# Corso sul linguaggio C++

## Modulo LC

## 1.2.1 - I tipi di dato



# Prerequisiti

- Concetti matematici di:
  - Funzione
  - Espressione
  - Insieme
- Bit, byte e numerazione binaria

# Introduzione

Un programma opera su *variabili* che, in base al problema da automatizzare, possono essere di vario tipo. Perciò ci chiediamo:

**Quali tipi di dato offre C++?**

**Quali operatori possiamo usare per ciascun tipo di dato?**

In questa Unità descriviamo i tipi di dato, mentre nella successiva, vedremo come utilizzare le variabili per scrivere espressioni anche complesse che il programma dovrà elaborare.



# Informazioni generali

- Esistono diversi **tipi di dato** che possiamo utilizzare, in base alle specifiche richieste del problema.
- Mediante questi tipi di dato possiamo dichiarare **costanti** e **variabili**
- Ogni tipo di dato occupa una precisa quantità di memoria

Vediamo in ordine questi concetti.

# Tipizzazione dei dati

C++, come molti altri linguaggi di programmazione, mette a disposizione del programmatore tipi di dato predefiniti, per questo motivo, Java, come molti linguaggi simili, si dice **linguaggio tipizzato**.

Vediamo i tipi di dato di C++.



# Tipi di dato

I tipi di dato elementare che descriviamo sono:

- **Carattere**
- **Stringa**
- **Booleano**
- **Intero**
- **Float**
- **Double**
- **Enumerato**

# Tipo char

Il tipo di dato **char** si usa per rappresentare caratteri.

- Occupa **1 byte** di memoria.
- Con 1 byte è possibile rappresentare **256 diversi** possibili caratteri ($2^8$=256)
- La dichiarazione di una variabile di tipo char è:

  **char** c;
- I caratteri costanti vengono indicati racchiudendoli tra apici (‘). Ad esempio:

```
char car;       /* dichiarazione variabile di tipo carattere */
car='?';        /* assegnazione del carattere costante */
cout<<car;      /* stampa a video il carattere '?' */
```



# Tipo char
## - codice ASCII

In programmazione i caratteri sono rappresentati mediante un codice detto **codice ASCII** (*American Standard Code For Information Interchange*) che associa a ciascun carattere un codice intero.

Possiamo supporre cioè che:

**ASCII**(carattere) = intero

**ASCII**$^{-1}$ (intero) = carattere

Ad es.

**ASCII**(‘A’)=65; **ASCII**$^{-1}$(65)=‘A’; **ASCII**(‘B’)=66; **ASCII**$^{-1}$(66)=‘B’;

Il codice **ASCII** consta di 256 caratteri (da 0 a 255) ciascuno individuabile mediante il corrispondente valore intero.

# Tipo char
## - conversione implicita

```
/* ConvCarInt.cpp */
#include <iostream>
using namespace std;
int main()
{   char c;
    int asc;
    cout<<"Immetti un carattere: ";
    cin>>c;
    asc=c;
    cout<<"ascii di "<<c<<" "<<asc<<endl;
    cout<<"Immetti un intero: ";
    cin>>asc;
    c=asc;
    cout<<"char di "<<asc<<" "<<c<<endl;
    system("Pause");
    return EXIT_SUCCESS;
}
```

In pratica il C++ esegue una **conversione implicita** a seconda del tipo della variabile usata.
La variabile letta c è di tipo carattere, ma viene assegnata ad *asc*, di tipo intero.

Viceversa, leggendo un intero *asc* e assegnandolo ad una variabile di tipo carattere *c* si realizza la funzione inversa.



# Tipo char
## - conversione implicita

```
/* ConvCarInt.cpp */
#include <iostream>
using namespace std;
int main()
{
    char c, prec, succ;
    cout<<"Immetti un carattere: ";
    cin>>c;
    prec=c-1;
    succ=c+1;
    cout<<"prec: = "<<prec<<endl;
    cout<<"succ: = "<<succ<<endl;
    system("Pause");
    return 0;
}
```

Altro esempio di **conversione implicita**.
Le operazioni aritmetiche su *c*, trattano *c* come fosse una variabile intera.

L'assegnazione successiva a *prec* e *succ*, ritrasforma gli interi in caratteri.

# Tipo char

## - funzioni get e put

Alcune funzioni sui caratteri

```
#include <iostream>    //getch-putchar.cpp
#include <conio.h>     // per l'uso di getch()
using namespace std;
int main(int argc, char *argv[])
{   int ch;
    cout<<"(IMMISSIONE SENZA ECHO) Premi un tasto: ";
    ch = getch();
    cout<<"\nHai premuto ";   putchar(ch);
    cout<<"\n(IMMISSIONE CON ECHO) Premi un tasto: ";
    ch = getche();
    cout<<"\nHai premuto ";   putchar(ch);
    cout<<"\n(IMMISSIONE SENZA INVIO) Premi un tasto: ";
    ch = getchar();
    cout<<"Hai premuto ";   putchar(ch);
    cout<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

**getch()** – legge un carattere senza darne l'eco

**getche**() – legge un carattere dandone l'eco

**getchar** () – legge un carattere senza darne l'eco



# Tipo char

## - funzioni predefinite

Altre funzioni sui caratteri

| | |
|---|---|
| **if** (**isalpha** (*car*)) *istruzione* | **vero** se *car* è alfabetico |
| **if** (**isupper** (*car*)) *istruzione* | **vero** se *car* è maiuscolo |
| **if** (**islower** (*car*)) *istruzione* | **vero** se *car* è minuscolo |
| *car1* = **tolower** (*car*); | converte in minuscolo |
| *car1* = **toupper** (*car*); | converte in maiuscolo |
| **if** (**ispunct** (*car*)) *istruzione* | **vero** se *car* è punteggiatura |
| **if** (**isdigit** (*car*)) *istruzione* | **vero** se *car* è una cifra |
| **if** (**isspace** (*car*)) *istruzione* | **vero** se *car* è lo spazio |

Il costrutto **if** (*condizione*) *istruzione* consente di eseguire una *istruzione* nel caso la *condizione* risulti vera. In caso *condizione* sia falsa, *istruzione* non viene eseguita.

# Tipo char

## - conversione esplicita (cast)

```
#include <iostream>    //tolower-toupper.cpp
#include <cstring>
using namespace std;

int main(int argc, char *argv[])
{  char c, conv;
   cout<<"Immetti un carattere: ";
   cin>>c;
   cout<<"Precedente maiuscolo: "<<(char)(toupper (c-1))<<endl;
   cout<<"Precedente minuscolo: "<<(char)(tolower (c-1))<<endl;
   system("PAUSE");
   return EXIT_SUCCESS;
}
```

**ATTIVITA**':Scrivere un programma che legga un carattere e stampi il precedente sia maiuscolo che minuscolo.

Poiché le funzioni *tolower*() e *toupper*() restituiscono un valore intero, è necessaria la **conversione esplicita**, che si realizza con l'operatore **cast** con:

(*tipo*) *espressione*

dove *tipo* è il tipo di arrivo in cui viene convertita *espressione*



# Tipo stringa

Mediante i caratteri possiamo formare sequenze dette **stringhe.**

- Una **variabile stringa** si dichiara usando il tipo **string**:

  **string** *nome, cognome*; // dichiara 2 variabili stringa

- Una **stringa costante** si indica racchiudendole tra virgolette.

  **#define** MSG "Immetti valore: " /* dichiar. stringa costante */

  **string** cognome("Bianchi"); /* assegnaz. stringa costante*/

- **Una stringa occupa in memoria un numero di byte uguale alla dimensione massima della stringa.**
  - Nell'esempio, *MSG* occupa 14 byte

# Tipo stringa
## - esempi

- Le stringhe si leggono e si stampano mediante le istruzioni **cin** e **cout**. Ad esempio:

```
……..
string cognome();                    /* dichiara stringa cognome*/
cin>>cognome;                        /* legge da input la stringa */
……                                   /* es. l'utente digita Rossi */
cout<<"Buongiorno Sig. "<<cognome;
```

in stampa:

Buongiorno Sig. Rossi



# Tipo stringa
## - carattere di fine stringa

Quando si legge una stringa da tastiera, alla pressione del tasto INVIO, dopo l'ultimo carattere immesso, nella stringa viene posto il carattere '\0' detto **terminatore,** che ha lo scopo di segnalare la fine della stringa.

*N.B.: è da notare che l'istruzione* **cin** *rispetta il carattere '\0', per cui nel modo detto non è possibile inserire ad esempio stringhe che contengono spazi (es: "Il linguaggio C++"), perché verrebbe letta solo la stringa "Il". Per leggere stringhe contenenti spazi è necessaria una variante dell'istruzione* **cin** *(v. seguito Unità)*

# Tipo stringa
## - funzioni predefinite

**string** s1,s2, s3; **char** c; **int** p;

| | |
|---|---|
| **cout**<<s1.**size**(); | Stampa la lunghezza di *s1* |
| s1="le"; | Assegnazione a *s1* di una stringa costante |
| s3=s1+s2; | Concatena *s1* con *s2* |
| s2=s1.**substr** (3, 5) | Pone in s2 i primi 5 caratteri di s1 a partire da posizione 3 |
| s2=s1.**replace** (6, 5, "casa") | Sostituisce 4 caratt. di *s1*, con "casa", partendo da pos. 6 |
| **getline** (**cin**, *s*); | Legge la stringa *s* anche se presenta spazi |
| c=s1.**at** (4); | Seleziona il carattere in posizione 4 di *s1* |
| p=s1.**find_first_of** ('a'); | Trova la prima occorrenza di 'a' in *s1* |
| p=s1.**find_last_of** ('?') | Trova l'ultima occorrenza di '?' in *s1* |
| s2=s1.**insert**(3, "tra") | Inserisce in *s1* la stringa "tra", in posizione 3 |
| p=s1.**compare**(s2); | *p*=0 (uguali), p=-1 (*s1*<*s2*) p=-1 (*s1*>*s2*) |
| s1.**swap**(s2); | Scambia *s1* con *s2* |



# Tipo stringa
## - funzioni predefinite

```
#include ….   //StrCatLen.cpp
using namespace std;
int main(int argc, char *argv[])
{ string nome, cognome;
  cout<<"Immetti il tuo nome: ";
  cin>>nome;
  cout<<"Ciao "<<nome<<endl;
  cout<<"Il tuo nome e' formato da "<<nome.size()<<" caratteri\n";
  cout<<"Qual e' il tuo cognome ?";
  cin>>cognome;
  cout<<"Il tuo cognome e' formato da "<<cognome.size()<<" caratteri\n";
  cout<<"O.K.! il tuo nome completo e' "<<nome + cognome<<endl;
  system("Pause");
  return EXIT_SUCCESS;
}
```

Esempi di funzioni predefinite sul tipo **string**.

# Tipo bool

Il tipo di dato **bool** si usa per rappresentare variabili di tipo logico ossia variabili che possono assumere uno solo tra due possibili valori: **true** o **false**.

- **true** e **false** sono due **costanti bool** di tipo logico (o booleano):
  - **false** è rappresentato dal valore 0
  - **true** dal valore 1
- Il tipo **bool** occupa **1 byte** di memoria.
- Per dichiarare una variabile di tipo **bool** si scrive ad esempio:

  **bool** finito;



# Tipo int

Il tipo di dato **int** si usa per rappresentare variabili di tipo intero con segno.

- Occupa **2 byte** di memoria: 15 bit per il valore ed 1 bit per il segno
- Con 2 byte è possibile rappresentare valori che vanno da –65536 a +65535.
- In generale il range dei valori è individuato dalle due costanti **INT_MIN** e **INT_MAX**
- Per dichiarare una variabile **int** si scrive ad esempio:

  **int** conteggio; /* dichiarazione variabile conteggio */

# Tipo float

Il tipo di dato **float** si usa per rappresentare variabili di tipo reale in virgola mobile.

- Il tipo float occupa **4 byte.**
- Con 4 byte è possibile rappresentare valori che vanno da ±1.2E-38 a ±3.4E38.
- Per dichiarare variabili di tipo **float** si scrive ad esempio:

  **float** raggio; /* dichiarazione variabile reale raggio */



# Tipo float

```
// ConvPiediMetri.cpp
#include <cstdlib>
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{  int piedi;
   float metri;
   cout<<"Immetti la misura espressa in piedi: ";
   cin>>piedi;
   metri = piedi * 0.304;
   cout<<piedi<<" piedi corrispondono a "<<metri<<" metri"<<endl;
   system("Pause");
   return EXIT_SUCCESS;
}
```

**ATTIVITA'**: Scrivere un programma che legga una misura intera *piedi* e la converta nella corrispondente misura *metri*, reale espressa in metri

(1 m = 0.304 piedi)

Il risultato dell'espressione *piedi*0.304* è reale e quindi adatto alla variabile di destinazione *metri*.

# Tipo double

Il tipo di dato **double** si usa per rappresentare variabili di tipo reale in virgola mobile ma con una **precisione doppia rispetto al float**.

- Occupa **8 byte** di memoria:
- Con 8 byte è possibile rappresentare valori che vanno da ±2.2E-308 a ±1.8E308.
- Per dichiarare una variabile di tipo **double** si scrive ad esempio:
  **double** raggio-orbita;



# Tipo enum

Il tipo **enum** consente di associare a delle costanti intere delle etichette scelte dal programmatore. Ciò consente una maggiore documentazione del programma.

La sintassi dichiarativa del tipo **enum** è la seguente:

**enum** *ident* {*listavalori*};

**enum** è la parola chiave per il tipo di dato

dove

- *ident* è il nome che il programmatore dà al tipo enumerato
- *listavalori* è un elenco di identificatori, scelti dal programmatore, ciascuno associato ad un numero intero progressivo iniziante dal valore 0.

# Tipo enum

- Esempio:
  **enum** colori {rosso, giallo, blu};  // definisce il tipo
  colori colore;  // dichiara la variabile colori di tipo colore

- Quando ciascuna etichetta deve essere associata ad un intero **NON** in progressione. Ad esempio:

  **enum** colori {rosso=3, giallo=5, blu=10};
  colori colore;
  …..
  colore = rosso;  // equivalente a  colore = 3;



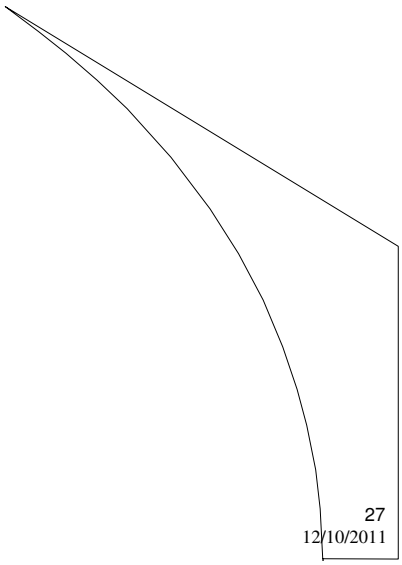
# Tipo enum

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

int main(int argc, char *argv[])
{    enum mesi{Gennaio, Febbraio, Marzo, Aprile,
          Maggio, Giugno, Luglio, Agosto,
          Settembre, Ottobre, Novembre, Dicembre};
     mesi mese;
     mese=Dicembre;
     cout<<"Dicembre corrisponde a "<<mese<<endl;
     system("PAUSE");
     return EXIT_SUCCESS;
}
```

# Argomenti

- Tipi di dato
- **char**
- Stringhe
- **bool**
- **int**
- **float**
- **double**
- **enum**





# Altre fonti di informazione

- J. Purdum, C – ed. Jackson
- Romagnoli-Ventura, C/C++ - Ed. Petrini
- A.Bellini-A.Guidi, Conoscere il C – ed. McGraw Hill
- A. Lorenzi et alii – Il linguaggio C++ - Ed. ATLAS