LINGUAGGIO C [1]

- STRUTTURA DI UN PROGRAMMA
  - Parte Dichiarativa Globale (file sorgente)
  - Programma Principale (file sorgente)
    - Parte Dichiarativa
    - Parte Esecutiva
  - Altre Funzioni (file esterni)
- PARTE DICHIARATIVA GLOBALE
  - Direttive di inclusione e compilazione (*#direttiva*)
- PROGRAMMA PRINCIPALE
  - Il programma vero e proprio (main(){…})
- ALTRE FUNZIONI
  - E' possibile importare funzioni esterne (librerie)

LINGUAGGIO C [2]

- UN ESEMPIO: SOMMA DI DUE NUMERI

```
#include <stdio.h>
main()
{
    // inizio parte dichiarativa
    int a,b;
    int somma = 0;
    /* inizio parte esecutiva */
    scanf("%d %d",&a,&b);
    somma=a+b;
    printf("La somma di a+b è:\n%d\narrivederci!\n",somma);
}
```

- OUTPUT

```
La somma di a+b è:
<somma>
arrivederci!
```

LINGUAGGIO C [3]

- PARTE DICHIARATIVA
  - Inclusione di file esterni (`#include <nomefile>`)
  - Direttive di compilazione (`#define nome valore`)
  - Dichiarazione di costanti (`const tipo nome valore`)
  - Dichiarazione di variabili (`tipo nome`)
- PARTE ESECUTIVA
  - Assegnamenti [a=1;]
  - Espressioni aritmetiche [a+b; 2+1; a+1;]
  - Espressioni logiche [a && b; a || b;]
  - Condizioni [if (a && b) …]
  - Cicli [while (a && b) …]
  - Input/output [scanf(…); printf(…);]
  - …

LINGUAGGIO C [4]

- PARTE DICHIARATIVA
  - **#include <stdio.h>**: copia (in compilazione) le funzioni e le variabili definite nel file header *stdio.h*
  - **;** : terminatore di istruzione
  - **{…}** : racchiudono un gruppo di istruzioni in un'unica istruzione composta
  - **int a,b** : dichiaro due variabili di tipo intero che userò durante tutto il programma
  - **int somma = 0** : dichiaro una variabile di tipo intero e la inizializzo ad un valore
  - **const float PiGreco = 3.14** : dichiaro una costante di tipo float e le assegno un valore per tutto il programma

LINGUAGGIO C [5]

- PARTE ESECUTIVA
  - **main() {...}**: il programma inizia sempre dal main ed e' interamente racchiuso da una coppia di parentesi graffe (ALT GR+SHIFT +[e ALT GR+SHIFT+])
  - **scanf(...)**: istruzione di input formattato
  - **somma=a+b** : assegno il valore dell'operazione di somma alla variabile *somma*
  - **printf(...)** : istruzione di output formattato
  - **//** : la linea è commentata
  - **/*...*/** : il testo compreso è commentato (c++)
  - **=, +, %, *, -, /** : alcune istruzioni semplici
  - **==, <, >, !=, &&, ||, !** : alcuni operatori logici
  - **... cicli e condizioni ...**

TIPI DI DATO [1]

- TIPO DI DATO
  - Insieme di valori ed operazioni che possono esservi applicate
  - Ogni dato ha una sua rappresentazione in memoria, ma è solitamente sconosciuta per il programmatore
  - Con la dichiarazione viene associato ad ogni variabile un preciso tipo di dato
- TIPOLOGIE
  - Semplici o built-in (numeri interi, float, caratteri …)
  - Strutturati o user-defined (vettori, record …)
- DEFINIZIONE DI DATI
  - Semplici o built-in
  - Costruttori di tipo (array, struct …)

TIPI DI DATO [2]

- DEFINIZIONE DI TIPI DI DATO
  - Ridefinizione di tipi semplici:
    ```
    typedef Tipo_esistente Nuovo_Tipo;
    ```
  - Enumerazione di valori (in realtà associo i nomi a degli interi):
    ```
    typedef enum {…} Nuovo_Tipo;
    ```
  - Applico costruttori di tipo ai tipi semplici (strutturati)
- ESEMPI (RIDEFINIZIONE ED ENUMERAZIONE)
  - Ridefinisco gli interi per indicare il tipo ANNO
    ```
    typedef int Anno;
    Anno AnnoCorrente, AnnoDiNascita;
    ```
  - Creo il tipo Boolean enumerando i possibili valori
    ```
    typedef enum {false, true} Boolean;
    Boolean Check, Condizione;
    ```

TIPI DI DATO SEMPLICI [1]

- TIPI SEMPLICI (CASE SENSITIVE)
  - int: numeri interi
  - char: caratteri (rappresentati come gli interi, ASCII)
  - float: reali in virgola mobile
  - double: reali in virgola mobile con doppia precisione
- QUALIFICATORI (PER TIPI SEMPLICI)
  - signed: con segno, si applica a char e int
  - unsigned: senza segno, si applica a char e int
  - short: si può applicare a int
  - long: si può applicare a int e double
- TOTALE DI 12 TIPI PREDEFINITI

TIPI DI DATO SEMPLICI [2]

- INT
  - Insieme dei numeri interi {… -1, 0, 1, 2, …}
  - Insieme delle operazioni che vi si possono applicare

=	Asegnamento di un **int** ad un **int**
+	Somma (tra **int** ha come risultato un **int**)
-	Sottrazione (tra **int** ha come risultato un **int**)
*	Moltiplicazione (tra **int** ha come risultato un **int**)
/	Divisione con troncamento (risultato **int**)
%	Resto della divisione intera
==	Relazione di unguaglianza
!=	Relazione di diversità
<	Relazione di "minore di"
>	Relazione di "maggiore di"
<=	Relazione di "minore o uguale di"
=>	Relazione di "maggiore o uguale di"

  - Dimensione dipende da compilatori (16 o 32 bit)
  - signed int (16): da -2^{15} a $2^{15}-1$
  - unsigned int (16): da 0 a $2^{16}-1$

TIPI DI DATO SEMPLICI [3]

- FLOAT E DOUBLE
  - Approssimazione dei numeri reali
  - Insieme delle operazioni che vi si possono applicare

=	Asegnamento
+	Somma
-	Sottrazione
*	Moltiplicazione
/	Divisione (a risultato reale)
==	Relazione di unguaglianza
!=	Relazione di diversità
<	Relazione di "minore di"
>	Relazione di "maggiore di"
<=	Relazione di "minore o uguale di"
=>	Relazione di "maggiore o uguale di"

  - Di solito 4 byte per float e 8 byte per double
  - float 6 cifre decimali valori tra 10^{-38} e 10^{+38}
  - double 15 cifre decimali valori tra 10^{-308} e 10^{+308}

TIPI DI DATO SEMPLICI [4]

- CHAR
  - Insieme dei caratteri
  - Ordinati come sulla tavola ASCII
  - Alcuni caratteri sono di controllo (\n, \t, \b, \r …)
  - Implementati con 1 byte prevedono le stesse operazioni degli interi (ne sono infatti un sottoinsieme)
  - Es: alla variabile carattere viene associato il valore intero corrispondente alla posizione nella tabella ASCII del carattere f (102)

```
char carattere;
carattere = 'f';
```

TIPI DI DATO SEMPLICI [4]

- CAST DI UN'ESPRESSIONE
  - A volte è necessario modificare il tipo di un dato al fine di conservare la consistenza nelle espressioni (o nella chiamata di funzioni)
  - Operatore di cast:
    (NomeTipo) Espressione
  - Es:

```
main()
{
   int x=5, y=2;
   double z;
   z = x / y ;
   /* z assume valore 2 */
   z = (double)x / (double)y ;
   /* z assume valore 2.5 */
}
```

TIPI DI DATO STRUTTURATI [1]

- COSTRUTTORI
  - array (lista di valori omogenei)
  - struct (insieme di elementi eterogenei)
  - union (…)
  - pointer (tipo puntatore)
- TIPI STRUTTURATI
  - Esigenza di aggregare dati di più tipi (omogenei ed eterogenei) per meglio catturare gli aspetti informatici di un concetto astratto
  - Es: l'identità di una persona è rappresentata da:

    […]

TIPI DI DATO STRUTTURATI [2]

- COSTRUTTORE DI ARRAY [1]
  - Vettore di elementi fra loro omogenei per tipo
  - Gli elementi vengono allocati in memoria in sequenza
  - Ogni elemento è accessibile attraverso un indice che indica la posizione nell'array
  - L'indice deve essere di tipo int (o char)
  - Il primo elemento ha indice 0 (quindi da 0 a N-1)
  - Possibile creare array di array (matrici)
  - Es:

```
int colori[20];
float temperature[2];
float matrice[3][3];
colori[0] = 3;
```

TIPI DI DATO STRUTTURATI [3]

- COSTRUTTORE DI ARRAY [2]
  - La dimensione (fissa) deve essere una costante intera
  - Posso utilizzarlo per creare nuovi tipi:

    typedef *Tipo_esistente Nuovo_Tipo*[*Numero_elem*];
  - La stringa di caratteri è un (particolare) array di char (zero ended - \0)
  - Non è consentito fare assegnamenti di interi array
  - Es:

```
char string[80];

typedef int ArrayDiVenti[20];
ArrayDiVenti array1, array2;
array1=array2;  // assegnamento scorretto
array1[0]=array2[0];
array1[1]=array2[1];
```

TIPI DI DATO STRUTTURATI [4]

- COSTRUTTORE STRUCT
  - Permette di definire strutture aggregando tipi fra loro non omogenei
  - Gli elementi che costituiscono una struttura vengono detti campi
  - Per accedere ad un campo di una struttura viene utilizzato il punto (dot notation)
  - E' possibile, con le strutture, fare operazioni che le coinvolgano nel complesso
  - Es:

```
typedef struct
        {
            int giorno;
            int mese;
            int anno;
        } Data;
```

TIPI DI DATO STRUTTURATI [5]

- ES: STRUTTURE PER UN PROGRAMMA DI GESTIONE [1]

```
[…]

typedef char stringa[80];
typedef struct
        {
            int giorno;
            int mese;
            int anno;
        } Data;
typedef struct
        {
            stringa destinatario;
            Data data_emissione;
            int importo;
        } Fattura;
```

TIPI DI DATO STRUTTURATI [6]

- ES: STRUTTURE PER UN PROGRAMMA DI GESTIONE [2]

```
[…]

Data data1;
Fattura fattura1, fattura2;

data1.giorno=7;
data1.mese=13;
data1.anno=1964;

fattura1.destinatario="pippo";
fattura1.data_emissione=data1;
fattura1.importo=199999999;

fattura2=fattura1;
fatura2.data_emissione.giorno=5;

[…]
```

INPUT ED OUTPUT [1]

- ISTRUZIONI DI I/O
  - Non esistono istruzioni built-in di I/O
  - Si utilizzano funzioni (sottoprogrammi) di libreria
  - Standard Library
    ```
    #include <stdio.h>
    ```
  - Principali funzioni di libreria per l'I/O formattato sono:
    - printf() -> stdout
    - scanf() <- stdin
  - Il formato di tali istruzioni è:
    ```
    printf(stringa di controllo, parametri);
    scanf(stringa di controllo, parametri);
    ```

INPUT ED OUTPUT [2]

- PRINTF(*STRINGA DI CONTROLLO, PARAMETRI*) [1]
  - La stringa di controllo definisce il formato di stampa, il numero e tipo dei parametri da stampare
  - Posso avere caratteri di controllo:
    - \n : a capo
    - \t : carattere di tabulazione
    - …
  - Posso avere caratteri di conversione per i parametri:
    - %d <-> parametro di tipo int
    - %c <-> parametro di tipo char
    - %f <-> parametro di tipo float
    - %s <-> parametro stringa di caratteri

INPUT ED OUTPUT [3]

- PRINTF(*STRINGA DI CONTROLLO, PARAMETRI*) [2]
  - Es1:

```
#include <stdio.h>;
[…]
char cognome[20], nome[20];
cognome="Matteucci";
nome="Matteo";
printf("%s\nil mio cognome è %s \ned il mio nome è %s,
            "Buon giorno",cognome, nome);
[…]
```

output: Buon giorno
il mio cognome è Matteucci
ed il mio nome Matteo

  - Es2:

```
printf("Il cerchio di raggio %f ha diametro %f",
             raggio, raggio*2);
```

output: Il cerchio di raggio 1.2 ha diametro 2.4

INPUT ED OUTPUT [4]

- SCANF(*STRINGA DI CONTROLLO, PARAMETRI*)
  - La stringa di controllo definisce il tipo ed il numero dei dati che verranno letti e memorizzati nelle variabili
  - Posso avere caratteri di conversione per i parametri:
    - %d <-> parametro di tipo int
    - %c <-> parametro di tipo char
    - %f <-> parametro di tipo float
    - %s <-> parametro stringa di caratteri
  - Es:

```
[…]
int i;
char c1,c2;
float x;
scanf("%c %c %f %d", &c1, &c2, &x, &i);
[…]
```

CD NOLEGGIO [1]

- DEFINIZIONE DI STRUTTURE DATI E PROGRAMMA MINIMO.
  - Si vuole gestire un piccolo noleggio di CD
  - Si vogliono memorizzare dati sui CD in nostro possesso (autore, titolo, anno, numero canzoni, titoli, importo noleggio, disponibilità, ecc. ecc.)
  - Si vogliono memorizzare i dati sui nostri clienti (nome, cognome, indirizzo, …)
  - Si vogliono memorizzare le informazioni sui noleggi (CD, noleggiatore, data, durata, importo, ecc. ecc.)
  - Il programma deve richiedere l'immissione di un CD, di un Cliente e la registrazione di un noleggio, quindi deve stampare a video una ricevuta che riporti gli estremi della transazione.

CD NOLEGGIO (CODICE) [1]

- DEFINIZIONE DI STRUTTURE DATI E PROGRAMMA MINIMO.

```
#include <stdio.h>;
#define MaxStringLength 50;
#define MaxNumClienti 20;
#define MaxNumCD 100;
#define MaxNumNoleggi 100;
main()
 {
     typedef char stringa[MaxStringLength];
     typedef enum {true false} boolean;
     typedef enum {lun, mar, mer, gio, ven, sab, dom} n_giorno;
     typedef enum {gen, feb, mar, apr, […], nov, dic} n_mese;
     typedef int n_anno;
     typedef struct
               {
                   n_giorno giorno;
                   n_mese mese;
                   n_anno anno;
               } data;
```

CD NOLEGGIO (CODICE) [2]

```
typedef struct
    {
        int codice;
        string autore;
        string titolo;
        int anno;
        int numero_canzoni;
        string titoli[25];
        int importo_noleggio;
        boolean disponibilità;
    } CD;
typedef struct
    {
        string via;
        int numero_civico;
        string città;
        int CAP;
        string provincia;
    } address;
```

CD NOLEGGIO (CODICE) [3]

```
typedef struct
    {
        int codice;
        string nome;
        string cognome;
        address indirizzo;
    } cliente;
typedef struct
    {
        int codice_noleggio;
        int codice_CD;
        int codice_Cliente;
        data data_noleggio;
        data data_restituzione;
        int importo;
        } noleggio;

/*  Fine della definizione nuovi tipi e strutture dati
    inizio la dichiarazione delle variabili */
```

CD NOLEGGIO (CODICE) [4]

```
CD Elenco_CD[MaxNumCD];
cliente Elenco_Clienti[MaxNumClienti];
noleggio Elenco_Noleggi[MaxNumNoleggi];

/*  suppongo di aver già inserito i nomi dei clienti e i
    CD nell'archivio, inoltre penso di aver già effettuato
    Un certo numero di noleggi e di farne uno nuovo */

int numero_att_clienti;
int numero_att_CD;
int numero_att_noleggi=43;
int codice_att_cliente;
int codice_att_CD;
[…]

printf("\nInserisci il codice del noleggiatore: ");
scanf("%d", &codice_att_cliente);
printf("\nInserisci il codice del CD: ");
scanf("%d", &codice_att_CD);
```

CD NOLEGGIO (CODICE) [5]

```
// Inizio operazioni di registrazione
numero_att_noleggi = numero_att_noleggi+1; // incremento
Elenco_Noleggi[numero_att_noleggi].codice_cliente =
                                        codice_att_cliente;
Elenco_Noleggi[numero_att_noleggi].codice_cd =
                                        codice_att_CD;
Elenco_Noleggi[numero_att_noleggi].codice_noleggio =
                                        numero_att_noleggi;
[…]
// Stampo a video lo scontrino
printf("Il CD %d è stato noleggiato da %s", codice_att_CD,
              Elenco_clienti[codice_att_cliente].cognome);
[…]
}
```