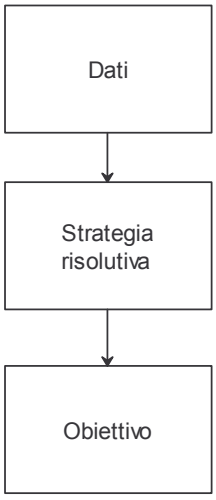
Dispense di informatica classi 3 LSA

La programmazione.

Algoritmi

Tutte le volte che si vuole far risolvere un problema dal computer occorre fornire ad esso dei dati e dei comandi (istruzioni) e il pc ci fornirà dei risultati.
Un **problema** è sempre caratterizzato da una situazione non completamente nota, di cui si cerca di conoscer determinati aspetti, avendo delle conoscenze iniziali .
Le conoscenze iniziali sono i dati in nostro possesso,il modo di ottenere i dati finali è la strategia risolutiva e i dati finali rappresentano l'obiettivo del problema.



Un esempio di matematica: determinare il perimetro di un quadrato , conoscendo l'area $S = 100\,cm^2$. Il dato iniziale è la superficie, la strategia risolutiva sono le due operazione radice quadrata e prodotto, l'obiettivo è il perimetro.
Ma se il problema non è matematico , la strategia risolutiva sarà una serie di azioni elementari (es. spiegare ad uno straniero come raggiungere l'aeroporto di Caselle partendo da Mirafiori).
La strategia risolutiva in ogni caso si chiama **algoritmo .**
L'algoritmo è quindi una serie ordinata di azioni necessarie , affinché , partendo dai dati iniziali di un problema , si riesca ad arrivare alla soluzione.
L'algoritmo deve essere eseguibile , cioè deve possedere le tre seguenti proprietà:

1. essere finito
    a. Considera un numero naturale
    b. Aggiungi 1
    c. Se la somma è < di 10, torna al punto a)

    Non finito (es.):
    a. Considera un numero naturale
    b. Aggiungi 1
    c. Torna al punto a)
2. non ambiguo
    a. Considera 2 numeri x e y
    b. Fa il prodotto
    c. Se x*y>100 ,va all'istruzione e)
    d. Torna ad a)
    e. Eleva x*y al quadrato

f. Scrivi il risultato.

Il seguente algoritmo invece è ambiguo:

a. Considera 2 numeri x e y
b. Fa il prodotto
c. Se x*y è grande ,va all'istruzione e) (ci si chiede cosa vuol dire grande)
d. Torna ad a)
e. Eleva x*y al quadrato
f. .Scrivi il risultato.

3. generalizzabile

Se vale in una situazione , dovrà valere in tutte le situazioni simili. Es.la regola per trovare il minimo comune multiplo vale per qualunque numero intero.

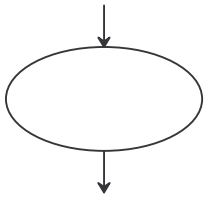
Una volta analizzato il problema e trovato l'algoritmo necessario a risolverlo, occorre descrivere la sequenza di azioni che si devono fare.

Diagramma di flusso.

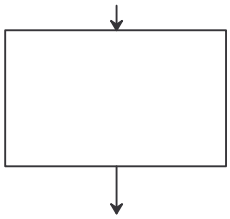
L'algoritmo può essere descritto da un'espressione matematica, oppure da una sequenza di frasi (discorsivo) oppure da **uno schema grafico che visualizza il fluire delle diverse operazioni da compiere. Lo schema grafico si chiama diagramma di flusso.**

Il diagramma di flusso (flow chart) utilizza una serie di simboli convenzionali di cui i principali sono

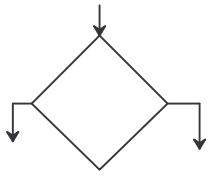
Inizio_ Fine



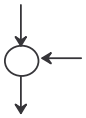
Blocco d'azione



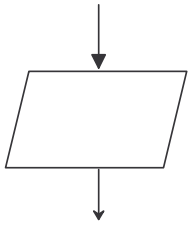
Blocco di controllo
o di selezione



collegamento



Blocco di input/output
(per immettere dati iniziali dall'esterno e per visualizzare o stampare i dati finali)

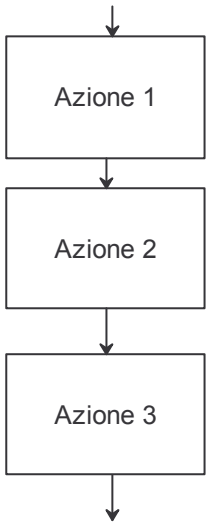


Ne esistono altri , ma per ora per i diagrammi di flusso base sono sufficienti.

Strutture fondamentali .

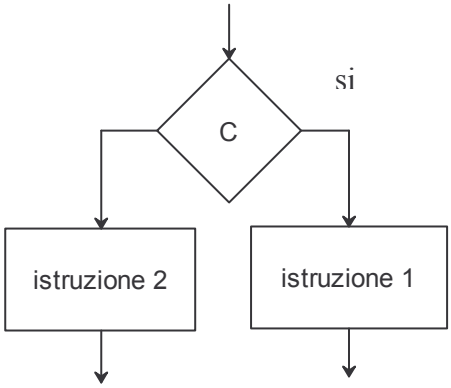
Le strutture fondamentali di un algoritmo e quindi di un diagramma di flusso sono 3 (nell'analisi di un problema ci possono essere tutte o solo qualcuna):

1. **sequenza** Le azioni sono eseguite una di seguito all'altra nell'ordine in cui sono state scritte(questo è lo schema che il computer segue , se non viene specificato altrimenti).

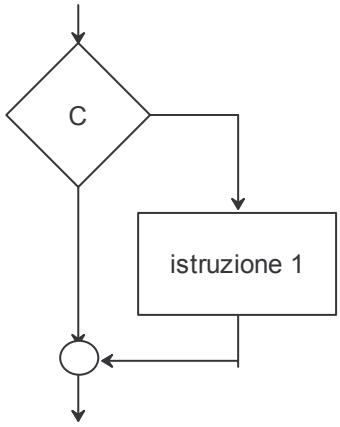


2. **selezione** E' un'istruzione di selezione o di controllo che permette di scegliere tra due percorsi differenti; può essere di due tipi:

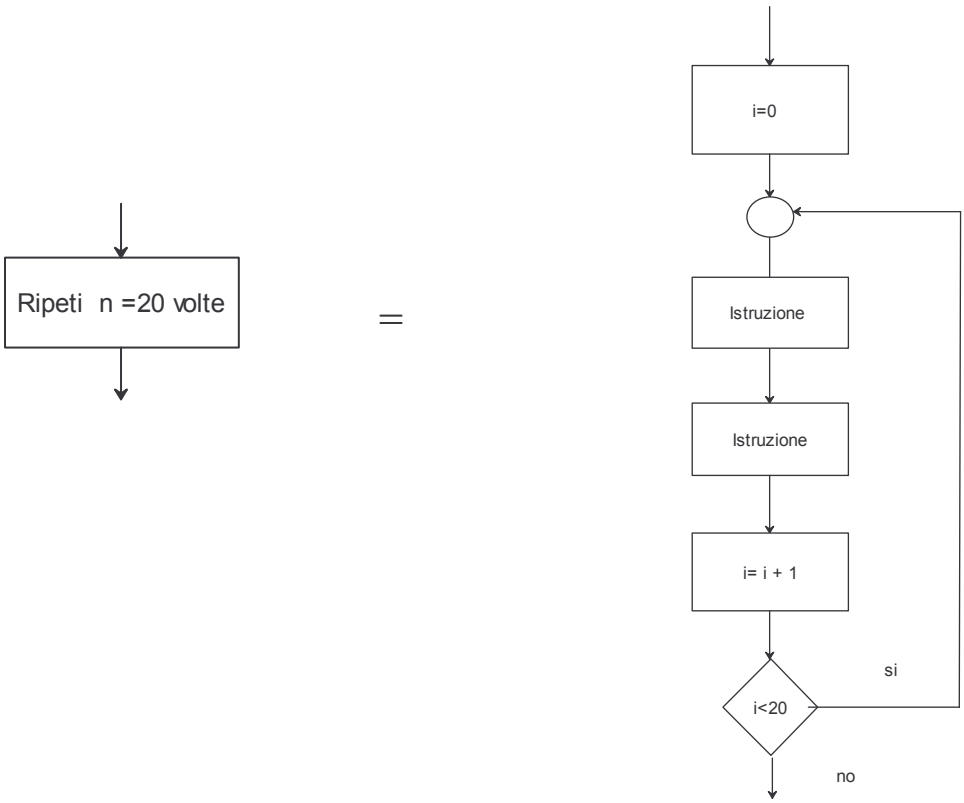
   a. Se è soddisfatta la condizione c, allora svolgi l'istruzione 1 , se no, allora la 2.



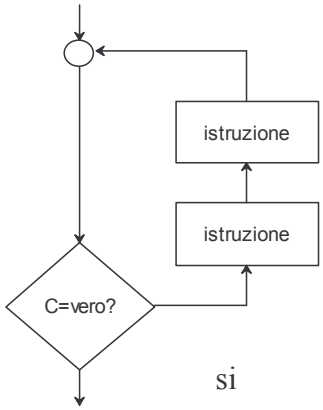
   b. se è soddisfatta la condizione C, allora esegui l'istruzione 1, se no , prosegui il programma



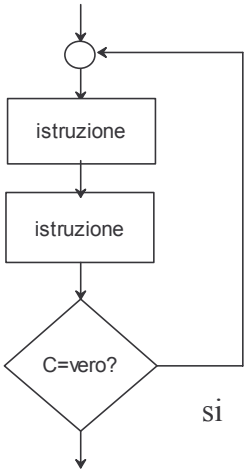
3. **iterazione o ciclo** si ripetono una o più istruzioni un certo numero di volte. Può essere di due tipi:
    a. Possiamo conoscere il numero di volte per cui un ciclo si ripete



    b. Non si conosce il numero di cicli, ma si esce su condizione:
        i. con un controllo di testa (il controllo è prima delle istruzioni)



        ii. o con un controllo di coda.(il controllo è dopo le istruzioni)



Esistono software appositi (tipo Diagram Designer software libero , scaricabile da Internet) per disegnare i diagrammi di flusso.
Per maggiori dettagli sui diagrammi di flusso e il loro uso consultare il libro in adozione al 3 anno del liceo SA Barbero, Vaschetto_ Corso di Informatica _ C.editrice Pearson Unità 2

Linguaggi di programmazione

Una volta trovato l'algoritmo e disegnato il diagramma di flusso occorre trasformarlo in un linguaggio comprensibile dal PC. **Un linguaggio è un insieme di parole collegate tra di loro da regole sintattiche.** Di linguaggi ad **alto livello** (vicini al nostro linguaggio, non al binario del PC) ne esistono tanti (C,C++, Basic, Pascal, HTML, Java) ciascuno adatto per un determinato compito (dall'analizzare problemi finanziari o matematici, allo scrivere e disegnare un sistema operativo o un pacchetto applicativo, es. Word, al costruire un sito web, ancora a comandare un microcontrollore che programma la lavatrice, o un PLC che controlla il braccio meccanico di una catena di montaggio). Per poter programmare è necessario conoscere un linguaggio di programmazione (cioè le parole, dette istruzioni, e le regole sintattiche) e avere gli strumenti software adatti a tradurre le istruzioni in linguaggio macchina (cioè nella sequenza di numeri binari che il PC può interpretare).
Gli strumenti software si chiamano IDE (Integrated Development Environement) cioè ambiente di sviluppo integrato e sono un insieme di programmi raccolti in un unico software che permettono di:

1. scrivere il programma (un **editor** di testo non formattato come notepad; Word è un editor formattato cioè inserisce altri caratteri oltre a quelli che noi scriviamo ad es. l'allineamento a destra a sinistra , testo giustificato).Il file ottenuto si chiama **file sorgente.**
2. tradurre le istruzioni in un codice interpretabile dal PC .Questa operazione viene svolta da un **compilatore** che crea un file che si chiama **file oggetto.**
3. trovare (contemporaneamente alla compilazione) gli errori lessicali e sintattici che il programmatore ha fatto .Il programma si chiama **debugger.**
4. inserire le librerie (cioè parti di programma standard che non è necessario riscrivere tutte le volte) che il programmatore ha richiesto di includere . Il programma si chiama **linker** e, oltre ad aggiungere le librerie, trasforma il file oggetto in **file eseguibile.**

A questo punto si può far "girare" (to run) il programma cioè vedere se svolge il compito che il programmatore voleva far svolgere.
Il compilatore trova gli errori sintattici e lessicali, ma non quelli **logici .** Se il programmatore ha sbagliato la formula o se non ha messo nel giusto ordine le azioni il risultato verrà sbagliato, perchè il PC è solo un esecutore. Occorre quindi verificare il programma per tutte le possibili soluzioni.
Vedi il libro in adozione al 3 anno del liceo SA Barbero, Vaschetto_ Corso di Informatica _ C.editrice Pearson Unità 7.

Il linguaggio di programmazione C++

Il linguaggio C++ deriva dal linguaggio C (un linguaggio ad alto livello che permetteva anche interazioni con la CPU) e, pur conservando le istruzioni e i vantaggi del precedente, permette **la programmazione ad oggetti** cioè una programmazione in cui è possibile costruire dei moduli "sicuri"(oggetti) facilmente riutilizzabili.
Sono spiegate solo le istruzioni base necessarie per codificare le strutture indicate nella parte sui diagrammi di flusso.
Prima di andare ad analizzare le principali istruzioni in C++, bisogna definire che cosa sono le **variabili** e quali **tipi di variabili** sono usate dal PC.

Variabili.
In **matematica e in fisica** una **variabile** è un carattere alfabetico che rappresenta un numero arbitrario, non completamente specificato o del tutto sconosciuto. Ad es. nell'equazione di una retta y=4x+5 x e y sono 2 variabili a cui io posso assegnare tutti i valori che soddisfano l'equazione. Si dice anche che una variabile può assumere tutti i valori possibili nell'intervallo di definizione. In **informatica** una variabile individua una porzione di memoria (una o più locazioni di memoria) destinata a contenere dei dati, suscettibili di modifica nel corso dell' esecuzione di un programma, cioè è "l'indirizzo" della cella di memoria in cui c'è o verrà inserito un valore . Una **variabile** è un **nome** (solitamente come una sequenza di caratteri e cifre es **voto, media1, b, A123**).
Ci possono essere variabili di tipo diverso : un numero intero, uno con la virgola, un carattere alfanumerico. A seconda del linguaggio di programmazione usato e della lunghezza del numerosi hanno nomi differenti. Sotto è allegata una tabella con i tipi di variabili usate.
Costanti . In **informatica** una costante é un dato non modificabile situato in una porzione di memoria (una o più locazioni di memoria) .

Rappresentazione dei numeri sul PC

Numeri interi. Il numero di bit con cui viene rappresentato dipende dal tipo di applicazione odi linguaggio che si sta usando.
Tipi interi senza segno:

N° byte(occupato in memoria)	intervallo di valori della variabile	nome in C o C++
1byte	0-255	**Unsigned char** (anche i caratteri alfanumerici)
2byte	0-65535	**Unsigned int**
4byte	0-4 294 964 295	**Unsigned long**

Tipi interi con segno:

N° byte (occupato in memoria)	intervallo di valori della variabile	nome in C o C++
1byte	-128-+127	**Char** (serve anche per i caratteri alfanumerici)
2byte	-32768-+32767	**Int**
4byte	-2 147 483 648- +2 147 483 647	**Long**

Numeri non interi. Il numero di bit con cui viene rappresentato dipende dal tipo di applicazione che si sta usando. I numeri **negativi** sono scritti in modulo e segno**.**
I numeri non interi si dicono anche numeri in **virgola mobile (floating point)** perché in forma normalizzata sono scritti come 0,…….. un numero di cifre che corrisponde alla precisione moltiplicato per una potenza di 10 (ad es. 75,46 viene salvato in memoria come $0{,}7546 * 10^{2)}$ .La parte dopo la virgola si chiama **mantissa** e l'esponente della potenza di 10 **caratteristica.** Se il numero non è normalizzato si può normalizzare come nell'esempio..
Tipi di numeri non interi:

N° byte (occupati in memoria)	segno(bit)	mantissa (bit)	caratteristica(bit)	nome in C o C++
4 (32bit)	1 +	23 +	8	**Float**
8 (64bit)	1 +	52 +	11	**Double**
10(80bit)	1 +	64 +	15	**Long Double**

Float.
I numeri in virgola mobile utilizzano 4 byte 3 byte per la mantissa e 1 per la caratteristica :

1° byte	2° byte	3° byte	4 ° byte
S mantissa	mantissa	mantissa	caratteristica

S indica il segno : 0 se positivo, 1 se negativo.

Istruzioni in C++

La funzione main ( )

Nella codifica (cioè nella scrittura di un programma in un determinato linguaggio di programmazione), le istruzioni (gli enunciati) sono racchiuse tra due parentesi graffe che indicano l'inizio e la finedel programma . In C++ la funzione principale si chiama int main( )

```
int main ()
    {.....
     .....
    .....
```

}

Le parentesi graffe (non essendo presenti su tastiera)si ottengono con :
{ Alt+123 (tastierino numerico) oppure Alt Gr+shift+[
} Alt+125 (tastierino numerico) oppure Alt Gr+shift+]

Al di fuori del main, prima, bisogna indicare quali librerie includere nel programma e i tipi di tutte le variabili utilizzate nel programma

Le istruzioni di input (cin>> ...) output (cout<<.....)

Se si desidera inserire da tastiera un valore occorre spiegare all'utente cosa deve inserire (cioè visualizzare sullo schermo una scritta) e poi richiedere il valore da tastiera .Il programma sarà:

```
#include  <iostream> //libreria  per il flusso di dati di input output
using namespace std; // direttiva per l'utilizzo dei comandi standard
int b;                //dichiarazione  variabili intere
main ()                       //inizio programma principale
    {cout<<"inserisci numero biro"; //visualizza sullo schermo la frase tra" "
     cin>>b;      //inserisce  da tastiera il valore della variabile b
     ......................
    ..............
    }
```

NB.

1. Tutti gli enunciati finiscono con un **;** che ha lo scopo di indicare al compilatore la fine dell'istruzione. Se manca il ; sulla riga successiva ci deve essere una parentesi { per indicare che l'istruzione continua su più righe.
2. il C++ è "**case sensitive**" cioè distingue tra maiuscolo e minuscolo ; le parole riservate (cioè ad es cout) sono sempre **minuscole**. La variabile **A** è diversa dalla variabile **a.**

cout visualizza sullo schermo la scritta tra " "
cin aspetta e preleva da tastiera il valore della variabile inserito.
Sono state aggiunte 3 righe di codice:

1. #include<iostream> Il C++ non ha istruzioni dirette per controllare la tastiera e lo schermo. iostream è una libreria (un insieme di parti di programma (funzioni, oggetti) già costruite che così non si deve riscrivere)
2. using namespace sdt; istruzione che serve a dire che utilizzo i nomi standard per l'input output
3. int b; questo enunciato si chiama dichiarazione e serve a dire al PC che ci sono 2 byte (è un intero) in memoria per la variabile che si chiama b.

Dopo i ; ci sono // (doppio slash) e poi un testo; il testo è un commento (una spiegazione di cosa fa l'istruzione).

I **commenti** hanno valore soltanto per il programmatore e vengono ignorati dal compilatore. E` possibile inserirli nel proprio codice in due modi diversi:

1. racchiudendoli tra i simboli **/*** e ***/**
2. facendoli precedere dal simbolo //

Nel primo caso e` considerato commento tutto quello che e` compreso tra **/*** e ***/**, il commento quindi si puo` estendere anche su piu` righe e trovarsi in mezzo al codice.

Nel secondo caso, proprio del C++, e` invece considerato commento tutto cio` che segue // fino alla fine della linea, ne consegue che non e` possibile inserirlo in mezzo al codice o dividerlo su piu` righe (a meno che anche l'altra riga non cominci con //).

Istruzioni per l'uso di un ambiente di sviluppo per il C++.

L'IDE, l'ambiente di sviluppo utilizzato, è il Dev C++ scaricabile liberamente da internet. Si apre il programma e si crea un nuovo file sorgente (attenzione non un progetto, perchè è più complicato). Si scrive il programma,lo si salvo (con estensione .cpp) lo si compila e, se ci sono degli errori di sintassi,questi appaiono nella parte inferiore dello schermo. Si corregge e si risalva. Quando il programma è giusto si può eseguire: a questo punto appare un piccolo schermo nero in cui si può provare il programma e verificare i risultati (ci possono ancora essere degli errori logici)..
Sotto è scritto un programma che utilizza le istruzioni viste fin qui e altre da spiegare (si può provare su DevC++):

```
/* Cartolaio.cpp  Uno studente acquista dal cartolaio quaderni e biro. Costruire  un programma che, ricevendo in ingresso il costo e  il numero di quaderni  e di biro  acquistate, stampi (visualizzi) l'importo complessivo */

#include  <iostream>              //libreria  per il flusso di dati di input output
using namespace std;             // direttiva per l'utilizzo dei comandi standard
int b,q;                         //dichiarazione  variabili intere
float cb,cq,importo;             //dichiarazione variabili reali ( con la virgola)
 main ()                         //inizio programma principale
        {cout<<"inserisci num biro"; //scritta sullo schermo
         cin>>b;                          //immissione da tastiera del valore di una variabile
         cout<<"inserisci num quaderni"; //scritta sullo schermo
         cin>>q;                          //immissione da tastiera del valore di una variabile
         cout<<"inserisci costo biro e costo quaderni "; //scritta sullo schermo
         cin>>cb>>cq;                     //immissione  da tastiera del valore di due variabili
        importo= b*cb+q*cq;   // assegna alla variabile importo il risultato dell'operazione
         cout<<endl<<" l'importo e'="<<importo<<endl; /*visualizza sullo schermo su una nuova linea
                                                 (endl)  la frase tra" " e il valore di importo*/
         system("PAUSE");       //aspetta la pressione di un tasto per chiudere la  schermata del programma
   }                        // Chiude  il programma principale
```

NB: Tutti i programmi devono avere come commento all'inizio il testo del problema e il nome del file sorgente (in questo caso cartolaio. cpp).
Tutte le volte che si apre una parentesi graffa le istruzioni vengono scritte più a destra fino alla chiusura : questo modo di scrivere un programma si chiama **indentazione** e permette una maggior visibilità del flusso di programma con la visualizzazione dei blocchi di istruzioni che vengono trattate insieme.
Le 4 istruzioni nuove sono:

1. **float cb,cq, importo** ; tutte le variabili utilizzate devono essere dichiarate e bisogna anche definirne il tipo (queste sono con la virgola)
2. **importo= b*cb+q*cq**; è un calcolo , ma dal punto di vista della programmazione è un'istruzione di **assegnazione** cioè l'= indica che il risultato del calcolo viene posto nella variabile (meglio nelle celle di memoria di nome importo) importo. In questa istruzione il simbolo = sembra avere lo stesso significato di = in matematica. Ma in un programma si può anche scrivere a=a+1 espressione che in matematica è sbagliata : qui vuol dire che la a che sta a destra dell'uguale viene aumentata di 1 e il risultato viene di nuovo posto nella variabile a ,quella a sinistra dell'=. (se a valeva 2 prima di quest'istruzione , dopo vale 3)
3. **cout<<endl<<" l'importo e'="<<importo<<endl;** endl (end line) indica solo che scrive il risultato su una nuova riga .E' una visualizzazione sullo schermo un po' più complessa: apparirà l'importo è = valore importo. E' questo il cout tipico per la visualizzazione dei risultati.

4. **system("PAUSE");** Il programma giro molto velocemente e quando finisce, il piccolo schermo aperto durante il programma, scompare ; per poter vedere i risultati occorre fermare il programma.

Il programma sopra scritto ha una struttura (per quanto riguarda il diagramma di flusso) di sequenza, cioè le istruzioni sono l'una sotto l'altra nel giusto ordine.

La selezione

Se il problema richiede di scegliere tra due algoritmi o azioni differenti occorre un'altra istruzione **if.....else.** La sintassi dell'istruzione ha due possibili formulazioni (vedi i due tipi di struttura della selezione nei diagrammi di flusso):

```
if (Condizione)
            Istruzione1  ;
```

oppure

```
if ( Condizione )
            Istruzione1 ;
  else
           Istruzione2 ;
```

L'**else** e` quindi opzionale e quindi se la condizione è falsa si esegue direttamente la prima istruzione successiva del programma. Se invece l'**else** viene utilizzato nessuna istruzione deve essere inserita tra il ramo **if** e il ramo **else** . In entrambi i casi se **Condizione** e` vera viene eseguita **Istruzione1**, altrimenti nel primo caso non viene eseguito alcunche` e il programma va avanti, nel secondo caso invece si esegue **Istruzione2**.

Un esempio:

```
if ( X==10 )
     cout<<"hai vinto";
  else
    {                 // le istruzioni sono 2 quindi occorre aprire la {
    Y=Y+1;
    cout "hai perso";
    }
```

NB. Gli operatori presenti nella condizione possono essere >,<,>=,<=, = =, != (diverso) o anche un'espressione matematica o logica. In questo caso = = non è un'operazione di assegnazione, cioè 10 non viene posto nella cella denominata X, ma si **confronta** quello che è nella cella X con 10 ; se il risultato è vero si esegue l'istruzione successiva, altrimenti si passa all'**else.**

È importante sottolineare che, quando un ramo if o un ramo else è composto da più di un'istruzione, è necessario racchiudere tutte le istruzioni che ne fanno parte all'interno di una coppia di parentesi graffe, in modo da formare un blocco.

Un programma con selezione:

```
/* Acqua.cpp. Per la fatturazione della bolletta dell'acqua si calcolano:
   15 euro fissi per consumi non superiori ai 40m^3 ,  0,80 euro a m^3 oltre i 40,  sul totale un'IVA del
   19%. Calcolare il totale e visualizzarlo */

#include  <iostream>
using namespace std;
float mcubi,importo, IVA;
int main()
   {  cout<<endl<<" Inserisci i mcubi";
    cin>>mcubi;
    if (mcubi>40.00)                    //se mcubi> 40.00
```

```
        importo= 15+(mcubi-40)*0.8;  //   allora si somma al fisso il costo dei mcubi in più
    else                             //altrimenti
       importo=15;                   //l'importo è solo il fisso
    IVA=importo*19/100;
    importo=importo+IVA;
    cout<<endl<<" Importo= "<<importo<<endl;
    system("pause");
}                          //le istruzioni già spiegate non sono commentate.
```

Istruzioni iterative.

Facciamo il caso di dover sommare 10 numeri forniti da tastiera . Con le istruzioni fornite fino a questo punto l'unico programma che possiamo scrivere è

```
#include  <iostream>
using namespace std;
float a,b,c,e,f,g,h,i,l,m, totale;
int main()
   {     cout<<endl<<" Inserisci numero";
         cin>>a;
         cout<<endl<<" Inserisci numero";
         cin>>b;
          cout<<endl<<" Inserisci numero";
          cin>>c;
         cout<<endl<<" Inserisci numero";
           cin>>d;
         ..............  inserisco gli altri numeri
         ..............
         ..............
         Totale=a+b+c+d+e+f+g+h+i+l+m;
         cout<<" Totale= "<<totale;
          system("pause");
  }
```

Cioè ci occorrono 11 variabili e una lunga sequenza di istruzioni per l'inserimento dei valori.
Ma si può usare un altro modo (algoritmo) per fare la somma, quella che si chiama **Somma ricorsiva** (è il modo in cui noi facciamo lunghe somme quando non abbiamo a disposizione un foglio di carta).
Si tiene conto che si parte da 0 (cioè si azzera una variabile che conterrà la somma). Si prende il primo numero e lo si somma al secondo. Si ricorda solo la somma . A questa si aggiunge il terzo numero. Di nuovo si ricorda la somma e avanti così. C'è sempre solo la somma parziale e un numero da ricordare. Dal punto di vista informatico è
S=0 Inserisci val S=S+val Inserisci val S=S+val Inserisci val S=S+val per 10 volte.
Ci sono solo 2 variabili S e val e 2 istruzioni che si ripetono 10 volte .

Istruzione for.
L'istruzione iterativa che permette di ripetere un numero conosciuto di volte una parte del programma è il **for** la cui sintassi è

for (espressione 1; espressione 2; espressione 3)
{…….....
…….....
}
dove espressione1 è la condizione iniziale, l'espressione2 è la condizione finale e l'espressione 3 è l'incremento. Nei casi più semplici diventa:

for (i=0; i<n; i=i+1)
{…….....
…….....
}

che significa: per **i** che parte da 0 fino a quando **i<n,** con un **incremento della i di 1,** ripeti la parte di programma tra le parentesi graffe. L'**intero i** si chiama **contatore** e serve a contare le ripetizioni del for.

Con quest'istruzione il programma sulla somma di 10 termini diventa:

```
// somma.cpp  Somma 10 numeri dati da tastiera.

#include <iostream>
int i,val,somma = 0;//le variabili possono essere inizializzate durante la dichiarazione
int main()
    {
    cout << "Inserire 10 valori da sommare:" << endl;
    for (i = 0; i < 10; i++) //ripeto le istruzioni tra le { }per 10 volte
         { cin >> val;
         somma = somma + val;
         }
    cout << "somma = " << somma << endl;
    system ("Pause");
}
```

Se non conosco quanti valori bisogna inserire si può chiedere la quantità da tastiera prima del for.

```
 // somma1.cpp  Somma gli n  numeri dati da tastiera.

#include <iostream>
int i,n,val, somma = 0;//le variabili possono essere inizializzate durante la dichiarazione
int main()
{
     cout << "inserire quanti valori sommare" << endl;
     cin>>n;
     cout << "Inserire gli n valori da sommare:" << endl;
     for (i = 0; i < n; i++)
          { cin >> val;
          somma = somma + val;
          }
     cout << "somma = " << somma << endl;
     system ("Pause");
}
```

Questo programma è adattabile a più situazioni del precedente, perchè posso cambiare il numero di valori..

Istruzione while.

Esistono però delle situazioni reali in cui non è possibile conoscere il numero di valori prima di fare la somma (es. la cassiera del supermercato non conosce quanti sono i prodotti prima di cominciare a inserire i prezzi).
In questo caso l'istruzione iterativa si chiama while e la codifica è:

```
a=3;
while( a= =3)  //fino a quando a=3 ripeti le istruzioni tra le { }
    {…….....
     ……....
     cout<<"Inserisci 3 se vuoi ripetere il ciclo, altro per uscire";
     cin>>a;
     }
```

Bisogna prima inizializzare una variabile che serve come condizione per ripetere il ciclo; poi si inizia il ciclo che si ripeterà fino a quando a=3. Per poterlo interrompere, bisogna poter cambiare il valore di a (è il cin >>a interno). L'operatore della condizione può essere >,<, >=, <=, !=, = = o una qualunque espressione composta.

Se il problema è quindi sommare un numero di valori che non si può conoscere a priori il programma diventa:

```
// somma2.cpp Somma un numero di valori inseriti da tastiera non conosciuto a priori.

#include <iostream>
int i,val, a=3, somma = 0;    //le variabili possono essere inizializzate durante la dichiarazione
int main()
    { i=0;                    //è un contatore che serve per contare il numero di valori che non conosco.

     while( a==3)          // mentre a=3 ripeti le istruzioni tra le graffe.
        {cout << "inserire il valore" << endl;
        cin >> val;
        somma = somma + val;
        i=i+1;                              // il numero di valori è il numero di ripetizioni del ciclo
         cout<<"Inserisci 3 se vuoi ripetere il ciclo, altro per uscire";
        cin>a;
         }
    cout << "somma = " << somma << endl;
    cout << "il numero di valori è   = " << i << endl;    //nella i è contenuto il numero di valori inserito
    system ("Pause");
  }
```

Variabili strutturate

Tante volte ci sono molti dati dello stesso tipo (costi, numeri, iniziali di prodotti etc) e in questo caso si possono aggregare in una sola variabile chiamata **vettore (array).**

A differenza di una variabile, che contiene un singolo valore, un array indica una variabile (o meglio un contenitore) che fa riferimento ad un insieme **omogeneo** di valori/oggetti. Ogni array ha un **nome** al quale viene associato un **indice** che individua **i singoli elementi** dell' array. E'possibile definire array di tipo: carattere, intero, float, double etc..
Le proprietà fondamentali di un array sono:

1. Gli oggetti che compongono l'array sono denominati elementi;
2. Tutti gli elementi di un array devono essere dello stesso tipo;
3. Tutti gli elementi di un array vengono memorizzati uno di seguito all'altro nella memoria del PC e l'indice del primo elemento è 0;
4. Il nome dell' array è un variabile che rappresenta l'indirizzo di memoria del primo elemento dell' array stesso.

Per dichiarare un array (analogamente ad una qualsiasi variabile) in C++ è necessario definirne il tipo, seguito da un nome valido e da una coppia di parentesi quadre che racchiudono un'espressione costante rappresentante la dimensione massima dell' array. Es.
int prova[100]; // Un array di nome prova di 100 interi
char pippo [20]; // Un array di nome pippo di 20 caratteri
Si noti che all'interno delle parentesi quadre non è possibile, in fase di dichiarazione dell' array, utilizzare nomi di variabili, perché la dichiarazione serve a definire il numero di locazioni di memoria occupate e quindi occorre una quantità definita o una costante (potrebbe essere anche una lettera, ma definita come costante).

Un limite dei programmi di somma ricorsiva precedenti con for e while è che le istruzioni iterative non salvano i valori (cioè se devo sommare 10 numeri utilizzo una sola variabile es. **a** per inserire da tastiera i 10 numeri ; alla fine in **a** mi rimane l'ultimo numero inserito, i 9 precedenti non sono stati conservati o meglio sono stati sovrascritti)**.** I vettori risolvono questo problema : questo vuol dire che i valori sono conservati nella memoria del computer e si possono utilizzare per altre operazioni o per visualizzarli in una lista . Ripetiamo il programma con la somma di n valori inserendoli in un vettore e visualizziamo anche la lista dei valori.

```
// somma3.cpp Somma  numeri dati da tastiera. Visualizza la somma e l'elenco dei valori.

#include <iostream>
int i,n,val[50], somma = 0;// si indica un numero di elementi del vettore opportuno(dipende dalla situazione)
int main()
     {
     cout << "inserire quanti valori sommare" << endl;
     cin>>n;
     cout << "Inserire gli n valori da sommare:" << endl;
     for (i = 0; i < n; i++)
          { cin >> val[i]; //tra le [] adesso si inserisce i , così ad ogni giro i incrementa di 1 e incrementa di 1
                    // anche la posizione dell'elemento in memoria in cui si vuole salvare il valore
          somma = somma + val [i];
          }
     cout << "somma = " << somma << endl;
     cout<<"Lista dei valori"<<endl:       // titolo della lista
      for (i=0; i<n; i=i+1)              //istruzione iterativa per visualizzare i valori
          cout<<val [i]<<endl;     // endl  (end line) serve per andare a capo dopo aver scritto un valore,
                                   // se non ci fosse i valori verrebbero visualizzati attaccati su un'unica riga.
    system ("Pause");
   }
```

e lo stesso programma con il while:

```
/*somma5.cpp Somma un numero di valori inseriti da tastiera non conosciuto a priori. Visualizza la somma e
l'elenco dei valori.*/

#include <iostream>
int i,val[50], a=3 somma = 0;//si indica un numero di elementi del vettore opportuno ( adatto al problema)
int main()
      { i=0;                //è un contatore che serve per contare il numero di valori che non conosco.

       while( a==3)
          {cout << "inserire il valore" << endl;
           cin >> val[i];
          somma = somma + val[i];
          i=i+1;                              // il numero di valori è il numero di ripetizioni del ciclo
```

```
        cout<<"Inserisci 3 se vuoi ripetere il ciclo, altro per uscire";
        cin>a;
        }
    n=i;    /* devo salvare  il valore di i (contiene il numero di valori) in un'altra variabile n , perchè nel for
              successivo riazzero i e non so più quanti sono i valori.*/
    cout << "somma = " << somma << endl;
    cout<<"Lista dei valori"<<endl:
    for (i=0; i<n; i=i+1)
       cout<<val [i]<<endl;       // endl  (end line) serve per andare a capo dopo aver scritto un valore
    system ("Pause");
}
```

NB. La visualizzazione dell'elenco si può fare sempre con il for , perchè a questo punto del programma si conosce il numero di elementi.

Le variabili carattere e le stringhe

Ogni simbolo alfanumerico della tastiera corrisponde all'interno della CPU a un numero in binario a 8 bit secondo il codice ASCII (vedi tabella). Il tipo di variabile per un simbolo alfanumerico si chiama **char (** 1 byte). Per il PC i caratteri alfanumerici e i corrispondenti codici ASCII sono intercambiabili; solo nella visualizzazione (cioè sullo schermo) occorre distinguere. Es.

```
/*Visualizzazione di 12 caratteri  inseriti da tastiera e del corrispondente codice ascii   */

#include <iostream>
using namespace std;
char frase [12];  // dichiara un vettore di 12 caratteri
int i;
int main ( )
{    for (i=0;i<12; i++)
      {    cout<<("inserisci una carattere");
           cin>>frase[i];// inserisce  lettere maiuscole e minuscole e i numeri uno per volta
           cin.ignore();//cin.ignore(100,'\n');//ignora 100 caratteri o almeno tutti i caratteri fino al new line
                        //( il ritorno a capo)
      }
    for (i=0;i<12; i++) //visualizza  il carattere e il corrispondente ASCII in decimale
           cout<<frase[i]<<" ="<<(int)frase[i]<<endl;// andando a capo per  ogni carattere
system ( "Pause");
}
```

L'istruzione **cin.ignore( )**; serve a svuotare **il buffer di tastiera** cioè a ignorare i caratteri presenti nel buffer fino all'End of line dell'Invio (da un punto di vista alfanumerico Invio è dato da 2 caratteri ASCII Carriage return e End of line, ritorno a capo).
Il **buffer di tastiera** è una **memoria** che conserva i caratteri inseriti; se i caratteri inseriti devono finire in una variabile numerica ad es. un int o un float non ci sono problemi, perchè il programma prende dal buffer tutti i numeri fino alla pressione di Invio. Se si inserisce per sbaglio un carattere alfanumerico questo non viene accettato, perchè non è del tipo richiesto ,un numero.
Invece se si tratta di inserire caratteri alfanumerici, il programma non distingue più tra i caratteri e l'Invio e conserva in memoria tutti i caratteri digitati .Se nel programma precedente si inserissero per errore 2 caratteri insieme ,il programma, nel giro successivo del for , non si fermerebbe al cin per aspettare che venga digitato un nuovo carattere, ma userebbe il carattere rimasto nel buffer di tastiera.

Tabella codice ASCII esteso

Nella tabella seguente ci sono tutti i simboli alfamumerici presenti sulla tastiera (e anche altri che si possono visualizzare con Alt+il numero ASCII corrispondente sul tastierino numerico); ad ogni simbolo corrisponde un numero da 0 a 255 in decimale o da 0 a FF in esadecimale.

N.B. $256 = 2^8$ è il numero max che si può scrivere in un byte (8 bit) ed è la dimensione del tipo **char.**

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`	128	80	Ç	160	A0	á	192	C0	└	224	E0	α
1	01	Start of heading	33	21	!	65	41	A	97	61	a	129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
2	02	Start of text	34	22	"	66	42	B	98	62	b	130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
3	03	End of text	35	23	#	67	43	C	99	63	c	131	83	â	163	A3	ú	195	C3	├	227	E3	π
4	04	End of transmit	36	24	$	68	44	D	100	64	d	132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
5	05	Enquiry	37	25	%	69	45	E	101	65	e	133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
6	06	Acknowledge	38	26	&	70	46	F	102	66	f	134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
7	07	Audible bell	39	27	'	71	47	G	103	67	g	135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
8	08	Backspace	40	28	(	72	48	H	104	68	h	136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i	137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j	138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k	139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l	140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m	141	8D	ì	173	AD	¡	205	CD	═	237	ED	ø
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n	142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o	143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
16	10	Data link escape	48	30	0	80	50	P	112	70	p	144	90	É	176	B0	░	208	D0	╨	240	F0	≡
17	11	Device control 1	49	31	1	81	51	Q	113	71	q	145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
18	12	Device control 2	50	32	2	82	52	R	114	72	r	146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
19	13	Device control 3	51	33	3	83	53	S	115	73	s	147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
20	14	Device control 4	52	34	4	84	54	T	116	74	t	148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u	149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v	150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
23	17	End trans. block	55	37	7	87	57	W	119	77	w	151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
24	18	Cancel	56	38	8	88	58	X	120	78	x	152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
25	19	End of medium	57	39	9	89	59	Y	121	79	y	153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z	154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{	155	9B	¢	187	BB	╗	219	DB	▌	251	FB	√
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|	156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}	157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~	158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	⌂	159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	□

In realtà i caratteri, oltre che essere usati come semplici variabili, vengono per lo più strutturati in **vettori di caratteri** o stringhe per scrivere parole o frasi. La differenza tra un vettore e una stringa è che la **stringa** ha un carattere in più alla fine , il carattere di fine stringa **'\0'** che occupa un elemento del vettore. . Il carattere di fine stringa è utile perché permette di individuare quando la stringa è finita e può essere utilizzato per le operazioni sulle stringhe che vengono che sono già preconfezionate nella libreria apposita **cstring.**

La stringa può essere dichiarata in differenti modi:

```
char frase [30]; // dichiara una stringa di 30 elementi 29+ '\0'
char frase []= "Oggi piove e la temperatura è bassa"/* dichiara una stringa , assegna ai singoli elementi
                                    i caratteri della frase tra " " e chiude la stringa con il carattere '\0' */
char frase[50]= "Oggi piove e la temperatura è bassa" /* dichiara una stringa e assegna ai singoli elementi
        i caratteri della frase tra " " e chiude la stringa con il carattere '\0', ma è poi possibile aggiungere in
        quella stringa altre parole fino ad arrivare a 49 caratteri */
```

NB. Una stringa è definita tra" ", un carattere tra ' '.
Per inserire e visualizzare una stringa si può usare di nuovo cin e cout, ma nell'istruzione non si definisce il numero di elementi:

```
cin>>frase;
cout<<frase;
```

l'istruzione **cin non accetta lo spazio bianco,** se lo trova termina la stringa e lascia il resto dei caratteri nel buffer di tastiera, creando problemi per i cin successivi .Per evitare questo problema occorre un **cin.ignore(20,'\n')** o, meglio, al posto di cin l'istruzione **cin.getline(frase,50)** (50 è il numero di caratteri della stringa frase).
Utilizzando la libreria **cstring** è possibile fare più operazioni sulle stringhe , ad esempio unire più stringhe, calcolarne la lunghezza, confrontare 2 stringhe per vedere se sono uguali.

Esempio:

```
/*conta il numero di caratteri di una  frase inizializzata nella dichiarazione, ne inserisce due da tastiera, le
confronta per vedere se sono uguali e unisce la prima stringa inserita a quella data nella dichiarazione */

#include <iostream>
#include <cstring>  // libreria di operazioni sulle stringhe
using namespace std;
char frase [50]="se il tempo è bello e il sole splende";
char dimmi[20], prova[20];
int a,b,c;
int main ( )
        {cout<<frase<<endl;
        c=strlen( frase);                //conta il numero di caratteri eccetto'\0'
        cout <<endl<<"lunghezza=  "<<c<<endl;
        cout<<"completa la frase";
         cin.getline(dimmi,20);             // acquisisce  una frase anche con  gli spazi bianchi
        b=strlen( dimmi);                  //conta il numero di caratteri eccetto\0
        cout <<endl<<"b=  "<<b<<endl;
         cin.ignore (20,'\0');         /*svuota il buffer di tastiera  se la frase inserita precedente fosse stata
                                      troppo lunga* /
        cout<<"scrivi la frase da confrontare con la prima";
         cin.getline(prova,20);            // acquisisce anche gli spazi bianchi
        a=strcmp( dimmi,prova);    //confronta i caratteri, se sono uguali, a=0
        cout<<"a= "<<a<<endl;
        if(a==0 )                          //a=0 le due frasi sono uguali
```

```
            cout <<endl<<" sono proprio uguali");
      else
          if (a==-1)
                {cout<<endl<<" dimmi è più corta di prova";
                 cout<<endl<<"a="<<a;
                 }
            else
               { cout<<endl<<" dimmi è più lunga di prova";
               cout<<endl<<"a="<<a;
               }
       cout<<endl<<"il confronto e' fatto sul corrispondente decimale del      codice ASCII"<<endl;
       strcat(frase,dimmi);                    //aggiunge dimmi in coda a frase
      cout<<"la frase completa è";
      cout<<endl<<frase;
      system ("Pause");
}
```