



Nel 1972, presso i laboratori della AT&T, Tennis Ritchie realizza il linguaggio C per sviluppare il sistema operativo UNIX, svincolandosi così dall'Assembler. Nel 1983 Bjarne Stroustrup, ricercatore AT&T, realizza il C++, un linguaggio orientato agli oggetti, evoluzione dello C.

In queste dispense vengono introdotti alcuni concetti del C++ procedurale (sostanzialmente linguaggio C), senza entrare nel merito della OOP, della programmazione visuale e dell'accesso ai database.

STRUTTURA .....	1
COSTANTI, VARIABILI, TIPI DI DATI .....	1
OPERATORI .....	2
LIBRERIE.....	2
MACROISTRUZIONI .....	2
USING NAMESPACE STD; .....	2
INPUT / OUTPUT .....	3
STAMPA .....	3
ISTRUZIONI.....	3
ESEMPIO DI PROGRAMMA .....	5
FUNZIONI .....	6
RICORSIONE .....	6
MODULI .....	6
ARRAY .....	7
STRINGHE .....	7
ENUMERAZIONI.....	7
STRUTTURE - STRUCT - (RECORD E TABELLE) .....	7
PUNTATORI [ &, * ] .....	9
STRUTTURE DINAMICHE .....	10
INPUT/OUTPUT.....	10
FILE (O ARCHIVI O FLUSSI) .....	11
APPENDICE 1 - LIBRERIE DI SISTEMA.....	13
APPENDICE 2 - LISTA .....	15
APPENDICE 3 - ARCHIVI.....	16
APPENDICE 4 - CASUALITÀ, POSIZIONAMENTO, COLORE, TASTI, ATTESA... ..	18
APPENDICE 5 - FUNZIONE SYSTEM .....	22

## Struttura

Un programma C++ è costituito da direttive, dichiarazioni, programma principale e funzioni o sottoprogrammi. Il programma principale ha nome **main**, le istruzioni sono racchiuse tra parentesi graffe "{" (codice Ascii 123) e "}" (codice Ascii 125) e terminano con il punto e virgola ";", gli spazi sono ignorati. Nella sintassi delle istruzioni si useranno i simboli { } per racchiudere le parti opzionali (in quanto le parentesi quadre sono usate da C++ per identificare gli elementi degli array).

Le funzioni restituiscono un valore - a meno che non siano precedute dalla clausola *void*, nel qual caso non restituiscono alcun valore - e terminano con l'istruzione *return*.

Il tipo di valore restituito dalla funzione viene dichiarato premettendolo al nome della funzione stessa; le funzioni non dichiarate restituiscono un dato intero.

Per inserire i **commenti** si utilizza la doppia barra "//" o la barra-asterisco "/\*" iniziale e asterisco-barra "\*/" finale.

## Costanti, variabili, tipi di dati

Gli identificatori del linguaggio possono essere lunghi fino a 1024 caratteri e possono contenere lettere dell'alfabeto (maiuscole e minuscole), cifre e il carattere "\_" (*underscore*, sottolineato), ma il primo carattere non può essere una cifra. Maiuscole e minuscole sono distinte.

Le costanti carattere sono racchiuse tra apici singoli, mentre quelle stringa tra doppi apici. Per i numeri si usa il "." (punto decimale). Le costanti stringa terminano con '\0' (se non presente, viene inserito automaticamente dal compilatore).

I tipi dati di C++ si distinguono in: Fondamentali - predefiniti (char, int, float, double, void) e definiti dall'utente (enumerativi) – e Derivati (array, funzioni, riferimenti, puntatori, strutture)

TIPI di dati PREDEFINITI

Parola chiave	Tipo	Dimensioni	Valori
<b>char</b>	Carattere singolo	1	1 codice ASCII (da 0 a 255)
<b>float</b>	Reale singola precisione	4	$\pm 1.2\text{E}-38 \div \pm 3.4\text{E}38$
<b>double</b>	Reale doppia precisione	8	$\pm 2.2\text{E}-308 \div \pm 1.8\text{E}308$
<b>short int</b>	Intero	1	-128 ÷ 127
<b>int</b>	Intero	2	-32768 ÷ 32768
<b>long int</b>	Intero	4	-2147483648 ÷ 2147483647
<b>bool</b>	Booleano	1	True (1) / False (0)
<b>string</b>	Stringa	n	lunghezza variabile, ASCII

**Definizione** di variabili e costanti:

{ **modificatore accesso** } { **modificatore tipo** } **tipo variabile** { = *valore* } , ....

modificatore di accesso: auto, register, extern, static, volatile

modificatore di tipo: const, signed, unsigned

tipo: char, float, double, short int, int, long int

**int i, somma=0;** definisce due variabili intere ed una la inizializza a 0.

**const float pigreco=3.14;** definisce la costante pigreco

Oltre alle variabili semplici, C++ prevede anche un notevole set di dati astratti (**ADT** - *Abstract Data Type*) e relative variabili strutturate – array, stringhe, enumerazioni, strutture, tabelle, puntatori, classi, ... - più avanti trattati.

## Operatori

ARITMETICI	
+	Addizione
-	Sottrazione
*	Moltiplicazione
/	Divisione
%	modulo
++	incremento
--	decremento
>>	shift di bit a destra
<<	shift di bit a sinistra

RELAZIONALI	
==	uguale
>	maggiore
>=	maggiore uguale
<	minore
<=	minore uguale
!=	diverso

CARATTERE	
+	concatenazione

CONDIZIONALE	
?	condizione

LOGICI	
&&	And
	Or
!	Not

LOGICI sui BIT	
&	And
	Or
^	Xor
!	Not
~	Complemento a 1

Con l'**assegnazione multipla** si può assegnare lo stesso valore a più variabili:

*var1 = var2 = var3 = valore;*

Con l'**assegnazione composta** (operatore=) si eseguono i calcoli prima dell'assegnamento (operazioni aritmetiche abbreviate):

```
a = a + b      a += b
a = a - b      a -= b
a = a * b      a *= b
a = a / b      a /= b
a = a % b      a %= b
i = i + 1      i += 1      i++      ++i
i = i - 1      i -= 1      i--      --i
```

Operatore condizionale ?: *espressione\_logica ? espr1 : espr2*  
se l'espressione logica è vera allora si ha espr1, altrimenti espr2.

esempio |x|, modulo di x: *valore\_assoluto = x>0 ? x : -x* // in C++  
*valore\_assoluto = IIF(x>0,x,-x)* // in VB

## Librerie

Moltissime funzioni aggiuntive sono utilizzabili grazie a file di intestazione/"header" con estensione **.h** che fanno riferimento alle "librerie" di sistema che le contengono e che vengono inseriti nel programma con la **direttiva #include <libreria>** (*#include <iostream>*, *#include <stdlib.h>*, etc ).

Al riguardo, si veda l'**Appendice 1 – Librerie di sistema**.

## Macroistruzioni

Con la direttiva **#define NOME valore** si possono definire delle costanti.

Esempio: *#define PIGR 3.14* / *#define AUTORE "Claudio Maccherani"*

Comunque, per definire le costanti, è meglio utilizzare la dichiarazione **const** vista precedentemente.

## Using Namespace Std;

I *namespace* sono dei contenitori di nomi di variabili e funzioni. Per riferirci a un nome si deve specificare *namespace::nome* (ad esempio *str::cout*). Utilizzando **Using Namespace Std;** si comunica al compilatore che, se non specificato altrimenti, i nomi di variabili e funzioni appartengono al *namespace STD* (standard).

## Input / Output

Per attivare l'I/O di C++ occorre includere la libreria **iostream.h** delle funzioni standard di I/O e quindi le funzioni console *input* e console *output*:

```
cin >> variabile; [ >> "leggi da", input ]
cout << espressione; [ << "scrivi su", output ]
```

Per l'output si possono utilizzare opportune sequenze di escape individuate dalla barra rovesciata:

'\n' <i>line feed</i>	'\f' <i>form feed</i> ,	'\a' <i>alert</i> ,	'\t' tabulazione orizzontale
'\v' tabulazione verticale	'\r' ritorno carrello	'\b' <i>backspace</i>	'\\' barra rovesciata
'\?' punto interrogativo	'\' ' apice	'\"' doppio apice	'\0' <i>nul</i> .

Nella libreria **stdio.h** sono presenti altre importanti funzioni di I/O quali :

<b>scanf</b> ( <i>elenco di puntatori alle variabili</i> );	input
<b>printf</b> ( <i>stringa formattata con costanti e variabili</i> );	output

Esempio: `printf("La somma di %d più %d è %d \n",a,b,c);`  
se a è 5 e b è 2, la stringa stampata sarà: "La somma di 5 più 2 è 7".

Con il carattere % si specifica il formato delle variabili che compaiono in fondo all'espressione:

%d	int – numero intero
%i	int – numero intero
%c	int – codice ASCII
%s	char* - stringa
%f	double – numero con 6 decimali
%x.yf	double – numero con x cifre di cui y decimali
%p	void* - indirizzo in memoria
%x	int – numero intero esadecimale
%e	double – numero decimale esponenziale
%%	carattere %
...	etc. etc. (altri tipi di formattazione)

Nell'espressione possono comparire anche caratteri di *sequenza di escape*:

\0	carattere vuoto, NUL (0)
\a	allarme, beep
\b	backspace
\f	salto pagina, FF (12)
\n	salto riga, LF (10)
\t	tab.orizzontale, HT (9)
\v	tab.verticale, VT (11)
\\	carattre \
\?	carattre ?
\'	carattre '
\"	carattre \"

## Stampa

Per attivare l'output sulla stampante di sistema occorre includere la libreria **fstream.h** ed utilizzare la funzione **ofstream** *stampante*("PRN") e poi *stampante* << *riga\_di\_stamp*a.

Esempio: `ofstream Printer("PRN");`  
`Printer << "il risultato è: " << ris;`

## Istruzioni

Le istruzioni racchiuse tra parentesi graffe "{ }" costituiscono un **blocco** di elaborazione. Le variabili dichiarate all'interno di un blocco sono **locali**, mentre quelle dichiarate esternamente da qualsiasi blocco sono **globali**. Esempio:

```
#include <iostream.h>
int flag = 0 // variabile globale
int main() {
    float euro = 1.936,27; // variabile locale
    ....
    return 0; }
```

Definizione: { *modificatore accesso* } { *modificatore tipo* } *tipo variabile* { = *valore* } ,....  
modificatore di accesso: auto, register, extern, static, volatile  
modificatore di tipo: const, signed, unsigned  
tipo: char, float, double, short int, int, long int

Assegnamento:

*variabile = espressione;*

```
a = 10; ciccio = x3; x = a + b / 2;
```

Istruzioni con etichetta:

*etichetta : istruzione;*

```
Continua: bic = 0;
```

Salto incondizionato:

**goto** *etichetta;*

```
goto Continua;
```

Selezione:

```
if (condizione)  
    { istruzione/ blocco A; }  
{ else  
    { istruzione/blocco B; } }
```

```
if (n > 0)  
    { media = tot / n;  
      cout << media; }  
else  
    { cout << "nessun dato!"; }
```

Selezione multipla:

```
switch (scelta) {  
    case costante1: istruzione/blocco 1;  
    case costante2: istruzione/blocco 2;  
        .....  
    { default: istruzione/blocco alternativo;  
    }
```

```
switch (scelta) {  
    case 1: inserimento(); break;  
    case 2: elenco(); break;  
    case 3: stampa(); break;  
    default: cout<<"errore"; break;  
}
```

Iterazioni (ciclano sempre per **vero**):

```
while (condizione) // test all'inizio  
    { istruzione/blocco; }
```

```
while (cont == 'S') {  
    cin >> x; tot = tot + x;  
    cout << "continui?"; cin >> cont;  
}
```

```
do // test alla fine  
    { istruzione/blocco; }  
while (condizione);
```

```
do {  
    cin >> x; tot = tot + x;  
    cout << "continui?"; cin >> cont; }  
while (cont == 'S');
```

```
for (inizio ; condizione ; incremento)  
    { istruzione/blocco; }
```

```
for (i=0; i<10; i++) {  
    cout << "importo"; cin>> imp;  
    tot = tot + imp; }
```

Nelle iterazioni (nidificate) si può utilizzare l'istruzione **break** (esce dal ciclo) o l'istruzione **continue** (passa all'iterazione successiva).

## Esempio di Programma

Un primo programma di esempio:

```
// programma C++ di esempio (file '000_Primo.cpp')

#include<iostream>      // file "header" per libreria di sistema di I/O
#include<string.h>      // file "header" per libreria gestione stringhe

using namespace std;    // direttiva per l'uso dei nomi standard

int main()              // programma principale (MAIN)
{                       // "{" inizio blocco programma
    float  voto;         // definizione variabile numerica decimale 'voto'
    string alu;          // definizione variabile stringa 'alu'
    int    tot,i,x;      // definizione variabili intere 'tot', 'i' e 'x'

    cout << "Primo programma" << endl << endl; // visualizza il titolo

    cout << "Nome: ";    // visualizza la richiesta del nome
    cin  >> alu;         // il nome digitato va in 'alu'

    cout << "Voto: ";    // visualizza la richiesta del voto
    cin  >> voto;        // il voto digitato va in 'voto'

    if (voto < 6)        // test sul voto, se minore di 6
    {                   // "{" inizio blocco del "vero" di "if"
        cout << alu << " deve fare il recupero"; // vis.nome + recupero
    }                   // "}" fine blocco del "vero" di "if"
    else                // "altrimenti", il voto non è < 6 (è maggiore)
    {                   // "{" inizio blocco del "falso" di "if"
        cout << "Bravo " << alu; // visualizza bravo + nome alunno
    }                   // "}" fine blocco del "falso" di "if"

    tot = 0;            // azzeramento accumulatore 'tot'
    i = 0;              // azzeramento contatore 'i'

    while (i<7)         // ciclo 'while' (finché 'i' è minore di 7)
    {                   // "{" inizio blocco ciclo "while"
        cout << "caramelle mangiate: "; // vis. richiesta caramelle
        cin  >> x;      // il numero di caramelle va in 'x'
        tot = tot + x; // aggiorna l'accumulatore 'tot' di 'x'
        i = i + 1;     // aggiorna il contatore 'i' di 1
    }                   // "}" fine blocco ciclo "while"

    // visualizza totale caramelle mangiate
    cout << alu << " ha mangiato " << tot << " caramelle " << endl;

    return 0;           // ritorno al sistema (con valore intero 0)
}                       // "}" fine blocco programma
```

Il programma richiede il nome di un alunno e il voto preso in un compito e dice se l'alunno deve fare il recupero (se il voto non è sufficiente) o se è bravo (il voto è sufficiente o più che sufficiente), quindi richiede quante caramelle ha mangiato giornalmente e comunica il totale di caramelle mangiate durante la settimana.

## Funzioni

In C++ la funzione è l'unico strumento per realizzare sottoprogrammi e procedure. Una funzione può avere dei parametri di ingresso e restituisce un risultato, se non definita di tipo **void** (nel qual caso restituisce un valore *vuoto*, nullo):

```
{ tipo risultato } nome_funzione ( { elenco parametri } )
{
    ....
    return { espressione } ;
}
```

Le funzioni possono essere definite, nel programma sorgente, prima del programma principale **main** oppure dopo, ma in questo caso occorre inserire prima del **main** il **prototipo** della funzione, cioè la sua intestazione completa. Oppure possono stare in un file **.h** da includere (**#include**).

### Passaggio dei parametri

I parametri possono essere passati per **valore** (*by value*), per default, o per **indirizzo** (*by reference*, riferimento). In quest'ultimo caso il parametro, nella dichiarazione della funzione, deve essere preceduto dalla e commerciale "&" (*indirizzo di*).

Nella definizione della funzione si possono avere dei parametri opzionali ai quali occorre assegnare dei valori di default. In questo caso tali parametri debbono essere gli ultimi dell'elenco e debbono essere inizializzati una sola volta, nel prototipo della funzione. Esempio:

```
int pippo(double x, float &a, int k=0)
```

"x" viene passato per valore, "a" per indirizzo e "k" per valore ed è opzionale

Con il modificatore di accesso **static** si possono definire variabili e funzioni "persistenti", il cui valore permane anche dopo l'uscita dal blocco. Tali elementi vengono inizializzati solo la prima volta che si incontra la loro definizione.

### Ricorsione

Una definizione ricorsiva è un algoritmo in cui una procedura o una funzione richiama se stessa.

La ricorsione può essere diretta (una funzione richiama se stessa) o indiretta (una funzione ne chiama un'altra che a sua volta la richiama). Il classico esempio di ricorsione è **n!**, il fattoriale:

```
0! = 1
n! = n*(n-1)!

double fatt(double n) {
    if (n==0) return 1; else return (n*fatt(n-1)); }
int main() { double x;
    cout<<"n="; cin>>x; cout<<x<<"! = "<<fatt(x);
    return 0; }
```

### Moduli

Una applicazione C++ complessa può essere strutturata in più moduli (file sorgente) che dovranno, dopo la compilazione, essere lincati tra loro. È possibile da qualsiasi modulo utilizzare le funzioni definite in altri moduli (che debbono essere definite una sola volta). La funzione **main** deve comparire una sola volta, nel modulo principale. È bene strutturare ciascun modulo in due parti:

- *intestazione* - file **.h** - contenente i tipi dati e le costanti (**#define**), i prototipi delle funzioni del modulo, le variabili esterne (**extern**), le inclusioni di altri file di intestazione (**#include**)
- *implementazione* - file **.cpp** – contenente le definizioni delle strutture dati impiegate nel modulo e il codice eseguibile organizzato in funzioni

## Array

In C++ sono implementati array monodimensionali (Vettori), bidimensionali (Matrici) e a più di due dimensioni (multidimensionali). La definizione di un array è la seguente:

`tipo nome_array [num.max.elementi] { [num.max.elementi] } {={valori iniziali} }`

Gli indici vanno da 0 a numero\_massimo\_elementi – 1.

**C e C++ non controllano l'accesso agli array per cui un accesso al di fuori dell'array (indice errato) andrà a "sporcare" altre aree di memoria del programma, con effetti imprevedibili.**

<code>int vett[10];</code>	vettore di 10 elementi, da 0 a 9
<code>int vet[4] = {5,3,8,12};</code>	definizione e inizializzazione vettore
<code>int vet[] = {5,3,8,12};</code>	" " " " "
<code>int mat[2][4];</code>	matrice di 2 righe (da 0 a 1) e 4 colonne (da 0 a 3)
<code>int mat[3][2] = {5,3,8,12,4,92};</code>	definizione e inizializzazione matrice (per righe)
<code>int arr[2][4][3][5];</code>	array multidimensionale a 4 dimensioni

## Stringhe

Per C++ una stringa è un vettore di caratteri che deve terminare con il simbolo speciale '\0'.

<code>char s[15];</code>	definisce una stringa di 15 caratteri (14 + '\0')
<code>char app[] = {"Info"};</code>	definisce ed inizializza la stringa "Info" di 4+1 caratteri
<code>char testo [10] [81];</code>	definisce un vettore di 10 stringhe lunghe 80 caratteri+1

C++ non dispone di operazioni predefinite sulle stringhe, ma tutte le operazioni debbono fare riferimento ai vettori di caratteri. Per confrontare se due stringhe sono uguali occorre fare un ciclo confrontando i relativi vettori, così come per fare la copia (che in VB si fa con `s2 = s1`):

```
void copia(char s1[], char s2[])  
{ int i = 0; while (s1[i]) { s2[i]=s1[i]; i++; } } // il ciclo termina su \0
```

I tool di programmazione dispongono - in `stdio.h` e in `string.h` - di alcune funzioni predefinite sulle stringhe:

- |  |   |
|--|---|
| ○ <code>gets(stringa)</code>               | input di una stringa da tastiera su <i>stringa</i>          |
| ○ <code>printf(stringa)</code>             | output su video di <i>stringa</i>                           |
| ○ <code>strcat(s1,s2)</code>               | concatena <i>s2</i> a <i>s1</i>                             |
| ○ <code>variabile = strlen(stringa)</code> | restituisce la lunghezza di <i>stringa</i> (senza il '\0')  |
| ○ <code>strep(s1,s2)</code>                | copia <i>s2</i> in <i>s1</i> ( <i>s2 = s1</i> in VB)        |
| ○ <code>strcmp(s1,s2)</code>               | confronta <i>s2</i> con <i>s1</i> e restituisce 0 se uguali |

Altra libreria contenente utili funzioni sulle stringhe [`strcpy()`, `strcat()`, `strstr()`, `strchr()`, `strcmp()`] è la `string.h` (si veda in fondo alle presenti dispense).

## Enumerazioni

Le enumerazioni sono liste di costanti intere simboliche - enumeratori - a ciascuna delle quali può essere assegnato un valore intero, a partire dal valore 0 assegnato alla prima.

`enum nome_enumerazione { id_costante {=valore iniziale} },... } {nome variabile}`

es: `enum trimestre {Gennaio, Aprile=4, Luglio=7, Ottobre=10} mese;`  
`trimestri trim1, trim2; // definisce 3 variabili – mese, trim1, trim2 – d tipo trimestre`

## Strutture - Struct - (record e tabelle)

La struct – struttura – permette di definire strutture dati di tipo **record**, cioè insiemi strutturati di campi (è l'equivalente della **TYPE** del VB).



```
struct nome_struttura { tipo nome_elemento; ... } {nome record}
```

```
es:  struct rec_alu {int matr; char nome[30]; int cla; char sez; char cor; };  
      rec_alu alunno;           // definisce il record alunno di tipo rec_alu
```

Le strutture, dopo la dichiarazione, diventano a tutti gli effetti dei nuovi tipi di dati. Le variabili di tipo struct possono essere inizializzate durante la definizione:

```
rec_alu alunno = {358, "Vera Lucia de Oliveira", 5, 'A', 'p'};
```

Per accedere ai singoli campi di una struttura si usa il punto "." (`alunno.matricola`) o la freccia "->" (`alunno->matricola`)

A variabili dello stesso tipo di dato struct è possibile applicare l'operatore di assegnazione "=".

Le strutture possono essere **annidate**, inserendo un campo strutturato dentro un'altra struttura:

```
struct rec_ind {char via[30]; char citta[20], char provincia[2]; };  
struct rec_alu {int matr; char nome[30]; int cla; char sez; char cor; rec_ind indir};  
rec_alu alunno;           // definisce l'alunno con il campo indir strutturato
```

Con le strutture si possono definire **tabelle** semplicemente definendo un *array di struct*:

```
rec_alu classe[20] // definisce la tabella classe di 20 alunni di tipo rec_alu
```

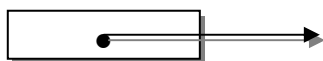
Si accede al singolo elemento specificando la riga e il campo:

```
classe[8].sez // è la sezione dell'ottava riga della tabella, cioè del nono alunno
```

### **Puntatori** [ &, \* ]

Per tutte le variabili esaminate precedentemente il compilatore riserva un ben determinato spazio di memoria che non si modifica durante l'esecuzione. Esistono però dei tipi di dati dinamici per i quali non si può conoscere a priori l'occupazione.

Il **puntatore** (\*p) è un particolare tipo di dati che permette di implementare le strutture dati dinamiche quali coda, pila, lista, albero e grafo. Esso contiene l'indirizzo iniziale di una zona di memoria dove è allocata una struttura dati.



Per definire un puntatore - che deve essere dello stesso tipo del dato che punterà - si utilizza l'operatore di dichiarazione asterisco '\*':

```
tipo_dato_puntato *puntatore {= indirizzo iniziale}
```

Una **variabile** e un **puntatore** individuano entrambi un contenitore, una cella di memoria, ma mentre la variabile restituisce direttamente il dato della cella - accesso diretto -, il puntatore restituisce l'indirizzo della cella che contiene il dato - accesso indiretto - .

È bene inizializzare sempre i puntatori a 0. Esempio: `char *punc = 0;`

Per assegnare un indirizzo a un puntatore (\*p) si usa l'operatore '&' (*reference*, "indirizzo di"):

```
double inf = 350, *pun; // definisce variabile e puntatore
pun = &inf; // assegna al puntatore l'indirizzo della variabile
cout << "Indirizzo di INF: " << hex << (long int) pun;
```

L'operatore '\*' serve per accedere al dato cui il puntatore fa riferimento (indirizzamento indiretto). \*pun viene detto **puntatore dereferenziato**.

```
Esempi: // c2 = c1 // tot = op1 + op2
char c1 = 'A', c2; double op1, op2, tot
char *pun; double p_op1, p_op2, p_tot
pun = &c1; p_op1 = &op1; p_op2 = &op2; p_tot = &tot
c2 = *pun; cout << "Primo "; cin >> *p_op1
cout << "Secondo "; cin >> *p_op2
*p_tot = *p_op1 + *p_op2
cout << "Somma: "; << *p_tot ;
```


Grazie ai puntatori ed agli operatori algebrici ad essi applicati ci si può muovere tra dati dello stesso tipo: ++, --, + intero, - intero. Es: `pun++` // passa a indirizzare il prossimo dato dello stesso tipo.

Finché non inizializzati i puntatori possono puntare a una qualsiasi parte della memoria. Per questo occorre inizializzare quelli non ancora utilizzati con NULL, '\0'. Es: `int *pun1 = '\0';`

Un puntatore NULL può indicare anche la terminazione di strutture dinamiche quali le liste.

In C++ il nome di un array contiene l'indirizzo del suo primo elemento, è cioè un puntatore. Il passaggio di un array ad una funzione avviene sempre per indirizzo. Si può accedere agli elementi di un vettore anche attraverso il puntatore:

```
int vet[4], *pun; pun = vet;
cout << *(pun+3);
cout << *pun[3];
cout << vet[3];
```



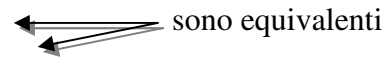
sono equivalenti

Con i puntatori si possono gestire anche le stringhe, che sono array di caratteri:

```
while (*p_vetchar) { cout << *p_vetchar; p_vetchar++; }
```

I puntatori possono indirizzare anche strutture complesse quali struct e union, accedendo alla struttura tramite il suo puntatore:

```
struct rec_voti { char nome[30]; int ita; int mat; int inf; };
rec_voti alunno = { "Giacomo Cerno", 7,9,10 };
rec_voti *p_voti;
p_voti = &alunno;
cout << alunno->inf;
cout << *p_voti->inf;
```



sono equivalenti

Anche i nomi delle funzioni sono dei puntatori alla zona di memoria che contiene la funzione. Si può definire un puntatore alla funzione che può essere usato per richiamare indirettamente la stessa funzione.

### Strutture dinamiche

Mediante i punatori si possono realizzare strutture dinamiche per le quali allocare e deallocare, durante l'esecuzione, memoria RAM (nei vecchi sistemi operativi tale memoria era detta *heap memory*).

Per allocare dinamicamente la memoria c'è l'istruzione **new** - *nome\_puntatore* = **new** *tipo\_dato* – che riserva, a *run time*, la memoria necessaria ed assegna al puntatore il suo indirizzo iniziale.

```
float *p_mem; p_mem = new float[10];           // alloca memoria per contenere 10 interi
for (int i=0; i<10; i++) p_mem[i]=(int) i;     // riempie la struttura allocata
```

Se non c'è più memoria disponibile new restituisce il puntatore nullo '\0'.

Per liberare la memoria allocata con new si usa l'istruzione **delete** - **delete** { [ ] } *nome\_puntatore* – (se è un array occorre specificarlo con "[ ]") che elimina il collegamento e libera memoria, ma non elimina il puntatore.

Le principali strutture dati dinamiche sono **Lista**, **Coda**, **Pila**, **Grafo**, **Albero**.

Nell'**Appendice 2 - Lista** è trattata la gestione di una lista semplice in C.

### Input/Output

C++ non dispone di istruzioni predefinite di I/O che è basato su librerie di classi predefinite, quali la **iostream.h**, da includere - **#include <iostream.h>**. Le periferiche di I/O sono viste come file:

```
cin >> variabile1 >> variabile2;           // input da tastiera
cout << espressione1 << espressione2;      // output su video
```

## File (o Archivi o Flussi)

In C++ i flussi possono essere di testo oppure binari. Un flusso di testo è costituito da caratteri rappresentati in base al ASCII ('\n' viene interpretato come "a capo"), mentre un flusso binario è una lista di caratteri char ai quali non è associato alcun codice particolare ('\n' non viene interpretato, sono solo due caratteri). I flussi di testo e i flussi binari possono essere, rispetto al tipo di accesso, sequenziali oppure relativi o diretti.

I file (o archivi, o flussi) sono visti come oggetti della classe `fstream.h` (`#include <fstream.h>`) che debbono sempre essere definiti con `ifstream flusso` (in input) e con `ofstream flusso` (in output).

**Apertura:** `flusso.open(pathname {, modalità | ... | ... } );`  
modalità: (di default è file di testo)

<code>ios::in</code>	in input
<code>ios::out</code>	in output
<code>ios::binary</code>	file binario
<code>ios::nocreate</code>	apre solo file esistenti
<code>ios::noreplace</code>	apre solo file non esistenti
<code>ios::ate</code>	apre e si posiziona alla fine
<code>ios::app</code>	apre in append (nuove scritture in fondo)
<code>ios::trunc</code>	apre in output (se il file esiste lo azzerà)

Si possono specificare più modalità di apertura utilizzando il separatore "|" (*pipe*).

**Chiusura:** `flusso.close();`

### 1) flussi Sequenziali

Un flusso logico sequenziale è un insieme di registrazioni chiuse da *eof* (*end of file*) nel quale le registrazioni avvengono in fondo e la lettura è sequenziale. Si legge con **write**, si scrive con **read**:

```
flusso.write/read((const unsigned char*)p_record, int numero_byte);  
flusso.write/read((const unsigned char*)p_record, sizeof(numero_byte));
```

dove *p\_record* è il puntatore del record in memoria centrale e *numero\_byte* è la sua lunghezza. La write scrive il record nel buffer di memoria centrale, che generalmente contiene più record logici. Per forzare la scrittura del buffer si può usare `flusso.flush();`

### 2) flussi Relativi

Un flusso logico relativo è un insieme di registrazioni dove ogni record è individuato dalla sua posizione (il primo carattere ha indirizzo 0). Il sistema di I/O gestisce due puntatori, uno per la scrittura ed uno per la lettura. Se il file contiene *N* record di *n* caratteri l'indirizzo iniziale del generico record *i*-esimo è  $(i - 1) * n$ . Anche i flussi relativi sono chiusi da *eof* (*end of file*), si legge con **write**, si scrive con **read**, ci si posiziona sul record con **seekp** (in scrittura) e con **seekg** (in lettura):

```
flusso.seekp/seekg(int spostamento, seek_dir posizione_riferimento);
```

dove *spostamento* è l'offset da aggiungere alla *posizione\_riferimento* per posizionarsi all'inizio del record. Se la posizione di riferimento è l'inizio del file, lo spostamento equivale alla posizione del record calcolata con  $(i-1)*n$ . La *posizione\_riferimento* specifica da dove si deve partire per calcolare lo spostamento: `ios::beg` (dall'inizio – *begin*), `ios::end` (dalla fine – *end*), `ios::cur` (dal record corrente –

*current*). Le funzioni *flusso.tellp()* e *flusso.tellg()* restituiscono la posizione corrente, rispettivamente, dei puntatori di scrittura e di lettura.

### 3) flussi Binari

In un flusso binario – *ios::binary* – si può leggere (*get*) e scrivere a (*put*) a livello di singolo carattere:

*flusso.put/get(char carattere);*

Le funzioni *put()* e *get()* possono essere applicate anche ai flussi della console quali **cout** e **cin**.

### 4) file di Testo

Un file fisico di testo è una sequenza di caratteri terminanti con *eof* che può essere gestito da un qualsiasi programma editore di testi. Tali file si aprono con **open** e si chiudono con **close**, mentre per scrittura e lettura si usano, rispettivamente, gli operatori **<<** ("scrivi su") e **>>** ("leggi da") applicati al flusso associato al file.

*flusso << var/cost1 << var/cost2 << ...;*      scrive I dati sul flusso di output

*flusso >> var1 >> var2 >> var3 >>...;*      legge i caratteri dal file di testo, a partire dalla posizione corrente, fino a che non incontra spazio ('\0') o LF ('\n') o tabulatore ('\t')

```
/* esempio di lettura/scrittura da/su flusso sequenziale ⇔ vettore */
#include <iostream>
#include <fstream>
#include <string>

int v[5], i;

void inx(){ // lettura dei dati dal file di input "input.txt"
    ifstream fin("input.txt");
    for (i=1; i<=5; i++) fin >> v[i];
    fin.close(); }

void elabora(){ // elaborazione dei dati (qui "raddoppia" il vettore)
    for (i=1; i<=5; i++) v[i] = v[i] * 2; }

void outx(){ // memorizzazione dati sul file di output "output.txt"
    ofstream fout("output.txt");
    for (i=1; i<=5; i++) fout << v[i] << "\n";
    fout.close(); }

int main() {
    cout<<"Programma C++ di Input/Output da/su file \n";
    cout<<"*** a - lettura dei dati dal file di input \n";
    inx(); // lettura dei dati dal file di input
    cout<<"*** b - elaborazione dei dati \n";
    elabora(); // elaborazione dei dati
    cout<<"*** c - memorizzazione risultato sul file di output \n";
    outx(); // memorizzazione dati sul file di output
    return 0; }
```

Nell'**Appendice 3- Archivi** è trattata la gestione degli archivi sequenziali e relativi (random) in C.

## **Appendice 1 - Librerie di sistema**

In C e C++ esistono numerose librerie predefinite - o "di sistema" - che mettono a disposizione molte funzioni predefinite, ampliando la potenza del linguaggio. Tali librerie vengono accluse al programma corrente con la direttiva **#include**, ad esempio **#include <stdlib.h>**.

### **STDLIB.H**

<b>atoi()</b>	converte una stringa in numero intero	<code>n = atoi(s);</code>
<b>itoa</b>	converte un numero intero in una stringa	<code>s = itoa(n);</code>
<b>atof()</b>	converte una stringa in numero reale float	<code>x = atof(s);</code>
<b>srand()</b>	inizializza il generatore di numeri casuali	<code>srand(time(0));</code>
<b>rand()</b>	restituisce un numero casuale intero	<code>n = rand() %10; [n = 0÷9]</code>
<b>system()</b>	esegue un comando MS-DOS	<code>system("dir");</code>
<b>malloc()</b>	alloca memoria centrale	<code>int *a; a = malloc(2);</code>
<b>free()</b>	libera la memoria centrale allocata	<code>free(a);</code>
<b>qsort()</b>	ordinamento (QuickSort)	
<b>bsearch()</b>	ricerca binaria	
...		

### **STDIO.H**

<b>printf()</b>	stampa una stringa (formattata con costanti e variabili)	<code>printf("totale %d\n",x);</code>
<b>scanf()</b>	richiede valori passando i puntatori alle variabili	<code>scanf("%d %d",&amp;x,&amp;y);</code>
<b>getchar()</b>	legge carattere da tastiera attendendo il tasto	<code>ch = getchar(); / getchar();</code>
<b>putchar()</b>	visualizza un carattere	<code>putchar('m');</code>
<b>gets()</b>	legge una stringa da tastiera	<code>gets(nomevar);</code>
<b>puts()</b>	visualizza una stringa	<code>puts(" buongiorno! ");</code>
<b>flushall()</b>	svuota il buffer di input	<code>flushall();</code>
<b>FILE</b>	tipo di dati "file", per descrittori di file dati	<code>FILE *fp</code>
<b>fopen()</b>	apre un file di dati	<code>fp = fopen("file.txt","r");</code>
<b>fclose()</b>	chiude un file di dati aperto	<code>fclose(fp);</code>
<b>fread()</b>	legge da un file di dati	
<b>fwrite()</b>	scrive su un file di dati	
<b>fseek()</b>	si posiziona su un record del file di dati	
<b>fputs()</b>	memorizza una stringa sul file	
<b>fputc()</b>	memorizza un carattere sul file	
<b>fgets()</b>	legge una stringa (che finisce con '\n' o '0') dal file	
<b>fgetc()</b>	legge un carattere dal file	
....		

### **IOSTREAM.H**

<b>cin</b>	input da tastiera	<code>cin &gt;&gt; a;</code>
<b>cout</b>	output a video	<code>cout &lt;&lt; "risultato: " &lt;&lt; x;</code>
...		

### **CONIO.H**

<b>getch()</b>	legge carattere da tastiera attende il tasto non visualizza	<code>ch=getch(); / getch();</code>
<b>getche()</b>	legge carattere da tastiera attende il tasto visualizza	<code>ch=getche(); / getche();</code>
<b>clrscr()</b>	pulisce il video	<code>clrscr();</code>
<b>gotoxy()</b>	posiziona il cursore a colonna e riga specificate	<code>gotoxy(40,10);</code>
<b>wherex()</b>	restituisce la posizione orizzontale del cursore	<code>x=wherex();</code>
<b>wherey()</b>	restituisce la posizione verticale del cursore	<code>y=wherey();</code>
<b>kbhit()</b>	controlla se è stato premuto un tasto da tastiera	<code>printf(" premi un tasto");</code> <code>while (!kbhit());</code>
...		

## FSTREAM.H

<b>ifstream()</b>	definisce (ed apre) un file di input (nell'esempio ' <u>fin</u> ')	ifstream fil("dati.txt");
<b>ofstream()</b>	definisce (ed apre) un file di output (nell'esempio ' <u>fou</u> ')	ofstream fou("risult.txt");
...		

## STRING.H

<b>strlen()</b>	restituisce la lunghezza della stringa	n=strlen(" buonasera");
<b>strcpy()</b>	copia la seconda stringa sulla prima	strip(s1,s2);
<b>strcat()</b>	concatena la seconda stringa alla prima	strcat(s1, "ciao");
<b>strstr()</b>	cerca s2 in s1 e se c'è da la sua posizione, se no NULL	p = strstr(s1, "pippo");
<b>strchr()</b>	cerca un carattere in una stringa e se c'è da la sua posizione, se no NULL	p = strstr(s1, "p");
<b>strcmp()</b>	confronta due stringhe e restituisce 0 se uguali, < 0 se s1 < s2, > 0 se s1 > s2	if (strcmp(s1,s2)= =0)...
...		

## MATH.H

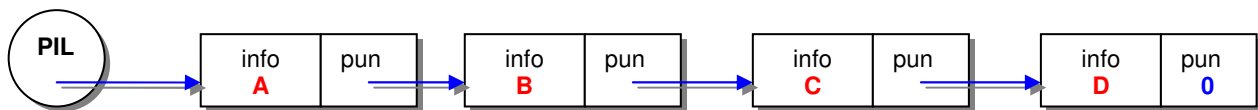
<b>abs()</b>	valore assoluto di un numero intero	a = abs(a);
<b>sqrt()</b>	radice quadrata di un numero double	x = sqrt(y);
<b>pow()</b>	elevamento a potenza (primo base, secondo esponente)	z = pow(x,y);
<b>sin()/cos()/tan()</b>	seno, coseno, tangente di un angolo	y = sin(x); y = cos(x); ...
<b>log()</b>	logaritmo naturale	y = log(x);
<b>log10()</b>	logaritmo in base 10	y = log10(x);
...		

## TIME.H

<b>time()</b>	da il numero di secondi associato al timer	srand(time(0));
<b>difftime()</b>	da la differenza in secondi tra due tempi/date	
<b>clock()</b>	da il numero di 'tick' della CPU dall'inizio del processo	
...		

## Appendice 2 - Lista

Esempio di struttura dati dinamica: LISTA (semplice)



Definizione degli elementi della lista:

```
struct elemento          // definisce il tipo dell'elemento della lista
{ int info;              // "info" = informazine
  elemento *pun; };      // "pun" = puntatore all'elemento successivo
elemento *elem;          // definisce il puntatore a un generico elemento della lista
elemento *pil;            // definisce il puntatore all'elemento iniziale (PIL)
```

Elenco di tutti gli elementi della lista:

```
elem = pil;              // da al punt.'elem' il valore del punt.'pil' (primo)
while (elem != NULL)     // fin quando non si raggiunge il puntatore a NULL:
{ cout<<elem->info;      // visualizza l'infomazione dell'elemento corrente
  elem=elem->pun; }      // passa all'elemento successivo (puntato dal pun)
                        // dando al punt.'elem' il valore del punt.'elem.pun'
```

Inserimento di un elemento come primo della lista (in cima):

```
elemento *nuovo;          // definisce il puntatore al nuovo elemento
nuovo = pil;              // da al punt 'nuovo' il valore del punt.'pil' (primo)
pil = new elemento;       // alloca il nuovo pil e da il suo indirizzo al punt.'pil'
pil->info = x;             // da il valore (info) all'elemento inserito
pil->pun = nuovo;          // da al suo punt. l'indirizzo del vecchio pil (ora secondo)
```

Inserimento di un elemento dopo un elemento - `elem` - della lista (in mezzo/in fondo):

```
elemento *nuovo;          // definisce il puntatore al nuovo elemento
nuovo = new elemento;     // alloca nuovo elemento e da il suo indir. al punt. nuovo
nuovo->info = x;           // da il valore (info) all'elemento inserito)
nuovo->pun = elem->pun;    // da al suo punt. il valore del punt. del precedente 'elem'
                        // (collegandolo così al successivo)
elem->pun = nuovo;         // da al punt. del precedente 'elem' l'indirizzo del nuovo
                        // (collegandolo così al prcedente)
```

Cancellazione del primo elemento della lista:

```
elemento *erase;          // definisce un puntatore di appoggio per eliminazione
erase = pil;              // da al punt.'erase' il valore del punt.'pil' (primo)
pil = pil->pun;            // da al punt.'pil' il valore del punt.del primo elemento
                        // (e quindi ora 'pil' punta al secondo)
free(erase);              // libera la memoria occupata dall'elemento cancellato
```

Cancellazione di un elemento successivo a un elemento - `elem` - della lista:

```
elemento *erase;          // definisce un puntatore di appoggio per eliminazione
erase = elem->pun;         // da al punt.'erase' il valore del punt.del precedente
'elem'
elem->pun = elem->pun->pun; // da al punt. del precedente 'elem' il valore del
                        // punt.dell'elemento da cancellare
free(erase);              // libera la memoria occupata dall'elemento cancellato
```



## Appendice 3 - Archivi

L'accesso al file fisico su disco viene fatto tramite il puntatore al suo indirizzo iniziale, che viene definito di tipo FILE e inizializzato all'apertura dell'archivio:

```
FILE *filepointer;
```

definizione puntatore al file

```
filepointer = fopen(nomefile, modalità);
```

apertura archivio

*modalità* per file sequenziali: **r** (lettura, inizio file), **r+** (lettura e scrittura, inizio file), **w** (scrittura, inizio file), **w+** (scrittura, inizio file), **a** (scrittura, fine), **a+** (lettura e scrittura, fine file)

*modalità* per file random (relativi): **rb** (lettura), **r+b** (lettura e scrittura), **wb** (scrittura), **w+b** (lettura e scrittura), **ab** (scrittura), **a+b** (lettura e scrittura)

```
esempio: FILE *fp;          fp = fopen("Clienti.dat", "r");
```

Dopo l'apertura si possono leggere e/o scrivere i record, dopo di che si chiude con **fclose**(*filepointer*);

archivio sequenziale

Scrittura record: **fprintf**(*filepointer*, *stringa formattata*, *elenco campi*);

```
esempio: fprintf(fp,"%i %s %c\n", codice, nome, sesso);
```

Letture record: **fscanf**(*filepointer*, *stringa formattata*, *elenco indirizzi campi*);

```
esempio: fscanf(fp,"%i %s %c\n", &codice, &nome, & sesso);
```

Il fine file viene segnalato dalla funzione booleana **fEOF**(*filepointer*);

archivio random

Per permettere l'accesso diretto, i record debbono essere a lunghezza fissa e definiti con **struct**:

```
struct struttura { elenco definizione campi };
```

## definizione struttura

*struttura record*

### definizione record

```
esempio: struct recalu { int matr; char nome[20], char sesso; };
         recalu alu;
```

I singoli campi vengono riferiti con *record.campo* (esempio: *alu.nome*)

Prima di leggere/scrivere un record ci si deve posizionare su di esso con **fseek**:

```
fseek(filepointer, posizione, SEEK_SET);
```

## posizionamento

dove  $posizione = \text{sizeof}(\text{record}) * \text{numero\_record}$

```
esempio: fseek(fp, sizeof(alu) * 30, SEEK_SET);
```

posizionamento sul 30-esimo record

Scrittura record: **fwrite**(*indirizzo record*, *lunghezza record*,1,*filepointer*);

```
esempio: fwrite(&alu,sizeof(alu),1,fp)
```

Lettura record: **fread**(indirizzo record, lunghezza record,1,filepointer);

```
esempio: fread(&alu,sizeof(alu),1,fp)
```

```

/* Gestione archivio SEQUENZIALE in Dev-C++, prof. Claudio Maccherani, 2008
+-----+-----+-----+-----+-----+
|matricola | nome | sesso | classe | sezione | corso | tasse | città |
+-----+-----+-----+-----+-----+ */
FILE *fp; // fp è il "file pointer" per il descrittore del file
int matr, classe, tasse; // matricola, classe, tasse (campi)
char sex, sez; // sesso, sezione (campi)
char nome[31], corso[3], citta[21]; // nome, corso, città (campi)
int main() { //-----
    ....
    .... }
void scrittura() { // memorizzazione di un nuovo record -----
    fp = fopen("FileSeq.txt", "a"); // file aperto in "a"-append
    cout<<"Matricola "; cin>>matr; cout<<"Nome "; cin>>nome;
    cout<<"Sesso "; cin>>sex; cout<<"Classe "; cin>>classe;
    cout<<"Sezione "; cin>>sez; cout<<"Corso "; cin>>corso;
    cout<<"Tasse "; cin>>tasse; cout<<"Città' "; cin>>citta;
    fprintf(fp, "%i %s %c %i %c %s %i %s\n",
            matr, nome, sex, classe, sez, corso, tasse, citta);
    fclose(fp); }
void elenco() { // elenco di tutti i record -----
    fp = fopen("FileSeq.txt", "r"); // file aperto in "r"-read
    while (!feof(fp)) {
        fscanf(fp, "%i %s %c %i %c %s %i %s\n",
            &matr, &nome, &sex, &classe, &sez, &corso, &tasse, &citta);
        cout<<matr<<nome<<sex<<classe<<sezione<<corso<<tasse<<citta<<"\n"; }
    fclose(fp); }

```

```

/* Gestione archivio RANDOM/DIRETTO in Dev-C++, prof. Claudio Maccherani, 2008
+-----+-----+-----+-----+-----+
|matricola | nome | sesso | classe | sezione | corso | tasse | città |
+-----+-----+-----+-----+-----+ */
struct recalu { int matr; char nome[31]; char sex; int classe;
                char sez; char corso[3]; int tasse; char citta[21]; };
recalu alu; // definisce il record 'alu' di tipo 'recalu'
FILE *fp; // fp è il "file pointer" per il descrittore del file
const int N = 21; // numero di record dell'archivio (da 0 a 20!)
int main() { //-----
    ....
    .... }
void scrittura() { // memorizzazione di un nuovo record -----
    fp = fopen("FileRnd.dat", "r+b"); // file aperto "r+b"-lett.scritt.
    cout<<"Matricola"; cin>>alu.matr; cout<<"Nome "; gets(alu.nome);
    cout<<"Sesso "; cin>>alu.sex; cout<<"Classe "; cin>>alu.classe;
    cout<<"Sezione "; cin>>alu.sez; cout<<"Corso "; scanf("%s", alu.corso);
    cout<<"Tasse "; cin>>alu.tasse; cout<<"Città' "; scanf("%s", alu.citta);
    cout<<"Numero record (posizione): "; cin>>r; // numero record da scrivere
    fseek(fp, sizeof(recalu)*r, SEEK_SET); // si posiziona sul record
    fwrite(&alu, sizeof(recalu), 1, fp); // scrive il record
    fclose(fp); }
void elenco() { // elenco di tutti i record -----
    fp = fopen("FileRnd.dat", "rb"); // file aperto in "rb": lettura
    for (int r=1; r<N; r++) {
        fseek(fp, sizeof(recstud)*r, SEEK_SET); // si posiziona sul record 'r'
        fread(&alu, sizeof(recstud), 1, fp); // lo legge
        cout<<alu.matr<<alu.nome<<alu.sex<<alu.classe<<alu.sez<<alu.corso
            <<alu.tasse<<alu.citta<<"\n"; }
    fclose(fp); }

```

## Appendice 4 - casualità, posizionamento, colore, tasti, attesa...

### Casualità

Per poter realizzare giochi, ad esempio di carte, occorre simulare la "casualità" degli eventi quali la distribuzione casuale delle carte prima della partita. Per questo C++ mette a disposizione la funzione **RAND() %n** che genera numeri pseudo casuali da 0 a n-1.

**rand() % n**                      **x = rand() %1000; // restituisce in x un numero da 0 a 999**

Ogni volta che viene "chiamata" la funzione rand restituisce un numero diverso, applicando un algoritmo matematico basato su una serie di moltiplicazioni e divisioni con presa in considerazione dei resti di tali operazioni. Questo algoritmo di generazione parte da un numero iniziale e questo implica che, dato lo stesso numero iniziale, la sequenza dei numeri restituiti da **rand** sarà sempre la stessa (numeri pseudo-casuali). Per poter avere effettiva casualità occorre cambiare il numero iniziale del generatore rand e per questo si usa la funzione **SRAND(x)**, dove x rappresenta tale numero iniziale. Per evitare di cambiare ogni volta **X** si può utilizzare la funzione **TIME(0)** che restituisce il tempo attuale in secondi a partire dal primo gennaio 1970. Occorre utilizzare gli header `stdlib.h` e `time.h`.

```
// 10 lanci di un dado
#include <iostream>
#include <time.h>
#include <stdlib.h>
using namespace std;
int main() {
    int i,dado;
    srand(time(0)); // inizializza il generatore di numeri casuali rand()
    cout << "Lancio del dado" << endl;
    for (i = 1; i <= 10; i++) {
        dado = rand() % 6 + 1; // rand da un numero da 0 a 5 (con +1 da 1 a 6)
        cout << dado << endl; }
    return 0; }
```

### Posizionamento

Il video dei vecchi sistemi operativi "a riga di comando" (cioè non grafici) quali MS-DOS e UNIX è una matrice di 25 righe per 80 colonne su ciascun elemento della quale si può visualizzare un singolo carattere associato ad un codice ASCII. In C++ gli output avvengono sequenzialmente e, quando si arriva in fondo inizia lo scrolling in alto della videata. Però, con la funzione **GOTOXY**, è possibile gestire il posizionamento del cursore su ciascun elemento della griglia 25x80.

**gotoxy(riga, colonna)**                      **gotoxy(10,35); // cursore a riga 10 e colonna 35**

Per poter utilizzare tale funzione occorre aggiungere l'header `windows.h` e, prima del main, la seguente intestazione:

```
int gotoxy(int r, int c) {
HANDLE Hout;
CONSOLE_SCREEN_BUFFER_INFO ConsoleInfo;
Hout = GetStdHandle(STD_OUTPUT_HANDLE);
ConsoleInfo.dwCursorPosition.Y = r;
ConsoleInfo.dwCursorPosition.X = c;
SetConsoleCursorPosition(Hout,ConsoleInfo.dwCursorPosition); }
```

## Colori

I programmi C++ generalmente sono "in bianco e nero", ma è possibile cambiare i colori di primo piano e di sfondo combinandoli a piacere. Per far ciò si utilizza la funzione **SetConsoleTextAttribute**.

**SetConsoleTextAttribute(hCon, color)      SetConsoleTextAttribute(hCon, 7); // b & w**

Dove color è il codice della combinazione di colori (da 0 a 255) primo piano/sfondo.

Per poter utilizzare tale funzione occorre aggiungere l'header `windows.h` e, prima del main, la seguente intestazione:

```
HANDLE hCon = GetStdHandle(STD_OUTPUT_HANDLE);
```

## Rilevazione TASTO premuto

Per rilevare quale tasto della tastiera è stato premuto si usa la funzione **GetAsyncKeyState(tasto)**.

**GetAsyncKeyState(tasto)      if (GetAsyncKeyState(VK\_UP) {...}) // freccia in alto**

La funzione (parametro TASTO da esaminare) restituisce il valore *True* se tale TASTO è stato premuto. Per poter utilizzare tale funzione occorre aggiungere l'header `windows.h`.

### TASTI (alcuni ...)

VK_BACK	backspace	VK_TAB	tab	VK_ENTER	Invio
VK_SHIFT	shift	VK_CONTROL	ctrl	VK_MENU	Alt
VK_PAUSE	pause	VK_CAPITAL	caps lock	VK_ESCAPE	Esc
VK_SPACE	spacebar	VK_PRIOR	page up	VK_NEXT	page down
VK_END	end	VK_HOME	home	VK_LEFT	left arrow
VK_RIGHT	right arrow	VK_DOWN	down arrow	VK_UP	up arrow
VK_DELETE	del	VK_INSERT	ins	VK_NUMLOCK	num lock
VK_F1	F1	VK_Fx	Fx	VK_F24	F24

## Attesa

Per attendere un intervallo di tempo si usa la funzione **Sleep(milisecondi)**.

**Sleep(milisecondi)      Sleep(1000) // attende un secondo**

La funzione (parametro TASTO da esaminare) restituisce il valore *True* se tale TASTO è stato premuto. Si può utilizzare la funzione `Sleep` per implementare il metodo `setInterval()` di JavaScript:

```
while(true) { Sleep(1000); mia_funzione(); }
```

Esempio di programma che sposta un "avatar" in base ai tasti di movimento cursore:

```
/*Dev-C++ - prof. Claudio Maccherani - Perugia - 2018/19
+-----+
| Per la gestione dei colori, il posizionamento del cursore (GOTORC) e |
| la rilevazione dei tasti premuti occorre utilizzare le API di Windows. |
| - #include <windows.h> |
| - HANDLE hCon = GetStdHandle(STD_OUTPUT_HANDLE); |
| - SetConsoleTextAttribute (hCon, x) |
|   dove 'x' è il codice colore (un numero intero compreso tra 0 e 255) |
| - goto(R,C) [per posizionare il colore a riga R e colonna C] |
|   dove 'R' va da 1 a 25 e 'C' va da 1 a 80 |
| - GetAsyncKeyState(TASTO) [per rilevare quale tasto è stato premuto] |
| - Sleep(TEMPO) [per attendere un TEMPO di millisecondi] |
+-----+*/

#include <iostream>
#include <string.h>
#include <windows.h>

HANDLE hCon=GetStdHandle(STD_OUTPUT_HANDLE); //per colori primo piano/sfondo

using namespace std;

int Rl=1,C1=1,R,C; char Avatar='O'; int attesa=100;
char tracciamento='n'; string G[24];

void gotorc(short r, short c) {
    COORD pos = {c, r};
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), pos); }

void Inizializza() { // inizializza il "campo di gioco" G
G[00]="01234567890123456789012345678901234567890123456789012345678901234567";
G[01]="#####";
G[02]="#";
G[03]="#          #####";
G[04]="#          #          #";
G[05]="#          #          #          #          #####";
G[06]="#          #####          #          #";
G[07]="#          #          #          #####          #";
G[08]="#          #####          #          #";
G[09]="#          #          #          #          #####";
G[10]="#          #####          #          #          #";
G[11]="#          #          #          #          #####          #";
G[12]="#          #####          #          #          #";
G[13]="#          #          #          #          #          #####";
G[14]="#          #####          #          #          #";
G[15]="#          #####          #          #####          #";
G[16]="#          #          #          #          #####          #";
G[17]="#          #          #          #####          #";
G[18]="#";
G[19]="#          #####          #####";
G[20]="#";
G[21]="#";
G[22]="#";
G[23]="#####";
}

void Disegna_Riquadro() { // disegna il terreno di gioco
    for(int i=1;i<=23;i++) { gotorc(Rl+i-1,C1-1); cout<<G[i]<<"\n"; }
    gotorc(Rl+23,C1);
    cout<<"usa i TASTI di Movimento Cursore - ESC per terminare"; }
```

```

void Spostamento(char Tasto) { // spostamento dell'avatar
    int r,c; char x = ' '; if (tracciamento=='s') x = Avatar;
    switch(Tasto) {
        case 'A': // freccia in alto
            if (G[R-1][C]==' ')
                {gotorc(R+R1-1,C+C1-1); cout<<x; Avatar = '^'; R--;}
            break;
        case 'B': // freccia in basso
            if (G[R+1][C]==' ')
                {gotorc(R+R1-1,C+C1-1); cout<<x; Avatar = 'v'; R=R+1;}
            break;
        case 'D': // freccia a destra
            if (G[R][C+1] == ' ')
                { gotorc(R+R1-1,C+C1-1); cout<<x; Avatar = '>'; C=C+1;}
            break;
        case 'S': // freccia a sinistra
            if (G[R][C-1] == ' ')
                { gotorc(R+R1-1,C+C1-1); cout<<x; Avatar = '<'; C=C-1;}
            break;
    }
    gotorc(R+R1-1,C+C1-1); cout << Avatar;
}

int main() {
    SetConsoleTextAttribute (hCon, 159); // setta colore "bianco/blu"
    system("cls");
    cout << "Vermino/Murino <Claudio Maccherani> con TRACCIAMENTO [s/n]?";
    cin >> tracciamento;
    Inizializza();
    Disegna_Riquadro(); // disegna il riquadro di spostamento
    R = 7; C = 37; gotorc(R+R1-1,C+C1-1);
    cout << Avatar; // posiziona l'atavar
    while (GetAsyncKeyState(VK_ESCAPE)==0) { // gioca (sposta l'avatar)
        if(GetAsyncKeyState(VK_DOWN)) { Spostamento('B'); }
        if(GetAsyncKeyState(VK_UP)) { Spostamento('A'); }
        if(GetAsyncKeyState(VK_LEFT)) { Spostamento('S'); }
        if(GetAsyncKeyState(VK_RIGHT)) { Spostamento('D'); }
        Sleep(attesa); // millisecondi di 'attesa'
    }
    system("cls"); cout << "\n CIAO \n";
    return 0;
}

```

## Appendice 5 - funzione SYSTEM

La funzione **SYSTEM** permette di eseguire, da programma C++, un qualsiasi comando MS-DOS (per l'elenco dei possibili comandi si veda la dispensa "Sistema Operativo MS-DOS").

```
system(comando)           system("cls"); // pulisce il video
```

Dove comando è un comando eseguibile del sistema operativo MS-DOS. Ad esempio:

<b>system("cls");</b>	pulisce (imbianca) il video
<b>system("pause");</b>	arresta l'esecuzione fino a che non si da INVIO
<b>system("dir");</b>	elenca tutti i file e le sottodirectory della directory corrente
<b>system("dir a*.cpp");</b>	elenca tutti i programmi C++ il cui nome inizia per 'a'
<b>system("notepad");</b>	manda in esecuzione il Blocco Note (Notepad)
<b>system("D:\pippo.exe");</b>	manda in esecuzione il programma 'pippo.exe' del disco D
<b>system("date");</b>	modifica la data di sistema con quella che l'utente inserisce
<b>system("md pluto");</b>	crea la directory/cartella 'pluto'
<b>system("print leggimi.txt");</b>	stampa il file 'leggimi.txt'
<b>system("tree");</b>	visualizza la struttura ad albero della directory/cartella corrente
<b>etc ...</b>	etc ...