Linguaggio C++

Uso ambiente **Dev C++** con creazione di progetto con scelta ***Basic --> Empty Project***

NB: E' **necessario:**

- adoperare la sintassi più **evoluta** per le direttive di precompilazione[1], usando come "contenitore" il **namespace** std

```
#include <iostream>        // nome della libreria di I/O
using namespace std;
```

Il **C++ evoluto** permette di dividere lo spazio globale di definizione delle variabili (e delle funzioni) in diverse parti dette **namespace** ad ognuna delle quali è associato un nome univoco predefinito nel linguaggio o definito dal programmatore. Tale raggruppamento dei nomi in **contenitori** detti "***spazio dei nomi***" serve ad evitare ambiguità sull'uso di identificatori in programmi complessi.

Un **namespace** è dunque un insieme di nomi che può essere usato in alternativa ad altri insiemi analoghi e gli *identificatori standard* del linguaggio C++ sono contenuti nel namespace **std**.

- provvedere, al termine del programma, a **mantenere aperta la finestra** dell'**applicazione *console*** (ad esempio fino alla pressione del tasto INVIO)

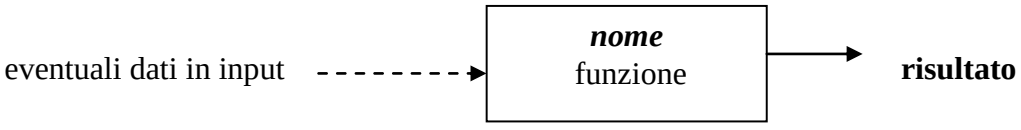
```
cout << "Premi ENTER per continuare .....";
cin.get();          // attesa della pressione del tasto "a capo" per proseguire
```

Modalità che richiede di porre particolare attenzione a svuotare il *buffer* da eventuali caratteri di "*a capo*".
Modalità alternativa è l'uso di *chiamate a sistema operativo* definite nella libreria standard **cstdlib**

```
system("PAUSE");
return EXIT_SUCCESS;
```

- ricordare che la funzione[2] principale deve essere definita di tipo **intero**

```
int main( )
    {
        // il compilatore inserisce automaticamente
        return 0;
    }
```



[1] **Direttive** che non codificano un comportamento del programma ma si limitano a dare *istruzioni di servizio* al compilatore per la generazione del file eseguibile

[2] Una **funzione** è un *modulo di programma* che può essere inteso come una "scatola nera" capace di realizzare una specifica elaborazione fornendo sempre un risultato (se la funzione non restituisce un valore è detta *void*) e potendo utilizzare dati che gli vengono passati, ad esempio come valori, al momento della *chiamata* da programma.

/*
Esempi con uso del costrutto: **SEQUENZA**

Es1.

concetti:

- **commento** /* anche su più righe */ oppure // su unica riga
- main() o **funzione principale**: dice alla CPU da dove eseguire
- blocco di istruzioni o corpo { }
- fine dell'istruzione con **;**
- **direttiva al preprocessore** (#include) per *includere librerie standard*.
  Equivale a inserire la definizione di *oggetti* che useremo nel programma
  (ad esempio un *canale* per scrivere a monitor ed uno per leggere da tastiera)
- ***cout*** con ***operatore*** << ***'scrivi su'*** o inserzione: per visualizzare su video
- ***cin*** con metodo ***get()*** per **acquisire da tastiera un carattere**
- sequenza di escape per **a capo** "\n" o modificatore di formato ***endl***

*/

#include <iostream> // **libreria** che descrive gli oggetti per colloqui con I/O

using namespace std; // clausola per evitare di ripetere il namespace cioè il
// contenitore detto "***spazio dei nomi***" **std**
// che contiene *identificatori standard* del linguaggio C++
// ad esempio invece di std :: cout si scriverà solo **cout**

```
int main()
    {
     cout << "Questo e' un semplice programma\n";
     cout << "in linguaggio C++" << endl;                    // operatori di inserzione

     cout << "Premi ENTER per continuare .....";      // in ambiente Dev C++ Empty Project
     cin.get();                                       // attende la pressione di ENTER
    }
```

/*==========

Attività:

Editare come file di testo ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) puro (uso EDITOR integrato nell'ambiente di sviluppo che crea un file *sorgente* con estensione .cpp)
poi **Compilare** (uso Traduttore sintattico o COMPILATORE che crea file oggetto.obj)
e, corretti gli eventuali errori, **Collegare** con librerie **e Mappare** gli indirizzi per **caricare in RAM** o Memoria di lavoro (uso LINKER LOADER che crea un file eseguibile.exe).

Per introdurre le ***parentesi graffe*** si possono digitare, in alternativa, le seguenti combinazioni di tasti:

{	ALT + 123 (tastierino numerico)	Ctrl +Alt + SHIFT + [	AltGr + SHIFT + [
}	ALT + 125 (tastierino numerico)	Ctrl +Alt + SHIFT +]	AltGr + SHIFT +]

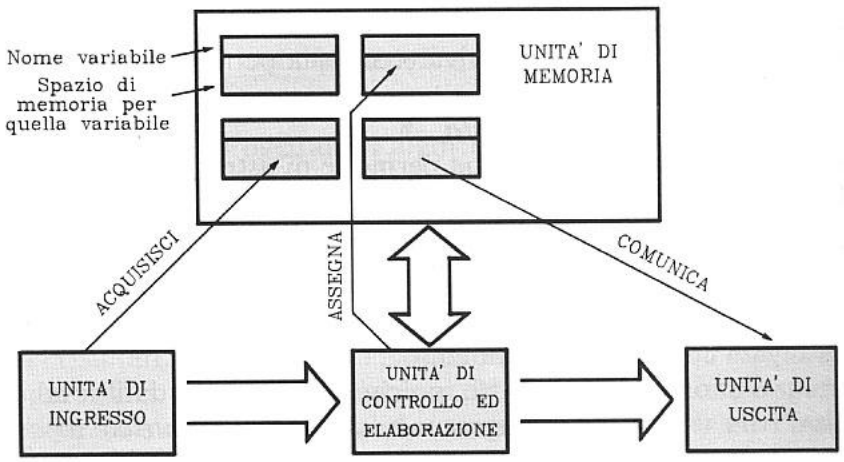
==========*/

```
/* Es2.

   concetti:
          - dichiarazione di variabile, associando al nome il tipo per  riservare il necessario spazio
            in memoria RAM
          - operatore di assegnamento = per inizializzare o modificare
          - operatori aritmetici
*/
#include <iostream>
using namespace std;

int main()
{
    int a, b;    // dichiarazione di variabili definendole di tipo intero
    a = 3;       // assegnazione di un valore  con uso operatore assegnamento
    b = 10;
    a = b-a;                                  // non è un'equazione è un'assegnazione
    cout << "risultato: " << a << "\n";
    a = a+b;
    cout << "risultato: " << a << "\n";
    a = a*b;
    cout << "risultato: " << a << "\n";
    a = a/b;
    cout << "risultato: " << a << "\n";
    b = a++;                                                    // post-incremento
    cout << "risulta b: " << b << " ed a : "<< a <<"\n";
    b = ++a;                                                    // pre-incremento
    cout << "risulta b: " << b << " ed a : "<< a <<"\n";
    a--;
    cout << "risultato: " << a << "\n";
    a = a % b;                                  // resto intero di a diviso b ( operatore modulo)
    cout << "risultato: " << a << "\n";

    cout << "Premi ENTER per continuare .....";      // in ambiente Dev C++
    cin.get();                                       // Empty Project
}
```



```
/*
  Esempi con uso del costrutto: RIPETIZIONE (iterazione)

  ESr1.

      concetto:

          - for con una sola istruzione

      notare che for è seguito da una parentesi che contiene:

      1) una assegnazione del valore iniziale del contatore
      2) una condizione  logica o aritmetica che indica la fine del ciclo
      3) un aggiornamento del contatore (in questo caso un incremento di + 1)

      Segue il corpo del for che può essere una sola istruzione oppure un blocco di
      istruzioni { } che viene ripetuto mentre è verifica la  condizione impostata.
*/

#include <iostream>
using namespace std;

int main()
{
      int i;
      for (i=1; i<=1; i++)
            cout << "passo " << i << " = " << 2*i << endl;

     cout << "Premi ENTER per continuare .....";       // in ambiente Dev C++
     cin.get();                                        // Empty Project
}




/*
   Attività : eseguire gli esercizi e commentare l'effetto prodotto inserendo breve descrizione nel
              testo di ogni esercizio
*/
```

/* ESr2.

concetti:

- ***cin*** per leggere da tastiera con ***operatore*** **>>** ***'leggi da'*** o estrazione

- **while** (ciclo con controllo in testa) con sintassi:

while (espressione)

istruzione

Ripete l'esecuzione dell'istruzione (o il blocco di istruzioni) **mentre** l'espressione è vera

In questo esempio il *ciclo* continua finché non viene digitato da tastiera un numero maggiore o uguale a 100
*/

```
#include <iostream>
using namespace std;

int main()
{
    int a = 0;    /* assegnazione del valore contemporaneamente
                     alla dichiarazione come variabile intera */

    while  (a <100)     // uso di operatore relazionale: minore
        {
         cout << "battere un numero : ";
         cin >> a;
        }
    cout << "realizzata la condizione di uscita dal while\n";
    ……
    cout << "Premi ENTER per continuare .....";       // in ambiente Dev C++
    cin.get();                                        // Empty Project
}
```

/*

Attività : eseguire l'esercizio e commentare l'***effetto*** prodotto inserendo breve descrizione nel testo dell'esercizio stesso

NB: Si verifica che **la lettura non avviene direttamente**, ma tramite un'area di memoria, detta buffer di input

Infatti al verificarsi della condizione di uscita la finestra si chiude senza che compaia la scritta "realizzata la condizione di uscita dal while". Questo perché per introdurre il numero si è premuto un "*a capo*" che è stato memorizzato nel "***buffer***" (implementazione del flusso logico o **stream**) ed è necessario "**svuotare il buffer**" da tale carattere di "*a capo*" ad esempio estraendolo con **cin.get()** oppure con altri metodi che applicheremo in seguito.

NB: tutti i canali o stream vengono implementati con un buffer (come approfondiremo) anche cout
*/

/* ESr3.

identico all'esercizio ESr2,
ma usa un **do...while** (*ciclo con controllo in coda*), anziché while, il che permette di non inizializzare con un valore di partenza la variabile con identificatore a.

concetti:
- do ... while con sintassi:

do
istruzione
while (espressione);

Ripete l'esecuzione dell'istruzione (o il blocco di istruzioni) **mentre** l'espressione è vera. Anche in questo esempio il *ciclo* continua finché non viene battuto un numero di valore maggiore o uguale a 100.
*/

```
#include <iostream>

using namespace std;

int main()
{
      int a;
      do
        {
            cout << "battere un numero : ";
            cin >> a;
         } while  (a <100);
      cout << "realizzata la condizione di uscita dal while\n";

     cin.get() ;          // svuota il buffer

     cout << "Premi ENTER per continuare .....";       // in ambiente Dev C++
     cin.get();                                        // Empty Project
}
```

/* ESr4.

concettualmete identico all'esercizio ESr2 e ESr3: il loop continua finché non viene battuto un numero di valore maggiore o uguale a 100.

Un **loop *INFINITO*** si può realizzare con while(1) poiché nel C il valore non zero significa *vero*, oppure, come nell'esempio, con un *for infinito*, cioè ***for(; ;)***

La condizione di uscita viene testata dentro il blocco di istruzioni del ciclo infinito usando un costrutto alternativa con sintassi:

if (espressione)
istruzione_vera

Se tale condizione è vera, si esce dal ciclo con l'istruzione ***break*** che forza l'uscita dal corpo in cui si trova.
*/

```
#include <iostream>

using namespace std;

int main()
{
      int a;
      for ( ; ; )
        {
           cout << "battere un numero : ";
           cin >> a;
           if (a >= 100)          // uso di operatore relazionale maggiore o uguale
             break;
         }
      cout << "realizzata la condizione di uscita dal ciclo infinito\n";

     cin.get() ;           // svuota il buffer

     cout << "Premi ENTER per continuare ....."; // in ambiente Dev C++
     cin.get();                                  // Empty Project

}
```

```
/*
   Esempi con uso del costrutto: ALTERNATIVA

   ESa1.

       sintassi:

         if  (espressione)
            istruzione_vera
         else
            istruzione_falsa

      Se espressione è Vera, esegue istruzione_vera
      altrimenti (else) esegue istruzione_falsa
*/

#include <iostream>

using namespace std;

int main()
{
       int i, j;
       cout << "Battere un numero = ";
       cin >> i;
       cout << "Battere un altro numero = ";
       cin >> j;
       if (i == j)                                    // uso di operatore relazionale: uguaglianza
          cout << "I due numeri sono uguali\n";
       else
          cout << "I due numeri sono diversi\n";

      cin.get() ;            // svuota il buffer

       cout << "Premi ENTER per continuare .....";       // in ambiente Dev C++
       cin.get();                                        // Empty Project

}
```

```
/* ESa2.

	concetti:

	- l'operatore condizionale ? :

	- operatori relazionali (maggiore >  minore < uguale == )
*/

#include <iostream>
using namespace std;

int main()
{				// variabili di tipo reale con rappresentazione FP o in
	double a = 5.2,	// virgola mobile con doppia precisione (word 32 bits)
		b = 10.5,
		minimo;

	a < b ? (minimo = a) : (minimo = b);

	cout << "Il minimo tra " << a << " e " << b << " e' " << minimo;
	cout << endl << endl;

	cin.get() ;		// svuota il buffer
	cout << "Premi ENTER per continuare .....";	// in ambiente Dev C++
	cin.get();					// Empty Project
}

/* ESa2 bis.

	- operatori logici: and  &&  or ||  not !
	- uso di operatore condizionale
*/

#include <iostream>

using namespace std;

int main()
{
	int   a = 0,	// inteso come FALSO
	      b = 1,	// non zero e' inteso come VERO
	      ris;

	(a && b) ? (ris = 1) : (ris = 0);  // prova a usare || e != diverso
	cout << "Il risultato e' " << ris;
	cout << endl << endl;
	cin.get() ;		// svuota il buffer
	cout << "Premi ENTER per continuare .....";	// in ambiente Dev C++
	cin.get();					// Empty Project
}
```

/* ESa3.

concetti:

- switch con sintassi:

switch (espressione)
{
case costante : istruzioni
.....
}

In realtà switch non è altro che una *sequenza di if*, dove però la condizione da valutare per verificare se è vera è una COSTANTE
*/

```
#include <iostream>

using namespace std;

int main()
{
  char ch;
  cout << "Battere una lettera minuscola ";
  cin >> ch;
  switch (ch)
       {
            case 'a' :
            case 'e' :
            case 'i' :
            case 'o' :
            case 'u' :  cout << "Hai battuto una vocale\n";
                        break;
            default :   cout << "Non e' una vocale\n";
       }

   cin.get() ;            // svuota il buffer

   cout << "Premi ENTER per continuare ....."       // in ambiente Dev C++
   cin.get();                                       // Empty Project
 }
```

```
/* CONTI con uso di FUNZIONI PREDEFINITE : abs , sqrt , pow

        per calcolare :  - il valore assoluto del numero intero "a"
                         - la radice quadrata di  "b"  (con b non negativo)
                         - l'elevazione a potenza (con esponente positivo)

        Funzioni predefinite in librerie attuali[3] cstdlib e cmath
        o nei file di intestazione (header file) math.h e stdlib.h
*/

#include <iostream>
                        // inserimento di librerie
                        // che contengono la definizione delle funzioni usate nel programma

#include <cstdlib>      // per chiamate al SO, per abs(int) ed eventualmente fabs (double)

#include <cmath>

using namespace std;

int main()  {
  int a = -2;
  int val_ass;
  double  b = 4.0,              // reali: virgola mobile in doppia precisione[4]
        radice, potenza;
  val_ass = abs (a);            // chiamate a funzioni con passaggio di valori:
  radice = sqrt(b);             // il controllo passa alla prima istruzione
  potenza = pow(a, b);          // della funzione e, tornando al flusso principale,
                                // il risultato si sostituisce al nome

  cout <<"Il valore assoluto di " << a <<" e' "<< val_ass<< endl;
  cout <<"La radice quadrata di " << b <<" e' "  << radice<< endl;
  cout <<"Il numero "<<a<<" elevato a "<< b<<" e' "<<potenza<<endl;

                // se uso solo in stampa le funzioni è più rapido

  cout <<"\nIl valore assoluto e' "<< abs(a);
  cout <<"\nLa radice quadrata e' " << sqrt(b);
  cout <<"\nIl potenza e' " << pow(a,b);

  system("PAUSE");              // avvisa l'utente "Premere un tasto per continuare. . ."
                                // e alla pressione di un tasto qualsiasi
  return EXIT_SUCCESS;          // chiude la finestra console
}

// Attività: implementa una calcolatrice con menu di scelta
```

[3] Tra le librerie attuali esiste anche **<cstdio>** per utilizzare le funzioni di I/O standard del linguaggio C infatti la **Libreria Standard del C++** ingloba la **Run Time Library** del **C**. Per approfondire

[4] Rappresentazione FP con **mantissa** e caratteristica (o **esponente**) tali che *numero = m.B^c* dove B è un intero detto **base** (tipicamente 2 o 16 per float o 32 per double) ; **precisione** è il numero di cifre decimali nella mantissa.

```
/*================================================================
  Tabella cinematica [unita MKS] :
  simulazione e presentazione dei risultati di  un'esperienza di laboratorio
  Trascinamento di corpo con  MASSA  nota, su  rotaia di LUNGHEZZA nota, grazie ad un
  contrappeso sottoposto alla forza di gravità
  Campionamento ogni secondo per un'ora
  ================================================================*/

#include <iostream>

#include <cstdlib>          // per chiamate al sistema operativo

using namespace std;

/* definizione di costanti simboliche con uso di define (una direttiva al preprocessore che ha
effetto durante la  prima fase della compilazione): prima di tradurre il programma con correzione
sintattica lo scorre tutto e sostituisce  il valore impostato al posto del simbolo; i  nomi infatti sono
più significativi dei numeri per il programmatore */

#define     TMAX    3600      // tempo di simulazione (3600 sec o 1 ora)
#define     DT      1         // intervallo di discretizzazione in secondi

#define     MASSA   0.4       // espressa in Kg
#define     G       9.8       // accelerazione di gravità in m/sec^2
#define     MAX     2         // lunghezza della rotaia

#define     RIGHE   17        // numero di righe per presentare su più schermate
                              // controllando lo scrolling
int main()
{
  char ch= 'y';

  double   peso, acc, pos, vel;
  int      numrighe,
           tempo;

  while( (ch =='y') || (ch =='Y') )
  {
     numrighe = 0;
     tempo = 0;
     pos = 0;

    system ("CLS");  // per pulire lo schermo (clear screen) con comando DOS

     cout << "Introduci il contrappeso [Kg] = ";    // es: tra 0.001 e 0.01
     cin >> peso;

     acc = (peso*G)/(peso+MASSA);

     cout.setf(ios::fixed);          // formato fisso necessario settando flag
     cout.precision(2);              // per fissare il numero di cifre decimali significative
```

```
    cout << "\n\tTabella Cinematica\n\n";                    // per inserire tabulatori \t


    cout << " tempo | posizione | velocita'\n";
    cout << "_______|____________|__________\n";        // prima colonna      7
                                                          // seconda colonna 11
                                                         // terza colonna      10 caratteri
    do {
          vel = acc*tempo;
          if (numrighe > RIGHE)
                              {
                                    cin.get();                              // per svuotare il buffer
                                    cout << "\npremi  INVIO per continuare ..\n";
                                    cin.get();
                                    numrighe = 0;
                              }
          cout.width(7);                  // stampo prima di aggiornare
          cout << tempo <<"|";

          cout.width(11);
          cout<< pos <<"|";

          cout.width(10);
          cout<< vel <<endl;

          pos = 0.5*vel*tempo;        // aggiorno la posizione

          tempo= tempo + DT;        // aggiorno il tempo

          numrighe++;                // aggiorno il numero delle righe

     } while (!( (pos >= MAX) || (tempo>=TMAX) ));

    cout << "\nVuoi rifare l'esperienza con altro contrappeso ? [y o n] ";
    cin >> ch;
  }
  cout << endl;

  system("PAUSE");                  // avvisa l'utente "Premere un tasto per continuare. . ."
                                    // e alla pressione di un tasto qualsiasi
   return EXIT_SUCCESS;            // chiude la finestra console

 }

/* NB:   il termine "PAUSE" non è "case sensitive" si può  usare anche "pause" o "Pause"...

        Si può verificare che EXIT_SUCCESS corrisponde al valore 0
        ad esempio con istruzione
                                    cout << EXIT_SUCCESS << endl;

*/
```

Attività:

- sostituisci come condizione ((pos< MAX) && (tempo<TMAX)) facendo pratica d'uso di ***operatori logici*** (OR || , AND && , e NOT !) e verificando il ***teorema di De Morgan*** cioè che il *"prodotto" in logica positiva equivale alla "somma" in logica negativa*

  Infatti: (pos >= MAX) è complementare a (pos < MAX)

  (tempo >= TMAX) è complementare a (tempo < TMAX)

  AND tra A e B (A*B)

  produce la stessa ***tabella di verità*** del NOR tra A/ e B/ ((A/) + (B/)) /

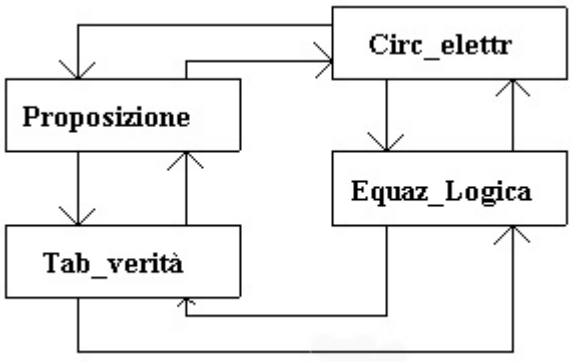
- **Verifica** poi, aggiornando i valori di posizione e tempo *prima* della stampa, la presenza di errore logico che causa la stampa del risultato calcolato anche la prima volta in cui pos > MAX

- Implementa poi il ***costrutto ripetizione*** con:

```
for ( ; ; ) {
   vel = acc*tempo;
   pos = 0.5*vel*tempo;
   if (pos > MAX)
     break;
   cout << tempo <<"\t"<< pos <<"\t" << vel <<endl;
   tempo++;
  }
```

- Implementa infine il ***costrutto ripetizione*** con *controllo in testa* con **while**

La logica delle proposizioni: GENERALITA'



Data un'affermazione è possibile verificarne l'esattezza mediante la logica delle proposizioni. Essa permette le seguenti conversioni.

E' cioè possibile rappresentare una frase con una equazione logica, con una tabella e con un circuito elettrico.

Si intende per proposizione il significato di un enunciato assertorio a cui si può assegnare un valore di verità

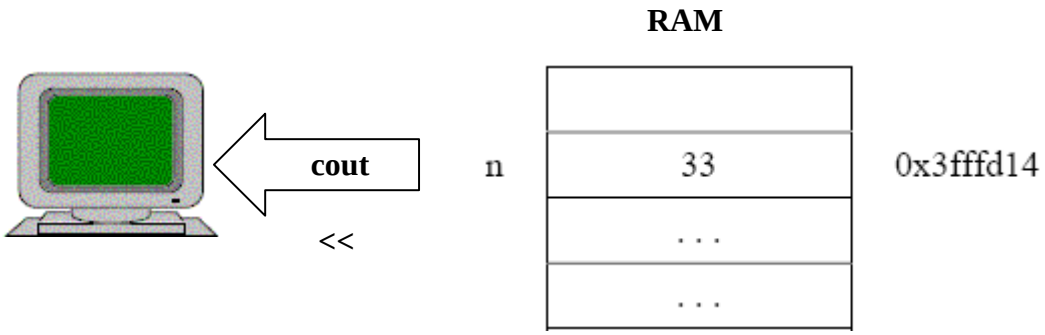
La logica delle proposizioni utilizza delle funzioni che si riferiscono ai connettivi *e*, *o* e all'avverbio *non*. La frase viene analizzata secondo le regole dell'analisi del periodo evidenziando la principale e le subordinate.

Ognuna di esse viene associata ad una variabile, detta *booleana*, che può assumere solo due valori: **vero** o **falso**. Queste variabili messe in relazione fra di loro, permetteranno la scrittura di equazioni logiche: *funzioni booleane*.

Il connettivo *e* corrisponde al *prodotto logico* **and**; il connettivo *o* corrisponde alla *somma logica* **or**; l'avverbio *non* corrisponde alla *negazione* **not**.

Appendice: flussi logici (*stream*) di input / output

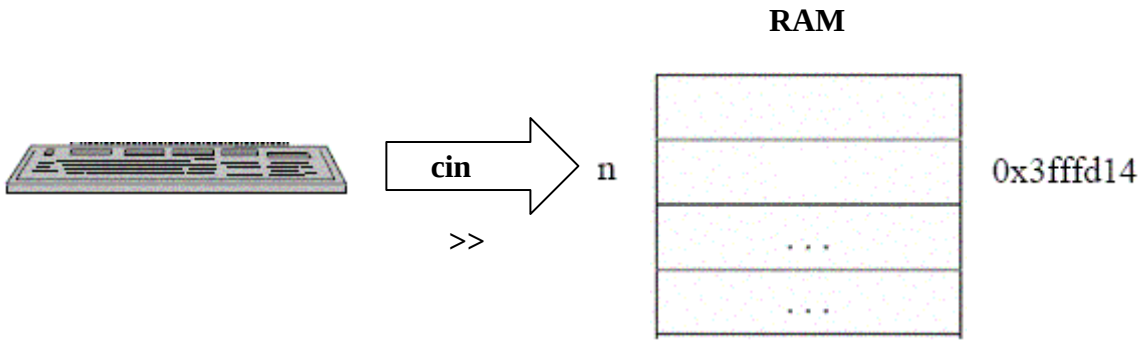
Operazione di scrittura a monitor con uso di **stream (flusso logico)** di **output**



cout << n; // stampa a monitor il ***valore*** 33

cout << &n // stampa a monitor l'***indirizzo*** esadecimale 3fffd14

Operazione di lettura da tastiera con uso di **stream (flusso logico)** di **input**

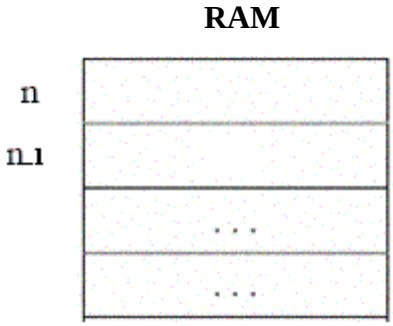


cin >> n; // inserisce il ***valore*** *digitato* da tastiera nella variabile con nome *identificatore* n

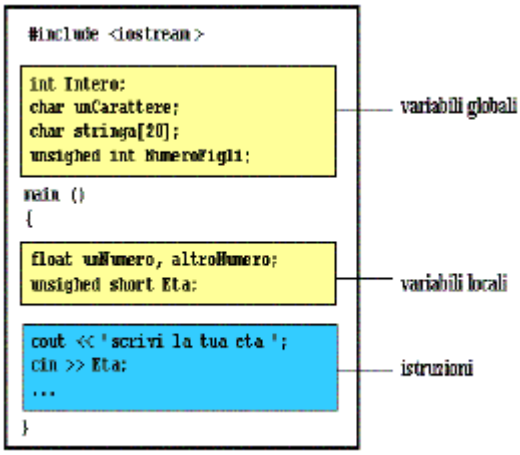
nb: con questa modalità ***non si legge lo spazio*** (blank)
viene interpretato come delimitatore al pari del *tabulatore* e dell''*a capo'* (tasto ENTER)
permettendo di realizzare letture successive:

cin>> n >> n_1;

due *caratteri* sono inseriti
in sequenza in celle con
nome *identificatore* n ed n_1
(occupano ognuno1 byte)



Appendice: ambiente *locale* e *globale*



Le variabili **globali** si possono usare in tutto il programma dal punto in cui sono dichiarate.

Lo *scope* delle variabili **locali** è invece limitato al *blocco* (cioè un gruppo di istruzioni racchiuse tra parentesi graffe) in cui sono dichiarate e non sono *visibili* se l'esecuzione del programma esce da tale blocco.

Eventuali ambiguità, nella filosofia del ***riuso***, si risolvono con l'operatore di ***scope resolution*** o risolutore di visibilità (simbolo **::**)**.**

Infatti quando una variabile locale ed una globale hanno lo stesso nome, quella locale "nasconde" la globale e per potervi accedere si utilizza in C++ tale operatore (che permette di accedere alla variabile globale omonima):

```
#include <iostream>
using namespacestd;
 int x = 0;                    // variabile globale
int main()
{
        int x = 5;             // variabile locale
        :: x = 4;                                  // modifica la variabile globale

        cout << x << endl;          // visualizza 5
        cout <<::x << endl;         // visualizza 4
        system("Pause");
        return 0;
}
```

Periodo di vita o di **esistenza (*lifetime*)**

DURATA: è un insieme di regole che ci dicono quando una variabile diventa attiva e quando è distrutta. Le variabili globali esistono sempre durante tutta l'esecuzione del programma; le variabili locali sono create nel momento della dichiarazione e distrutte alla fine dell'esecuzione dell'istruzione composta più interna che le racchiude.

Ambito di definizione o **campo di validità**

SCOPE: è il termine che descrive le regole che specificano quando un nome è utilizzabile, cioè quando si può usare un nome di variabile di funzione.

Ambito di visibilità (*scope* o *linkage*): porzione di programma nel quale può essere referenziato per nome un identificatore cioè nel quale è accessibile.

VISIBILITA': quando in un programma si hanno due variabili con lo stesso nome e a livelli diversi di portata, la visibilità dice quale delle variabili è utilizzata quando se ne usa il nome.

Appendice: assegnamento e **casting** con ***stile C++***

Quando dichiariamo una variabile il suo valore è indeterminato (i bit della zona di memoria riservata alla variabile hanno il valore loro assegnato da qualche precedente programma).

Se nel dichiarare una variabile si vuole anche assegnarle un valore iniziale si può usare l'operatore di **assegnamento** (stile C) oppure **parametrizzare** il dato (stile C++):

```
tipo identificatore = valore_iniziale ;   <-- Stile C
tipo identificatore(valore_iniziale);     <-- Stile C++
```

Ad esempio se si vuole dichiarare una variabile "a" di tipo intero con valore iniziale 0, si potrà scrivere:

```
int a = 0;   <-- Stile C
int a(0);    <-- Stile C++
```

In C++ si può utilizzare sia lo stile C che lo stile C++

Operatori espliciti di conversione

Gli operatori di conversione servono per convertire valori appartenenti ad un tipo in valori di un altro tipo. Vi sono diversi modi per fare questo in C++:

- Si usa l'**operatore** di ***casting*** cioè si fa precedere l'operando o l'espressione da convertire dal nome del nuovo tipo racchiuso tra parentesi **( )**

  Sintassi: *(tipo) espressione*

Stile C

```
int i;
float f = 3.14;
i = (int) f;
```

Converte il numero float 3.14 nel valore intero 3.
L'operatore di conversione è rappresentato da (int).

- Si fa precedere l'operando o l'espressione da convertire, racchiusa tra parentesi **( )** dal nome del **nuovo tipo** con uso della *funzione costruttore*

  Sintassi: *tipo (espressione)*

Stile C++

```
int i;
float f = 3.14;
i = int( f );
```

Usa la funzione "costruttore".

Il **TYPECASTING** è il meccanismo che permette di convertire un valore da un tipo di dato ad un altro.