**INTRODUZIONE A C++**

La funzione **main** è il cuore del nostro proramma, ogni funzione verrà eseguita dal programma dall’alto verso il basso: int main()

{

}

Parlando della **struttura** di un programma:

* **#include <iostream>**: prende una libreria e tutto il contenuto vieni importato nel nostro programma
* **;** : bisogna usarlo alla fine di un commando (non i tutti i casi).
* **<<:** ci permettono di concatenare più stringhe/variabili.
* **Endl, /n:** ci permette di andare a capo, EX: cout<<”ciao mondo”<< **endl** <<”gabriele”;
* **Return 0:** dobbiamo sapere che ogni funzione deve ritornare un valore.

**INPUT DEGLI UTENTI**

Ci permette di prendere gli input scritti dagli utenti, e salvarli nelle variabili:

int numero;

cout<< “inserisci un numero: ”; **(10)**

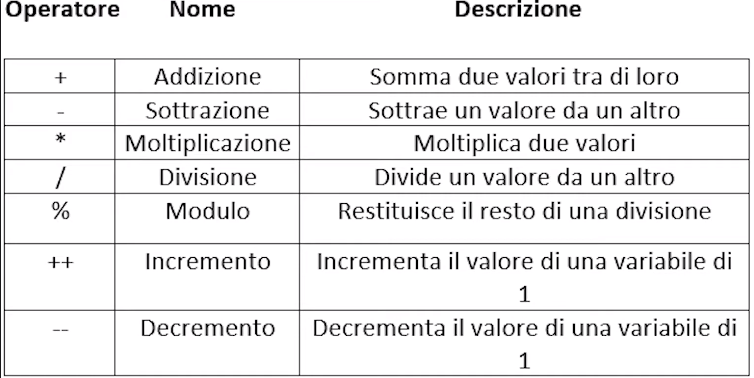
cin>> numero;

cout<< “il tuo numero è: “ << numero; 🡪 **il tuo numero e 10**

**GLI OPERATORI**

Sono usati per eseguire **determinate operazioni sulle variabili:**

* **operatori aritmetici ( + - / \*….):** possiamo ad esempio sommare numeri con numeri, numeri co variabili… se abbiamo risultati/numeri con la virgola, dobbiamo utilizzare la variabile **float.**

****

* **operatori di assegnazione:** molto importate da ricordare l’operatore += (idem tutti)

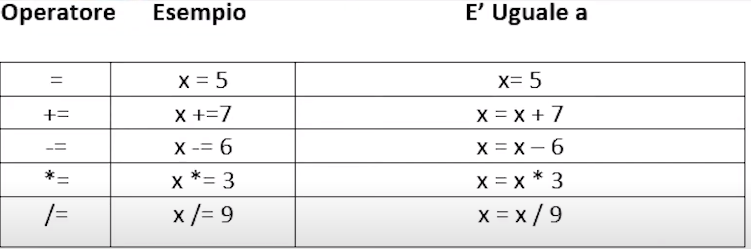
aggiungiamo 5 al valore x:

int x = 10;

x += 5;

cout << x;

**x = 15**

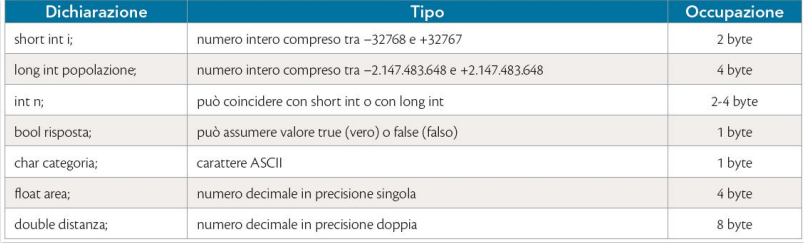
****

* **operatori da comparazione, operatori logici**

****

**VARIABILI**

All’interno di una variabile c’è un contenuto che può essere di vari tipi, insomma, è un **contenitore che salva dei dati:**



Int eta = 19; (se non diamo nessun valore, assumerà il valore di 0) int eta;

cout<< eta; eta = 20;

**19**  coul<< eta;

**20**

Int x = 3;

int y = 5;

int somma = x+y;

cout<<somma;

**8**

per ciascuna variabile, quindi:

* **assegniamo un nome**
* **il tipo di variabile**
* **capire se è di input o output**
* **inserire una descrizione**

Tramite le istruzioni si forniscono **dati di input**, e riceviamo **risultati in output.**

**IF (SE)**

Utilizzare se ci si trova a prendere una decisione, il programma poi dirà se la condizione è VERA o FALSA

**if**(numero>10) ---------🡪 se il numero è maggiore di 10 deve succedere qualcosa.

{

Cout<<”*esatto*”<<endl;

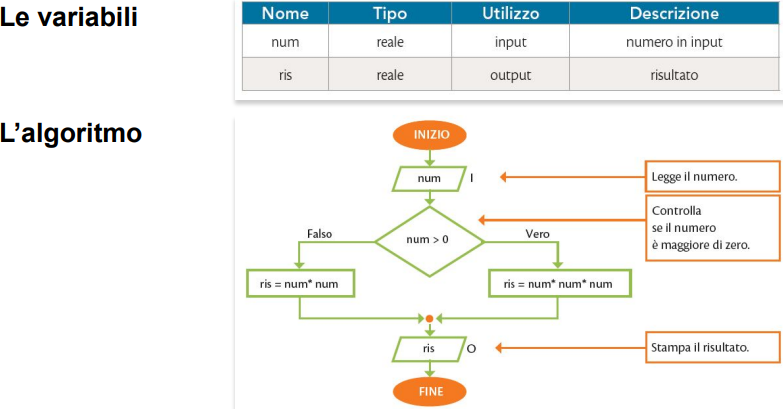
} **else if** (numero == 10) { -------------------🡪 o altrimenti…

Cout<<”*hai inserito il numero 10*”<<endl;

} **else** { -------------------🡪 o altrimenti… (dato che è l’ultimo non si usa if!!)

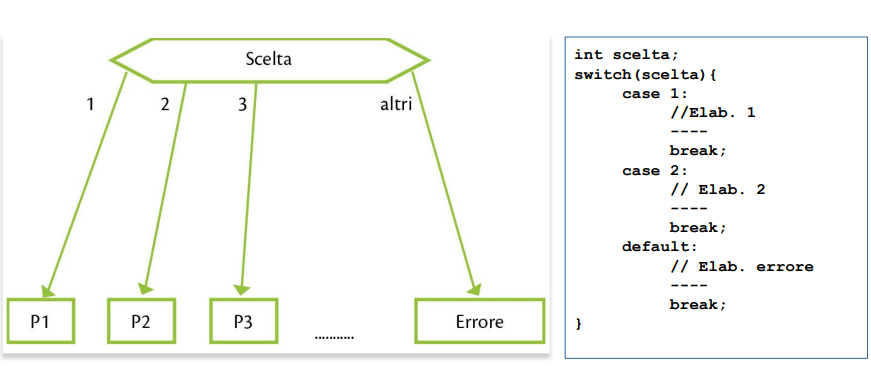
Cout<<”*sbagliato*” <<endl;

}

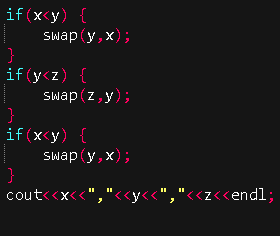
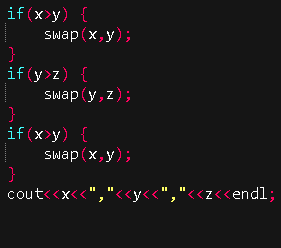
****

**LO SWITCHCASE**

viene chiamato anche selezione multipla, ***presenta + di 2 uscite***, ha la stessa funzione dell’IF ma può essere utilizzato in processi più lunghi rendendo il tutto più schematico. (ex: voglio sapere che giorno della settimana è, dovrei utilizzare 7 if…)



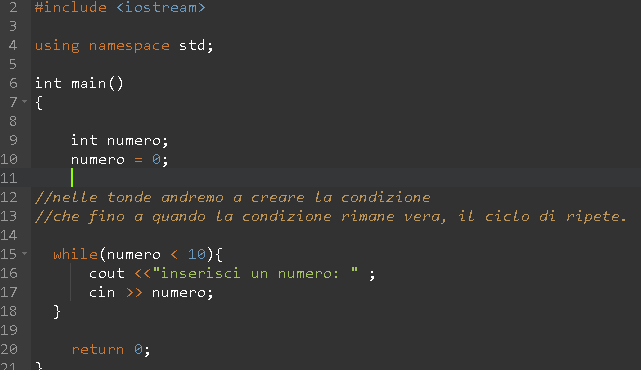
**NUMERI ORDINE CRESCENTE NUMERI ORDINE DECRESCENTE**

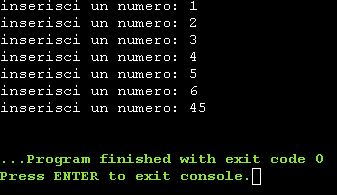
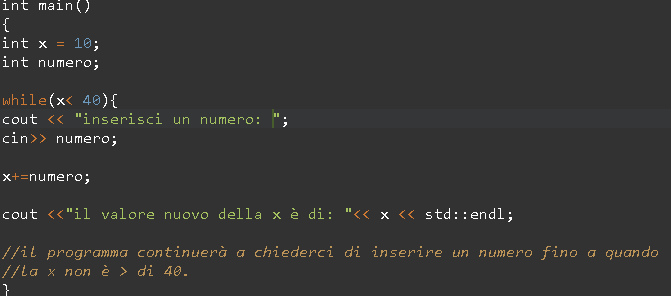


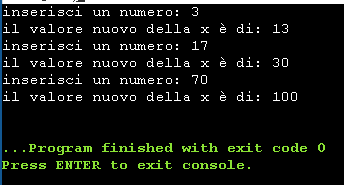
**CICLO WHILE / DO WHILE**

Sono dei blocchi di codici che contengono delle istruzioni, che a differenza dell’IF non vanno a dir faccio quello o quell’altro, ma **finché una condizione è verificata continua a fare la stessa cosa** (ex ho dei compiti per casa, quando avrò finito di farli, potrò fare altre attività), viene utilizzata molto la variabile **i** che sta per iteratori.

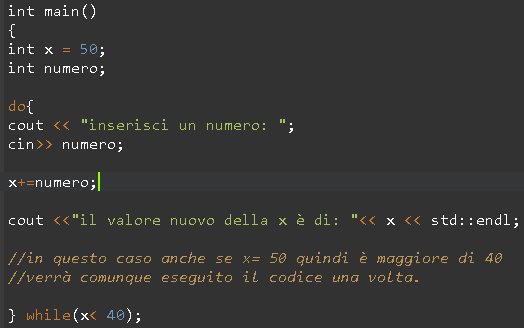
* **WHILE** :(mentre) una condizione è verificata, fa qualcosa:

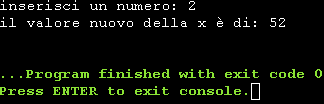


****

****

* **DO WHILE:** ci permette di eseguire una volta il codice all’interno del ciclo (anche se la condizione è vera)

****

****

**CICLO FOR**

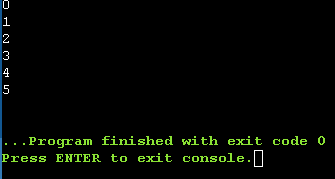
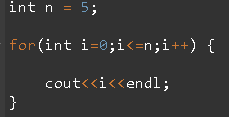
* **FOR:**  ciclo di ripetizione che usiamo quando quante volte sappiamo di volerlo ripetere.

ripete n volte delle istruzioni.

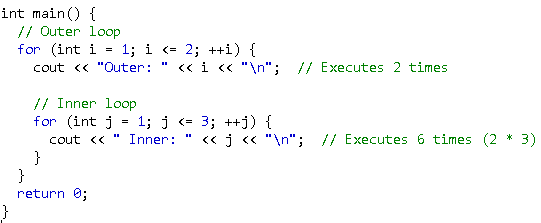
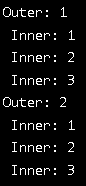
**i** parte da 0 e arriva a 9 ---🡪 10 turni/iterazioni (0,1,2,3,4,5,6,7,8,9) ---🡪 **10 ripetizioni**



Facciamo un esempio in cui io voglio stampare i ***primi 5 numeri naturali***:



È anche possibile posizionare un loop all'interno di un altro loop, è chiamato ***ciclo annidato***:

**BOOL/FLAG**

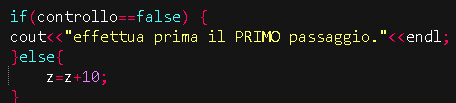
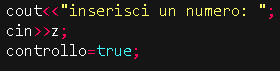
* **BOOL:** ha 2 valori 🡪 **TRUE, YES (0)**

**🡪 FALSE, NO (1)**

Ex: mettiamo che abbiamo un programma in cui compiere 2 PASSAGGI DISTINTI, se però NON abbiamo effettuato il primo passaggio, non possiamo effettuare il secondo.



**1 PASSAGGIO: 2 PASSAGGIO:**



Come possiamo osservare, se andiamo ad effettuare direttamente il SECONDO PASSAGGIO, il programma ci bloccherà, dato che appunto se il controllo è = FALSE, non ci farà andare avanti.

se andremo ad effettuare direttamente il PRIMO PASSAGGIO, il booleano diventerà controllo=TRUE. Così facendo il programma non ci ostacolerà più e potremo proseguire con gli altri PASSAGGI.

**GENERAZIONE NUMERI CASUALI**

Per permettere la generazione di numeri casuali all’interno del nostro programma, abbiamo bisogno di 3 strumenti fondamentali:

🡪raccolta di funzioni e macro per facilitare il codice C++

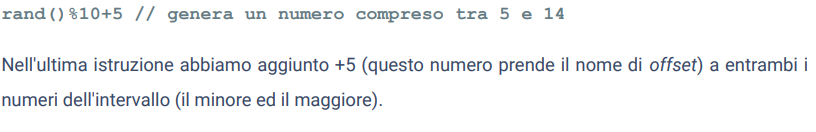
🡪…

🡪serve a inizializzare la funzione per la generazione di numeri casuali

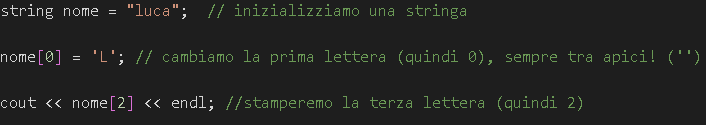
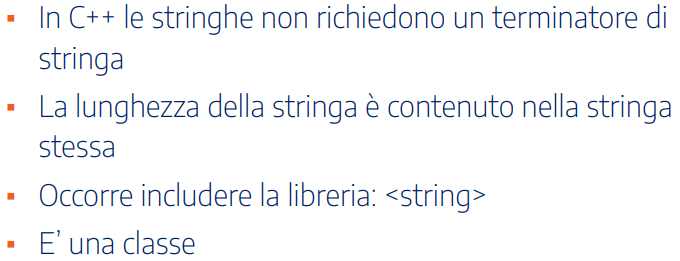
Se vogliamo decidere di generare un numero casuale da 1 a 100, ad una variabile (ex: numeroCasuale)



Se vogliamo generare numeri oltre 1, seguiamo questo schema:

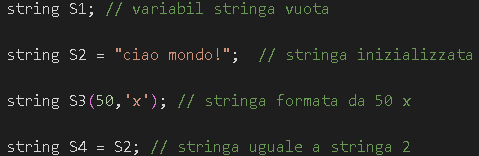


**STRINGHE (*Si inizia a contare da 0!!)***

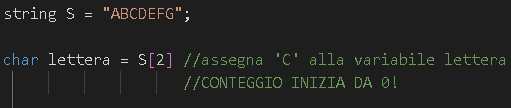
****

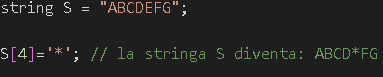


esempi di ***inizializzazione*** di una variabile **STRING**  IN C++**:**

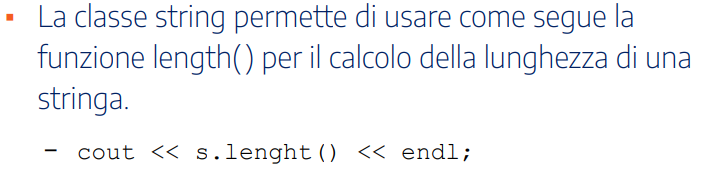


per accedere ai ***singoli caratteri*** in C++:



se volessimo modificare una lettera della nostra stringa:

Per vedere ***la lunghezza*** di una stringa in C++, tramite questo codice, il programma ci dirà la lunghezza della stringa in numeri:



al posto di .**lenght()**, si potrebbe utilizzare **.size(),** in questo esempio uscirà **3**

****

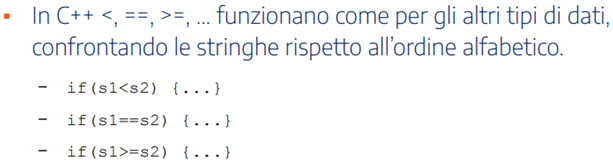
Per ***invertire*** una stringa in C++:

****

inoltre è necessario inserire una ***LIBRERIA:***

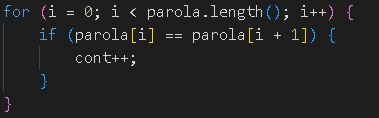


Parlando degli ***operatori di confronto*** tra stringhe:

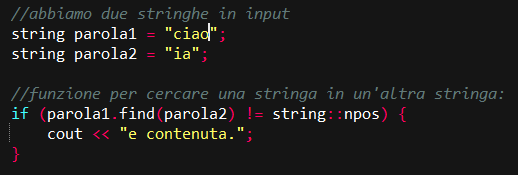




In questo esempio, dobbiamo dire in **output** quante ripetizioni ci sono, all’interno della stringa

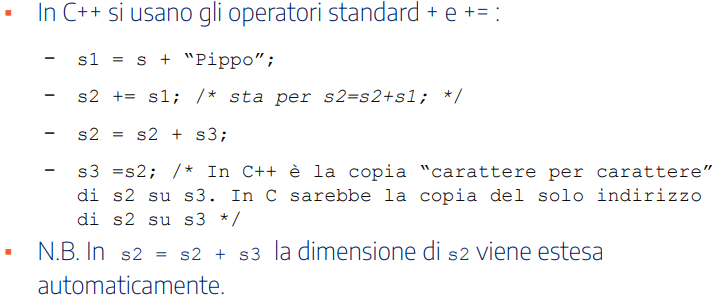


per ***cercare*** una stringa in un’altra stringa:

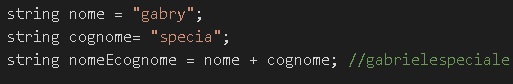


il messaggio ***String::npos*** dice se la stringa non viene trovata (si puo utilizzare == o !=)

Per ***concatenare*** delle stringhe in C++:



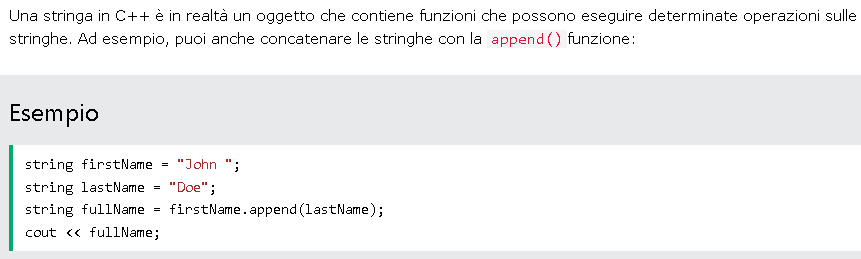
EX: Mettiamo che vogliamo unire 2 stringhe (NOME+COGNOME):



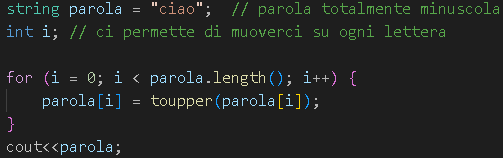
Se vogliamo mettere lo spazio tra nome e cognome:



Oppure possiamo concatenare le stringhe anche con **append()**



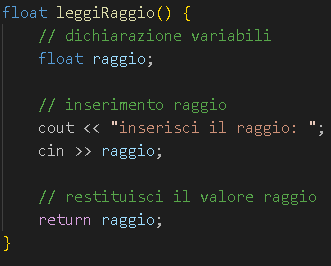
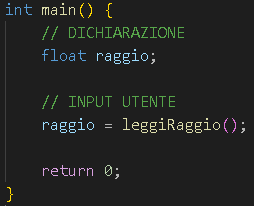
per verificare se un carattere è ***maiuscolo***  o ***minuscolo*** , in questo esempio le lettere minuscole diventeranno maiuscole (le maiuscole rimarrannò invariate):

 🡪 

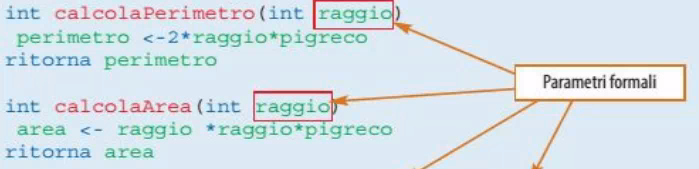
**INTODUZIONE FUNZIONI**

le funzioni ci permettono di effettuare delle operazioni al ***di fuori del main*** (il cuore del programma), però al suo interno creeremo *un collegamento* alle funzioni:

**MAIN**  **FUNZIONE**







Andiamo ad analizzare più nel dettaglio ***le parti costitutive*** del **main:**







Andiamo ad analizzare più nel dettaglio ***le parti costitutive*** delle **funzioni:**

****

****

****



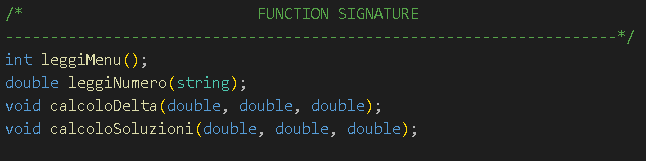
Vogliamo cambiare i valori degli argomenti della funzione originale o ritornare più di un valore dalla funzione, in questo caso utilizziamo i parametri **passati per riferimento.**

per usare le funzioni con il PASSAGGIO RIFERIMENTObisogna utilizzare **&**.

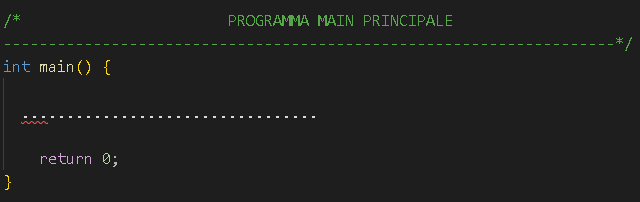
 🡪 una volta calcolato X ed Y nella funzione, i

valori vengono salvati anche nel **main**

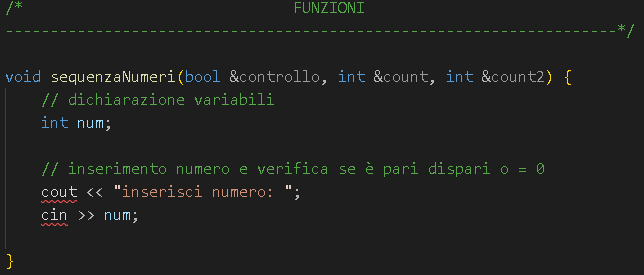
seguiamo questo schema a cascata per applicare il **function signature** nel nostro programma:



Successivamente andrà **il main:**



Successivamente metteremo **le funzioni:**



Piu avanti vedremo come collegare + file divisi nello stesso programma **(file.h)**

Domande fino alle funzioni

20 domande in 30 minuti

Risposta multipla vero/falso

**TIPI AGGREGATI**

**ENUMERAZIONI**

LE ENUMERAZIONI SONO SERIE DI VALORI INTERI CHE VENGONO ASSOCIATI A NOMI DI COSTANTI

SE NON VIENE SPECIFICATO IL PRIMO VALORE DELLE ENUMERAZIONI è = 0

**ARRAY (vettori)**

È uno dei più importanti tipo di aggregato, è un insieme di elementi omogeneo.

possiamo dare tanti dati dello stesso tipo con un unico nome al posto di utilizzare tanti nomi per indicare diversi dati, l’array viene passato sempre per riferimento, MAI per valore!

Tipo (int…) nomeArray [dimensione] { … }

Ex:

int temperature [4] = {3, 2, 5, 1} (un array di 4 numeri) 🡪 *la numerazione degli indici parte da* ***0 (zero)!***

Cout << temperature [1] 🡪 “stampera 2”

**MATRICE**