miglia in una distanza espressa in km sapendo che 1 miglio (US/UK) = 1,609 km

Corso di laurea in Informatica e Comunicazione Digitale (sede di Taranto) – a.a. 2009-10
Laboratorio di Programmazione - Veronica Rossano

42/43

Conversione da Miglia a Km

```
/*
Name: ConvertiInKM
Copyright:
Author:
Date: 23/10/06 18.26
Description: Questo programma converte una distanza espressa in miglia
            ad una distanza espressa in KM
*/
#include <stdio.h>
#include <stdlib.h>
#define KMS_PER_MIGLIA 1.609 /* 1609e-3 costante utile per la conversione */

int main ()
{
    float miglia, kms;
    printf("Inserire la distanza in Miglia ----> ");
    scanf("%f", &miglia);

    /* conversione della distanza in km */
    kms = KMS_PER_MIGLIA * miglia;
    printf("La distanza %f corrisponde a %f chilometri. \n", miglia, kms);
    system("pause");
    return 0;
}
```

Le costanti

- In alcuni programmi si utilizzando dati che non variano mai (o quasi mai)
- Tali dati, detti **costanti**, si definiscono nella parte del programma che contiene le dichiarative per il preprocessore con la seguente sintassi

```
#define NOME_COSTANTE valore
#define KMS_PER_MIGLIA 1.609
```

- Per convenzione si utilizzano le maiuscole per gli identificatori

Corso di laurea in Informatica e Comunicazione Digitale (sede di Taranto) – a.a. 2009-10
Laboratorio di Programmazione - Veronica Rossano

44/43

Strutture di controllo

Sequenza – Selezione – Iterazione in C

Dott.ssa Veronica Rossano
rossano@di.uniba.it
<http://www.di.uniba.it/~rossano>

Le strutture di controllo

- Sequenza
 - Concatenazione di azioni
- Selezione
 - Scelta di azioni alternative
 - Dipendenza da una condizione
- Iterazione
 - Ripetizione di una certa azione
 - Dati potenzialmente diversi
 - Dipendenza da una condizione

Le strutture di controllo in C...

■ Sequenza

```
Istruzione1;  
Istruzione2;  
Istruzione3;  
...  
IstruzioneN;
```

```
a=6;  
b=10;  
media=(a+b)/2;  
printf("Media tra 6 e 10: %d", media);
```

La selezione...

- La struttura di controllo selezione consente di definire quale blocco di istruzioni eseguire al verificarsi o meno di una condizione

...La selezione

■ Selezione

```
if (condizione)
{
    Sequenza di istruzioni da eseguire se la
    condizione è verificata;
}
else
{
    Sequenza di istruzioni da eseguire se la
    condizione non è verificata;
}
```

```
if (a>b)
    printf("Il primo numero e' maggiore del secondo");
else
    printf("Il secondo numero e' maggiore del primo");
```

Laboratorio di Programmazione - Veronica Rossano

5

Le condizioni...

- Le condizioni sono espressioni particolari che stabiliscono se eseguire o meno un gruppo di istruzioni
- Gli operatori utilizzabili sono:
 - Operatori relazionali
 - Operatori di uguaglianza
- Le condizioni possono essere operazioni di confronto tra variabili e/o costanti

Laboratorio di Programmazione - Veronica Rossano

6

...Le condizioni...

■ La sintassi in C:

variabile	operatore relazionale	variabile
variabile	operatore relazionale	costante
variabile	operatore di uguaglianza	variabile
variabile	operatore di uguaglianza	costante

■ Operatori relazionali

- < minore di
- > maggiore di
- <= minore uguale di
- >= maggiore uguale di

Laboratorio di Programmazione - Veronica Rossano

7

...Le condizioni

■ Operatori di uguaglianza

- == uguale a
- != diverso da

Esempi di condizioni:

```
a<b
x <= 0
x>=y
risp=='s'
conta!=10
```

Laboratorio di Programmazione - Veronica Rossano

8

Operatori logici...

■ Gli operatori logici:

- && → AND
- || → OR
- ! → NOT

consentono di creare espressioni logiche più complesse

```
voto>=18 && voto<=30
anni<=10 || anni>=65
!(voto==30)
```

...Operatori logici

■ Esempi

- La temperatura odierna è compresa tra [TempMin, TempMax]
 - (TempMin <=TempOggi) && (TempOggi<=TempMax)
- Gli studenti che alla prova di laboratorio non hanno ottenuto un voto compreso tra 18 e 30 non sono ammessi all'esame orale
 - !((18<= voto) && (voto<=30))

Ordine di precedenza

- La valutazione di una espressione è determinata dall'ordine di precedenza degli operatori

! + -	(operatori unari)	alta
* / %		
+ -		
< <= >= >		
== !=		
&&		
=		bassa

...Ordine di precedenza...

■ Esempi

- $-x - y * z \rightarrow (-x) - (y * z)$
- $x + y < \min + \max \rightarrow (x + y) < (\min + \max)$
- $x < y \mid \mid x < z \&\& x > 0 \rightarrow (x < y) \mid \mid (x < z \&\& x > 0)$

- Per cambiare l'ordine delle precedenze è possibile utilizzare le parentesi come nell'algebra

...Ordine di precedenza

- Supponendo che le seguenti variabili siano così avvalorate:
 - $x=3.0$
 - $y=4.0$
 - $z=2.0$
 - $flag=0$
- Determinare i valori delle seguenti espressioni:
 - $!flag$
 - $x + y / z <= 3.5$
 - $!flag \mid \mid (y + z >= x - z)$ ←
 - $!(flag \mid \mid (y + z >= x - z))$ ←

Short circuit evaluation

Assegnazioni logiche

- L'ordine di precedenza consente di realizzare le assegnazioni logiche a variabili intere che assumono valore 0 se la condizione risulta falsa e 1 altrimenti

```
int eta, anziano;
anziano = (eta > 65);
```

- La variabile anziano assumerà valore 1 se eta è maggiore di 65 e 0 viceversa

```
int lettera;
char ch;
lettera = ('A' <= ch && ch <= 'Z' ) || ('a' <= ch && ch <= 'z' );
```

Confronto tra caratteri

- In C è possibile utilizzare gli operatori relazionali e di uguaglianza anche per i caratteri
 - $'a' < 'b'$
 - Vero
 - $'9' > '0'$
 - Vero
 - $'a' <= car \ \&\& \ car <= 'z'$
 - Vero se car è un carattere minuscolo
 - $'a' == 'A'$
 - Falso

Selezione ad una alternativa

- È possibile costruire selezioni con un'unica alternativa
- La sequenza di istruzioni è eseguita solo se la condizione è verificata, in caso contrario è ignorata

```
if (x > 0.0)
    prodotto = prodotto * x;
```

Esercizio

- Scrivere un algoritmo e il relativo programma che dati due numeri interi in input restituisca il massimo dei due

Massimo.c

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

int main ()
{
    int a, b, max;
    printf("Inserire il primo valore ----> ");
    scanf("%d", &a);
    printf("Inserire il secondo valore ----> ");
    scanf("%d", &b);
    if (a>b)
        max=a;
    else
        max=b;
    printf("Il massimo e' %d \n", max);
    system("pause");
    return 0;
}
```

Selezioni innestate

- In un programma le condizioni possono avere più alternative, in questo caso è possibile utilizzare istruzioni di selezione innestate

```
if (condizione1)
{
    Sequenza di istruzioni da eseguire se condizione1 è verificata;
}
else
{
    if (condizione2)
    {
        Sequenza di istruzioni da eseguire se condizione2 è verificata;
    }
    else
    {
        Sequenza di istruzioni da eseguire se condizione2 non è verificata;
    }
}
```

Esempio

- In una serie di numeri calcolare quanti di essi sono positivi, quanti negativi e quanti uguali a zero

```
if (x > 0)
    numeri_positivi = numeri_positivi + 1;
else
    if (x < 0)
        numeri_negativi = numeri_negativi + 1;
    else
        numeri_zero = numeri_zero + 1;
```

Selezione Multipla (if innestati)

- All'aumentare del numero di alternative aumenta anche il livello di indentazione e la possibilità di commettere errori

```

if (condizione1)
{
    Sequenza di istruzioni da eseguire se condizione1 è verificata;
}
else if (condizione2)
{
    Sequenza di istruzioni da eseguire se condizione2 è verificata;
}
...
else if (condizioneN)
{
    Sequenza di istruzioni da eseguire se condizioneN è verificata;
}
else
{
    Sequenza di istruzioni da eseguire se nessuna condizione è verificata;
}

```

...Selezione Multipla (if innestati)

```

if (x > 0)
    numeri_positivi = numeri_positivi + 1;
else if (x < 0)
    numeri_negativi = numeri_negativi + 1;
else
    numeri_zero = numeri_zero + 1;

```

Esercizio

CategoriaVento.c

- Scrivere un algoritmo e poi un programma che data in input la velocità del vento (in Km/h) restituisca la categoria di appartenenza secondo la seguente tabella

Velocità del vento (in Km/h)	Classificazione
sotto i 25	Vento debole
25 – 38	Vento forte
39 – 54	Tempesta
55 – 72	Forte tempesta
Oltre i 72	Uragano

Selezione Multipla (switch)

- L'istruzione switch è utilizzata in C per le selezioni che valutano se singole variabili o semplici espressioni assumano valore all'interno di un certo insieme di costanti intere

```

switch (espressione)
{
    case valore1: Sequenza di istruzioni1;
                  break;
    case valore2: Sequenza di istruzioni2;
                  break;
    ...
    case valoreN: Sequenza di istruzioniN;
                  break;
    default:
        Sequenza di istruzioni di default;
}

```

ATTENZIONE!!!

- Lo SWITCH traduce nel linguaggio di programmazione una serie di SELEZIONI innestate, non esiste un costrutto della programmazione strutturata per realizzare lo SWITCH

Esempio

Velocità del vento (in Km/h)	Classificazione	Classe
sotto i 25	Vento debole	D
25 – 38	Vento forte	V
39 – 54	Tempesta	T
55 – 72	Forte tempesta	F
Oltre i 72	Uragano	U

```
switch (Classe)
{
    case 'D': printf("Vento debole, velocità sotto i 25 KM/h");
              break;
    case 'V': printf("Vento forte, velocità tra i 25 e i 38 KM/h");
              break;
    case 'T': printf("Tempesta, velocità tra i 39 e i 54 KM/h");
              break;
    case 'F': printf("Forte tempesta, velocità tra i 55 e i 72KM/h");
              break;
    case 'U': printf("Uragano, velocità oltre i 72 KM/h");
              break;
}
```

Laboratorio di Programmazione - Veronica Rossano

26

Esercizio

AreaToR.c

- Scrivere un algoritmo e un programma che consenta di calcolare l'area di un triangolo o l'area di un rettangolo semplicemente chiedendo all'utente di inserire la lettera iniziale della figura e le dimensioni

Esercizio

ContaGiorni.c

- Scrivere un algoritmo e un programma che fornita una data in input fornisca come risultato il numero di giorni dall'inizio dell'anno

Il controllo del programma in C

- La maggior parte dei programmi richiede delle iterazioni o cicli
- Il ciclo è un insieme di istruzioni che è eseguito fino al soddisfacimento di una determinata condizione
- I tipi di iterazioni possono essere :
 - Condizionali
 - Pre-condizionali
 - Post-condizionali
 - Limitati

I cicli condizionali

- I cicli condizionali sono utili quando non è possibile determinare a priori il numero di volte in cui il ciclo dovrà essere ripetuto
- Si utilizza un valore che consenta di rendere falsa la condizione in un determinato istante dell'esecuzione
- I valori prendono i nomi di:
 - Sentinella che può assumere un valore qualsiasi
 - Flag (bandiera) che assume solo valori booleani

L'iterazione pre-condizionale in C...

■ While

```
while (condizione)
{
    Sequenza di istruzioni da eseguire fino a
    quando la condizione resta verificata;
}
```

```
conta=0;
while (conta<10)
{
    conta=conta+1;
    printf("%d. Hello World!\n", conta);
}
```

...L'iterazione pre-condizionale in C

WhileSentinella.c

```
#include <stdlib.h>
int main ()
{
    int numero, quadrato;
    printf("***** Il programma calcola il quadrato di una serie di interi positivi *****");
    printf("\n*****   dati in input dall'utente. Digitare -1 per terminare *****");
    printf("\n*****");
    printf("\n\nInserire il valore da elevare al quadrato --> ");
    scanf("%d", &numero);
    while (numero != -1)
    {
        quadrato=numero*numero;
        printf("Il quadrato di %d e' %d", numero, quadrato);
        printf("\n\nInserire il valore da elevare al quadrato --> ");
        scanf("%d", &numero);
    }
    system ("pause");
    return 0;
}
```

La sentinella deve essere scelta tra valori che l'utente non inserirà mai

L'iterazione post-condizionale in C...

■ Do-While

```
do
{
    Sequenza di istruzioni da eseguire fino a
    quando la condizione resta verificata;
} while (condizione);
```

```
/* esegui le operazioni fino al primo numero pari */
do
{
    ...
    printf("Inserire un numero ->");
    scanf ("%d", &num);
} while ( (num % 2) != 0);
```

ATTENZIONE!!!

- Il DO-WHILE non è la traduzione precisa del REPEAT-UNTIL
- Nell'algoritmo è necessario continuare ad usare il costrutto REPEAT UNTIL che poi deve essere tradotto nel linguaggio di programmazione con il DO-WHILE

...L'iterazione post-condizionale in C

DoWhileSentinella.c

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int numero, quadrato;
    printf("*****");
    printf("\n**** Il programma calcola il quadrato di una serie di interi positivi ****");
    printf("\n**** dati in input dall'utente. Digitare -1 per terminare ****");
    printf("\n*****");
    do
    {
        printf("\n\nInserire il valore da elevare al quadrato --> ");
        scanf ("%d", &numero);
        if (numero!=-1)
        {
            quadrato=numero*numero;
            printf("Il quadrato di %d e' %d", numero, quadrato);
        }
    }while (numero!=-1);
    system ("pause");
    return 0;
}
```

Esercizio

- Scrivere un programma che consenta all'utente, dato un importo in input, di scegliere una tra le seguenti operazioni:
 - ❑ Calcolare uno sconto del 10 per cento
 - ❑ Calcolare uno sconto del 20 per cento
 - ❑ Calcolare un aumento del 30 per cento
 - ❑ Calcolare un aumento del 40 per cento

ScontoAumento.c

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

int main ()
{
    /*Dichiarazione delle variabili locali*/
    float importo ;/* I/O, */
    int numero ; /* I/O, */

    printf("*****\n");
    printf("***          Calcola sconto e aumento          **\n");
    printf("*** Dato in input un importo e il numero dell'operazione **\n");
    printf("***          da eseguire il programma restituisce          **\n");
    printf("***          l'importo aggiornato          **\n");
    printf("*****\n");

    do
    {
        printf ("\n\n---> 1 <--- per uno sconto del 10 per cento");
        printf ("\n---> 2 <--- per uno sconto del 20 per cento ");
        printf ("\n---> 3 <--- per un aumento del 30 per cento");
        printf ("\n---> 4 <--- per un aumento del 40 per cento ");
        printf ("\n---> 5 <--- per uscire");
        printf ("\n\nDigitare il numero dell'operazione da eseguire---> ");
        scanf ("%d", &numero);
```

```
switch (numero) {
    case 1:
        printf ("\nInserire l'importo da scontare ---> ");
        scanf ("%f", &importo);
        importo = importo * 0.9 ;
        printf ("\n Importo aggiornato= %.2f  \n\n", importo);
        break ;
    case 2:
        printf ("\nInserire l'importo da scontare ---> ");
        scanf ("%f", &importo);
        importo = importo * 0.8;
        printf ("\n Importo aggiornato= %.2f  \n\n", importo);
        break ;
    case 3 :
        printf ("\nInserire l'importo da aumentare ---> ");
        scanf ("%f", &importo);
        importo = importo * 0.3;
        printf ("\n Importo aggiornato= %.2f  \n\n", importo);
        break ;
    case 4 :
        printf ("\nInserire l'importo da aumentare ---> ");
        scanf ("%f", &importo);
        importo = importo * 0.4;
        printf ("\n Importo aggiornato= %.2f  \n\n", importo);
        break ;
    default :
        printf ("\n");
} /* fine switch */

system ("PAUSE");
} while ( numero != 5 ) ;

return 0;
}
```

Iterazione limitata o controllata da un contatore

- Un contatore (o accumulatore) è una variabile che consente di controllare il numero di volte in cui un ciclo deve essere ripetuto
- È necessario:
 - Inizializzare la variabile fuori dal ciclo (assegnarle un valore iniziale)
 - Definire una condizione che consenta di ripetere le istruzioni del ciclo
 - Incrementare la variabile accertandosi che dopo un numero finito di volte raggiunga il valore indicato nella condizione del ciclo

Esempio

MediaWhile7.c

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

int main ()
{
    int conta_numeri, numero;
    float media;
    /* Inizializzazione del contatore e dell'accumulatore*/
    conta_numeri = 1;
    media=0;
    printf ("\n\n*****\n");
    printf ("**** Il programma calcola la media tra 7 numeri interi dati in input ****\n");
    printf ("*****\n");
    /* Il ciclo while terminerà dopo l'inserimento del 7° numero */
    while (conta_numeri <= 7)
    {
        printf ("Inserisci il %d numero -->", conta_numeri);
        scanf ("%d", &numero);
        conta_numeri=conta_numeri+1;
        media= media+numero;
    }
    media=media/7;
    printf ("\n\nLa media e' --> %f \n\n", media);
    system ("pause");
    return 0;
}
```

Iterazione Limitata in C

HelloFor.c

■ For

```
for (InizializzazioneContatore; Condizione; Incremento)
{
    Sequenza di istruzioni eseguite per un numero di passi
    prestabilito;
}
```

```
for (conta=0; conta<10; ++conta)
{
    printf("%d. Hello World!\n", conta+1);
}
```

L'istruzione for

- L'istruzione for è particolarmente indicata per costruire iterazioni controllate da un contatore
- Consente in un'unica istruzione di:
 - inizializzare il contatore
 - testare la condizione di fine ciclo
 - aggiornare il contatore (variabile di controllo del ciclo)

Esempio

Conviene definire una costante N che rende flessibile il programma

MediaFor7.c

```
#include <stdio.h>
#include <stdlib.h>
#define N 7

int main ()
{
    int conta_numeri, numero;
    float media;
    /*inizializzazione dell'accumulatore */
    media=0;

    printf("\n\n **** Il programma calcola la media di N numeri ****\n\n");
    /* L'istruzione FOR consente di eseguire un gruppo di istruzioni N volte */
    for (conta_numeri = 1; conta_numeri <= N; conta_numeri+=1)
    {
        printf("Inserisci il %d numero -->", conta_numeri);
        scanf("%d", &numero);
        media= (float)media+numero;
    }
    media=media/N;

    printf("\n\nLa media e' --> %.2f \n\n", media);
    system ("pause");
    return 0;
}
```

conta_numeri+=1
consente di incrementare il contatore

%.2f
consente di visualizzare il dato definito double con sole due cifre decimali

Come funziona il for

- Prima è eseguita l'inizializzazione, poi è verificata la condizione di ripetizione
- Se la condizione è vera viene eseguita l'istruzione da ripetere, poi viene eseguita l'istruzione di aggiornamento e verificata nuovamente la condizione di ripetizione
- Quando la condizione di ripetizione assume valore falso allora l'istruzione eseguita è l'istruzione scritta dopo la chiusura del for

Esercizio

PagaOraria.c

- Costruire un algoritmo e il relativo programma che calcola e mostra il salario di un numero predeterminato (richiesto in input) di impiegati
- Si assume che
 - Il salario di un impiegato è dato dalla formula

$$\text{salario} = \text{orelavorate} * \text{pagaoraria}$$

Abbreviazione dell'istruzione di assegnazione...

- Le istruzioni di assegnazione seguono la sintassi

variabile = variabile operatore espressione

- `conta_numeri = conta_numeri + 1`
- `totale = totale + paga`

- C fornisce una notazione molto più concisa che consente di abbreviare le operazioni di assegnamento

...Abbreviazione dell'istruzione di assegnazione

- Le istruzioni di assegnazione abbreviate seguono la sintassi

variabile operatore = espressione

- `conta_numeri + = 1`
- `totale += paga`

- In C questa notazione può essere utilizzata con tutti gli operatori aritmetici `+`, `-`, `*`, `/` e `%`

Incremento e decremento dei contatori...

- Il C fornisce gli operatori unari di incremento e decremento la sintassi è la seguente

Notazione prefissa

`++ variabile`

`-- variabile`

Notazione postfissa

`variabile ++`

`variabile --`

- `conta_numeri ++`
- `++indice`
- `temperatura--`

- Il valore della variabile dipende dalla posizione dell'operatore

...Incremento e decremento dei contatori...

- L'istruzione
 - `alfa = beta++;`
- È equivalente all'esecuzione delle istruzioni:
 - `alfa = beta;`
 - `beta = beta + 1;`
- L'istruzione
 - `alfa = ++beta;`
- È equivalente all'esecuzione delle istruzioni:
 - `beta=beta + 1;`
 - `alfa=beta;`

...Incremento e decremento dei contatori

```
Se n= 4 le istruzioni seguenti
printf("%d", --n);
printf("%d", n);

stampano 3 3

Se n= 4 le istruzioni seguenti
printf("%d", n--); /*prima usa n e poi usa
                  l'operatore*/
printf("%d", n);

stampano 4 3
```

Esercizi

- Scrivere l'algoritmo e il programma che calcoli il MCD tra due numeri dati in input usando l'algoritmo euclideo
- Scrivere l'algoritmo e il programma che per N coppie di numeri inserite in input dall'utente calcoli il massimo di ognuna di esse

Tipi di dati definiti dall'utente

Tipi enumerativi

Dott.ssa Veronica Rossano
rossano@di.uniba.it
<http://www.di.uniba.it/~rossano>

Tipi definiti dall'utente

- In C è possibile definire nuovi tipi di dati che consentono di rappresentare il mondo reale del problema in maniera più accurata

Tipi enumerativi

- Un tipo enumerativo consente di definire un tipo di dato che può assumere valori solo in un insieme finito di valori
- La sintassi per la definizione del tipo di dato è la seguente:

```
typedef enum  
{ valore1, valore2, ..., valoreN }  
nome_tipo;
```

- La sintassi per la definizione della variabile è la seguente:

```
nome_tipo nome_variabile;
```

Esempio

```
typedef enum  
{lunedì, martedì, mercoledì, giovedì, venerdì, sabato, domenica}  
settimana;  
  
settimana giorno; /*dichiarazione di una  
variabile*/
```

- La definizione di un tipo di dato enumerativo consente di utilizzare gli operatori relazionali e aritmetici sulle variabili dichiarate di tale tipo

```
lunedì<domenica  
giorno!=mercoledì  
lunedì<=giorno && giorno<=venerdì
```

Tipi di dati definiti dall'utente

Array

Dott.ssa Veronica Rossano
rossano@di.uniba.it
<http://www.di.uniba.it/~rossano>

Dichiarazione di un vettore

- La dichiarazione di un array richiede la specifica del tipo di dato e del numero di dati che la struttura deve contenere.
- La sintassi per un array monodimensionale è la seguente:

```
tipo_dato nome_vettore[num_elementi];
```

- `double x[8];`
- `char nome[20];`
- `int mesi[12];`

Array

- Un array è una collezione di due o più celle di memoria adiacenti chiamate **elementi**
- Gli array possono essere:
 - Monodimensionali, detti anche **vettori**
 - Multidimensionali, detti anche **tabelle** o **matrici**

Usare un vettore

- Per riferirsi a ciascun elemento memorizzato in un array è necessario semplicemente indicare la posizione dell'elemento nel vettore secondo la sintassi

```
nome_vettore[indice];
```

- `x[3];`
/*si riferisce al quarto elemento del vettore x*/
- `nome[10];`
/*si riferisce all'11 lettera del vettore nome*/
- `mesi[11];`
/*si riferisce dicembre */

Il valore
dell'indice parte
da **0**
Il primo
elemento del
vettore è
vett[0]

Usare gli indici...

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
12.2	23	52.3	1	89.23	4.25	10.1	-563

1. float x[8] ; i = 5 ;
2. x[i - 1] = x[i] ;
3. x[i] = x[i + 1] ;
4. printf ("%d %.1f", i, x[i])
5. printf ("%d %.1f", x[--i]) ;
6. printf ("%d %.1f", x[i++]) ;
7. printf ("%d %.1f", x[2 * i - 3]) ;

Output

```
x[4]=x[5]=4.25
x[5] = x[6]=10.1
Printf 5 e 10.1
Printf x[4]= 4.3
Printf x[4]=4.3
Printf x[7]=-563
```

Attenzione solo le istruzioni n. 5 e 6 provocano una modifica dell'indice i

Acquisizione degli elementi

- Avvalorare gli elementi di un vettore equivale ad assegnare un valore ad una variabile qualsiasi in un programma
- La memorizzazione di un valore può avvenire tramite operazioni di:
 - assegnazione
 - input

Esempio

- Scrivere un algoritmo e il relativo programma che acquisisca gli elementi di un vettore come input digitati da tastiera dall'utente

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10

int main ()
{
    int vettore[MAX];
    int i;
    for (i=0; i<MAX; i++)
    {
        printf("\nInserire il %d valore --> ", (i+1));
        scanf("%d", &vettore[i]);
    }
    system("pause");
    return(0);
}
```

RiempiVettore.c

Costante per la dimensione del vettore

Dichiarazione del vettore

Iterazione limitata per avvalorare gli elementi del vettore

Inizializzare un vettore

- È possibile dichiarare un vettore mediante inizializzazione seguendo la seguente sintassi

```
tipo_dato nome_vettore[] = {dato1, dato2,...,datoN};
```

- `char vocali[] = {'a', 'e', 'i', 'o', 'u'};`
- `int mesi[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};`

Scandire un vettore

- Per visualizzare i singoli elementi di un vettore è necessario utilizzare una iterazione limitata che scandisca il vettore dal primo all'ultimo elemento

VisualizzaVettore.c

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
```

```
int main ()
{
    int i;
    int vettore[] = {0, 1, 1, 2, 3, 5, 8, 13, 21, 34};
    for (i=0; i<MAX; i++)
    {
        printf("Il %d valore e' --> %d\n", (i+1), vettore[i]);
    }
    system("pause");
    return(0);
}
```

Inizializzazione del vettore

Iterazione limitata per visualizzare gli elementi del vettore

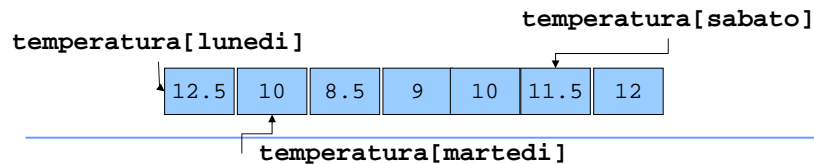
Vettori e tipi enumerativi

- È possibile definire un vettore che abbia come indice un tipo enumerativo
- È necessario assicurarsi che il numero degli elementi del vettore sia uguale alla cardinalità degli elementi del tipo enumerativo

Esempio

- Definire un vettore che raccolga le temperature per ciascun giorno della settimana

```
typedef enum
{
    lunedì, martedì, mercoledì, giovedì, venerdì,
    sabato, domenica
}
settimana;
float temperatura[7];
```



Laboratorio di Programmazione - Veronica Rossano

17

Scandire un vettore enumerativo

- Anche per leggere gli elementi di un vettore enumerativo si usa una iterazione limitata come indicato di seguito:

```
typedef enum
{
    lunedì, martedì, mercoledì, giovedì,
    venerdì, sabato, domenica
}
settimana;
float temperatura[7];
settimana i;
for(i=lunedì; i<=domenica; i++)
{
    scanf("%f", &temperatura[i]);
}
```

Laboratorio di Programmazione - Veronica Rossano

18

Esempio

- Scrivere un algoritmo e il relativo programma che memorizzi in un vettore le temperature di una settimana e ne restituisca la media

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

typedef enum
{
    lunedì, martedì, mercoledì, giovedì, venerdì, sabato, domenica
}
settimana;

int main ()
{
    float temperatura[7];
    float media=0;
    settimana i;

    printf("\n\n***** Il programma calcola la media della temperatura di una settimana *****\n\n");
    for(i=lunedì; i<=domenica; ++i)
    {
        printf("Inserisci la temperatura di ");
        switch (i)
        {
            case lunedì: printf("lunedì -->");
                        break;
            case martedì: printf("martedì -->");
                        break;
            case mercoledì: printf("mercoledì -->");
                        break;
            case giovedì: printf("giovedì -->");
                        break;
            case venerdì: printf("venerdì -->");
                        break;
            case sabato: printf("sabato -->");
                        break;
            case domenica: printf("domenica -->");
                        break;
        }
        scanf("%f", &temperatura[i]);
        media+=temperatura[i];
    }
    media/=7;
    printf("La temperatura media della settimana e' stata : %.2f\n", media);
    system("pause");
}
```

Dichiarazione Tipo di dato
enumerativoDichiarazione indice del
vettore

Scansione del vettore

Laboratorio di Programmazione - Veronica Rossano

19

20

Esercizio

- Scrivere un algoritmo e il relativo programma che generi e memorizzi i primi $n \geq 1$ (definito in input dall'utente) elementi della successione di Fibonacci in un vettore e che lo visualizzi all'utente
 - I primi due termini sono 0 e 1
 - Ogni termine successivo è ottenuto come
 - somma degli ultimi 2 termini
 - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Array multidimensionali

Dott.ssa Veronica Rossano
rossano@di.uniba.it
<http://www.di.uniba.it/~rossano>

Array multidimensionali...

- Gli array possono essere multidimensionali cioè avere due o più dimensioni
- Si utilizzano per rappresentare tabelle e/o matrici e qualsiasi altro oggetto a più dimensioni
- La sintassi è la seguente:

```
Tipo_valori nome_array [dim1] [dim2]...[dimN];
```

- `x[2][3]` /* si riferisce all'elemento in terza riga quarta colonna */
- `retribuzione [gennaio][2006]` /* si riferisce alla retribuzione di gennaio del 2006

Laboratorio di Programmazione - Veronica Rossano

2

...Array multidimensionali

ArrayMultidim-ValoriAssegnati.c

- L'inizializzazione di un array multidimensionale segue la stessa sintassi degli array monodimensionali

```
tipo_dato nome_vettore [N] [M]=  
{ {dato1, dato2,...,datoM},... {val1, val2,...,valM} };  
/*N volte*/  
  
double temp [7] [2]= { {10, 16.5}, {9, 12}, {-1, 5}, {5,  
9.5}, {4.5, 8}, {-2, 3}, {7, 12}};
```

Esempio

- Scrivere un programma che richieda in input temperature minime e massime per ogni giorno della settimana e ne visualizzi la media delle temperature minime e la media delle temperature massime

Rappresentazione grafica

	Min	Max
Lunedì	10	16.5
Martedì	9	12
Mercoledì	-1	5
Giovedì	5	9.5
Venerdì	4.5	8
Sabato	-2	3
Domenica	7	12

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

```
typedef enum
{min, max}
```

```
min_max;

typedef enum
{lunedì, martedì, mercoledì, giovedì, venerdì, sabato, domenica}
settimana;
```

```
int
main (void)
{
    double temperatura[7][2];
    double media_min=0, media_max=0;
    settimana i;
    for(i=lunedì; i<=domenica; ++i)
    {
        printf("Inserire le temperature minima e massima del %d giorno", i+1);
        printf(" separate da uno spazio -->");
        scanf("%lf %lf", &temperatura[i][min], &temperatura[i][max]);
        media_min+=temperatura[i][min];
        media_max+=temperatura[i][max];
    }
    media_min/=7;
    media_max/=7;
    printf("La temperatura minima media della settimana e' stata : %.2f\n", media_min);
    printf("La temperatura massima media della settimana e' stata : %.2f\n", media_max);
    system("pause");
    return(0);
}
```

Tipi definiti dall'utente

ArrayMultidim-Temperatura.c

Dichiarazione dell'array multidimensionale

Esercizio

TavolaPitagorica.c

- Scrivere un algoritmo e un programma che consenta di memorizzare e visualizzare la tavola pitagorica dei primi n numeri naturali. Consentire all'utente di decidere il valore di n (che comunque non dovrà essere superiore a 20). La visualizzazione dovrà essere nel formato:

```
1x1=1
1x2=2
1x3=3
1x4=4
1x5=5
...
```

Le stringhe

Dott.ssa Veronica Rossano
rossano@di.uniba.it
<http://www.di.uniba.it/~rossano>

Il tipo stringa

- In C una stringa (sequenza di caratteri) è implementata come un array
- La sintassi per la dichiarazione di una stringa è la seguente:

```
char nome_stringa[numero_caratteri];
```

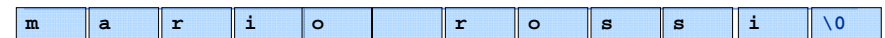
- `char nome[20];`

Inizializzazione

- La stringa in quanto array può essere inizializzata nei modi già discussi

```
char nome[20] = {'m','a','r','i','o',' ','r','o','s','s','i','\0'};
char nome[20] = "mario rossi"
```

- La stringa è rappresentata in memoria come un insieme di celle contigue. Nell'ultima cella è memorizzato il carattere `\0` di fine stringa



Array stringa multidimensionali

- È possibile dichiarare un array multidimensionale di caratteri
- Ogni elemento dell'array sarà una stringa di caratteri

```
char mesi [12] [10] = {"Gennaio", "Febbraio", "Marzo", "Aprile",
"Maggio", "Giugno", "Luglio", "Agosto", "Settembre", "Ottobre",
"Novembre", "Dicembre"};
```

NB. In una assegnazione non è possibile andare a capo
Nella slide è riportato con questa formattazione solo per motivi di presentazione

Stringa: Input e Output...

- Entrambe le funzioni `printf` e `scanf` possono utilizzare come argomenti una stringa semplicemente utilizzando il placeholder `%s`
- Nella `printf` è possibile specificare quanti caratteri minimo devono essere visualizzati
 - Scrivere `%8s` significa che **almeno** otto caratteri della stringa saranno visualizzati

...Stringa: Input e Output

- Nella Scanf non è necessario inserire l'operatore di indirizzo & in quanto il passaggio di un array avviene sempre per indirizzo
 - La scanf riceve una stringa in input con lo stesso processo adottato per gli input numerici
 - Ogni carattere digitato è memorizzato in celle contigue di memoria quando è digitato uno spazio, un tab o invio la scanf termina la memorizzazione

Formattazione dell'output

Stringhe.c

```
char nome[20];
printf("Inserire un nome (max 20 caratteri)");
scanf("%s", nome);
printf("*****s*****\n", nome);
printf("*****4s*****\n", nome);
printf("*****20s*****\n", nome);
printf("*****-20s*****\n", nome);
```

G:\Lezioni\Programmazione\Prove\Array\stringhe.exe

```
Mario
*****Mario*****
*****Mario*****
*****          Mario*****
*****Mario          *****
Press any key to continue . . .
```

Errore tipico

- La stringa **non è una variabile semplice** quindi non è possibile fare un assegnamento diretto del tipo

```
char nome[20];
nome = "Mario Rossi";
```

errore in compilazione

La libreria <string.h>...

- La libreria string.h fornisce funzioni per rendere la gestione delle stringhe più agevole
 - strcpy(str_dest, str_sorg)
 - Consente di copiare la stringa sorgente nella stringa destinazione aggiungendo il carattere \0
 - strncpy(str_dest, str_sorg, n)
 - Consente di copiare i primi n caratteri della stringa sorgente nella stringa destinazione non aggiunge il carattere \0

...La libreria <string.h>...

- ❑ Strcat(str_dest, str_sorg)
 - Concatena la stringa sorgente alla stringa destinazione
- ❑ Strncat(str_dest, str_sorg, n)
 - Concatena i primi n caratteri della stringa sorgente alla stringa destinazione
- ❑ Strcmp(str1, str2)
 - Confronta due stringhe. Ritorna un **valore negativo** se str1 < str2, **0** se le stringhe sono uguali e un **valore positivo** se str1 > str2

...La libreria <string.h>...

- ❑ Strcmp(str1, str2, n)
 - Confronta i **primi n caratteri** delle due stringhe. Ritorna un **valore negativo** se str1 < str2, **0** se le stringhe sono uguali e un **valore positivo** se str1 > str2
- ❑ Strlen(str)
 - Ritorna il numero di caratteri della stringa non includendo il carattere di fine stringa

...La libreria <string.h>...

- ❑ Strtok(str, car)
 - Cerca nella stringa le sottostringhe delimitate dal carattere **car**, restituendone il puntatore al primo carattere della sottostringa. Le chiamate successive alla prima devono sostituire a str il valore NULL. La funzione restituisce NULL quando non trova più alcuna occorrenza del carattere car (token)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define LUNG 20
```

GestioneStringhe.c

```
int
main()
{
```

Dichiarazione e
inizializzazione delle
stringhe

```
char stringa1[LUNG] = "Mario Rossi";
char stringa2[LUNG] = "Studente Matr.33965";
char *punt;
printf("Stringa 1 -->%s\n", stringa1);
printf("Stringa 2 -->%s\n", stringa2);
```

Copia delle stringhe

```
printf("\n***** strcpy (stringa1, stringa2)*****\n");
strcpy(stringa1, stringa2);
printf("Stringa 1 -->%s\n", stringa1);
printf("Stringa 2 -->%s\n", stringa2);
```

Confronto tra stringhe

```
printf("\n***** strcmp (stringa1, stringa2)*****\n");
if (strcmp(stringa1, stringa2) == 0)
    printf("Le due stringhe sono uguali\n");
```

```

printf("\n***** strcpy (stringa1, stringa2, 5)*****\n");
strcpy(stringa1, "Mario Rossi"); /* prima di provare strcpy riportia
strcpy(stringa1, stringa2, 5);
printf("Stringa 1 -->%s\n", stringa1); /* da notare che strcpy non a
                                continua ad avere sempre lo

printf("Stringa 2 -->%s\n", stringa2);

if (strcmp(stringa1,stringa2)==0)
    printf("Le due stringhe sono uguali\n");
else if (strcmp(stringa1,stringa2)<0)
    printf("Stringa1 precede Stringa2\n");

printf("\n***** strcmp (stringa2, stringa1)*****\n");
if (strcmp(stringa2,stringa1)>0)
    printf("Stringa2 segue Stringa1\n");

printf("\n***** strncmp (stringa2, stringa1, 5)*****\n");
if (strncmp(stringa2,stringa1,5)==0)
    printf("I primi 5 caratteri delle stringhe sono uguali\n\n");

printf("\n***** strlen(stringa2)*****\n");
printf("La stringa 2 ha %d caratteri \n\n", strlen(stringa2));

```

Confronto dei primi 5
caratteri

Lunghezza della stringa2

```

printf("\n***** strtok(stringa2, \",\") *****\n");
strcpy(stringa2, "Maria,Elena,Giovanni,Antonio,Ugo");
punt=strtok(stringa2,",");
while (punt!=NULL)
{
    printf("Il token estratto e' %s\n\n", punt);
    punt=strtok(NULL,",");
}

```

Sottostringhe della
stringa 2 utilizzando il
carattere ,

```

C:\eclipseC\workspace\Programmi Lezione 5\FunzioniStringhe\Debug\FunzioniStringhe.exe
Stringa 1 -->Mario Rossi
Stringa 2 -->Studente Matr.33965

***** strcpy (stringa1, stringa2)*****
Stringa 1 -->Studente Matr.33965
Stringa 2 -->Studente Matr.33965

***** strcmp (stringa1, stringa2)*****
Le due stringhe sono uguali

***** strcpy (stringa1, stringa2, 5)*****
Stringa 1 -->Stude Rossi
Stringa 2 -->Studente Matr.33965
Stringa 1 -->Stude
Stringa1 precede Stringa2

***** strcmp (stringa2, stringa1)*****

***** strncmp (stringa2, stringa1, 5)*****
I primi 5 caratteri delle stringhe sono uguali

***** strlen(stringa2)*****
La stringa 2 ha 19 caratteri

***** strtok(stringa2, ",")*****
Il token estratto e' Maria
Il token estratto e' Elena
Il token estratto e' Giovanni
Il token estratto e' Antonio
Il token estratto e' Ugo

Premere un tasto per continuare . . . _

```

Overflow di stringhe

- Quando si assegna ad una stringa un valore che contenga più caratteri della lunghezza dichiarata della stringa si può incorrere in un errore logico
 - Il compilatore segnala solo un warning

La libreria <ctype.h>...

- Nella gestione delle stringhe alcune volte può essere utile conoscere la natura di ciascun tipo di carattere
 - isalpha(ch)
 - Restituisce vero se ch è un carattere alfabetico
 - isdigit(ch)
 - Restituisce vero se ch è una cifra decimale
 - islower(ch) – isupper(ch)
 - Restituiscono vero se ch è rispettivamente un carattere minuscolo o maiuscolo

... La libreria <ctype.h>

- ispunct(ch)
 - Restituisce vero se ch è un carattere di punteggiatura
- isspace(ch)
 - Restituisce vero se ch è uno spazio
- Tolower(ch) – toupper(ch)
 - Convertono ch da un carattere maiuscolo in minuscolo e viceversa

GestioneCaratteri.c

```
main()
{
    char stringa[LUNG] = "Studente Matr.33965";
    char ch;
    if (isalpha(stringa[0]))
        printf("\nIl carattere --> %c e' un carattere alfabetico", stringa[0]);
    if (isdigit(stringa[14]))
        printf("\nIl carattere --> %c e' un numero", stringa[14]);
    if (islower(stringa[10]))
        printf("\nIl carattere --> %c e' un carattere minuscolo", stringa[10]);
    ch=toupper(stringa[10]);
    printf("\nIl carattere e' trasformato in maiuscolo --> %c", ch);
    if (isupper(stringa[9]))
        printf("\nIl carattere --> %c e' un carattere MAIUSCOLO", stringa[9]);
    ch=tolower(stringa[9]);
    printf("\nIl carattere e' trasformato in minuscolo --> %c", ch);
    if (ispunct(stringa[13]))
        printf("\nIl carattere --> %c e' un carattere di punteggiatura", stringa[13]);
    if (isspace(stringa[8]))
        printf("\nIl carattere --> %c e' uno spazio\n", stringa[8]);

    system("pause");
}
```

```
C:\Documents and Settings\ssis\Desktop\Prove\Array e Stringhe\Gestione caratteri.exe
Il carattere --> S e' un carattere alfabetico
Il carattere --> 3 e' un numero
Il carattere --> a e' un carattere minuscolo
Il carattere e' trasformato in maiuscolo --> A
Il carattere --> M e' un carattere MAIUSCOLO
Il carattere e' trasformato in minuscolo --> m
Il carattere --> . e' un carattere di punteggiatura
Il carattere -->   e' uno spazio
Premere un tasto per continuare . . .
```

Esercizio

AnalisiStringa.c

- Scrivere l'algoritmo e il programma data una stringa in input ne analizzi i singoli caratteri restituendo il numero di caratteri alfabetici e, di questi, specifichi quanti sono maiuscoli e quanti sono minuscoli, il numero di cifre e il numero di segni di punteggiatura

Funzioni e Procedure

Dott.ssa Veronica Rossano
rossano@di.uniba.it
<http://www.di.uniba.it/~rossano>

Sottoprogramma

- Insieme di istruzioni
 - Individuate da un nome
 - Che concorrono a risolvere un problema
 - Ben definito
 - Sensato
 - Non necessariamente fine a sé stesso
 - Supporto per la risoluzione di problemi più complessi
- Esempi:
 - Scambio, Ricerca del Minimo, Ordinamento,...

Sviluppo top-down

- Costruzione del programma per livelli successivi
 - Corrispondenza con la scomposizione del problema cui è relativo
 - Strumento concettuale per la costruzione di algoritmi
 - Raffinamento successivo delle parti in cui viene scomposto (sottoprogrammi) fino al codice finale
 - Strumento operativo per l'organizzazione e lo sviluppo di programmi complessi

Procedure vs Funzioni

- Procedure
 - Sottoprogrammi il cui compito è quello di produrre un effetto
 - Modifica del valore di variabili
 - Comunicazione di informazioni all'utente
- Funzioni
 - Sottoprogrammi che hanno come risultato il calcolo di un valore
 - All'interno della dichiarazione della funzione vi è un'istruzione che ritorna il risultato

Sottoprogramma in C

- Il C consente la definizione di sottoprogrammi (sia procedure che funzioni) utilizzando sempre lo stesso costrutto
- Durante la costruzione di un programma sarà possibile definire una funzione o una procedura per ogni sottoprogramma definito durante la fase di progettazione

Dichiarare una funzione o una procedura

- La dichiarazione di una funzione/procedura in C:
 - prende il nome di **prototipo**
 - è obbligatoria
 - deve essere inserita subito dopo le direttive del precompilatore (`#include` e `#define`) e prima del `main`
- La sintassi è la seguente:

```
tipo_risultato nome_sottoprogramma (elenco_argomenti)
■ void calcola_max (int a, int b)
■ int fattoriale (int n)
```

Procedure vs Funzioni in C

- Poiché in C esiste un unico costrutto per costruire sia funzioni che procedure si usa la parola chiave **VOID** per segnalare che un sottoprogramma non restituisce nulla ad un programma chiamante

- ```
■ int fattoriale (int n)
 □ È una funzione che restituisce il fattoriale di un numero n
■ void draw_cerchio (int n)
 □ È una procedura che disegna n volte un cerchio
```

## Schema di un programma C

```
/*Direttive per il preprocessore */
#include <stdio.h>

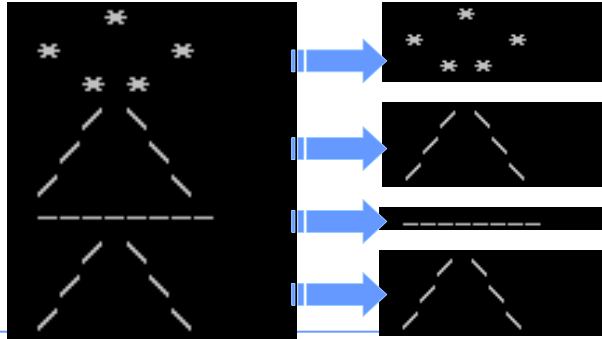
/*Dichiarazione dei prototipi */
tipo_risultato nome_sottoprogramma (elenco_argomenti);

int main()
{
 nome_sottoprogramma (elenco_argomenti);
 /* Chiamata del sottoprogramma*/
}

tipo_risultato nome_sottoprogramma (elenco_argomenti)
{
 ...
}
```

## Esempio...

- Supponendo di voler scrivere un programma che disegni la figura seguente sullo schermo



## ...Esempio

- Scomponiamo il problema in diverse funzioni che siano in grado di:
  - Disegnare un cerchio con degli asterischi
  - Disegnare una intersezione
  - Disegnare una linea di base
- I prototipi delle procedure saranno
  - void cerchio (void)
  - void intersezione (void)
  - void base (void)

**N.B.** La parola chiave **void** si utilizza quando non si prevede nessun dato (sia per l'argomento che per il risultato)

## Definizione di una funzione/procedura

- La definizione di una funzione/procedura ricalca la definizione del main
- La sintassi:

```
tiporisultato
nomsottoprogramma (elenco argomentazioni)
{
 /* dichiarazioni locali */
 /* istruzioni del sottoprogramma */
}
```

NB. Alcune volte il tipo di dato restituito è posto sulla riga che precede il nome del sottoprogramma. La sintassi è comunque corretta

## Esempio

- La procedura che disegna un cerchio di asterischi sarà definita come segue:

```
void
cerchio (void)
{
 printf (" * \n ");
 printf (" * * \n ");
 printf (" * * \n ");
}
```

## Le dichiarazioni locali

- Gli eventuali identificatori dichiarati all'interno del sottoprogramma sono nomi locali e possono essere usati (sono visibili) solo ed esclusivamente all'interno di esso
- Le istruzioni eseguibili descrivono le operazioni che effettua il sottoprogramma
- L'ordine delle definizioni non influisce sull'ordine delle esecuzioni che è determinato dall'ordine delle istruzioni di chiamata dei sottoprogrammi nel programma chiamante

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

### Prototipi delle procedure

Procedure.c

```
void cerchio (void);
void intersezione (void);
void base (void);
void triangolo (void);
```

```
int
main (void)
{
 cerchio();
 triangolo();
 intersezione ();
 system ("pause");
 return (0);
}
```

### Chiamate delle procedure

### Dichiarazione delle procedure

```
void
cerchio (void)
{
 printf(" * \n");
 printf(" * * \n");
 printf(" * \n");
}
```

```
void
intersezione (void)
{
 printf(" / \ \ \n");
 printf(" / \ \ \n");
 printf(" / \ \ \n");
}
```

```
void
base (void)
{
 printf(" ----- \n");
}
```

```
void triangolo (void)
{
 intersezione();
 base();
}
```

## Funzioni e Procedure con parametri (argomenti)

- I parametri (argomenti) sono utilizzati per consentire l'interazione tra i vari sottoprogrammi e il main
  - In altre parole per portare informazioni dal main ai sottoprogrammi e viceversa
- I parametri consentono di costruire sottoprogrammi più versatili che possono essere utilizzati in contesti completamente differenti
  - Argomenti di input usati per passare informazioni ad un sottoprogramma
  - Argomenti di output usati per passare informazioni al programma chiamante

## Parametri di input

- I parametri devono essere dichiarati dopo il nome del sottoprogramma e devono essere racchiusi tra parentesi
- Per ogni parametro è necessario dichiarare il tipo

```
void
calcola_area (double base, double altezza)
{
 double area;
 area=base*altezza;
 printf("l'area e' pari a %.2f metri", area);
}
```

## Chiamata di una procedura

- Per chiamare una procedura è sufficiente inserire nel main (o in una qualunque altro sottoprogramma) il nome della procedura seguito dai parametri reali (o attuali) da sostituire a quelli formali
- La corrispondenza è determinata dalla posizione

```
void calcola_area (double base, double altezza);

main(void)
{
 double b, h;
 ...
 calcola_area (b, h);
 ...
}
```

Laboratorio di Programmazione - Veronica Rossano

17

## Chiamata di una funzione

- La chiamata di una funzione avviene sempre a destra di una istruzione di assegnazione

```
double calcola_area (double base, double altezza);

int main(void)
{
 double b, h, area_rett;
 ...
 area_rett=calcola_area (b, h);
 printf("L'area del rettangolo è: %.2f", area_rett);
 ...
 return (0);
}
```

Laboratorio di Programmazione - Veronica Rossano

18

## Restituzione del risultato in una funzione

- Nella dichiarazione di una funzione è necessario indicare il tipo di risultato che la funzione deve restituire
- L'istruzione Return consente di restituire il valore computato dalla funzione al programma chiamante

```
double
calcola_area (double base, double altezza)
{
 double area;
 area=base*altezza;
 return (area);
}
```

Laboratorio di Programmazione - Veronica Rossano

19

## Regola fondamentale

- Quando si usano funzioni/procedure a più argomenti è necessario assicurarsi che siano verificate le corrispondenze tra parametri formali e parametri attuali relativamente a:
  - Numero di argomenti
  - Ordine degli argomenti
  - Tipo di ciascun argomento

Laboratorio di Programmazione - Veronica Rossano

20

## Esercizio

- Riscrivere il programma per calcolare il massimo tra tre numeri usando una funzione che calcola il massimo tra due numeri.



## Massimo tra tre numeri senza funzioni

```

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

int main ()
{
 int a, b, c, max;
 printf("Inserire il primo valore ----> ");
 scanf("%d", &a);
 printf("Inserire il secondo valore ----> ");
 scanf("%d", &b);
 printf("Inserire il terzo valore ----> ");
 scanf("%d", &c);
 if (a>b)
 max=a;
 else
 max=b;
 if (c>max)
 max=c;
 printf("Il massimo e' %d \n", max);
 system("pause");
 return 0;
}

```

## Massimo tra tre numeri con funzioni

Max3Funzioni.c

```

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

int maxcoppia(int primo, int secondo);

int main ()
{
 int a, b, c, massimo;
 printf("Inserire il primo valore ----> ");
 scanf("%d", &a);
 printf("Inserire il secondo valore ----> ");
 scanf("%d", &b);
 printf("Inserire il terzo valore ----> ");
 scanf("%d", &c);
 massimo=maxcoppia(a,b);
 massimo=maxcoppia(massimo,c);
 printf("Il massimo e' %d \n", massimo);
 system("pause");
 return 0;
}

int maxcoppia(int primo, int secondo)
{
 int max;
 if (primo>secondo)
 max=primo;
 else
 max=secondo;
 return (max);
}

```

## Esempio

- Scrivere un algoritmo e il relativo programma che calcoli la temperatura ad una certa profondità in gradi Celsius e Fahrenheit chiedendo in input la profondità (in km) e utilizzando le formule seguenti:
  - Celsius = 10 \* profondita + 20
  - Fahrenheit = 1.8 \* Celsius + 32
- Utilizzare due funzioni



## Generazione di numeri casuali

- Per poter generare una serie di numeri casuali si usa la funzione `rand()`
  - È inclusa nella libreria standard `<stdlib.h>`
  - Restituisce un intero compreso tra 0 e `RAND_MAX` che è una costante definita nella libreria standard
  - Solitamente il valore di `RAND_MAX` è 32767 il massimo valore rappresentabile per un intero di due byte

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

CelsiusFahrenheit.c

```
double celsius(double km);
double fahrenheit(double temp_cels);

int
main (void)
{
 double profondita, temperatura_c, temperatura_f;
 printf("Inserisci la profondita' espressa in km --> ");
 scanf("%lf", &profondita);
 temperatura_c=celsius(profondita);
 printf("\n\nLa temperatura alla profondita di %.2f km in gradi Celsius e' %.3f",
 profondita, temperatura_c);

 temperatura_f=fahrenheit(temperatura_c);
 printf(" mentre in gradi Fahrenheit e' %.3f\n\n", temperatura_f);
 system("pause");
 return(0);
}

double celsius(double km)
{
 return(10*km+20);
}
double fahrenheit(double temp_cels)
{
 return(1.8*temp_cels+32);
}
```

I valori restituiti dalla funzione possono anche essere calcolati direttamente nella return senza usare variabili locali

## Esercizio

Monetine.c

- Realizzare l'algoritmo e il programma per simulare il lancio di una moneta. Consentire all'utente di scegliere quante volte lanciare la moneta e per ogni lancio il programma dovrà visualizzare Testa o Croce e contare il numero di occorrenze per la comparsa di ogni faccia della moneta. Usare una funzione distinta che non riceverà argomenti e che restituirà il risultato del lancio della moneta (croce o testa)

## Esercizio

TavolaPitagorica.c

- Scrivere un programma che aiuti uno studente di scuola elementare ad allenarsi nel ricordare la tavola pitagorica. Utilizzare la funzione `rand()` per produrre due interi positivi di una cifra. Il programma dovrà chiedere "Quanto fa 4 per 5?". In seguito lo studente dovrà digitare la risposta. Nel caso lo studente risponda correttamente visualizzare un feedback positivo (es. BRAVO!!!) in caso contrario un feedback di incoraggiamento con la risposta corretta.
- Consentire all'utente di riprovare più volte fino a quando non vorrà uscire dal programma

## La randomizzazione...

- Si noterà che la funzione rand produce sempre la stessa serie di numeri
- Per rendere realmente casuale la produzione dei numeri sarà necessario “inizializzare” la funzione rand() utilizzando la funzione srand()
  - Prende in input un valore unsigned (intero senza segno)

## ...La randomizzazione

- Per usare la funzione srand() sarà necessario aggiungere le seguenti righe di codice all'inizio del programma

```
unsigned seme;
scanf("%u", &seme);
srand(seme);
```

- È possibile inizializzare la funzione rand usando anche la funzione time() che restituisce l'ora corrente del pc in questo caso le righe da aggiungere saranno

```
#include <time.h>
...
srand(time(NULL));
```

Il parametro di input NULL consente alla funzione time() di restituire un intero senza segno invece della stringa con l'indicazione dell'ora

## Parametri di un sottoprogramma

- L'elenco degli argomenti fornisce i legami di comunicazione tra il programma chiamante e il sottoprogramma chiamato
- I parametri permettono di mandare in esecuzione il sottoprogramma, ogni volta che viene chiamato, con valori differenti.

## Parametri di output

- L'istruzione return consente alla funzione di restituire un risultato al programma chiamante
- È possibile definire parametri di output anche per le procedure che consentano di restituire anche più di un risultato al programma chiamante
- È necessario che il programma crei delle celle di memoria in cui conservare i parametri formali per renderli disponibili successivamente al programma chiamante

## Passaggio di parametri per indirizzo...

- Quando una procedura deve restituire un risultato è necessario che si usi il **passaggio di parametri per referenza (o indirizzo)**
- L'elenco dei parametri formali deve essere presente sia nel prototipo che nella definizione della procedura
- Anche nelle funzioni è possibile usare il passaggio di parametri per indirizzo
- La sintassi per la dichiarazione di un parametro passato per indirizzo è:

```
Tipo risultato
Nome_funzione(tipo_par_in nome_par_in, tipo_par_out *nome_par_out)
```

## ...Passaggio di parametri per indirizzo

- Nell'istruzione di dichiarazione l'operatore **\*** dichiara che il parametro formale indicato è un **puntatore** ad una variabile del tipo indicato prima del suo nome
- La dichiarazione di un puntatore comunica al compilatore che la variabile dichiarata conterrà **l'indirizzo** della variabile del tipo indicato

## I puntatori in C...

- La dichiarazione del tipo:

```
double *temperatura;
```

- Indica che la variabile nome sarà un **puntatore ad una variabile di tipo double**, in altre parole conterrà **l'indirizzo della cella di memoria in cui è conservata una variabile di tipo double**

temperatura



## ...I puntatori in C...

temperatura



- Nelle computazioni se ci si vuole riferire:
  - all'indirizzo della cella di memoria che contiene la temperatura dovrà essere utilizzato l'operatore di indirizzo **&**
  - al valore della temperatura dovrà essere utilizzato l'operatore **\***

```
indirizzo=&temperatura;
```

```
temp_corporea=*temperatura;
```

## Puntatori e sottoprogrammi in C

- Nell'istruzione di chiamata di un sottoprogramma i parametri attuali corrispondenti ai parametri formali saranno preceduti dall'operatore di indirizzo &

## Esempio

- Scrivere un programma che prenda tre numeri in input e li restituisca in ordine

Università degli Studi di Bari – Dipartimento di Informatica

```

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

void ordina(double *num1, double *num2)

int
main (void)
{
 double primo, secondo, terzo;
 printf("Inserire tre numeri separati da spazi --->");
 scanf("%lf%lf%lf", &primo, &secondo, &terzo);
 ordina(&primo, &secondo);
 ordina(&primo, &terzo);
 ordina(&secondo, &terzo);
 printf("I numeri in ordine sono: %lf %lf %lf\n", primo, secondo, terzo);
 system("pause");
 return (0);
}

void
ordina(double *num1, double *num2)
{
 double scambio;
 if (*num1 > *num2)
 {
 scambio = *num1;
 *num1 = *num2;
 *num2 = scambio;
 }
}

```

**Prototipo della procedura**

**Parametri formali**  
dichiarazione dei puntatori

**Scanf a tre parametri**

**Chiamata della procedura**  
passaggio dei parametri attuali

**Corpo della procedura**  
uso delle variabili puntatori

## La chiamata di un sottoprogramma con passaggio per indirizzo

- L'istruzione di chiamata al sottoprogramma `ordina(&primo, &secondo)` indica come i parametri attuali siano istanziati con gli indirizzi delle due celle di memoria che conterranno i parametri attuali

## Significato dell'operatore \*...

- Operatore aritmetico
  - Nelle istruzioni di un programma è utilizzato per calcolare la moltiplicazione tra numeri
- Definizione di puntatori
  - Nelle dichiarazioni è utilizzato per la definizione di una variabile puntatore
- Recupera il dato puntato da
  - Nelle istruzioni di un programma è utilizzato per recuperare il dato puntato da una variabile puntatore

## ...Significato dell'operatore \*

```
int quantita, prezzo, totale;
totale=quantita*prezzo;
```

```
int quantita, prezzo;
int *totale;
...
calcola_prezzo(quantita, prezzo, &totale)
```

```
int *totale;
...
printf("La spesa totale e' %d", *totale);
```

## L'istruzione scanf...

- L'istruzione rappresenta una chiamata ad una funzione che utilizza come parametri formali dei puntatori a delle celle di memoria che restituiscono l'indirizzo della cella in cui l'input è memorizzato
- È possibile introdurre più input in un'unica istruzione

```
scanf("%lf%lf%lf", &primo,&secondo,&terzo);
```

## ...L'istruzione scanf

- La funzione scanf restituisce un numero che consente di capire se l'istruzione è andata a buon fine. Il numero restituito rappresenta il numero di parametri inseriti dall'utente in input.

```
status=scanf("%lf%lf%lf", &primo,&secondo,&terzo);
restituisce 3 in status
```

```

void ordina(double *num1, double *num2);

int
main (void)
{
 double primo, secondo, terzo;
 int val_input;
 char continua;
 do
 {
 printf("Inserire tre numeri separati da spazi (es. 23.7 1.32 67.5)--->");
 val_input=scanf("%lf%lf%lf", &primo,&secondo,&terzo);
 if (val_input!=3)
 {
 printf("Si e' verificato un errore nell'input. Riprocedere\n\n");
 scanf("%c", &continua);
 }
 } while (val_input!=3);
 ordina(&primo,&secondo);
 ordina(&primo, &terzo);
 ordina(&secondo, &terzo);
 printf("I numeri in ordine sono: %lf %lf %lf\n\n", primo, secondo, terzo);
 system("pause");
 return (0);
}

```



# Array e sottoprogrammi

Dott.ssa Veronica Rossano  
[rossano@di.uniba.it](mailto:rossano@di.uniba.it)  
<http://www.di.uniba.it/~rossano>

# Array e sottoprogrammi

- Piuttosto che passare singoli elementi dei vettori ad un sottoprogramma è possibile passare interi array come argomenti
- La dichiarazione di un sottoprogramma nella forma seguente

```
tipo_risultato nome_sottoprogramma (tipo_elem nome_array[]);
■ void rileva_temperatura (double temp[]);
```

indica che il parametro formale conterrà solo l'indirizzo del primo elemento dell'array e che tutte le operazioni avranno come risultato la **manipolazione dell'array originale e non di una sua copia**

## ...Array e sottoprogrammi

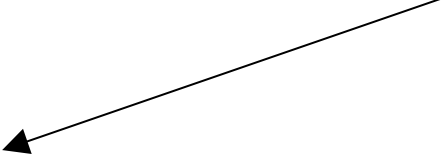
- È da notare che:
  - ❑ La dichiarazione del parametro formale non indica il numero di elementi contenuti nel vettore
  - ❑ Il passaggio di parametri potrebbe avvenire allo stesso modo semplicemente utilizzando un puntatore al primo elemento dell' array

## Esempio

- Definire un sottoprogramma che inizializzi tutti gli elementi di un qualsiasi vettore ad un valore specificato dall'utente


```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100
```

Dichiarazione di un  
vettore come parametro  
formale



```
void inizializza(int vettore[], int dim, int valore);
void visualizza(int vettore[], int dim);
```

```
int
main (void)
{
 int n, val_inizializzazione;
 int vett[MAX];
 printf("Inserire il numero degli elementi del vettore (Massimo 100) -->");
 scanf("%d", &n);
 printf("Inserire il valore di inizializzazione degli elementi del vettore -->");
 scanf("%d", &val_inizializzazione);
 inizializza(vett, n, val_inizializzazione);
 visualizza(vett, n);
 system("pause");
 return(0);
}
```



Passaggio di un vettore  
come parametro reale

```
void inizializza(int vettore[], int dim, int valore)
{
 int i;
 for (i=0; i<dim; i++)
 {
 vettore[i]=valore;
 }
}
```

Inizializzazione di un  
vettore

```
void visualizza(int vettore[], int dim)
{
 int i;
 for (i=0; i<dim; i++)
 {
 printf("Il %d valore e' --> %d\n", (i+1), vettore[i]);
 }
}
```

Visualizzazione degli  
elementi di un vettore



## Esercizio

- Scrivere un programma che restituisca il più grande elemento in un vettore qualsiasi dato in input

# Massimo

## Algoritmo

- Inserire il numero di valori
- Leggere il primo
- Porre il massimo al primo
- Mentre non si sono letti tutti gli elementi
  - Leggere il successivo
  - Se è maggiore del massimo
    - Porre il massimo a questo numero
- Comunicare il massimo

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 100

void inizializza(int vettore[], int dim);
int valore_massimo(int vettore[], int dim);

int
main (void)
{
 int n, massimo;
 int vett[MAX];
 printf("Inserire il numero degli elementi del vettore (Massimo 100) -->");
 scanf("%d", &n);
 inizializza(vett, n);
 massimo=valore_massimo(vett, n);
 printf("L'elemento piu' grande del vettore e' --> %d \n", massimo);
 system("pause");
 return(0);
}
```

Inizializzazione di un vettore

Calcolo del massimo elemento di un vettore

```
void inizializza(int vettore[], int dim)
{
 int i;
 for (i=0; i<dim; i++)
 {
 printf("\nInserire il %d valore --> ", (i+1));
 scanf("%d", &vettore[i]);
 }
}
```

Inizializzazione di un  
vettore con input  
dall'utente

```
int valore_massimo(int vettore[], int dim)
{
 int i, massimo_corrente;
 massimo_corrente=vettore[0];
 for (i=1; i<dim; i++)
 {
 if (vettore[i]>massimo_corrente)
 massimo_corrente=vettore[i];
 }
 return(massimo_corrente);
}
```

Calcolo del massimo  
elemento in un vettore

# Vettori come parametri di input

- È possibile indicare nell'intestazione di una funzione se un array deve essere considerato solo di input, ogni tentativo di modifica ai singoli elementi sarà segnalato come errore sintattico dal compilatore
- La sintassi

```
Tipo_ris nome_funzione (const tipo_elem nome_array[]);
■ void inizializza (const int vettore[], int dim);
■ void visualizza (const int vettore[], int dim);
```

# Array multidimensionali e sottoprogrammi

- Il passaggio di array multidimensionali come parametri di un sottoprogramma ricalca quanto già detto per gli array monodimensionali con il vincolo che l'unica dimensione che può essere lasciata non definita è la prima

```
Tipo_risultato nome_funzione (tipo_elem nome_array[] [dim2]);
```

- ```
void rileva_temperatura (double temp[ ][mesi]);
```

Chiamata di un sottoprogramma

```
rileva_temperatura (temperatura);
```

Strutture e file

Dott.ssa Veronica Rossano
rossano@di.uniba.it
<http://www.di.uniba.it/~rossano>

Laboratorio di Programmazione - Veronica
Rossano

1

Tipi definiti dall'utente: i record

- Un record è una collezione di informazioni riguardanti uno specifico oggetto
- La struttura del record è determinata dalla struttura e dalla natura dell'oggetto che si vuole rappresentare
- Per definire una struttura è necessario specificare tutti i singoli elementi che la compongono

Laboratorio di Programmazione - Veronica Rossano

2

Il tipo struttura

- Il C consente di definire un record utilizzando la **structure type definition** che consentirà di definire variabili con una determinata struttura
- La sintassi è la seguente

```
typedef struct {  
    tipo1 comp1;  
    tipo2 comp2;  
    ...  
} nome_struttura;
```

Laboratorio di Programmazione - Veronica Rossano

3

Esempi

- Definiamo un record che contenga i dati identificativi di uno studente

```
typedef struct {  
    char cognome[20];  
    char nome[20];  
    char matricola[6];  
    char corso_di_laurea[20];  
} studente_t;
```

- Definiamo la struttura di un numero complesso

```
typedef struct {  
    double parte_reale;  
    double parte_immaginaria;  
} complesso_t;
```

Laboratorio di Programmazione - Veronica Rossano

4

Riferirsi ad una componente

- Per manipolare ogni singola componente della struttura si utilizza l'operatore di selezione `.` che divide il nome della struttura dal nome della componente

```
studente_t stud;
stud.nome /* si riferisce al nome */
stud.cognome /* si riferisce al cognome */
stud.matricola /* si riferisce alla matricola */

complesso_t numero;
numero.parte_reale /* si riferisce alla parte reale */
numero.parte_immaginaria /* parte immaginaria */
```

Riferirsi all'intera struttura

- Se necessario è possibile riferirsi all'intera struttura semplicemente utilizzando il nome della variabile dichiarata del tipo struttura
- Istruzioni del tipo seguente creano una copia della struttura che può essere manipolata indipendentemente dalla struttura originaria

```
studente_t stud, stud2;
stud2=stud;
complesso_t numero, numero2;
numero2=numero;
```

Strutture e sottoprogrammi

- È possibile passare un tipo struttura come parametro di input/output di un sottoprogramma
 - Per default il passaggio avviene per **valore**
 - I valori di tutte le componenti del parametro attuale sono copiate nelle componenti del parametro formale
 - Se il passaggio deve essere fatto per **indirizzo** si utilizza come di consueto il puntatore alla struttura
 - Gli operatori `*` e `&` devono essere applicati come di consueto

Restituire una struttura come risultato di una funzione

- Al contrario di quanto avviene per gli array è possibile restituire una struttura come se fosse un dato elementare del C
- Una funzione che restituisce una struttura è definita esattamente come una funzione che restituisce un dato elementare
- La funzione restituisce il valore della struttura non un indirizzo

Tipo Struttura.c

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

Definizione della
struttura studente_t

```
typedef struct
{
    char cognome[20];
    char nome[20];
    char matricola[6];
    char corso_di_laurea[20];
} studente_t;
```

Parametro
formale

```
void inserisci_dettagli(void);
void stampa_dettagli(studente_t stud);
```

```
int
main ()
```

```
{
    studente_t studente;
    printf("*****\n");
    printf("*** Inserisci i dati identificativi dello studente **\n");
    printf("*****\n");
    studente=inserisci_dettagli();
    printf("\n\n*****\n");
    printf("*** I dati identificativi dello studente inseriti sono: **\n");
    printf("*****\n");
```

Chiamata della
procedura

Chiamata della
funzione

```
    stampa_dettagli(studente);
    system("pause");
    return(0);
}
```

```
studente_t inserisci_dettagli(void)
```

```
{
```

```
    studente_t stud;
    printf("\n\n        COGNOME  --> " );
    scanf("%s", stud.cognome);
    printf("\n\n        NOME    --> " );
    scanf("%s", stud.nome);
    printf("\n\n        MATRICOLA --> " );
    scanf("%s", stud.matricola);
    printf("\n\n        CORSO DI LAUREA IN  --> " );
    scanf("%s", stud.corso_di_laurea);
    return(stud);
}
```

Uso del
parametro
attuale

```
void stampa_dettagli(studente_t stud)
```

```
{
```

```
    printf("\n\n        COGNOME  --> %s", stud.cognome);
    printf("\n\n        NOME    --> %s", stud.nome);
    printf("\n\n        MATRICOLA --> %s", stud.matricola);
    printf("\n\n        CORSO DI LAUREA IN  --> %s\n\n", stud.corso_di_laurea);
}
```

```
int
main ()
```

```
{
    studente_t studente;
    printf("*****\n");
    printf("*** Inserisci i dati identificativi dello studente **\n");
    printf("*****\n");
    inserisci_dettagli(&studente);
    printf("\n\n*****\n");
    printf("*** I dati identificativi dello studente inseriti sono: **\n");
    printf("*****\n");
```

Passaggio del parametro per
indirizzo

```
    stampa_dettagli(studente);
    system("pause");
    return(0);
}
```

Tipo Struttura Puntatore.c

```
void inserisci_dettagli(studente_t *stud)
{
    printf("\n\n        COGNOME  --> " );
    scanf("%s", (*stud).cognome);
    printf("\n\n        NOME    --> " );
    scanf("%s", (*stud).nome);
    printf("\n\n        MATRICOLA --> " );
    scanf("%s", (*stud).matricola);
    printf("\n\n        CORSO DI LAUREA IN  --> " );
    scanf("%s", (*stud).corso_di_laurea);
}

void stampa_dettagli(studente_t stud)
{
    printf("\n\n        COGNOME  --> %s", stud.cognome);
    printf("\n\n        NOME    --> %s", stud.nome);
    printf("\n\n        MATRICOLA --> %s", stud.matricola);
    printf("\n\n        CORSO DI LAUREA IN  --> %s\n\n", stud.corso_di_laurea);
}
```

Riferimento ad una componente
della struttura puntata dal
puntatore

Operatore di selezione indiretta delle componenti

- Quando si utilizza un puntatore ad una struttura il riferimento alle singole componenti è più complesso
- Si utilizza l'operatore di selezione indiretta -> che consente di selezionare i valori delle singole componenti di una struttura puntata da un puntatore
- Le seguenti istruzioni sono equivalenti

(*stud).cognome
stud->cognome

Struttura Puntatore 2.c

```

void inserisci_dettagli(studente_t *stud)
{
    printf("\n\n      COGNOME    --> " );
    scanf("%s", stud->cognome);
    printf("\n\n      NOME      --> " );
    scanf("%s", stud->nome);
    printf("\n\n      MATRICOLA   --> " );
    scanf("%s", stud->matricola);
    printf("\n\n      CORSO DI LAUREA IN    --> " );
    scanf("%s", stud->corso_di_laurea);
}

```

Esercizio

- Scrivere un programma che definisca una struttura per la memorizzazione e la visualizzazione dei dati di un libro (autore, ISBN, titolo, disponibilità di magazzino)
 - NB: ricordate che le stringhe in C non possono contenere spazi

Struttura Libro.c

```

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

typedef struct
{
    char autore[20];
    char titolo[50];
    char isbn[11];
    int disponibilita;
} libro_t;

void inserisci_dettagli(libro_t *book);
void stampa_dettagli(libro_t book);

int
main ()
{
    libro_t libro;
    printf("*****\n" );
    printf("***          Inserisci i dati del libro          **\n" );
    printf("*****\n" );
    inserisci_dettagli(&libro);
    printf("\n\n*****\n" );
    printf("***    I dati del libro inseriti sono:    **\n" );
    printf("*****\n" );

    stampa_dettagli(libro);
    system("pause");
    return(0);
}

```

```

void inserisci_dettagli(libro_t *book)
{
    printf("\n\n      AUTORE    --> " );
    scanf("%s", book->autore);
    printf("\n\n      TITOLO    --> " );
    scanf("%s", book->titolo);
    printf("\n\n      ISBN     --> " );
    scanf("%s", book->isbn);
    printf("\n\n      DISPONIBILITA' --> " );
    scanf("%d", &book->disponibilita);
}

void stampa_dettagli(libro_t book)
{
    printf("\n\n      AUTORE    --> %s", book.autore);
    printf("\n\n      TITOLO    --> %s", book.titolo);
    printf("\n\n      ISBN     --> %s", book.isbn);
    printf("\n\n      DISPONIBILITA' --> %d \n\n", book.disponibilita);
}

```

Array di strutture

- È possibile combinare la definizione di un array con la definizione di una struttura per poter utilizzare delle collezioni di dati che contengano elementi simili e a loro volta composti da componenti differenti.

```
typedef struct
{
    char cognome[20];
    char nome[20];
} studente_t;
studente_t studente[MAX];
```

Laboratorio di Programmazione - Veronica Rossano

17

Esercizio

- Costruire un programma che memorizzi e visualizzi tutti i risultati delle varie prove dell'esame di programmazione degli studenti presenti in quest'aula. Per ciascuno studente memorizzare e visualizzare il voto finale calcolato come la parte intera della media tra i tre voti.

Laboratorio di Programmazione - Veronica Rossano

18

```
typedef struct
{
    char cognome[20];
    char nome[20];
    int laboratorio;
    int scritto;
    int orale;
    int media;
} studente_t;

studente_t inserisci_dettagli(void);
void stampa_dettagli(studente_t stud);

int
main ()
{
    studente_t studente[MAX];
    int n,i;
    printf("Inserire il numero degli studenti (Massimo 60) -->");
    scanf("%d", &n);
    for(i=0; i<n; ++i)
    {
        printf("\n\n**   Inserisci i dati del %d studente   **", i+1 );
        studente[i]=inserisci_dettagli();
    }
    for(i=0; i<n; ++i)
    {
        printf("\n\n**   I voti del %d studente sono   **", i+1 );
        stampa_dettagli(studente[i]);
    }
    system("pause");
    return(0);
}
```

19

```
studente_t inserisci_dettagli()
{
    studente_t stud;
    printf("\n\n      COGNOME    --> " );
    scanf("%s", stud.cognome);
    printf("\n\n      NOME      --> " );
    scanf("%s", stud.nome);
    printf("\n\n      VOTO DELLA PROVA DI LABORATORIO    --> " );
    scanf("%d", &stud.laboratorio);
    printf("\n\n      VOTO DELLA PROVA SCRITTA    --> " );
    scanf("%d", &stud.scritto);
    printf("\n\n      VOTO DELLA PROVA ORALE    --> " );
    scanf("%d", &stud.orale);
    stud.media=(stud.laboratorio+stud.orale+stud.scritto)/3;
    return(stud);
}

void stampa_dettagli(studente_t stud)
{
    printf("\n\n      COGNOME    --> %s", stud.cognome);
    printf("\n\n      NOME      --> %s", stud.nome);
    printf("\n\n      PROVA DI LABORATORIO    --> %d", stud.laboratorio);
    printf("\n\n      PROVA SCRITTA    --> %d", stud.scritto);
    printf("\n\n      PROVA ORALE    --> %d\n\n", stud.orale);
    printf("\n\n      VOTO FINALE --> %d\n\n", stud.media);
}
```

I file...

- Il C consente di utilizzare file di testo
- Un file di testo è una collezione di caratteri salvati in memoria secondaria
- Un file non ha una dimensione fissa, la fine del file è indicata con un carattere speciale di *end of file* denotato con EOF
- Il ritorno a capo in C è identificato dai caratteri \n

...I file

- Per utilizzare un file è necessario definire una variabile puntatore al file
- Il sistema deve preparare il file per ricevere l'input e l'output prima di permettere l'accesso
- Le funzioni che consentono di accedere ai file si trovano nella libreria <stdio.h>

Definire e aprire un file

- La sintassi per la definizione di un file è la seguente

```
FILE *nome_puntatore;
```

- FILE *input_file;
- FILE *studenti;

- La sintassi per l'apertura di un file è la seguente

```
nome_puntatore= fopen("nome_file.txt", "opzione");
```

Dove opzione indica l'operazione che si intende elaborare sul file

Aprire un file...

- La funzione fopen
 - Consente di aprire un file
 - Il primo argomento indica il nome del file da elaborare
 - Il secondo indica quale operazione si intende compiere sul file
 - Ritorna un puntatore ad un FILE
 - Se l'operazione non va a buon fine il puntatore assume il valore NULL

... Aprire un file

■ Le opzioni di apertura di un file

- r lettura
- w scrittura
- a appendere in coda
- r+ lettura/scrittura
- w+ creazione lettura/scrittura
- a+ appende o crea un file per lettura/scrittura

Chiudere un file

- Quando un programma non deve più utilizzare un file è necessario chiuderlo per liberare la memoria allocata al momento della sua apertura
- La funzione `fclose` chiude il file e impedisce ogni altro accesso al file

Leggere un file

- La funzione `getc` consente di leggere un carattere alla volta all'interno di un file

- Prende in input un puntatore ad un file
- Restituisce un carattere
- Alla fine del file restituisce EOF

```

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

int
main ()
{
    FILE *primo_file;
    char ch;
    primo_file=fopen("CiaoMondo.txt", "r");
    if (primo_file==NULL)
    {
        printf("Impossibile aprire il file\n");
    }
    else
    {
        for(ch=getc(primo_file); ch!=EOF; ch=getc(primo_file))
            printf("%c", ch);
        fclose(primo_file);
    }
    printf("\n");

    system("pause");
    return(0);
}

```

Leggi File.c

Definizione del FILE

Apertura del file

Legge un carattere alla volta

Scrivere un file

- La funzione **putc** consente di scrivere un carattere alla volta all'interno di un file
 - Prende in input un puntatore ad un file e il carattere da scrivere

```

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

int
main ()
{
    FILE *primo_file;
    char ch[60];
    int i, lung;
    strcpy(ch, "Questo e' un nuovo file\nE' la prima prova di un file di testo");
    lung=strlen(ch);
    primo_file=fopen("CiaoMondo.txt", "w+");
    if (primo_file==NULL)
    {
        printf("Impossibile aprire il file");
    }
    else
    {
        for(i=0; i<=lung; i++)
            putc(ch[i], primo_file);
        printf("Il file e' stato creato con successo.\n");
        fclose(primo_file);
    }
    system("pause");
    return(0);
}

```

Leggi File.c

Stringa da scrivere nel file

Apertura del file

Scrittura di un carattere nel file

fprintf e fscanf

- Per inserire dati in un file sequenziale:
 - in cui inserire dati di diverso tipo
 - senza dover scandire carattere per carattere

possono essere usate le funzioni
fprintf e **fscanf**

```

int
main ()
{
    FILE *primo_file;
    char insegnamento[60];
    int voto;
    primo_file=fopen("Esami.txt", "a+"); // appende le informazioni alla fine del file
    printf("Il programma memorizza in un file Esami.txt le votazioni riportate \n da un s");
    if (primo_file==NULL)
        printf("\n\n\n***** Impossibile aprire il file***** \n\n");
    else
    {
        printf("Inserire gli insegnamenti e le rispettive votazioni riportate (Es. Programmazione");
        scanf("%s %d", insegnamento, &voto);
        while (!feof (stdin))
        {
            fprintf(primo_file, "%s %d\n", insegnamento, voto);
            scanf("%s %d", insegnamento, &voto);
        }
        printf("File creato\n\n");
        fclose(primo_file);
    }

    system("pause");
    return(0);
}

```

Fprintf.c


```

int
main ()
{
    FILE *primo_file;
    char insegnamento[60];
    int voto;
    primo_file=fopen("Esami.txt", "r"); // appende le informazioni alla fine del file
    printf("Il programma memorizza in un file Esami.txt le votazioni riportate \n da un s
    if (primo_file==NULL)
        printf("\n\n\n***** Impossibile aprire il file***** \n\n");
    else
    {
        printf(" %-20s %-6s\n\n", "Insegnamento", "Voto");
        fscanf(primo_file,"%s%d", insegnamento, &voto);
        while (!feof (primo_file))
        {
            printf(" %-20s %d\n\n", insegnamento, voto);
            fscanf(primo_file,"%s %d", insegnamento, &voto);
        }
        printf("*****File Terminato*****\n\n");
        fclose(primo_file);
    }

    system("pause");
    return(0);
}

```

Fscanf.c

File ad accesso casuale

- In C è possibile creare file ad accesso casuale e quindi file di record
- La funzione **fwrite** trasferisce in un file un numero specificato di byte partendo da una data posizione in memoria
- La funzione **fseek** sistema il puntatore di posizione del file su un byte specifico
- La funzione **fread** legge uno specifico numero di byte da un file

Fwrite

File Accesso Casuale.c

```

do
{
    printf("Inserisci i dati dello studente: \n");
    scanf(stdin,"%s%s%s", stud.cognome, stud.nome,
        stud.matricola, stud.corso_di_laurea );
    fwrite(&stud, sizeof(studente_t), 1, primo_file);
    printf("\nVuoi Continuare? (s/n)\n");
    scanf("%c", &risp);
}while ((risp=='s') || (risp=='S'));

```

Fseek...

```
fseek(puntatoreFile,sizeof(struttura), posizionePartenza);
```

- Dove:
 - **puntatoreFile** è il puntatore al file su cui si intende operare
 - **Struttura** è il nome della struttura che si intende scrivere nel file
 - **PosizionePartenza** può assumere i seguenti valori:
 - **SEEK_SET** indica l'inizio del file
 - **SEEK_CUR** indica la posizione corrente
 - **SEEK_END** indica la fine del file

...Fseek

File Casuale Fseek.c

```

do
{
    printf("Inserire i dati dello studente (-1 nel codice per terminare) -->");
    printf("\n\n      CODICE    --> " );
    scanf("%d", &stud.codice);
    if (stud.codice!=-1)
    {
        printf("      COGNOME    --> " );
        scanf("%s", stud.cognome);
        printf("      NOME      --> " );
        scanf("%s", stud.nome);
        printf("      MATRICOLA  --> " );
        scanf("%s", stud.matricola);
        printf("\n      CORSO DI LAUREA IN  --> " );
        scanf("%s", stud.corso_di_laurea);
        fseek(primo_file, (stud.codice-1)*sizeof(studente_t), SEEK_SET);
        fwrite(&stud, sizeof(studente_t), 1, primo_file);
    }

    }while (stud.codice!=-1);

```

Fread...

```

fread(IndirizzoStruttura, sizeof(struttura), numElementi,
puntatoreFile);

```

■ Dove:

- **IndirizzoStruttura** è l'indirizzo della struttura in cui memorizzare i dati della struttura che si intende leggere dal file
- **numElementi** è il numero di elementi da leggere dal file
- **puntatoreFile** è il puntatore al file su cui si intende operare

...Fread

```

while (!feof(primo_file))
{
    fread(&stud, sizeof(studente_t), 1, primo_file);
    if (!feof(primo_file))
        printf(" %-10s %-10s %-10s %-10s\n\n", stud.cognome,
            stud.nome, stud.matricola,
            stud.corso_di_laurea );
}

```

Librerie personali

- È possibile creare le proprie librerie personali che contengano funzioni già sviluppate e pronte per essere riutilizzate
 - la direttiva del preprocessore **#include** consente di richiamare i file di libreria
 - I file di libreria devono avere estensione **.h**

Header file...

- Un header file è un file di testo che contiene tutte le informazioni necessarie al compilatore durante la fase di compilazione di un programma che utilizza funzioni definite nella libreria
- La struttura di un file header prevede
 - Un blocco di commento che definisca l'obiettivo della libreria
 - Le direttive che definiscono le costanti
 - Le eventuali definizioni di tipi
 - I corpi delle varie funzioni

...Header file

- Quando si usa un header file di sistema la sintassi è

`#include <stdio.h>`

- Quando si usa un header file personale che si trova nella stessa directory del file che stiamo realizzando la sintassi è

`#include "array.h"`

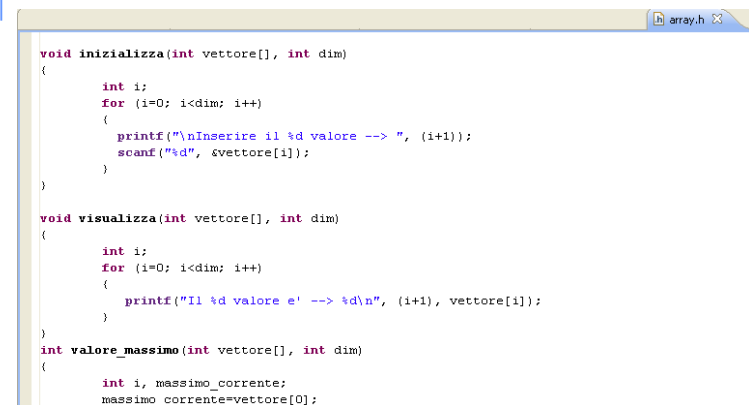
```
#include <stdio.h>
#include <stdlib.h>
#include "array.h"
#define MAX 100

int cerca_in_vettore(int vettore[], int dim, int valore);

int
main (void)
{
    int n, valore, pos;
    int vett[MAX];
    printf("Inserire il numero degli elementi del vettore (Massimo 100) -->");
    scanf("%d", &n);
    inizializza(vett, n);
    printf("\nInserire il numero da cercare -->");
    scanf("%d", &valore);
    pos=cerca_in_vettore(vett, n, valore);
    if (pos==-1)
        printf("\nElemento cercato non esiste\n\n");
    else
        printf("\nElemento cercato si trova in posizione %d\n\n", (pos+1));
    system("pause");
    return(0);
}

int cerca_in_vettore(int vettore[], int dim, int valore)
{
    int i, posizione;
    posizione=-1;
    i=0;
    while ((i<dim) && (posizione==-1))
    {
        if (vettore[i]==valore)
            posizione=i;
        ++i;
    };
    return (posizione);
}
```

formatica



```
void inizializza(int vettore[], int dim)
{
    int i;
    for (i=0; i<dim; i++)
    {
        printf("\nInserire il %d valore --> ", (i+1));
        scanf("%d", &vettore[i]);
    }
}

void visualizza(int vettore[], int dim)
{
    int i;
    for (i=0; i<dim; i++)
    {
        printf("Il %d valore e' --> %d\n", (i+1), vettore[i]);
    }
}

int valore_massimo(int vettore[], int dim)
{
    int i, massimo_corrente;
    massimo_corrente=vettore[0];
```

```
#include <stdio.h>
#include <stdlib.h>
#include "array.h"
#define MAX 100

void ordina_vettore(int vettore[], int dim);

int
main (void)
{
    int n;
    int vett[MAX];
    printf("Inserire il numero degli eleme");
    scanf("%d", &n);
    inizializza(vett, n);
    ordina_vettore(vett, n);
    visualizza(vett, n);
    system("pause");
    return (0);
}
```

File
header

```
void inizializza(int vettore[], int dim)
{
    int i;
    for (i=0; i<dim; i++)
    {
        printf("\nInserire il %d valore --> ", (i+1));
        scanf("%d", &vettore[i]);
    }
}

void visualizza(int vettore[], int dim)
{
    int i;
    for (i=0; i<dim; i++)
    {
        printf("Il %d valore e' --> %d\n", (i+1), vettore[i]);
    }
}

int valore_massimo(int vettore[], int dim)
{
    int i, massimo_corrente;
    massimo_corrente=vettore[0];
    for (i=1; i<dim; i++)
    {
        if (vettore[i]>massimo_corrente)
            massimo_corrente=vettore[i];
    };
    return(massimo_corrente);
}
```

Laboratorio di Programmazione

Altre librerie utili

- Math.h
 - Contiene funzioni matematiche
 - Sin (seno), cos (coseno), sqrt (radice quadrata),...
- Time.h
 - Contiene funzioni per il calcolo di tempo e data
 - Strftime (formato tempo), clock (tempo di esecuzione),...

Strutture

Dott.ssa Veronica Rossano
rossano@di.uniba.it
<http://www.di.uniba.it/~rossano>

Laboratorio di Programmazione - Veronica
Rossano

1

Tipi definiti dall'utente: i record

- Un record è una collezione di informazioni riguardanti uno specifico oggetto
- La struttura del record è determinata dalla struttura e dalla natura dell'oggetto che si vuole rappresentare
- Per definire una struttura è necessario specificare tutti i singoli elementi che la compongono

Laboratorio di Programmazione - Veronica Rossano

2

Il tipo struttura

- Il C consente di definire un record utilizzando la **structure type definition** che consentirà di definire variabili con una determinata struttura
- La sintassi è la seguente

```
typedef struct {  
    tipo1 comp1;  
    tipo2 comp2;  
    ...  
} nome_struttura;
```

Laboratorio di Programmazione - Veronica Rossano

3

Esempi

- Definiamo un record che contenga i dati identificativi di uno studente

```
typedef struct {  
    char cognome[20];  
    char nome[20];  
    char matricola[6];  
    char corso_di_laurea[20];  
} studente_t;
```

- Definiamo la struttura di un numero complesso

```
typedef struct {  
    double parte_reale;  
    double parte_immaginaria;  
} complesso_t;
```

Laboratorio di Programmazione - Veronica Rossano

4

Riferirsi ad una componente

- Per manipolare ogni singola componente della struttura si utilizza l'operatore di selezione `.` che divide il nome della struttura dal nome della componente

```

studente_t stud;
    stud.nome /* si riferisce al nome */
    stud.cognome /* si riferisce al cognome */
    stud.matricola /* si riferisce alla matricola */

complesso_t numero;
    numero.parte_reale /* si riferisce alla parte reale */
    numero.parte_immaginaria /* parte immaginaria */

```

Riferirsi all'intera struttura

- Se necessario è possibile riferirsi all'intera struttura semplicemente utilizzando il nome della variabile dichiarata del tipo struttura
- Istruzioni del tipo seguente creano una copia della struttura che può essere manipolata indipendentemente dalla struttura originaria

```

studente_t stud, stud2;
    stud2=stud;
complesso_t numero, numero2;
    numero2=numero;

```

Strutture e sottoprogrammi

- È possibile passare un tipo struttura come parametro di input/output di un sottoprogramma
 - Per default il passaggio avviene per **valore**
 - I valori di tutte le componenti del parametro attuale sono copiate nelle componenti del parametro formale
 - Se il passaggio deve essere fatto per **indirizzo** si utilizza come di consueto il puntatore alla struttura
 - Gli operatori `*` e `&` devono essere applicati come di consueto

Restituire una struttura come risultato di una funzione

- Al contrario di quanto avviene per gli array è possibile restituire una struttura come se fosse un dato elementare del C
- Una funzione che restituisce una struttura è definita esattamente come una funzione che restituisce un dato elementare
- La funzione restituisce il valore della struttura non un indirizzo

Tipo Struttura.c

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

```
typedef struct
```

```
{
    char cognome[20];
    char nome[20];
    char matricola[6];
    char corso_di_laurea[20];
}
```

```
studente_t;
```

```
studente_t inserisci_dettagli(void);
```

```
void stampa_dettagli(studente_t stud);
```

```
int
```

```
main ()
```

```
{
    studente_t studente;
    printf("*****\n");
    printf("*** Inserisci i dati identificativi dello studente **\n");
    printf("*****\n");
    studente=inserisci_dettagli();
    printf("\n\n*****\n");
    printf("*** I dati identificativi dello studente inseriti sono: **\n");
    printf("*****\n");
}
```

```
    stampa_dettagli(studente);
    system("pause");
    return(0);
}
```

Chiamata della procedura

Chiamata della funzione

Definizione della struttura studente_t

Parametro formale

```
studente_t inserisci_dettagli(void)
```

```
{
```

```
    studente_t stud;
```

```
    printf("\n\n      COGNOME    --> " );
```

```
    scanf("%s", stud.cognome);
```

```
    printf("\n\n      NOME      --> " );
```

```
    scanf("%s", stud.nome);
```

```
    printf("\n\n      MATRICOLA  --> " );
```

```
    scanf("%s", stud.matricola);
```

```
    printf("\n\n      CORSO DI LAUREA IN  --> " );
```

```
    scanf("%s", stud.corso_di_laurea);
```

```
    return(stud);
```

```
}
```

```
void stampa_dettagli(studente_t stud)
```

```
{
```

```
    printf("\n\n      COGNOME    --> %s", stud.cognome);
```

```
    printf("\n\n      NOME      --> %s", stud.nome);
```

```
    printf("\n\n      MATRICOLA  --> %s", stud.matricola);
```

```
    printf("\n\n      CORSO DI LAUREA IN  --> %s\n\n", stud.corso_di_laurea);
```

```
}
```

Uso del parametro attuale

```
int
main ()
```

```
{
```

```
    studente_t studente;
```

```
    printf("*****\n");
```

```
    printf("*** Inserisci i dati identificativi dello studente **\n");
```

```
    printf("*****\n");
```

```
    inserisci_dettagli(&studente);
```

```
    printf("\n\n*****\n");
```

```
    printf("*** I dati identificativi dello studente inseriti sono: **\n");
```

```
    printf("*****\n");
```

```
    stampa_dettagli(studente);
```

```
    system("pause");
```

```
    return(0);
```

```
}
```

Tipo Struttura Puntatore.c

Passaggio del parametro per indirizzo

```
void inserisci_dettagli(studente_t *stud)
```

```
{
```

```
    printf("\n\n      COGNOME    --> " );
```

```
    scanf("%s", (*stud).cognome);
```

```
    printf("\n\n      NOME      --> " );
```

```
    scanf("%s", (*stud).nome);
```

```
    printf("\n\n      MATRICOLA  --> " );
```

```
    scanf("%s", (*stud).matricola);
```

```
    printf("\n\n      CORSO DI LAUREA IN  --> " );
```

```
    scanf("%s", (*stud).corso_di_laurea);
```

```
}
```

```
void stampa_dettagli(studente_t stud)
```

```
{
```

```
    printf("\n\n      COGNOME    --> %s", stud.cognome);
```

```
    printf("\n\n      NOME      --> %s", stud.nome);
```

```
    printf("\n\n      MATRICOLA  --> %s", stud.matricola);
```

```
    printf("\n\n      CORSO DI LAUREA IN  --> %s\n\n", stud.corso_di_laurea);
```

```
}
```

Riferimento ad una componente della struttura puntata dal puntatore

Operatore di selezione indiretta delle componenti

- Quando si utilizza un puntatore ad una struttura il riferimento alle singole componenti è più complesso
- Si utilizza l'operatore di selezione indiretta -> che consente di selezionare i valori delle singole componenti di una struttura puntata da un puntatore
- Le seguenti istruzioni sono equivalenti

(*stud).cognome
stud->cognome


```

void inserisci_dettagli(studente_t *stud)
{
    printf("\n\n      COGNOME    --> " );
    scanf("%s", stud->cognome);
    printf("\n\n      NOME      --> " );
    scanf("%s", stud->nome);
    printf("\n\n      MATRICOLA   --> " );
    scanf("%s", stud->matricola);
    printf("\n\n      CORSO DI LAUREA IN    --> " );
    scanf("%s", stud->corso_di_laurea);
}

```

Esercizio

- Scrivere un programma che definisca una struttura per la memorizzazione e la visualizzazione dei dati di un libro (autore, ISBN, titolo, disponibilità di magazzino)
 - NB: ricordate che le stringhe in C non possono contenere spazi

```

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

typedef struct
{
    char autore[20];
    char titolo[50];
    char isbn[11];
    int disponibilita;
} libro_t;

void inserisci_dettagli(libro_t *book);
void stampa_dettagli(libro_t book);

int
main ()
{
    libro_t libro;
    printf("*****\n");
    printf("***      Inserisci i dati del libro      **\n");
    printf("*****\n");
    inserisci_dettagli(&libro);
    printf("\n\n*****\n");
    printf("***      I dati del libro inseriti sono:      **\n");
    printf("*****\n");

    stampa_dettagli(libro);
    system("pause");
    return(0);
}

```

```

void inserisci_dettagli(libro_t *book)
{
    printf("\n\n      AUTORE    --> " );
    scanf("%s", book->autore);
    printf("\n\n      TITOLO    --> " );
    scanf("%s", book->titolo);
    printf("\n\n      ISBN      --> " );
    scanf("%s", book->isbn);
    printf("\n\n      DISPONIBILITA' --> " );
    scanf("%d", &book->disponibilita);
}

void stampa_dettagli(libro_t book)
{
    printf("\n\n      AUTORE    --> %s", book.autore);
    printf("\n\n      TITOLO    --> %s", book.titolo);
    printf("\n\n      ISBN      --> %s", book.isbn);
    printf("\n\n      DISPONIBILITA' --> %d \n\n", book.disponibilita);
}

```

Array di strutture

- È possibile combinare la definizione di un array con la definizione di una struttura per poter utilizzare delle collezioni di dati che contengano elementi simili e a loro volta composti da componenti differenti.

```
typedef struct
{
    char cognome[20];
    char nome[20];
} studente_t;

studente_t studente[MAX];
```

Laboratorio di Programmazione - Veronica Rossano

17

Esercizio

- Costruire un programma che memorizzi e visualizzi tutti i risultati delle varie prove dell'esame di programmazione degli studenti presenti in quest'aula. Per ciascuno studente memorizzare e visualizzare il voto finale calcolato come la parte intera della media tra i tre voti.

Laboratorio di Programmazione - Veronica Rossano

18

```
typedef struct
{
    char cognome[20];
    char nome[20];
    int laboratorio;
    int scritto;
    int orale;
    int media;
} studente_t;

studente_t inserisci_dettagli(void);
void stampa_dettagli(studente_t stud);

int
main ()
{
    studente_t studente[MAX];
    int n,i;
    printf("Inserire il numero degli studenti (Massimo 60) -->");
    scanf("%d", &n);
    for(i=0; i<n; ++i)
    {
        printf("\n\n**   Inserisci i dati del %d studente   **", i+1 );
        studente[i]=inserisci_dettagli();
    }
    for(i=0; i<n; ++i)
    {
        printf("\n\n**   I voti del %d studente sono   **", i+1 );
        stampa_dettagli(studente[i]);
    }
    system("pause");
    return(0);
}
```

19

```
studente_t inserisci_dettagli()
{
    studente_t stud;
    printf("\n\n      COGNOME    --> " );
    scanf("%s", stud.cognome);
    printf("\n\n      NOME      --> " );
    scanf("%s", stud.nome);
    printf("\n\n      VOTO DELLA PROVA DI LABORATORIO    --> " );
    scanf("%d", &stud.laboratorio);
    printf("\n\n      VOTO DELLA PROVA SCRITTA    --> " );
    scanf("%d", &stud.scritto);
    printf("\n\n      VOTO DELLA PROVA ORALE    --> " );
    scanf("%d", &stud.orale);
    stud.media=(stud.laboratorio+stud.orale+stud.scritto)/3;
    return (stud);
}

void stampa_dettagli(studente_t stud)
{
    printf("\n\n      COGNOME    --> %s", stud.cognome);
    printf("\n\n      NOME      --> %s", stud.nome);
    printf("\n\n      PROVA DI LABORATORIO    --> %d", stud.laboratorio);
    printf("\n\n      PROVA SCRITTA    --> %d", stud.scritto);
    printf("\n\n      PROVA ORALE    --> %d\n\n", stud.orale);
    printf("\n\n      VOTO FINALE --> %d\n\n", stud.media);
}
```

Librerie personali

- È possibile creare le proprie librerie personali che contengano funzioni già sviluppate e pronte per essere riutilizzate
 - ▢ la direttiva del preprocessore `#include` consente di richiamare i file di libreria
 - ▢ I file di libreria devono avere estensione `.h`

Header file...

- Un header file è un file di testo che contiene tutte le informazioni necessarie al compilatore durante la fase di compilazione di un programma che utilizza funzioni definite nella libreria
- La struttura di un file header prevede
 - ▢ Un blocco di commento che definisca l'obiettivo della libreria
 - ▢ Le direttive che definiscono le costanti
 - ▢ Le eventuali definizioni di tipi
 - ▢ I corpi delle varie funzioni

...Header file

- Quando si usa un header file di sistema la sintassi è

`#include <stdio.h>`

- Quando si usa un header file personale che si trova nella stessa directory del file che stiamo realizzando la sintassi è

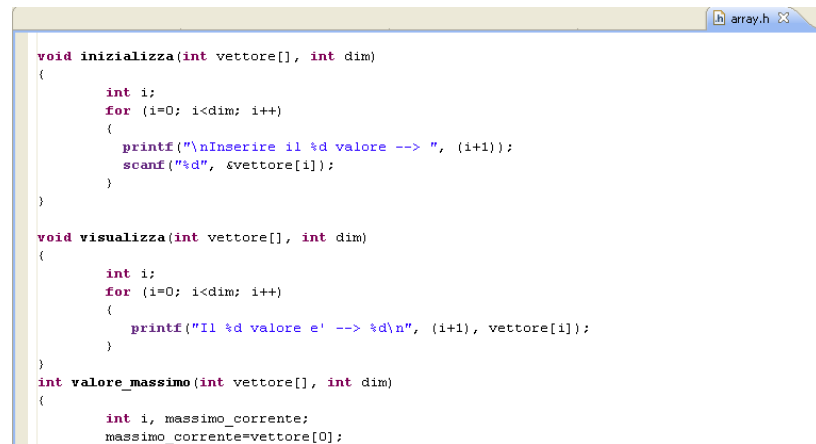
`#include "array.h"`

```
#include <stdio.h>
#include <stdlib.h>
#include "array.h"
#define MAX 100

int cerca_in_vettore(int vettore[], int dim, int valore);

int
main (void)
{
    int n, valore, pos;
    int vett[MAX];
    printf("Inserire il numero degli elementi del vettore (Massimo 100) -->");
    scanf("%d", &n);
    inizializza(vett, n);
    printf("\n\nInserire il numero da cercare -->");
    scanf("%d", &valore);
    pos=cerca_in_vettore(vett, n, valore);
    if (pos!=-1)
        printf("\nL'elemento cercato non esiste\n\n");
    else
        printf("\nL'elemento cercato si trova in posizione %d\n\n", (pos+1));
    system("pause");
    return(0);
}

int cerca_in_vettore(int vettore[], int dim, int valore)
{
    int i, posizione;
    posizione=-1;
    i=0;
    while ((i<dim) && (posizione==-1))
    {
        if (vettore[i]==valore)
            posizione=i;
        ++i;
    };
    return (posizione);
}
```



```
void inizializza(int vettore[], int dim)
{
    int i;
    for (i=0; i<dim; i++)
    {
        printf("\nInserire il %d valore --> ", (i+1));
        scanf("%d", &vettore[i]);
    }
}

void visualizza(int vettore[], int dim)
{
    int i;
    for (i=0; i<dim; i++)
    {
        printf("Il %d valore e' --> %d\n", (i+1), vettore[i]);
    }
}

int valore_massimo(int vettore[], int dim)
{
    int i, massimo_corrente;
    massimo_corrente=vettore[0];
```

Altre librerie utili

■ Math.h

- Contiene funzioni matematiche
 - Sin (seno), cos (coseno), sqrt (radice quadrata),...

■ Time.h

- Contiene funzioni per il calcolo di tempo e data
 - Strftime (formato tempo), clock (tempo di esecuzione),...