

## Per una più semplice programmazione in C++.

Venezia, 1998  
ultima revisione gen 2009 .

Questo modulo è stato realizzato all'ITIS "C.Zuccante" di Mestre - Venezia - utilizzando l'ambiente di sviluppo DEVC++ versione 5 ( <http://www.bloodshed.net> ), per gli studenti che imparano a programmare con il linguaggio C++, sotto SO.WindowsXP o superiori. Può essere distribuito liberamente purché non se ne alteri il contenuto. Sono graditi tutti i tipi di suggerimento per migliorare il prodotto.

Originariamente il modulo era stato realizzato con il compilatore Borland C++ 3.1 e la libreria BGI e lavorava a 16 bit sotto DOS, ora, con il compilatore *open source* utilizzato dal DevCpp si può lavorare sotto Windows e, inoltre, sono stati superati molti dei limiti del vecchio compilatore.

Questa libreria contiene una serie di tipi per realizzare contenitori di informazioni: Vettori e Matrici per interi e numeri in virgola mobile, Stringhe di caratteri (oggetti molto flessibili) e gli Insiemi di caratteri.

A questi contenitori si aggiungono alcuni utili strumenti: tipi od oggetti predefiniti.

Il tipo Tartaruga, simile all'oggetto Tartaruga del linguaggio Logo, ci permette di realizzare oggetti Tartaruga, ai quali possiamo inviare una nutrita serie di comandi.

Il tipo Cronometro ci permette di realizzare oggetti per misurare il tempo trascorso tra una operazione e l'altra e funziona con gli stessi comandi dei cronometri che tutti conosciamo.

Il tipo Bottone ci permette di realizzare oggetti che reagiscono quando vengono premuti dal Mouse.

L'ambiente grafico lavora in coppia con l'ambiente testo, per cui possiamo avere due finestre

Oltre ad una serie di definizioni di utili costanti e la ridefinizione di operatori od identificatori che rendono più semplice l'approccio alla programmazione, la libreria contiene un sofisticato meccanismo di gestione degli errori ed invio di suggerimenti al programmatore. Per esempio vi sono i controlli per l'indicizzazione fuori dai limiti dei contenitori di informazioni (range error); suggerimenti per quando si passano per copia oggetti troppo grandi alle funzioni; e altro ancora.

Dopo aver installato l'ambiente di sviluppo DevCpp, per poter usare il modulo winEasy\_C++, bisogna inserire alcuni file, sviluppati dall'autore, in particolari cartelle (o directory) come di seguito indicato:

Cartella **Template** i file :

**Easy\_C.cpp.txt** , **Easy\_Cpp.ico**, **Easy\_Cpp.project.ico**, **Easy\_Cpp.template**

Cartella **lib** i file libreria: "**lib\_easy\_c7.a**"

Cartella **include** il file d'inclusione: "**easy\_c7.h**"

Per il buon funzionamento dei programmi è consigliabile aprire un progetto di tipo "console application" (o meglio, se è presente, "*introduction – Easy\_Cpp*" ), con indicati i percorsi delle due librerie utilizzate (Project - Project Options - Parameters – Linker ) e ricordatevi di inserire nel vostro file sorgente, col quale state lavorando, la direttiva di inclusione `#include <easy_c7.h>`.

Su richiesta viene fornito un file compresso "dev\_cpp\_aggiornamenti" che contiene: cartelle, i file menzionati, esempi, file di help e documentazione: basta scompattarlo e ricopiare la struttura cartelle-file direttamente sulla cartella dev-cpp dell'installazione di DevC++, dentro ; alcuni file presenti nell'installazione originale verranno sostituiti.

Per il funzionamento con altri compilatori C++ si può realizzare autonomamente la libreria, prendendo spunto dal file di inclusione e dalle Classi descritte nel libro: Renato Conte "Il mondo degli oggetti: Programmazione in C++" - ed. Libreria Progetto - PADOVA.

Per segnalazioni di errori, suggerimenti e per avere aggiornamenti rivolgersi:

**Renato Conte**  
e-mail: [conte@dsi.unive.it](mailto:conte@dsi.unive.it)

## Valori predefiniti

PiGreco

### Colori

Nero, GrigioScuro, Grigio, GrigioChiaro, Bianco, BluScuro, Blu, Azzurro, BluChiaro, Cyan, Celeste, Verde, VerdeChiaro, Rosa, Arancio, Rosso, RossoChiaro, Magenta, Violetto, Marrone, Giallo, Salmone, Argento;

### Codice Tasti

KEY_HOME	KEY_UP	KEY_PGUP	KEY_LEFT	KEY_CENTER	KEY_RIGHT
KEY_END	KEY_DOWN	KEY_PGDN	KEY_INSERT	KEY_DELETE	KEY_F1
KEY_F2	KEY_F3	KEY_F4	KEY_F5	KEY_F6	KEY_F7
KEY_F8	KEY_F9				

### *Sinonimi di simboli o gruppi di simboli*

I seguenti identificatori possono esser utili per un principiante

<b>AND</b>	sostituisce	<b>&amp;&amp;</b>
<b>OR</b>	sostituisce	<b>  </b>
<b>NOT</b>	sostituisce	<b>!</b>
<b>BEGIN</b>	sostituisce	<b>{</b>
<b>END</b>	sostituisce	<b>}</b>

<b>InizioProgramma</b>	sostituisce	<b>int main() {</b>
<b>FineProgramma</b>	sostituisce	<b>return 0; }</b>
<b>PerSempre</b>	sostituisce	<b>for(;;)</b>

### Passaggio parametri nelle funzioni

<b>PerIndirizzo</b>	sostituisce	<b>&amp;</b>
<b>PerReferenza</b>	sostituisce	<b>&amp;</b>
<b>PerCopia</b>		
<b>PerValore</b>		

```

#include <easy_c7.h>

void Scambia( int PerReferenza A, int PerReferenza B )
{
    int C=A;      // creo una variabile di supporto
    A= B;
    B= C;
}

InizioProgramma
int Min, Max; // dichiaro due variabili intere

cout << "inserisci due valori interi: "; // mando a video (cout) la scritta indicata
cin >> Max >> Min; // i valori numerici inseriti da tastiera vanno nelle variabili Max e Min

if (Max < Min ) Scambia(Max, Min);
cout << " I valori, in ordine crescente, sono: " << Min << " " << Max;
FineProgramma

```

## Funzioni per la gestione Video, Tastiera

**void WaitESC();** *// aspetta che l'utente prema <ESC> (l'attesa viene segnalata a video)*

**void WaitESC( string frase );**

*// aspetta che l'utente prema <ESC> e mette a video la "frase" (utile per il debug)*

**int KeyCode();**

*aspetta che l'utente prema un tasto restituendo il codice interno. Se gli 8 bit piu' significativi sono uguali a zero, allora gli 8 bit meno significativi rappresentano il carattere premuto sulla tastiera; altrimenti keyCode restituisce il codice esteso del tasto negli 8 bit piu' significativi*

**int Getch();** *// legge qualsiasi tasto da tastiera*

**int Kbhit();** *// 0 se nessun tasto e' premuto; diverso da zero se un tasto e' premuto*

**void clrscr();** *// pulisce lo schermo testo*

**void ClearScreen();** *// sinonimo di clrscr*

**void gotoxy(int x, int y);** *// posiziona il cursore testo nella posizione x, y (1,1 alto a sinistra)*

```
/**
 * programma di prova che legge un tasto dalla tastiera
 * per scoprire il codice del tasto
 */

#include <easy_c7.h>

InizioProgramma

cout << endl<< "Prova lettura tasti speciali e non (per uscire premere Q)" << endl;

PerSempre
{
    if ( Kbhit())          // se e' premuto un tasto della tastiera
    { int c= KeyCode(); // leggo il codice da tastiera
      cout << "codice:" << c
            << " corrispondente carattere (se c'e):" << char(c)<< endl;

      // se viene inserito il carattere Q oppure il tasto FINE usciamo dal ciclo
      if (c== 'Q' OR c=='q' ) break;
    } // if

} //FinePerSempre

WaitESC("fine programma");

FineProgramma
```

## Oggetti Cronometro

```
void Start();// parte il cronometro o riparte dopo uno Stop( )
void Stop();// ferma il cronometro
void Reset();// azzera il cronometro
float Time(); oppure Tempo(); // restituisce il valore (float) corrente del cronometro
```

Esempio:

```
#include <easy_c7.h>
int main( )
{
    Cronometro CX; // dichiarazione di un cronometro di nome CX

    CX.Reset( );
    CX.Start( );
    int dato;
    cin >> dato;
    CX.Stop( );
    cout << CX.Time( ); // visualizza il tempo impiegato dall'utente ad inserire l'intero
    CX.Start( ); // riprende la misura del tempo
    cin >> dato;
    CX.Stop( );
    cout << CX.Time( ); // visualizza il tempo totale impiegato dall'utente ad inserire due interi

    WaitESC("fine programma");
    return 0;
}
```

## Oggetti ColoreRGB

Un colore e' un oggetto con il quale possiamo fare alcune operazioni tipiche dei colori naturali. Si puo' usare in ambiente grafico per la gestione del colore del pennello della Tartaruga, ma si presta ugualmente in ambiente testo per lo studio del comportamento delle componenti dei colori come illustra l'esempio sotto riportato. Ogni componente RGB puo' assumere valori compresi tra zero e 255.

Esempio:

```
#include <easy_c7.h>
....
// definizione di un colore di nome ca arancione tramite un intero scritto in forma esadecimale
ColoreRGB ca (0xffff00);

// definizione di un colore di nome cb arancione chiaro tramite le componenti RGB
ColoreRGB cb (255,255,100);

cout << ca << " " <<cb <<endl; // scrittura a video dei componenti dei due colori

ColoreRGB cx= ca - cb; // creazione di un colore differenza di altri due colori

ColoreRGB cc= - cx; // creazione del colore complementare di cx
```

# CONTENITORI DI INFORMAZIONI

## VettoreInt e VettoreFloat *vettore di numeri (interi o reali)*

```
int N = 100;
VettoreInt vettA, vettB; // dichiarazione di un vettore
vettA.Dimensione( N ); // impone la dimensione 100 al vettore

for (int i=0; i< N-1; i++) // lettura di tutti gli elementi (riempimento del vettore da tastiera)
{ cin >> vettA[i]; }

vettB = vettA; // assegnamento di tutto un vettore
vettB[10] = 123; // assegnamento ad un solo elemento del vettore
int elem = vettA[ 7]; // restituzione di un elemento del vettore
int Lung = vettA.Dimensione( ); // restituisce la dimensione del vettore

// visualizzazione di tutti gli elementi a video
for (int k=0; k< Lung -1; k++) cout << vettA[k];
```

*/// Ordinamento a bolle semplice per un vettore di interi*

```
void BubbleSort( VettoreInt & Vett )
{
    int N=Vett.Dimensione( ); // chiedo la dimensione del vettore
    for (int i=0; i< N-1; i++) // i" contatore "passate"
    {
        for (int k=0; k< N-i-1; k++)
        {
            if ( Vett[k] > Vett[k+1]) // Scambia: Vett[i] <--> Vett[i+1]
            { int Buff=Vett[k]; Vett[k]=Vett[k+1];Vett[k+1]=Buff; }
        }
    }
}

int main( )
{
    VettoreInt V;
    int M=100;
    V.Dimensione(M);
    for (int i=0; i<M; i++) V[i]= random(M); // riempio il vettore con valori interi casuali
    cout <<"Vettore disordinato: "<< V;
    BubbleSort(V);
    cout <<"Vettore ordinato: "<< V;
    return 0;
}
```

## **MatriceInt e MatriceFloat    vettore bidimensionale di numeri (interi o reali)**

```
int Righe=20, Col=10;
MatriceInt Tabella, MatA; // dichiarazione
MatA.Dimensioni ( Righe, Col )      // impone le dimensioni alla Matrice

for( int r=0; r< Righe; r++)  // riempimento di una matrice di interi
for( int c=0; c< Col; c++)
{ MatA[r][c]= random(100); } // random(N) restituisce un numero intero casuale tra 0 e N-1

Tabella = MatA; // assegnamento di tutta una matrice
Tabella [ 5][ 7 ]= 73; //   modifica  di un solo elemento
int elem = Tabella[ 3 ][ 9]; // osservazione di un elemento
```

## SetCaratteri    *contenitore*    *“Insieme di caratteri”*

```
SetCaratteri();    // Set vuoto:    SetCaratteri A;  
SetCaratteri(char); // inizializzazione con un insieme di caratteri:    SetCaratteri B="asdfg";
```

### Operatori per gli insiemi di caratteri

```
+    oppure    Unione    // A =A+B;  
.    oppure    Intersezione    // A =A*B;  
-    oppure    Differenza    // A =A-B;  
!    oppure    Complemento    // A =!A;  
<=   oppure    IN    // appartenenza    if ( 'x' IN A ) ...  
+=   // aggiunta di un elemento    A += 'x';  
+=   // aggiunta di una sequenza di caratteri    B += "iuyuioiu";  
<=   // inclusione in senso largo  
<    // inclusione in senso stretto  
==   // equivalenza tra due insiemi  
cout <<    // visualizzazione del contenuto di un insieme:    cout << A;
```

```
SetCaratteri SetA="ABXY", SetB="ABCXYZ", SetC="ADHXW";  
cout << "SetA"<< SetA << endl  
    << "SetB"<< SetB << endl  
    << "SetC"<< SetC << endl;
```

```
cout << "Intersezione SetA, SetB =" << (SetA, SetB) << endl;  
cout << "Unione SetA + SetB =" << (SetA + SetB) << endl;  
cout << "Differenza SetA - SetB =" << (SetA - SetB) << endl;  
if (SetA < SetB) cout << "Inclusione stretta SetA < SetB \n";  
if (SetA <= SetB) cout << "Inclusione larga SetA <= SetB \n";  
else cout << " il SetA non include il SetB \n";
```

```
if ( 'X' IN SetA ) cout << "Il Carattere X e' presente nel SetA";
```

```
if (SetA < SetC) cout << "Inclusione stretta SetA < SetC \n";  
if (SetA <= SetC) cout << "Inclusione larga SetA <= SetC \n";  
else cout << " il SetA non include il SetC \n";
```

Uscita a video:

```
SetA [ ABXY ]  
SetB [ ABCXYZ ]  
SetC [ ADHWX ]  
Intersezione SetA, SetB =[ ABXY ]  
Unione SetA + SetB =[ ABCXYZ ]  
Differenza SetA - SetB =[ ]  
Inclusione stretta SetA < SetB  
Inclusione larga SetA <= SetB  
Il Carattere X e' presente nel SetA  
il SetA non include il SetC
```



# L'ambiente grafico ed i suoi oggetti

**void Grafica::FinestraVisibile( float Xi, float Ys, float Xs, float Yi );** sinonimo: **SetLimits**

Lo spazio video viene imposto dalla finestra che guarda al piano cartesiano.

Gli estremi degli assi cartesiani vengono imposti nel seguente modo:

Asse X: Xi -----> Xs      Asse Y: Yi-----> Ys

Angolo sinistro alto Xi,Ys - angolo destro basso Xs,Yi

Il rapporto Altezza/Larghezza della finestra dovrebbe essere  $\frac{3}{4}$  altrimenti la scala delle ascisse risulta essere diversa dalla scala delle ordinate. Per default il piano della Tartaruga viene impostato con lo zero al centro:

Grafica::FinestraVisibile(-319, +240, +320, -239);

Si possono orientare opportunamente gli assi. Per far coincidere le coordinate tartaruga con le coordinate pixel dello schermo: Grafica::FinestraVisibile(0,0, 639, 479);

## Tartaruga

La tartaruga si muove su un piano cartesiano XY inizialmente centrato sullo schermo.

**void Tana();** sinonimo: **Home**

// La Tartaruga si colloca al centro del piano cartesiano (0,0), con direzione 90 gradi (asse Y)

**void PulisciSchermo ( ColoreRGB );** sinonimo: **ClearScreen**

// La Tartaruga pulisce lo schermo col colore C (per default colore corrente)

**void TempoPasso( int T );** sinonimo **Time( int T )**

// Per ogni passo la Tartaruga impiega un tempo di T millisecondi

**void Velocita( int V );** sinonimo: **Speed**

// è all'incirca il reciproco del comando precedente: più alto è il valore V e più veloce sarà la Tartaruga

**void AlzaPennello()**  sinonimi: **Up** , **Su**

// alza il pennello (non disegna, attivo finché non lo abbassa)

**void AbbassaPennello()**  sinonimi **Giu** , **Down**

// abbassa il pennello (disegna, attivo finché non lo alza)

**void VaiAvanti( float P );** sinonimi: **Avanti**    **A**

// avanza di P passi (disegnando se pennello giù)

**void VaiIndietro( float P );** sinonimi: **Indietro**    **I**

// indietreggia di P passi (disegnando se pennello giù)

**void GiraDestra( float G );** sinonimi: **Destra**    **D**

// ruota la direzione attuale della tartaruga di G gradi a destra (se G positivo)

**void GiraSinistra( float G );** sinonimi: **Sinistra**    **S**

// ruota la direzione attuale della tartaruga di G gradi a sinistra (se G positivo)

**float Direzione( ) const;** restituisce la direzione corrente in gradi( a partire dall'asse X )

**void Direzione( float A );** // imposta la direzione assoluta di movimento della tartaruga

**void CambiaColorePennello(ColoreRGB C);** sinonimi: **Colore**    **Color**

// imposta il colore di scrittura e/o disegno a C )

**ColoreRGB ColorePennello ( ) const;** restituisce il colore di scrittura e/o disegno

**ColoreRGB ColoreFondo ( ) const;** restituisce il colore del fondo impostato da pulisci

**ColoreRGB ColoreSottoLaCoda ( ) const;** restituisce il colore del fondo presente in quel punto sotto la coda

**void UsaPennelloLargo( );**

**void UsaPennelloSottile( );**

```

float X( ) const; //funzione che restituisce la coordinata X corrente della tartaruga
float Y( ) const; //funzione che restituisce la coordinata Y corrente della tartaruga

// procedura che riempie le variabili X e Y con i valori delle coordinate della posizione corrente della Tartaruga
void DoveSei( float & X, float & Y);

void MettiUnPunto ( float Xo, float Yo ); sinonimi: PointXY , // mette un punto in Xo Yo

void MettiUnPunto ( float X, float Y, int Colore );   sinonimi: FastPointXY
// mette un punto in Xo Yo colorato

void Vai( float Xo, float Yo); sinonimo: GoToXY
// pone la tartaruga nella posizione Xo Yo se il pennello e' abbassato la Tartaruga traccia una linea

void Salta( float Xo, float Yo);  sinonimo: Jump
// pone la tartaruga nella posizione Xo Yo senza lasciare traccia col pennello

void SpostatiDi( float DeltaX, float DeltaY);  // Spostamento relativo di DeltaX e di DeltaY

float MaxX( ) const; // restituisce la dimensione X massima dello schermo

float MaxY( ) const; // restituisce la dimensione Y massima dello schermo

float MinX( ) const; // restituisce la dimensione X minima dello schermo

float MinY( ) const; // restituisce la dimensione Y minima dello schermo

void DirezioneAlPunto( float Xo, float Yo );   sinonimi: GiraVerso  SetDirectionToPoint
// imposta la direzione dal punto attuale verso Xo, Yo

void RiempiColore(ColoreRGB ColoreRiemp, ColoreRGB ColoreBordo ); sinonimo: Fill
// colora con ColoreRiemp la figura geometrica (disegnata con ColoreBordo ) che racchiude la tartaruga

void RiempiColore( );
// La tartaruga "tenta" di colorare la figura chiusa che ha appena disegnato, usando come colore di
// riempimento quello della figura. Se la figura non e' chiusa gli effetti sono imprevedibili

void CambiaColorePennello( ColoreRGB c);

void Mostrati();           sinonimo: Show // mostra il simbolo della tartaruga
void Nasconditi();        sinonimo: Hide // nasconde il simbolo della tartaruga

void Cerchio( double R );

void Griglia(ColoreRGB ColoreAssi=Rosso, ColoreRGB ColoreGriglia=Verde );

void GraficoFunzione
( Espressione E, ColoreRGB ColoreGrafico=Bianco, ColoreRGB ColoreAssi=Trasparente, ColoreRGB
ColoreGriglia=Trasparente );

void GraficoFunzione
( double (*F)(double) , ColoreRGB ColoreGrafico, ColoreRGB ColoreAssi=Trasparente, ColoreRGB
ColoreGriglia= Trasparente);

sinonimo: PlotFunction
// disegna il grafico di una funzione col colore specificato da ColoreGrafico, con una Griglia opzionale.

```

Esempi:

```
T.GraficoFunzione( sin, Rosso );
T.GraficoFunzione( FuncMia, Verde ); // "FuncMia" funz. definita dall'utente
T.GraficoFunzione( 12*X*X+123); //
```

```
double F(double x)
{   return 200*sin(x/80)+ 100*cos(x/40)*sin(3*x+3);   }

int main()
{   Tartaruga T;
    T.GraficoFunzione(F, Giallo, Rosso, Verde); // visualizzazione del grafico della funzione

    Bottone B1(-10,10,Rosso, "Prova bottone" ); // creazione di un oggetto bottone
    while(! B1.PremutoeRilasciato()); // attende che l'utente prema il bottone

    return 0;
}
```

```
int Interpreta( char * Comandi ); // interpreta la stringa Comandi.
// in caso di errore restituisce la posizione del carattere che ha causato errore, altrimenti restituisce 0
// interpreta la stringa Comandi del tipo : " A(100) Destra 90 indietro12.3 "
// I comandi riconosciuti sono: "su", "up", "giu", "down", "a", "i", "d", "s", "avanti", "indietro", "destra", "sinistra",
// "color", "colore", "clear", "pulisci", "gotoxy", "vai", "tana", "home", "mostra", "show", "nascondi", "hide",
// "riempicolore" "fill"
```

```
T.Interpreta(" A 100 Destra 90 indietro (12.3) " );
```

```
Operatore di flusso: << // Scrittura su video della maggior parte degli oggetti di tipo primitivo
OggettoTartaruga << Caratteri; // scrive a video la stringa o il carattere
```

```
Operatore di flusso: >> // lettura da tastiera per la maggior parte degli oggetti di tipo primitivo
// L'oggetto "Tartaruga" legge da tastiera riportando il contenuto sia sullo schermo sia nel parametro
// ( lettura con eco ).
// Sono a disposizione alcuni comandi per l'editing sulla linea di immissione:
// <backspace>, <Del>, le frecce <Dx>, <Sx> e il tasto <Ins>.
// Se si preme il tasto <ESC>, la Tartaruga termina la lettura e restituisce 0 nel caso di parametro numerico
// non inserito oppure la stringa originale.
// Utilizzando il comando LeggiStringa e' necessario assicurarsi che esista lo spazio sufficiente a
// contenere la stringa: il numero L di caratteri che
// al massimo si puo' immettere in S comprende anche '\0' che viene inserito quando si preme il tasto <ENTER>.
```

```
bool FuoriSchermo( ) const;
```

```
void UsaPennelloLargo( );
```

```
void UsaPennelloSottile( );
```

Esempio: utilizzo della Tartaruga

```
#include <easy_c7.h>

void SpiraleCrescente( Tartaruga & T, float L )
{ if ( L < 500 ) // se il lato disegnato non supera il valore 500
  {   T.Avanti( L );
      T.Destra( 12);
      SpiraleCrescente( L*1.02 ); // chiamata ricorsiva
  }
}

void Poligono(Tartaruga & T, float L , int Nlati ) // disegna un poligono regolare con Nlati
{ for(int i=0; i< Nlati; i++) { T.Avanti(L); T.Destra(360.0/ Nlati); }

int main( )
{ Tartaruga T; // creazione dell'oggetto T di tipo Tartaruga
  SpiraleCrescente(T, 10); // disegno di una spirale ricorsiva
  T.PulisciSchermo(Rosso);
  Poligono(T, 100, 7); // disegna ettagono con il lato lungo 100
return 0;
}
```

Esempio: utilizzo di più oggetti di tipo Tartaruga:

```
#include <easy_c7.h>

void AZonzo( ) // N Tartarughe vanno a zonzo per lo schermo
{ const N =7;
  Tartaruga T[N]; // Contenitore (vettore) di Tartarughe

  for(int i=0; i< N; i++) // a ciascuna Tartaruga imponi:
  { T[i].TempoPasso(4); // di essere lenta (un passo 4 millisecondi);
    T[i].Su( ); // di tirare su il pennello
  }

  do{ // muovi le Tartarughe
    for(int i=0; i<N; i++)
    { T[i].Avanti(random(50));
      if (T[i].FuoriSchermo( )) T[i].Indietro(55);
      if (random(5)==0) T[i].Destra(random(100)); // ogni tanto gira
    }
  } while ( !kbhit( ) ); // Fintantoche` non e` premuto un tasto

}/* AZonzo */

//-----
int main( )
{
  Tartaruga T; // creo un oggetto Tartaruga di nome T
  T.PulisciSchermo(Blu); // Gli comando di pulire lo schermo col colore Blu
  T.Su( ); // Gli invio un messaggio per alzare il pennello
  T.Vai(-300,0); // Gli comando di andare nella posizione X=-300, Y=0
  WaitESC( ); // Il computer aspetta che l'utente prema il tasto ESC

  T <<"PROVA DELLE CAPACITA` DEGLI OGGETTI TARTARUGA con il modulo EASY_C3";
  // Invio alla Tartaruga T una frase da mettere sullo schermo

  AZonzo( );
  // chiamo una subroutine che crea un gruppo di Tartarughe che camminano nello spazio delimitato dallo schermo

  WaitESC( );
  return 0;
}
```

## **Bottone**

**Bottone**( float Xi, float Yi, ColoreRGB Colore, const char \*Frase=0);  
// Xi, Yi : coordinate del bottone (angolo alto a sinistra)  
// Colore: colore del bottone  
// Il parametro "Frase" e' una stringa che viene riportata all'interno del Bottone;

```
#include <easy_c7.h>

int main( )
{
    Tartaruga T;

    char * S = " PROVA BOTTONI ";

    T.salta(-200,220); T.<<S;
    T.Nasconditi();

    Bottone B1(-10,10,Rosso, "ciao a tutti " );

    WaitESC( );

    return 0;
}
```

## ESEMPI DI COMPITI IN CLASSE

```
// Si considerino due rettangoli con i lati paralleli agli assi cartesiani, la cui posizione è definita sul piano
// cartesiano da due punti per ogni rettangolo: Po (Xo, Yo) angolo alto-sinistra, Pi(Xi, Yi) angolo basso a destra.
// Creare una funzione in C++ che date le posizioni di due rettangoli trovi se il primo
// è interamente contenuto nel secondo
//----- soluzione:
bool Rettangolo_A_ContenutoNelRettangolo_B
(int XAo, int YAo, int XAi, int YAi, int XBo, int YBo, int XBi, int YBi)
{ // Po (Xo,Yo): alto-sinistra  Pi(Xi,Yi): basso-destra
  return (X Ao > XBo) && (X Ai < XBi)
        && (Y Ao < YBo) && (Y Ai > YBi);
}
```

```
// Creare una procedura per il riempimento di un Vettore di interi in maniera random garantendo
// che tutti gli elementi siano diversi tra loro.
// Scambiare poi l'elemento più piccolo del Vettore con l'elemento più grande.
// Si consiglia l'uso di procedure e/o funzioni.
```

```
//----- soluzione:
```

```
void Scambia( int & A, int & B)
{ int C= A; A= B; B= C; }
```

```
void MescolaVett( VettoreInt & V )
{ int N= V.Dimensione();
  for( int i=0; i<N; i++) Scambia( V[i], V[random(N)] );
}
```

```
void RiempiVettAcaso( VettoreInt & V )
{ int N= V.Dimensione();
  V[0]= random(10);
  for( int i=1; i<N; i++) // riempio con valori crescenti a caso
    V[i]= random(10)+ 1 + V[i-1];
  MescolaVett(V);
}
```

```
int PosMax( const VettoreInt & V )// trova la posizione del valore Massimo
{ int N= V.Dimensione();
  int iMax= 0;
  for( int i=0; i<N; i++) { if ( V[i] > V[iMax] ) iMax= i; }
  return iMax;
}
```

```
int PosMin( const VettoreInt & V ) // trova la posizione del valore Minimo
{ int N= V.Dimensione();
  int iMin= 0;
  for( int i=0; i<N; i++) { if ( V[i] < V[iMin] ) iMin= i; }
  return iMin;
}
```

```
int main()//===== prova soluzione =====
{
  VettoreInt Vx;
  Vx.Dimensione(20);
  RiempiVettAcaso( Vx );    cout << Vx << endl;
  int s= PosMax( Vx );      int i= PosMin( Vx );
  Scambia( Vx[i], Vx[s] );
  cout << Vx << endl;
  return 0;
}
```

```

// Disegnare, utilizzando i comandi della Tartaruga, una serie di poligoni regolari concentrici
// aventi tutti la stessa misura del lato, ma numero di lati crescente:
// triangolo, quadrato, pentagono, esagono, ... e così via fino ad N lati.

//-----soluzione:
void PoligoniConcentrici( float L, int N )
{
    // Disegna poligoni regolari concentrici di lato L, con numero di lati N
    Tartaruga T;
    T.GraficoFunzione(sin, Rosso); // per disegnare gli assi cartesiani e controllare la centratura
    T.Nasconditi(); T.Su();
    for( int p= 3; p<= N; p++ )
    {
        float alfa= 180- 360.0/p; // angolo interno del poligono
        float R= (L/2)/ cos( (alfa/2)*PiGreco/180 ); // Raggio cerchio circ.
        T.Avanti(R);
        T.Destra(alfa/2+360.0/p);
        T.Giu();
        for(int i=0; i<p; i++) { T.Avanti(L); T.Destra(360.0/p); }
        T.Su();
        T.Sinistra(alfa/2+360.0/p);
        T.Indietro(R);
    } //for-p
}

WaitESC();
}/*PoligoniConcentrici*/

```



```

#include <easy_c7.h>
// Consideriamo la serie di Fibonacci:
// 0, 1, 1, 2, 3, 5, 8, 13, 21, 34...
// 0 1 2 3 4 5 -
// Ciascun numero, dopo il secondo, e' costituito dalla somma dei primi due

//-----
//funzione ricorsiva che calcola l'ennesimo valore di Fibonacci
// si fa uso di un parametro di uscita "nRic" per contare le due chiamate ricorsive

double fibR( int n, double & nRic )
{ if ( n<2 )
  { return n; // restituisco l'ennesimo valore di Fibonacci
  }
  else { nRic= nRic+2; // incremento il contatore di chiamate ricorsive
        return fibR(n-1, nRic) + fibR(n-2, nRic); // chiamate ricorsive
  }
} //fibR

//-----
// Soluzione iterativa al problema di Fibonacci
double fib( int n )
{ if (n<2) return n; // fib(0) e fib(1)

  double a=0; // penultimo intero
  double b=1; // ultimo intero
  double ris= b+a; // somma dei due precedenti: fib(2)

  for( int i=2; i< n; i++ )
  { a= b; // sostituisco il penultimo con l'ultimo
    b= ris; // sostituisco l'ultimo col risultato
    ris= b+a; // calcolo il nuovo risultato: fib(i+1)
  }
  return ris;
}
//----- programma di prova Fibonacci -----
int main()
{ ClearScreen(Blu);

  double nRic=0; // contatore di chiamate ricorsive di Fibonacci

  for(int i=10; i<32; i++)
  cout<< "fib("<<i <<" ) " << fib(i) <<" " << fibR(i, nRic)<<"   chiamate ricorsive:" << nRic<< endl;

  cout <<"-----"<<endl;
  Cronometro CK;
  int N= 40;
  CK.Reset();   CK.Start();
  cout << fib(N);
  CK.Stop();
  cout << " tempo fib(" << N <<" ) sec." <<CK.Time()<< endl;
  cout <<"-----"<<endl;

  nRic= 0;
  CK.Reset(); CK.Start();
  cout << fibR( N, nRic );
  CK.Stop();
  cout << " tempo fibR(" << N <<" ) sec." <<CK.Time()<< endl;
  WaitESC();
  return 0;
}

```

# La classe `std::string`

(principali operazioni)

La libreria standard del C++ mette a disposizione la classe `std::string` per la gestione delle stringhe come normali oggetti (quindi trasferibili per copia nelle chiamate di funzioni). Per poter utilizzare questa classe è necessario includere il file `<string>`.

In seguito faremo riferimento alla classe `std::string` con il semplice nome `string`. In verità il prefisso `std::` viene utilizzato per essere espliciti che stiamo utilizzando la classe `string` standard e non qualche altra classe `string`. Per evitare di dover porre il prefisso `std::` prima della parola `string`, possiamo inserire il comando all'inizio del nostro codice sorgente:

```
using namespace std;
```

ATTENZIONE: La libreria `<string.h>` è una libreria standard del C di contenuto completamente diverso da quello analizzato in questo tutorial.

## Dichiarazione e definizione

Per definire un oggetto di tipo `string` si opera come se si trattasse di un tipo predefinito:

```
string nome;  
string matricola("1234");  
const string colore = "Rosso";
```

Nel momento in cui viene dichiarato un tipo `string`, lo spazio di memoria iniziale ad esso allocato è nullo, a meno che la stringa non venga inizializzata. Nel caso in cui la memoria allocata non risulti sufficiente, viene allocato un ulteriore spazio di memoria.

Riempire una stringa con n caratteri uguali:

```
string(unsigned int n, char val)
```

esempio: `string str1( 5, 'c' );`

Costruttore di copia che crea un oggetto di tipo `string` di n caratteri riempiendolo con il carattere `val` passato per parametro.

## Accesso ai dati

Una stringa può essere vista come un vettore di caratteri. Per questo motivo è lecito accedere direttamente all'i-esimo carattere di una stringa data utilizzando la funzione membro

```
char& string::operator[ ] (unsigned int i)
```

Vediamo, in un esempio, come può essere utilizzata questa funzione. Il codice seguente

```
string str = "Prova";  
char cA= str[0];  
char cB= str[3];  
cout << cA << " " << cB;
```

produrrà l'uscita seguente `P v`

N.B.

Occorre ricordare che il metodo `operator[ ]` non controlla che l'argomento indice sia valido(compreso nel range) e che quindi è necessario eseguire tali controlli manualmente.

## Concatenazione

**string operator+(const string& str1, const string& str2)**

Di questi operandi, uno è sempre del tipo **const string&**, mentre l'altro può essere del tipo **const string&**, **const char\*** oppure **char**.

Vediamo, in un esempio, come può essere utilizzata questa funzione. Il codice seguente

```
string p1 = "Questa è una ";  
string p2 = "stringa";  
string p3 = p1 + p2;  
cout << p3;
```

produrrà l'uscita seguente **Questa è una stringa**

+ =

**string& string::operator+=(const string& str)**

Il codice seguente

```
string p1 = "Questa è una ";  
string p2 = "stringa";  
string p1 += p2;  
cout << p1;
```

produrrà l'uscita seguente **Questa è una stringa.**

## Confronti tra stringhe

È possibile confrontare il valore di due oggetti di tipo **string**. Per far questo possiamo utilizzare le seguenti funzioni:

**bool operator ==(const string& str1, const string& str2)**

Questo operatore restituisce **true** nel caso in cui gli oggetti di tipo **string** passati come parametro sono coincidenti. Altrimenti la funzione restituisce **false**. Due oggetti di tipo **string** sono coincidenti se hanno la stessa dimensione e se tutti i caratteri corrispondenti sono uguali.

**bool operator !=(const string& str1, const string& str2)**

Questo operatore restituisce 1 nel caso in cui gli oggetti di tipo **string** passati come parametro NON sono coincidenti. Altrimenti la funzione restituisce 0. Due oggetti di tipo **string** sono coincidenti se hanno la stessa dimensione e se tutti i caratteri corrispondenti sono uguali.

**bool operator > (const string& str1, const string& str2)**

Questo operatore restituisce 1 nel caso in cui l'oggetto **string str1** sia maggiore dell'oggetto **string str2** in base all'ordinamento lessicografico. Altrimenti la funzione restituisce 0. Un oggetto **str1** è maggiore di **str2** in base all'ordinamento lessicografico se, e solo se, è verificata una delle seguenti condizioni: o tutti gli elementi corrispondenti sono uguali e la dimensione di **str1** è maggiore della dimensione di **str2**, o indipendentemente dalla loro dimensione, il primo elemento di **str1** non uguale al corrispondente elemento di **str2** è maggiore del corrispondente elemento di **str2**.

**bool operator < (const string& str1, const string& str2)**

Questo operatore restituisce 1 nel caso in cui l'oggetto **string str1** sia minore dell'oggetto **string str2** in base all'ordinamento lessicografico. Altrimenti la funzione restituisce 0. Un oggetto **str1** è minore di **str2** in base all'ordinamento lessicografico se, e solo se, è verificata una delle seguenti condizioni: o tutti gli elementi corrispondenti

sono uguali e la dimensione di **str1** è minore della dimensione di **str2**.o Indipendentemente dalla loro dimensione, il primo elemento di **str1** non uguale al corrispondente elemento di **str2** è minore del corrispondente elemento di **str2**.

**bool operator <= ( const string& str1, const string& str2)**

Questo operatore restituisce 1 nel caso in cui l'oggetto **string str1** sia minore o coincidente, in base all'ordinamento lessicografico, rispetto all'oggetto **string str2**. Altrimenti la funzione restituisce 0.

**bool operator >= (const string& str1, const string& str2)**

Questo operatore restituisce 1 nel caso in cui l'oggetto **string str1** sia maggiore o coincidente, in base all'ordinamento lessicografico, rispetto all'oggetto **string str2**. Altrimenti la funzione restituisce 0.

**int string::compare(const string& s) const**

Questa funzione permette di confrontare tra di loro l'oggetto **string** su cui viene invocato e l'oggetto **string** passato come parametro. Tale metodo ritorna i seguenti valori: 0 nel caso in cui le due stringhe siano identiche.

Un numero negativo nel caso in cui l'oggetto su cui viene invocato precede lessicalmente la stringa passata per argomento. Un numero positivo nel caso in cui l'oggetto su cui viene invocato segue lessicalmente la stringa passata per argomento. L'oggetto passato come parametro può essere anche una stringa C (**char\***). Consideriamo il seguente esempio:

```
string p1 = "questa è ";
string p2 = "una stringa";
p1 == p2; //La funzione restituisce false
p1 != p2; //La funzione restituisce true
p1 >= p2; //La funzione restituisce false (lessicalmente il carattere 'q' precede il carattere 'u')
p1 == p2; // La funzione restituisce true (lessicalmente il carattere 'q' precede il carattere 'u')
p1.compare(p2) //La funzione restituisce un valore negativo
                //( lessicalmente il carattere 'q' precede il carattere 'u')
p2.compare(p1) //La funzione restituisce un valore positivo
                //(lessicalmente il carattere 'q' precede il carattere 'u')
```

## Gestione di sottostringhe

Vediamo adesso una serie di funzioni che permettono di gestire le sottostringhe.

**string& string::insert (int i, string str)**

Funzione che inserisce, all'interno dell'oggetto **string** su cui è stato invocato, la stringa **str** passata come parametro, a partire dall'**i**-esima posizione. Ad esempio il codice

```
string str = "Mario Rossi";
str = str.insert(10, "on");
cout << str;
```

produce la seguente uscita               **Mario Rossoni**

**string& string::erase(int x, int y)**

Funzione che cancella **y** caratteri dalla stringa su cui viene invocata a partire dalla posizione **x**. Ad esempio il codice

```
string str = "Maria Rossi";
str = str.erase(4,3);
cout << str;
```

produce la seguente uscita               **Mariosi**

**string& string::substr(int x=0, int y=npos)**

Funzione che restituisce una sottostringa ricavata dall'oggetto **string** su cui è stata invocata. La sottostringa restituita è costituita da **y** caratteri a partire dalla posizione **x**. Ad esempio il codice

```
string str1 = "Mario Rossi";
string str2;
str2 = str1.substr(6,4);
```

produce la seguente uscita **Rossi**

Nel caso in cui si specifichi un solo parametro, la funzione restituisce la sottostringa ricavata dall'oggetto **string** su cui è stata invocata a partire dalla posizione specificata come parametro fino alla fine della stringa.

Nel caso in cui non si specifichi alcun parametro, la funzione restituisce l'intera stringa su cui è stata invocata.

```
int string::find(string s)
```

La funzione restituisce la posizione della sottostringa **s** passata come parametro all'interno della stringa su cui viene invocata. Ad esempio il codice

```
string str1 = "Mario Rossi";
int i;
i = str1.find("os");
```

memorizza nella variabile **i** il valore 7. In caso di insuccesso questa funzione ritorna come risultato **npos**, ma non genera eccezioni.

```
string& string::replace(int x, int y, string s)
```

La funzione sostituisce nella stringa su cui viene invocata **y** caratteri, a partire dalla posizione **x**, con la stringa **s**. Ad esempio il codice

```
string str = "Mario Rossi";
str = str.replace(3, 3, "ta e Tina ");
cout << str;
```

L'esecuzione di questo codice produce il seguente output:

**Marta e Tina Rossi**

## Operazioni di input ed output

Per stampare gli oggetti di tipo **string**, si utilizza la funzione **operator<<()** in modo standard, come per le stringhe C. Ad esempio questo pezzo di codice

```
string str = "Hallo world";
cout << str;
```

produce come risultato **Hallo world**

Per quanto riguarda l'input degli oggetti di tipo **string**, non è opportuno utilizzare la funzione **operator>>()**, perché in questo modo l'input viene formattato e quindi gli spazi vengono eliminati. Inoltre la dimensione della stringa in cui viene memorizzato l'input viene posta pari alla dimensione della stringa letta.

Per avere un input non formattato (e quindi fare in modo che gli spazi non vengano eliminati), possiamo utilizzare la funzione

```
istream& getline( istream& flusso, string& str, char eol='\n')
```

dove **flusso** è il flusso da cui leggere l'input (ad esempio **cin**), mentre **str** è l'oggetto di tipo **string** su cui vogliamo memorizzare l'input. L'estrazione dall'oggetto **flusso** termina nel momento in cui si incontra il carattere **eol** che viene rimosso dall'oggetto **flusso**, ma non viene posto nell'oggetto **str**. Ad esempio, potremo scrivere

```
string str;
getline(cin, str);
```

Se non si specifica il parametro **eol**, la funzione **getline()** preleva una riga di caratteri (\*) (spazi inclusi) dal flusso di input passato come parametro e la assegna alla stringa.

Occorre tenere presente che tutte le operazioni di input formattato (ad esempio **cin >> str**) lasciano il carattere di fine riga (**\n**) all'interno del flusso. Perciò, nel caso in cui l'input formattato sia seguito da un input non formattato, l'input non formattato si limiterà a leggere un carattere di fine riga. Per capire meglio supponiamo di leggere dal flusso **cin** che

contiene i seguenti caratteri:

```
2 3 \n a b c \n
```

supponiamo di avere il seguente codice:

```
int i;  
string str;  
cin >> i;  
getline (cin, str);
```

Dopo aver eseguito questa parte di codice avremo la situazione seguente:

```
i : 23  
str : \n
```

Può, quindi, essere necessario eliminare il carattere di fine riga presente all'inizio del flusso da cui si deve eseguire l'input. Per eliminare il primo carattere contenuto nel flusso è possibile invocare la funzione **ignore()** sul flusso stesso. Ad esempio

```
cin.ignore()
```

(\*) Con "riga di caratteri" si intendono tutti i caratteri presenti nel flusso fino al primo carattere di fine riga (\n) incluso.

## Funzioni di utilità

Elenchiamo adesso alcune funzioni che possono risultare molto utili nell'utilizzo dei tipi **string**:

```
unsigned int string::size() const
```

Funzione che restituisce la dimensione corrente dell'oggetto di tipo **string** su cui viene invocata.

```
unsigned int string::length() const
```

Funzione che restituisce la dimensione corrente dell'oggetto di tipo **string** su cui viene invocata.

```
bool string::empty() const
```

Funzione che restituisce **true** se l'oggetto di tipo **string** su cui viene invocata risulta vuoto.

```
unsigned int string::max_size() const
```

Funzione che restituisce la dimensione massima che un oggetto di tipo **string** può raggiungere (è un valore normalmente molto grande che dipende dalla stessa dimensione del tipo **char** e dall'implementazione)

## Gestione degli errori

Molte funzioni membro della classe **string** hanno, fra i parametri, due tipi **unsigned int** consecutivi, dove il primo rappresenta una posizione all'interno della stringa e il secondo rappresenta il numero di caratteri da considerare (\*). In tutti i casi il primo argomento è sempre controllato (generando un'eccezione in caso di errore), mentre il secondo non è controllato. Nel caso in cui il secondo parametro abbia un numero troppo alto, tale valore viene semplicemente interpretato come "il resto della stringa".

In seguito ci riferiremo a questa coppia di parametri con il termine coppia "posizione-numero".

Nel caso in cui la coppia "posizione-numero" si riferisca ad una stringa C (**char\***), la funzione non esegue nessun controllo sulla validità dei parametri attuali. Nel caso in cui il parametro posizione valga **NULL**, il programma abortisce.

(\*)

poiché la posizione iniziale e il numero di caratteri sono oggetti di tipo **unsigned int**, nel caso in cui vengano assegnati loro dei numeri negativi questi vengono convertiti in numeri positivi molto grandi.

Un altro tipo di errore si verifica quando si tenta di costruire una stringa più lunga del massimo consentito (dato da **max\_size**).

## INDICE ALFABETICO DEI COMANDI

### DISPONIBILI NELLA LIBRERIA **lib\_easy\_c**

( per ulteriori informazioni consultare il file di inclusione `easy_c7.h` )

```
void delay(int ms)
int Getch();
void gotoxy( int x, int y)
int KeyCode( )
int Kbhit( )
int Random(int)
void WaitESC( )
void WaitESC( const Stringa & )
```

### **Bottone**

Bottone::Bottone( float, float, int, const char \* )

### **ColoreRGB**

```
ColoreRGB( int r, int g, int b);
ColoreRGB( int c );
static ColoreRGB Random( );
ColoreRGB operator ++ (int); /// schiarisce il colore
ColoreRGB operator -- (int); /// scurisce il colore
ColoreRGB operator ++ (); /// schiarisce il colore
ColoreRGB operator -- (); /// scurisce il colore
ostream & operator << (ostream & s, const ColoreRGB )
ColoreRGB operator - (); /// colore complementare
ColoreRGB operator + (const ColoreRGB &); /// somma di colori
ColoreRGB operator - (const ColoreRGB &); /// differenza di colori
```

### **Cronometro**

```
Cronometro::Cronometro( )
void Cronometro::Reset( )
void Cronometro::Start( )
void Cronometro::Stop( )
float Cronometro::Time( ) const sinonimo Tempo()
```

### **Espressione**

```
Espressione Espressione::Derivata( )
Espressione D( Espressione)
Espressione::Espressione( )
Espressione::Espressione( char * )
Espressione::Espressione( const Espressione & )
char * Espressione::FunzioniRiconosciute( )
Espressione::operator ( ) ( )
Espressione::operator ( ) ( double )
Espressione Espressione::operator = ( const Espressione & )
operator <<( ostream &, Espressione & )
operator >>( istream &, Espressione & )
```

### **Grafica**

```
float Grafica::Altezza( )
float Grafica::CentroX( )
```

```
float Grafica::CentroY()
void Grafica::FinestraVisibile( float, float, float, float )
float Grafica::Larghezza()
float Grafica::ScalaX()
float Grafica::ScalaY()
```

## **MatriceFloat**

```
void MatriceFloat::Dimensioni( unsigned int, unsigned int )
MatriceFloat::MatriceFloat()
MatriceFloat::MatriceFloat( const MatriceFloat & )
int MatriceFloat::NumeroColonne() const
int MatriceFloat::NumeroRighe() const
void MatriceFloat::NumeroColonne( unsigned int )
void MatriceFloat::NumeroRighe( unsigned int )
MatriceFloat & MatriceFloat::operator [ ] ( unsigned int )
MatriceFloat MatriceFloat::operator [ ] ( unsigned int ) const
MatriceFloat & MatriceFloat::operator = ( const MatriceFloat & )
```

## **MatriceInt**

( come MatriceFloat )

## **SetCaratteri**

```
SetCaratteri::CreaInsiemeVuoto()
SetCaratteri::operator ! ( ) const
SetCaratteri::operator - ( const SetCaratteri & ) const
bool SetCaratteri::operator * ( const SetCaratteri & ) const
bool SetCaratteri::operator + ( const SetCaratteri & ) const
void SetCaratteri::operator += ( char * )
void SetCaratteri::operator += ( char )
bool SetCaratteri::operator < ( const SetCaratteri & ) const
bool SetCaratteri::operator <= ( const SetCaratteri & ) const
bool SetCaratteri::operator == ( const SetCaratteri & ) const
SetCaratteri::SetCaratteri()
SetCaratteri::SetCaratteri( char * )
```

## **Tartaruga**

```
Tartaruga::Tartaruga( const Tartaruga & )
void Tartaruga::Avanti( float )
void Tartaruga::CambiaColorePennello( ColoreRGB )
void Tartaruga::Cerchio( double )
void Tartaruga::ClearScreen( ColoreRGB )
void Tartaruga::Color( ColoreRGB )
ColoreRGB Tartaruga::ColoreFondo() const
ColoreRGB Tartaruga::ColorePennello() const
void Tartaruga::Destra( float )
float Tartaruga::Direzione() const
void Tartaruga::Direzione( float )
void Tartaruga::DirezioneAlPunto( float, float )
void Tartaruga::DoveSei( float &, float & ) const
bool Tartaruga::FuoriSchermo() const
void Tartaruga::Giu()
void Tartaruga::GraficoFunzione( double( * )( double ), ColoreRGB, ColoreRGB, ColoreRGB )
void Tartaruga::GraficoFunzione( Espressione, ColoreRGB, ColoreRGB, ColoreRGB )
void Tartaruga::Griglia( ColoreRGB, ColoreRGB )
void Tartaruga::Hide()
```



```

void Tartaruga::Indietro( float )
void Tartaruga::Interpreta( char * )
void Tartaruga::LeggiStringa( char *, unsigned int )
float Tartaruga::MaxX() const
float Tartaruga::MaxY() const
void Tartaruga::MettiUnPunto( float, float )
void Tartaruga::MettiUnPunto( float, float, ColoreRGB)
float Tartaruga::MinX() const
float Tartaruga::MinY() const

Tartaruga & Tartaruga::operator <<( char * )
Tartaruga & Tartaruga::operator <<( const Stringa & )
Tartaruga & Tartaruga::operator <<( double )
Tartaruga & Tartaruga::operator <<( int )
Tartaruga & Tartaruga::operator <<(Tartaruga &(*) ( Tartaruga & ) )
Tartaruga & Tartaruga::operator >>( char & )
Tartaruga & Tartaruga::operator >>( double & )
Tartaruga & Tartaruga::operator >>( float & )
Tartaruga & Tartaruga::operator >>( int & )
Tartaruga & Tartaruga::operator >>( long & )
Tartaruga & Tartaruga::operator >>( Stringa & )

void Tartaruga::RiempiColore( )
void Tartaruga::RiempiColore(ColoreRGB, ColoreRGB)
void Tartaruga::Salta( float , float )
void Tartaruga::Show( )
void Tartaruga::Sinistra( float )
void Tartaruga::SpostatiDi( float, float )
void Tartaruga::Su( )
void Tartaruga::Tana( )
void Tartaruga::Tartaruga( )
void Tartaruga::TartaZero( )
void Tartaruga::TempoPasso( int )
void Tartaruga::UsaPennelloLargo( )
void Tartaruga::UsaPennelloSottile( )
void Tartaruga::Vai( float, float )
void Tartaruga::Velocita( int )
float Tartaruga::X() const
float Tartaruga::Y() const
endl( Tartaruga & )

```

## ***VettoreFloat***

```

unsigned int VettoreFloat::Dimensione( ) const
void VettoreFloat::Dimensione( unsigned int )
VettoreFloat & VettoreFloat::operator [ ] ( unsigned int )
VettoreFloat VettoreFloat::operator [ ] ( unsigned int ) const
VettoreFloat & VettoreFloat::operator = ( const VettoreFloat & )
VettoreFloat::VettoreFloat( const VettoreFloat & )

```

## ***VettoreInt***

( come VettoreFloat )