

# Lezione 15

---

## Compendio C/C++

# Contenuto lezione

---

- Libreria standard
- Input/Output in C
- Macro e costanti
- Dichiarazioni e typedef
- Gestione memoria dinamica in C

- Il solo insieme di istruzioni di un linguaggio di programmazione come il C/C++ è sufficiente a scrivere un qualsiasi programma in grado per lo meno di interagire con l'utente?
  - Decisamente no
- Gli oggetti *cin* e *cout* erano oggetti predefiniti del linguaggio?

# Librerie 2/2

---

- No, sono oggetti appartenenti ad una **libreria**
- Una libreria è una raccolta di funzioni ed oggetti che permettono di effettuare determinati insiemi di operazioni
  - Esistono ad esempio librerie per l'ingresso/uscita, librerie matematiche, librerie grafiche e così via

# Libreria standard 1/2

---

- Sia nel linguaggio C che nel linguaggio C++ è prevista la disponibilità di una **libreria standard**
- La libreria standard del C++ è più o meno un sovrainsieme di quella del C
- Sia la libreria standard del C che quella del C++ sono costituite da molti moduli, ciascuno dei quali è praticamente una libreria a se stante, che fornisce funzioni ed oggetti per un determinato scopo
- Per utilizzare ciascun modulo è tipicamente necessario includere un ben determinato *header file*

# Libreria standard 2/2

---

- Alcuni moduli di base della libreria standard per il C ed il C++ sono ad esempio:

Libreria	Header C	Header C++
Matematica	<code>math.h</code>	<code>cmath</code>
Ingresso/Uscita	<code>stdio.h</code>	<code>iostream</code>
Limiti numerici	<code>limits.h</code>	<code>limits</code>

# Uso moduli C in C++

---

- Per utilizzare i moduli della libreria standard C++ a comune col C è bene includere degli header file il cui nome si ottiene, a partire dal nome del corrispondente header file per il C, eliminando il suffisso `.h` ed aggiungendo una `c` all'inizio del nome
  - Es.: la libreria matematica è presentata nell'header file `math.h` in C, mentre in C++ è presentata nell'header file `cmath`
  - Volendo, anche in C++ si possono includere gli header file originali, ma è una pratica sconsigliata
- Nel caso del C++, i nomi delle funzioni e degli oggetti di queste librerie sono definiti nello spazio dei nomi *std*
- In pratica, per usarli, bisogna aggiungere sempre la direttiva:  
`using namespace std ;`

# Ingresso/uscita formattato in C

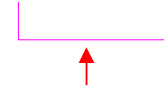
---

- Diversamente dal C++, in C l'Input/Output formattato è realizzato mediante funzioni di libreria presentate in `<stdio.h>`
  - `<cstdio>` se volete utilizzare tali funzioni in C++
- Tra le funzioni principali:
  - `printf`: output formattato su *stdout*
  - `scanf`: input formattato da *stdin*



# printf 1/2

```
void printf(const char format[], ... ) ;
```



Lista valori da stampare

- La stringa `format` può contenere due tipi di oggetti:
  - Caratteri ordinari (incluso quelli speciali), copiati sullo *stdout*
  - *Specifiche di conversione*, che controllano la traduzione in caratteri del valore dei successivi argomenti
    - Una per ogni successivo valore da stampare

# printf 2/2

---

- Esempio:  

```
int a = 15; double b = 16.5 ;  
printf("Il valore di a è %d, quello di b è %g\n", a, b) ;
```
- Equivale a  

```
cout<<"Il valore di a è "<<a  
  <<" , quello di b è "<<b<<endl;
```
- Si sono utilizzate due delle seguenti specifiche di conversione:
  - %d Numero intero, stampare in notazione decimale
  - %g Numero reale, stampare in notazione decimale
  - %c Carattere, tipicamente codifica ASCII
  - %s Stringa, tipicamente codifica ASCII

# scanf 1/2

```
void scanf(const char format[], &<nome_variabile> ) ;
```



Variabile da modificare

- In questa forma semplificata, la stringa **format** può contenere solo una specifica di conversione
  - Tale specifica controlla l'interpretazione da dare ai caratteri letti da *stdin* per determinare il valore da memorizzare nella variabile passata come secondo argomento

- Esempio:  

```
int a ;  
scanf("%d", &a) ; // equivale a cin>>a ;
```
- Si è utilizzata una delle seguenti specifiche di conversione:
  - %d Numero intero, in notazione decimale, da memorizzare in un **int**
  - %lg Numero reale, in notazione decimale, da memorizzare in un **double**
  - %c Carattere, tipicamente codifica ASCII, da memorizzare in un **char**
  - %s Stringa (lo spazio è un separatore), tipicamente codifica ASCII, da memorizzare in un **char []**

- Che succede se ci si sbaglia con le specifiche di conversione?
  - Errore logico
  - Errore di gestione della memoria
    - Meno pericoloso nella `printf`
    - Estremamente dannoso nella `scanf`: corruzione della memoria
- Esempio:

```
char a ;  
scanf ("%d", &a) ; // corruzione della memoria
```
- Altro tipico errore molto pericoloso:

```
int a ;  
scanf ("%d", a) ; // corruzione della memoria
```

# File ed IO non formattato in C

---

- In questo corso non vedremo l'uso dei file e l'Input/Output non formattato in C

# Direttiva `#define`

---

- Solo negli ultimi standard del C è stato introdotto il qualificatore `const`
- Per le costanti in C si usa ancora spesso la direttiva `#define`
- Esempi:  
`#define a 5`  
`#define b2 5.5`
- E' una direttiva C/C++ per il preprocessore
- Comporta una sostituzione **testuale** del simbolo con qualsiasi sequenza di caratteri lo segua, prima della compilazione
  - Nessuna dichiarazione/controllo di tipo
    - Il simbolo sparisce **prima** della compilazione
  - Può essere utilizzata anche per sostituzioni più complesse

# Tipo `struct` ed `enum` in C

---

- Anche in C si dispone dei tipi `struct` ed `enum`
- Però, data la dichiarazione di due tipi:

```
struct <nome_tipo_struct> { ... } ;  
enum <nome_tipo_enum> { ... } ;
```

a differenza del C++, in C la definizione di oggetti dei due tipi va fatta ripetendo ogni volta rispettivamente `struct` ed `enum`:

```
struct <nome_tipo_struct> <nome_variabile1> ;  
enum <nome_tipo_enum> <nome_variabile2> ;
```



# typedef 1/2

- Sia in C che in C++ si possono definire dei sinonimi di tipi primitivi, oppure di tipi precedentemente dichiarati.
  - Si fa mediante le dichiarazioni di nomi **typedef**
- Esempi:

**Nuovo nome (sinonimo) per il tipo**

```
typedef unsigned int u_int ;  
u_int a ; // equivalente a unsigned int a ;
```

```
typedef struct persona Persona ;
```

```
Persona p ;
```

**Solo in C vanno aggiunti  
struct ed enum**

```
typedef enum colore colore_t ;
```

```
colore_t c ;
```

# typedef 2/2

---

- Sia in C che in C++ le dichiarazioni typedef possono aiutare tantissimo a **migliorare la leggibilità** dei programmi
  - Permettono di evitare di dover ripetere in più punti dichiarazioni molto complesse
  - Permettono di sostituire nel programma nomi di tipo di basso di livello con nomi di tipo significativi nel dominio del problema

# Allocazione array dinamici in C

---

- Mediante funzione di libreria `malloc`
  - presentata in `<stdlib.h>` (`<cstdlib>` se si vuole utilizzarla in C++)
  - prende in ingresso la dimensione, in byte, dell'oggetto da allocare
  - ritorna l'indirizzo dell'oggetto, oppure 0 in caso di fallimento (NULL in C)
- Allocazione di un array dinamico:

```
<nome_tipo> * <identificatore> =  
    malloc(<num_elementi> * sizeof(<nome_tipo>)) ;
```

# Deallocazione in C

---

- Mediante funzione di libreria **free**
  - presentata in `<stdlib.h>` (`<cstdlib>` se si vuole usarla in C++)
  - prende in ingresso l'indirizzo dell'oggetto da deallocare
- Deallocazione di un array dinamico:

```
free (<indirizzo_array>) ;
```