

# Laboratorio II – 1° modulo

## Lezione 1

### Introduzione al C/C++

# Introduzione al C/C++

# Contenuto del corso di calcolo

---

- Questo corso ha lo scopo di insegnare alcuni strumenti di **programmazione** e di **calcolo** per l'**analisi statistica** dei dati
- Svolgeremo con il calcolatore alcuni esercizi proposti durante le lezioni frontali di statistica
- Utilizzeremo i seguenti strumenti:
  - Il linguaggio **C++**, per creare programmi
  - Il **framework ROOT** (sviluppato al CERN per l'analisi dati)
- Il corso è molto pratico, pertanto buona parte della lezione sarà dedicata allo svolgimento (da parte vostra) di esercizi

**Attenzione:** l'esame prevede una parte pratica in laboratorio e pertanto più vi esercitate durante la lezione e maggiori possibilità avrete di far bene durante l'esame

# Indice

---

- Introduzione
- Variabili e tipi predefiniti
  - L'operazione di cast tra tipi
- Operatori matematici ed operatori logici
- Le strutture di controllo
  - if ... else
  - for
  - while
- Le direttive al preprocessore
  - `#include`
  - `#define`
- Le funzioni
- Esercizi
- Alcuni comandi Linux
- *Flag* di compilazione
- Referenze



# Cosa è il C++

Il C per costruzione fornisce costrutti che si mappano in maniera efficiente con le tipiche “istruzioni macchina”, quindi ha trovato un uso duraturo in applicazioni che erano state precedentemente codificate in linguaggio *assembly*, inclusi i sistemi operativi, nonché vari software applicativi per computer che vanno dai *supercomputer* ai sistemi *embedded* (i.e. sistemi ad uso dedicato, e.g. biglietterie automatiche delle ferrovie dello stato, centralina delle moderne auto, ecc...). Per questo motivo la sua esecuzione è tipicamente più veloce, rispetto ad altri linguaggi

Il **C++** (vi ricordo che in C l'operatore **++** ha l'effetto di incrementare la variabile che lo precede) è una **estensione del C** nella quale vengono forniti nuovi elementi sintattici in maniera tale che il programmatore, se lo vuole, può implementare programmi secondo il paradigma della **programmazione orientata agli oggetti**

Versioni standardizzate del C++: **C++98** (useremo noi) → C++03 → **C++11** (accenni) → C++14 → C++17

Il codice (o istruzioni) macchina è un programma per computer scritto in linguaggio macchina, cioè un linguaggio che può essere eseguito direttamente dalla Central Processing Unit (CPU) del computer

```

MONITOR FOR 6802 1.4      9-14-80  TSC ASSEMBLER PAGE 2
ORG ROM+$0000 BEGIN MONITOR
START LDS #STACK

*****+
* FUNCTION: INITA - Initialize ACIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A
*****+
RESETA EQU %00010011
CTLREG EQU %00010001
INITA LDA A #RESETA |RESET ACIA
STA A ACIA
LDA A #CTLREG |SET 8 BITS AND 2 STOP
STA A ACIA
JMP SIGNON |GO TO START OF MONITOR

*****+
* FUNCTION: INCH - Input character
* INPUT: none
* OUTPUT: char in acc A
* DESTROYS: acc A
* CALLS: none
* DESCRIPTION: Gets 1 character from terminal
*****+
INCH LDA A ACIA |GET STATUS
ASR A |SHIFT RDRE FLAG INTO CARRY
BCC INCH |RECEIVE NOT READY
LDA A ACIA+1 |GET CHAR
AND A #$7F |MASK PARITY
JMP OUTCH |ECHO & RTS

*****+
* FUNCTION: INHEX - INPUT HEX DIGIT
* INPUT: none
* OUTPUT: Digit in acc A
* CALLS: INCH
* DESTROYS: acc A
* Returns to monitor if not HEX input
*****+
INHEX BSR INCH |GET A CHAR
CMP A #0 |ZERO
BMI HEXERR |NOT HEX
CMP A #'9 |NINE
BLE HEXRTS |GOOD HEX
CMP A #'A |TEN
BMI HEXERR |NOT HEX
C02C 81 46 |NOT HEX
C02E 28 05 |NOT HEX
C030 80 07 |NOT HEX
C032 84 0F |NOT HEX
SUB A #7 |FIX A-F
HEXRITS AND A #$0F |CONVERT ASCII TO DIGIT
RTS
C034 39 |RTS

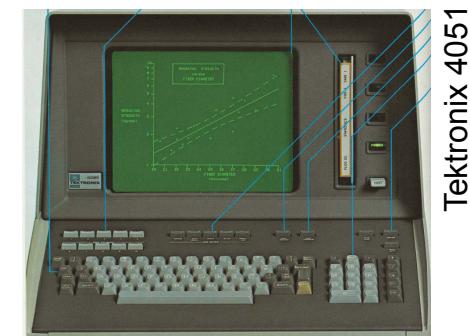
*****+
HEXERR JMP CTRL |RETURN TO CONTROL LOOP

```

**Codice macchina**

**Motorola MC6800 Assembly**

**Codice assembly**



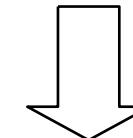
Tektronix 4051

# Sorgente ed eseguibile

---

È necessario avere ben presente la distinzione tra:

- Codice sorgente: versione “leggibile dagli esseri umani”
- Codice eseguibile: versione che il computer può interpretare ed eseguire (scritta in istruzioni macchina)



Il compito di trasformare il primo nel secondo è assolto dal compilatore, nel nostro caso `c++` (`o g++`)

- Il compilatore esegue il controllo sintattico: `itn` invece di `int` da errore
- Il compilatore esegue il controllo grammaticale: `int i = "hello";` da errore
- Il compilatore non esegue il controllo semantico (i.e. logico)

Per scrivere il codice sorgente si usa un editor di testo (e.g. `gedit`, `emacs`, `vim`, ... per progetti complessi `eclipse`)

Per compilare il codice sorgente il comando e`:

`c++ -o nome_esegibile nome_sorgente`

# Il mio primo programma C++

Commenti ignorati dal compilatore, molto utili per il programmatore. Altro metodo, usare //, ma vale solo per una riga

```
/*
c++ -o esempio01 esempio01.cpp
*/
#include <iostream>
int main()
{
    std::cout << "This is my first C++ program.\n";
    return 0;
}
```

Termina il programma e **ritorna** il valore 0 (intero) al processo che ha invocato il programma stesso (tipicamente il sistema operativo)

Inclusione dell'**header** della libreria **iostream** (input/output stream): contiene le funzioni per leggere e scrivere su terminale

Dichiarazione della funzione principale, **obbligatoria**. Ogni programma inizia chiamando **main**. Preceduto da dichiarazione del tipo di valore ritornato (**int**)

- **std::cout** → chiamata al **metodo** (i.e. funzione) cout (console output) contenuta nella **classe** delle librerie std (standard) del C++
- **<<** → **operatore di output**
- **"This..."** → **stringa** di testo
- **;** → tutte le istruzioni **terminate** dal punto e virgola

# Compiliamolo e eseguiamolo

Compiliamo ed avviamo da terminale, così:

**Compilazione**: avviene con il comando `c++` (o `g++`). Il parametro `-o` specifica il nome del file eseguibile che deve essere creato dal processo di compilazione

```
studente@lezi_fislab02:~$ c++ -o esempio01 esempio01.cpp
studente@lezi_fislab02:~$ ./esempio01
This is my first C++ program.
studente@lezi_fislab02:~$ █
```

**Esecuzione**: una volta compilato il programma può essere eseguito richiamando il nome dell'eseguibile preceduto da `./`

# Un programma più completo...

Provate il seguente programma:

```
#include <iostream>

int main()
{
    int num;

    std::cout << "Inserisci un numero: "
              << std::endl;
    std::cin >> num;
    std::cout << "Il numero inserito e': "
              << num << std::endl;
    return 0;
}
```

- Operatore di input `>>` acquisire da terminale
- Funzione `endl` per andare a capo
- Le istruzioni possono essere suddivise su più righe di codice
- Ogni istruzione deve sempre terminare con ;
- Porzioni di codice contigue comprese tra due parentesi graffe `{ ... }` prendono il nome di **scope**
- La variabile `num` può essere usata solo nello scope in cui è stata dichiarata

# Le variabili

Le quantità di interesse si gestiscono con le variabili

```
int num1 = 0;  
int num2 = 3;  
int somma = num1 + num2;  
std::cout << "Somma: "  
    << somma << std::endl;  
  
float razionale1 = 3.1416;  
double razionale2 = 1.4142;  
  
char lettera = 'a';  
  
bool condizione = true;  
  
int vettore [10];  
for (int i = 0; i < 10; i++)  
{  
    vettore[i] = i*2;  
}  
  
int valori [] = {2,4,6,8};
```

- Le variabili possono essere di diversi **tipi** (char, int, float, bool)
- Una variabile è **l'istanza di un tipo** (num1 è una istanza di int)
- Le normali operazioni matematiche sono definite per i diversi tipi
- Si possono creare vettori di un tipo di variabili, chiamati **array**
- La lista degli oggetti di un array occupa una zona contigua della memoria (**N.B:** l'indice del primo elemento è 0)
- La dimensione di un array è una costante
- Gli elementi di un array possono essere inizializzati in fase di definizione (in questo caso non è necessario dichiarare esplicitamente la dimensione)

# I tipi delle variabili

char

Type	Bit Width	Common Range
char	8	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127
int	16	-32,768 to 32,767
unsigned int	16	0 to 65,535
signed int	16	-32,768 to 32,767
short int	16	same as int
unsigned short int	16	same as unsigned int
signed short int	16	same as short int
long int	32	-2,147,483,648 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295
signed long int	32	-2,147,483,648 to 2,147,483,647
float	32	3.4E-38 to 3.4E+38
double	64	1.7E-308 to 1.7E+308
long double	80	3.4E-4932 to 1.1E+4932
bool	N/A	true or false
wchar_t	16	0 to 65,535

bool

# L'operatore `sizeof`

L'operatore `sizeof(variabile)` restituisce il numero di bytes (1 byte = 8 bit) necessari a memorizzare la variabile indicata tra parentesi

```
char lettera = 'a';
int Nbytes = sizeof(lettera);
std::cout << Nbytes << std::endl;
```

Quale numero verrà stampato a schermo?

È possibile utilizzare anche la sintassi `sizeof(tipo)`

```
int valori [] = {2, 4, 6, 8, 10};
int N_elementi = sizeof(valori)/sizeof(int);
std::cout << N_elementi << std::endl;
```

In questo modo ricavo il numero di elementi di un array

# Cast dei tipi (standard C)

**Attenzione:** durante le operazioni matematiche è necessario prestare attenzione alla **conversione** tra i tipi delle variabili

- in una operazione tra tipi diversi, tutti gli **operandi vengono convertiti al tipo “più grande”** prima di compiere l'operazione
- Potete utilizzare, se necessario, il **cast** per convertire un tipo di variabile in un altro tipo mentre eseguite un'operazione. La sintassi è: **(tipo) variabile**

```
int x = 1;  
x / 2; // restituirà 0 (divisione tra interi)  
((float)x) / 2; // restituirà 0.5 (conversione di x a float prima  
di eseguire la divisione)  
x / 2.0; // restituirà 0.5 perché 2.0 è float
```

# Cast dei tipi (C++ *oriented*)

**Attenzione:** durante le operazioni matematiche è necessario prestare attenzione alla **conversione** tra i tipi delle variabili

- in una operazione tra tipi diversi, tutti gli **operandi vengono convertiti al tipo “più grande”** prima di compiere l'operazione
- Potete utilizzare, se necessario, il **cast** per convertire un tipo di variabile in un altro tipo mentre eseguite un'operazione. La sintassi è: `static_cast<tipo>(variabile)`

```
int x = 1;  
x / 2; // restituirà 0 (divisione tra interi)  
static_cast<float>(x) / 2; // restituirà 0.5 (converte x a float  
prima di eseguire la divisione)  
x / 2.0; // restituirà 0.5 perché 2.0 è float
```

# Gli operatori

Avete a disposizione diversi operatori definiti in C++:

- Operatore di **assegnamento** (=)

E.g. `b = 5;` Il valore dell'espressione a destra dell' = viene  
`a = 2+b;` assegnato alla variabile che sta a sinistra

- Operatori **aritmetici**: +, -, \*, /, %

+ addizione

- sottrazione

\* moltiplicazione

/ divisione

% resto della divisione tra interi

- Operatori di **assegnamento composti** +=, -=, \*=, /=, %=

E.g. `a += 2;` equivale ad `a = a+2;`

- Operatori di **incremento e decremento**: ++, --

E.g. `i++;` equivale a `i = i+1` (oppure `i += 1`)

# Gli operatori

---

Gli operatori logici e relazionali confrontano i valori delle espressioni a destra e a sinistra dell'operatore e restituiscono vero/falso

- Operatori relazionali:

`==` (uguale), `!=` (diverso), `>`, `<`, `>=`, `<=` (maggiore, minore)

E.g. `a = 5;`

`bool answer = (a==7);` la variabile `answer` conterrà `false`

- Operatori logici:

`&&` (AND)      `||` (OR)      `!` (NOT)

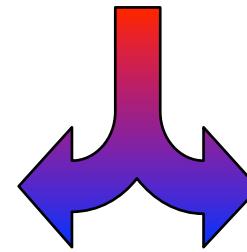
E.g. `! (6==5)` restituisce `true`    `! true` restituisce `false`

`(a==b) && (c != d)` restituisce `true` solo se è vero sia che `a` è uguale a `b`, sia che `c` è diverso da `d`

# Gli operatori

- Fate sempre attenzione alla precedenza tra operatori:

<b>Più alta</b>	<code>++ --</code>
	<code>/ * %</code>
<b>Più bassa</b>	<code>+ -</code>



<b>Più alta</b>	<code>!</code>
	<code>&gt; &gt;= &lt; &lt;=</code>
	<code>== !=</code>
	<code>&amp;&amp;</code>
<b>Più bassa</b>	<code>  </code>

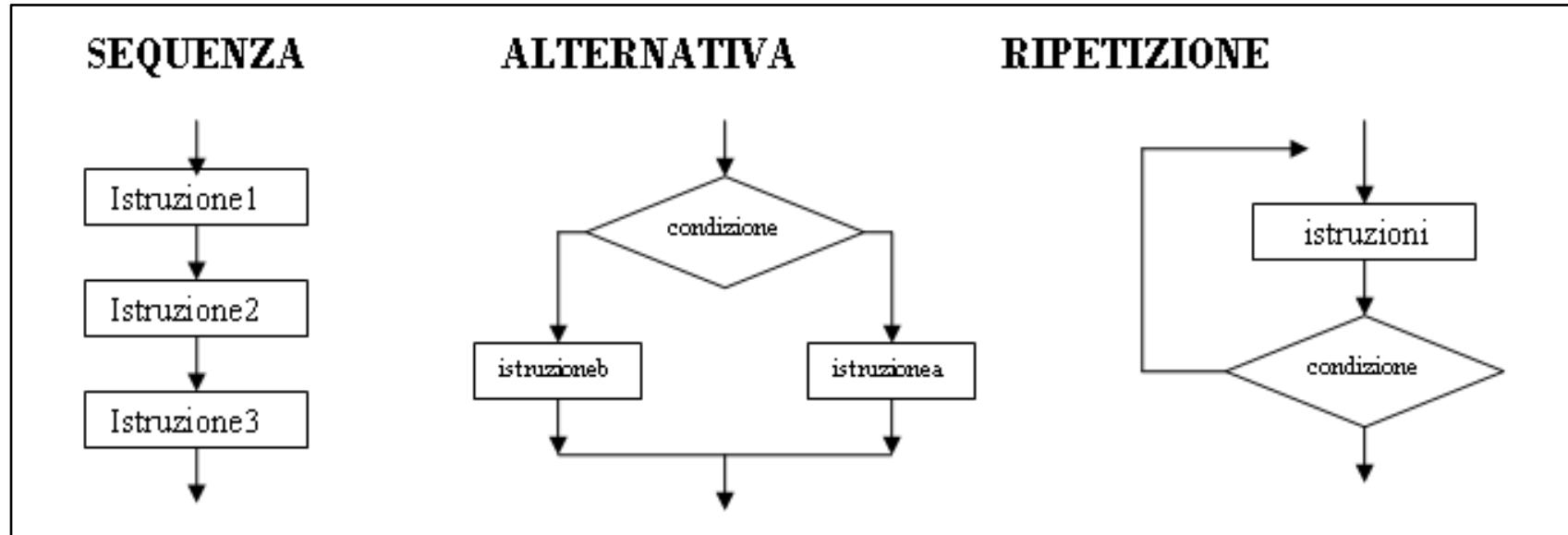
E.g. `a = 5 + 7 % 2;` equivale `a = 5 + ( 7 % 2 );`

- Se avete dubbi fate uso delle **parentesi** nelle espressioni matematiche

# Strutture di controllo

I programmi possono essere organizzati secondo diversi schemi a seconda della necessità. Ad esempio, oltre a semplici istruzioni in sequenza, vogliamo:

- implementare diverse istruzioni a seconda del verificarsi (o meno) di una certa condizione
- ripetere ciclicamente alcune operazioni variando uno o più parametri



A questo scopo il C/C++ fornisce **strutture di controllo** (**if, for, while**, ecc ...)

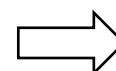
I blocchi di istruzioni all'interno delle strutture di controllo sono delimitati da parentesi `{ . . . }` (i.e. scope)

# Strutture condizionali: if - else

```
int numero;
std::cin >> numero;
if (numero > 10 || numero < -10)
{
    std::cout << "Hai inserito " << numero
        << " che e` un numero a due"
        << " o piu` cifre" << std::endl;
}
else
{
    std::cout << "Hai inserito " << numero
        << " che e` un numero ad una"
        << " sola cifra" << std::endl;
}
```

- if ed else si possono combinare per verificare anche più di due alternative

Se una struttura di controllo necessita di una sola istruzione, non è obbligatorio utilizzare le parentesi { . . . }



```
if (x > 0)
    std::cout << "x is positive";
else if (x < 0)
    std::cout << "x is negative";
else
    std::cout << "x is zero";
```

# Strutture iterative: ciclo for

```
int valori [] = {7,20,21,-4,22,34};  
int numArg = 6;  
  
for(int index = 0; index < numArg; index++)  
{  
    std::cout << index << "-esimo argomento: "  
        << valori[index] << std::endl;  
}
```

- Il ciclo viene impostato in tre parti:
  - Definendo `index`, che esiste soltanto nello scope del ciclo
  - Stabilendo la condizione di uscita (il ciclo viene eseguito fintanto che questa condizione è vera, ovvero `index < numArg`)
  - Dettando l'istruzione di incremento della variabile di controllo
    - E` inoltre possibile saltare all'indice successivo mediante l'istruzione `continue`, e.g. nel ciclo `for` si aggiunge l'istruzione `if (condizione) continue; // Se condizione == true riprende il ciclo for al valore di index successivo`
    - E` anche possibile fermare il ciclo prima che `index == numArg` mediante l'istruzione `break`, e.g. nel ciclo `for` si aggiunge l'istruzione `if (condizione) break; // Se condizione == true esce dal ciclo for`

# Strutture iterative: ciclo for

```
int valori [] = { 7, 20, 21, -4, 22, 34 };
int numArg = 6;

for(int index = 0; index < numArg; index++)
{
    if (valori[index] % 2 == 0)
    {
        // Azzera l'argomento e passa al valore di index successivo
        valori[index] = 0;
        continue;
    }
    else if (valori[index] < 0)
        // Esce dal ciclo for
        break;

    std::cout << index << "-esimo argomento dispari e positivo: "
          << valori[index] << std::endl;
}
```

# Strutture iterative: ciclo while

```
int valori [] = {7,20,21,-4,22,34};  
int numArg = 6;  
int index = 0;  
  
while (index < numArg)  
{  
    std::cout << index << "-esimo argomento: "  
        << valori[index] << std::endl;  
    index++;  
}
```

- Le istruzioni nello scope vengono eseguite finché `index < numArg`
- La variabile `index` esiste anche al di fuori del ciclo
- È possibile utilizzare le istruzioni `continue` e `break`

## Attenzione:

- la variabile di controllo va inizializzata prima dell'inizio del ciclo
- la variabile di controllo va incrementata all'interno del ciclo

# Strutture iterative: ciclo while

---

```
int valori [] = {7,20,21,-4,22,34};  
int numArg = 6;  
int index = 0;  
  
while (index < numArg)  
{  
    if (valori[index] % 2 == 0)  
    {  
        // Azzera l'argomento e passa al valore di index successivo  
        valori[index] = 0;  
        index++;  
        continue;  
    }  
    else if (valori[index] < 0)  
        // Esce dal ciclo while  
        break;  
  
    std::cout << index << "-esimo argomento dispari e positivo: "  
          << valori[index] << std::endl;  
    index++;  
}
```

# Operazioni matematiche

---

- Gli operatori e le funzioni matematiche non sono presenti nelle librerie standard
- È necessario includere la libreria **cmath**
- Per consultare la lista delle funzioni disponibili:  
[www.cplusplus.com/reference/cmath/](http://www.cplusplus.com/reference/cmath/)

```
#include <iostream>
#include <cmath>

int main()
{
    double due = 2;
    double radice_due = sqrt(2);
    std::cout << "Radice 2 = " << radice_due << std::endl;

    return 0;
}
```

# Direttive al preprocessore

In C++ (come in C), prima che il compilatore inizi a compilare, viene attivato un programma, detto **preprocessore**, che ricerca nel file sorgente speciali istruzioni, chiamate **direttive al preprocessore**

Una direttiva inizia sempre con il carattere **#** e occupa una sola riga (non ha un terminatore, in quanto finisce alla fine della riga)

Il preprocessore crea una copia del file sorgente (da far leggere al **compilatore**) e, ogni volta che incontra una direttiva, la esegue sostituendola con il risultato dell'operazione. Pertanto il preprocessore, eseguendo le direttive, non produce codice macchina, ma codice sorgente per il compilatore

Ogni file sorgente, dopo la trasformazione operata dal preprocessore, prende il nome di **translation unit**. Ogni *translation unit* viene poi compilata separatamente, con la creazione del corrispondente **file oggetto**, in codice binario. Spetta al **linker**, infine, collegare tutti i file oggetto, generando un unico programma eseguibile

**#include <nomefile>** oppure **#include "nomefile"** è una direttiva al preprocessore che serve per includere la libreria *nomefile*

Un'altra direttiva al preprocessore è **#define**

**#define identificatore valore**

- *identificatore*: nome simbolico
- *valore*: espressione qualsiasi

# Direttive al preprocessore

**Esempio:**

```
#define N 100
int main()
{
    ...
    for (unsigned int i = 0; i < N; i++)
    ...
```

Utile nel caso sia necessario introdurre nel codice dei **valori “hard-coded”** per ritrovarli immediatamente, la direttiva `#define` è infatti sempre posta **all'inizio del programma**

È possibile anche definire delle macro:

```
#define identificatore(argomenti) espressione
```

**Esempio:**

```
#define quadrato(x) (x) * (x)
int main()
{
    ...
    int a = quadrato(2+3); // Dopo il passaggio del preprocessore
                           // l'espressione diventa int a = (2+3) * (2+3);
    ...
```

# Le funzioni

Le funzioni sono utilissime per implementare blocchi di istruzioni che si ritiene di voler utilizzare in diversi punti del programma principale (o in programmi diversi)

```
#include <iostream>
#define START 4
double raddoppia(double input).
{
    return input*2;
}

int main()
{
    double valore_iniziale = START;
    double valore_finale = raddoppia(valore_iniziale);
    std::cout << valore_iniziale << " x 2 = "
          << valore_finale << std::endl;

    return 0;
}
```

- La funzione si definisce prima del programma principale (`main`)  
Può avere diverse variabili come input ed (al più) un solo output
- All'interno del programma principale viene chiamata secondo il prototipo dichiarato

# Le funzioni ed i prototipi

---

- Abbiamo detto che la funzione si definisce prima del `main`, perché così la funzione risulta già nota al compilatore quando la funzione viene chiamata nel `main`
- In realtà è sufficiente che al compilatore sia noto il nome della funzione ed il tipo dei suoi parametri, ovvero il **prototipo della funzione**  
`tipo nome_funzione(tipo par1, tipo par2, ...);`
- L'implementazione vera e propria della funzione può essere riportata anche in seguito (o in un file separato, da compilare insieme a quello contenente il `main`)
- L'uso dei prototipi è molto utile quando un programma contiene molte funzioni e con istruzioni molto lunghe
- Nel prototipo è possibile anche assegnare dei valori di *default* agli "ultimi" parametri così da poter essere omessi all'atto della chiamata della funzione nel `main`: e.g. `tipo nome_funzione(tipo par1, tipo par2, tipo par3 = val);`  
Nel `main`: `val_output = nome_funzione(val_par1, val_par2);`

# Le funzioni ed i prototipi

---

```
#include <iostream>
#define START 4
// Indicazione del prototipo
double raddoppia (double input);

int main()
{
    double valore_iniziale = START;
    double valore_finale = raddoppia(valore_iniziale);
    std::cout << valore_iniziale << " x 2 = "
                  << valore_finale << std::endl;

    return 0;
}

// Definizione della funzione
double raddoppia (double input)
{
    return input*2;
}
```

# Ordine

- Prestate attenzione alla **SINTASSI** ma soprattutto alla **SEMANTICA** del codice
- Per non commettere errori banali, scrivete il codice con ordine, adottando una serie di sane abitudini. Ad esempio:
  - **indentare** il codice: le istruzioni di uno stesso scope vengono allineate più a destra
  - **commentare** il codice (per se stessi e per gli altri)
  - scegliere un **modo coerente** di scrivere il codice (parentesi graffe a capo o non a capo, spaziature, ecc ...)
  - scegliere **nomi lunghi ed esplicativi** per le variabili
- Ricordatevi di **non** usare caratteri speciali (come le lettere accentate) nei nomi delle variabili
- Ricordatevi che il C++ fa distinzione tra caratteri maiuscoli e minuscoli (e.g. la variabile Num è diversa da num)

# Leggere i messaggi d'errore

Alcuni **errori di sintassi**:

- **Punto e virgola mancante:**

```
esempio01.cpp: In function 'int main()':  
esempio01.cpp:10:2: error: expected ';' before 'return'  
    return 0;  
    ^
```

Nome del file

Numerò della riga

Nome della funzione

Tipo di errore

- **Errore di dichiarazione:**

```
esempio01.cpp: In function 'int main()':  
esempio01.cpp:9:3: error: 'variabile' was  
not declared in this scope  
    variabile=1;
```

**Attenzione:** Un errore sintattico può generare altri errori in cascata, risolvete quindi un errore alla volta partendo da quello che il compilatore vi presenta per primo

**Consiglio:** se non capite il messaggio di errore, cercate in internet

# Esercizi

---

- **Esercizio 1:** Scrivere un programma che, letti due numeri interi da terminale, restituisca il loro rapporto
- **Esercizio 2:** Scrivere un programma che scrive a terminale la radice quadrata di 2, il cubo di 2 il seno di  $\pi/4$
- **Esercizio 3:** Scrivere un programma che chieda all'utente di inserire un intero a scelta tra 1 e 2, e restituisca a terminale il valore inserito, o un messaggio di errore in caso di inserimento di altri interi
- **Esercizio 4:** Scrivere un programma che richieda all'utente di inserire un numero intero e che sappia riconoscere se il numero è pari o dispari (utilizzare l'operatore `%`, implementando due funzioni che stampino a schermo messaggi diversi nei due casi)
  - Implementare nel `main` un ciclo infinito con l'istruzione `while (true)`. Uscite dal ciclo facendo uso dell'istruzione `break` se il numero inserito è negativo

# Esercizi

---

- **Esercizio 5:** Scrivere una funzione che calcoli il fattoriale di un numero intero non negativo
- **Esercizio 6:** Scrivere la funzione fattoriale in modo ricorsivo, cioè facendo in modo che la funzione che calcola il fattoriale chiami se stessa
- **Esercizio 7:** Scrivere un programma che chieda all'utente di inserire  $N=10$  numeri da tastiera e li memorizzi in un array. Implementare le funzioni:

```
double Media (float array[], int N);
```

```
double Varianza (float array[], int N);
```

e utilizzarle nel main per calcolare la media e la varianza dei numeri inseriti

# Alcuni comandi Linux

- `cd nome_directory` → cambia directory
- `mkdir nome_directory` → crea una directory
- `ls -latr` → mostra la lista dei file nella directory corrente
- `cp nome_file nome_directory` → copia un file in una directory
- `more nome_file` → mostra il contenuto di un file
- `pwd` → mostra la directory corrente
- È possibile usare le *wildcard* \*, ?, e [...] per esempio `ls -latr *.cpp` → mostra tutti i file con estensione .cpp
- `grep "testo" *.cpp` → cerca la stringa testo in tutti i file con estensione .cpp
- `tar -czvf nomefile.tar.gz *` → crea un archivio compresso di tutti i file nella directory
- `tar -xzvf nomefile.tar.gz` → decomprime l'archivio
- `ssh userID@computer.remoto` (e.g. `mario.rossi@lxplus.cern.ch`) → consente di collegarsi ad un computer remoto utilizzando la cifratura dei dati
- `scp *.cpp userID@computer.remoto:/dir/di/destinazione/` → copia tutti i file con estensione .cpp presenti sul computer locale, nella directory del computer remoto, utilizzando lo stesso protocollo di cifratura di ssh (nel caso in cui il sistema operativo del computer remoto sia Windows è necessario installare un client ssh, e.g. [www.bitvise.com/ssh-client-download](http://www.bitvise.com/ssh-client-download))
- `man comando_linux` oppure `elemento_sintattico_C/C++` → fornisce il manuale di alcuni comandi Linux oppure di alcuni elementi sintattici del C o del C++

# Flag di compilazione

---

E` possibile rendere più severo il controllo del compilatore abilitando dei *flag*. Alcuni di questi sono qui riportati:

**-ansi**: impone lo standard c90 per il C o c++98 per il C++. Nel caso fosse necessario richiedere uno standard più avanzato, come il c++11, è necessario usare il *flag*: **-std=c++11**

**-pedantic**: abilita tutti i *warning* richiesti dallo standard impostato con il *flag* **-std=**

**-Wall**: abilita tutti i *warning* di costruzione che sono facili da evitare (e.g. variabili usate non preventivamente inizializzate)

**-Wextra**: abilita *extra-warning* oltre a quelli di **-Wall**

Per usarli basta aggiungerli alla riga di comando per la compilazione, e.g. `c++ -o test test.cpp -pedantic`

Avere i *flag* abilitati può risultare utile per essere guidati nella pratica della buona programmazione. Inoltre possono risultare interessanti questi altri due *flag* speciali:

**-E**: genera solo la *translation unit*, e.g. `c++ -E -o name_preproc name.cpp`

**-S**: genera solo il codice assembly, e.g. `c++ -S -o name_assembly name.cpp`

Per maggiori dettagli sul significato dei *flag* guardare:

[gcc.gnu.org/onlinedocs/gcc/Option-Summary.html](http://gcc.gnu.org/onlinedocs/gcc/Option-Summary.html)

# Referenze

---

- “*C programming language*” B.W. Kernighan, D.M. Ritchie
- “*The C++ programming language*” Bjarne Stroustrup
- “*Thinking in C++*” Bruce Eckel