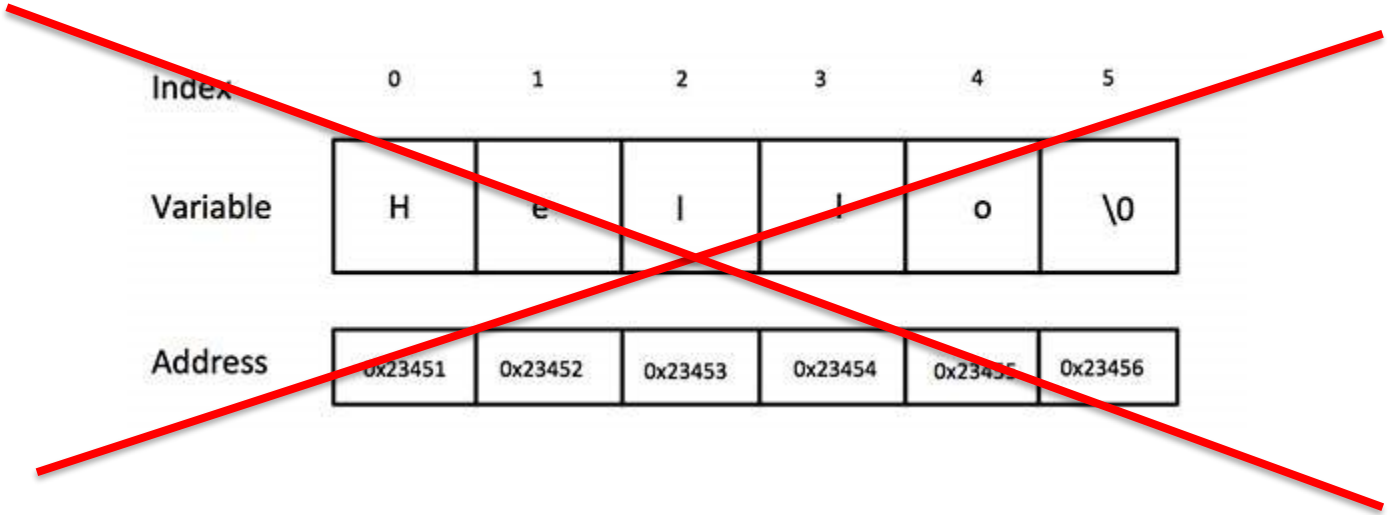
UNIVERSITÀ
DI PARMA

C++

Alberto Ferrari



la classe

string

string

- una stringa è una sequenza di 0 o più caratteri racchiusi fra doppi apici
  - `string corso = "Informatica e laboratorio di programmazione";`
- la classe string non fa parte del linguaggio C++ ma è inclusa nella libreria standard
- per utilizzare oggetti della classe string (variabili di tipo string) è «necessario» includere la libreria
  - `#include <string>`

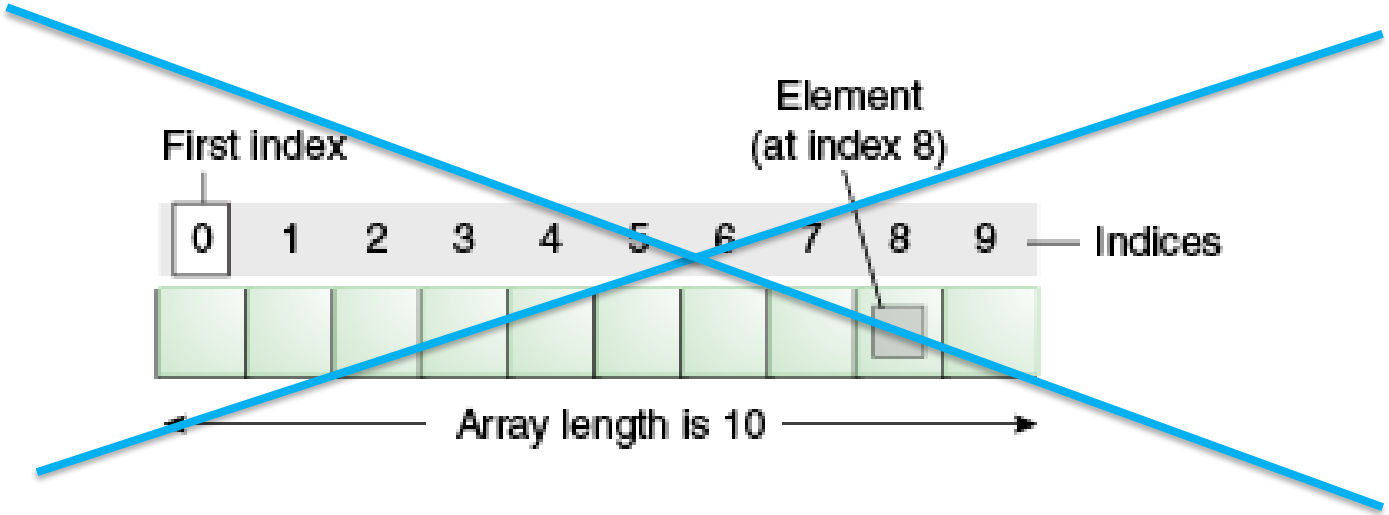
operatori sulle stringhe

- operazione di indicizzazione **[ ]**
  - l'indice del primo carattere è 0 e quello dell'ultimo è uguale alla lunghezza della stringa -1
- operatore di concatenazione **+**
  - almeno uno dei due operandi deve essere un oggetto di tipo string
- funzioni definite sulle stringhe
  - **length**() o **size**()
    - restituisce il numero di caratteri presenti nella stringa
  - **find**(**s**)
    - ricerca la prima occorrenza della stinga s nella stringa in cui è invocata
  - **substr**(**i_inizio**,**lung**)
    - restituisce la sottostringa di lunghezza lung a partire dal carattere di indice i_inizio

string - esempio

```
#include <iostream>
#include <string>
using namespace std;
int main() {
    string s1,s2,s3;
    s1 = "Informatica";
    s2 = s1 + " e laboratorio " + "di programmazione";
    // error: invalid operands of types '...' and '...' to binary 'operator+'
    // s3 = "Informatica" + " e laboratorio";
    s2[0] = 'i';          // sostituzione del primo carattere della stringa
    cout << "contenuto della stringa s2: " << s2 << endl;
    cout << "numero di caratteri della stringa s2: " << s2.length() << endl;
    s3 = "Ingegneria dei Sistemi Informativi";
    int pos;
    pos = s3.find("In");
    cout << "nella stinga " << s3 << endl << "la sottostringa " << "In"
         << " si trova in posizione " << pos << endl;
    pos = s3.find("out");
    cout << "nella stinga " << s3 << endl << "la sottostringa " << "out"
         << " si trova in posizione " << pos << endl;
    cout << s3.substr(15,7) << endl;
    return 0;
}
```

http://www.cplusplus.com/reference/string/string/

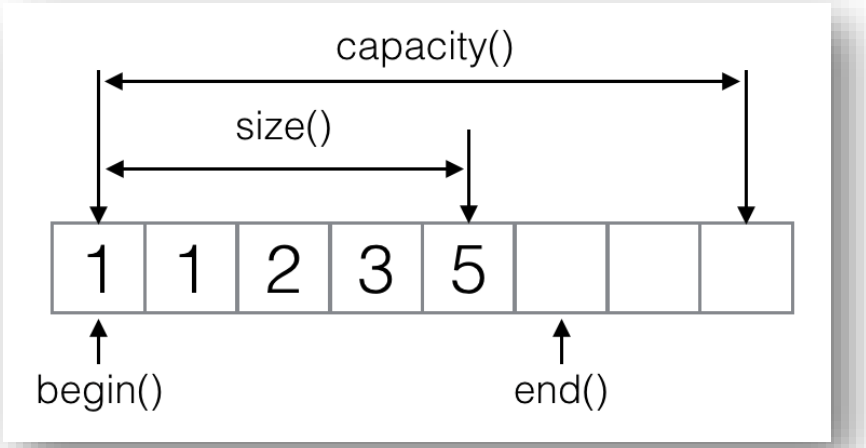


la classe

vector

vector – accesso agli elementi

- ***vector***
  - è un *array* con *dimensione variabile*
- la notazione **[]**
  - può essere usata per
    - ***leggere*** un elemento
    - ***cambiare*** un elemento che ha già un ***valore***
  - ***non*** può essere usata per
    - ***inizializzare*** un elemento
  - ***non controlla*** se l'elemento esiste
- per ***aggiungere*** elementi (in ordine di posizione) si usa il metodo ***push_back()***



vector (generics)

- «necessario» specificare il tipo degli elementi del vector

```
vector<DataType> nameOfVector
vector<int> v1    vector<string> v2    vector<Ball> v3
```

```
vector<string> myVector;

// push_back aggiunge un elemento alla fine del vector e lo ridimensiona
myVector.push_back("el_1");
myVector.push_back("el_2");
myVector.push_back("el_3");
// pop_back elimina l'ultimo elemento
myVector.pop_back();

// [index] accesso all'elemento di indice index
myVector[1] = "el_002";

vector<int> intVect(5,0);       // 5 elementi inizializzati a 0
```

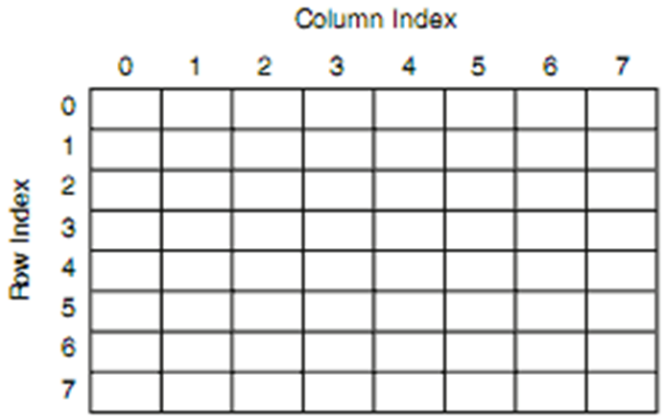
accesso agli elementi

```
// e : vector (accesso sequenziale agli elementi di vector)
for (auto e : myVector)
    cout << e << " ";
cout << endl;

// [index] accede all'elemento di indice index
// size restituisce il numero di elementi presenti
for (unsigned i=0; i<myVector.size(); ++i)
    cout << myVector[i] << " ";
cout << endl;

// at(index) restituisce il valore dell'elemento di indice index
for (unsigned i=0; i<myVector.size(); ++i)
    cout << myVector.at(i) << " ";
cout << endl;
```

insert – erase – empty - clear


```
// begin() restituisce il riferimento al primo elemento
// insert(index_ref, value) inserisce value dopo l'elemento riferito
myVector.insert(myVector.begin() +2,"el_5");

// erase(index_ref) elimina l'elemento riferito da index_ref
myVector.erase(myVector.begin() +2);

// empty() restituisce true se non sono presenti elementi
// clear() elimina tutti gli elementi
if (!myVector.empty())
{
    cout << "sono presenti elementi -> li elimino" << endl;
    myVector.clear();
}
if (myVector.empty())
    cout << "non sono presenti elementi" << endl;
```

```
// ricerca di un elemento
// begin(myVector) riferimento al primo elemento
// end(myVector) riferimento all'alemento dopo l'ultimo
auto pos = find(begin(myVector), end(myVector), "el_2");
if (pos != end(myVector)) {      // trovato
    myVector.erase(pos);
}

//assegnamento valori iniziali
vector<string> lista;
lista.assign(10, ""); // 10 stringhe vuote
for (auto e : lista)
    cout << e << " - ";
cout << endl;
```



C++

matrici

matrice (inizializzazione)

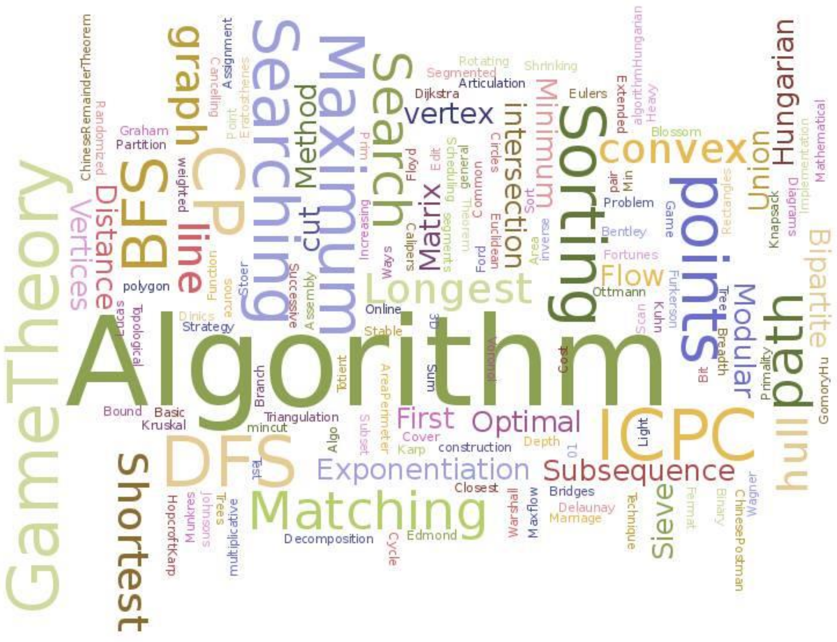
```
// matrice inizializzata
vector<vector<int>> matrice = { {2, 4, 3, 8},
                                {9, 3, 2, 7},
                                {5, 6, 9, 1} };

auto righe_m = matrice.size();       // numero di righe
auto colonne_m = matrice[0].size(); // numero di colonne

for (auto r = 0; r < righe_m; ++r) {
    for (auto c = 0; c < colonne_m; ++c) {
        cout << matrice[r][c] << " ";
    }
    cout << endl;
}
```

matrice «vuota»

```
auto righe_v = 6;
auto colonne_v = 8;

vector <vector<int> > vuota(righe_v, vector<int>(colonne_v));

for (auto r = 0; r < righe_v; ++r) {
    for (auto c = 0; c < colonne_v; ++c) {
        cout << vuota[r][c] << " ";
    }
    cout << endl;
}
```

assign

```
auto righe_c = 3;
auto colonne_c = 4;

vector<vector<char>> matrice_car;
matrice_car.assign(righe_c, vector<char>(colonne_c, '-'));

for (unsigned r = 0; r < matrice_car.size(); ++r) {
    for (unsigned c = 0; c < matrice_car[0].size(); ++c) {
        cout << matrice_car[r][c] << " ";
    }
    cout << endl;
}
```



STL

algoritmi

algoritmi

- nella STL sono presenti numerosi ***algoritmi generici*** per eseguire le operazioni più comuni
- gli algoritmi sono **indipendenti dai contenitori**
- per l'utilizzo degli algoritmi della STL è necessario includere l'header ***<algorithm>***
- ***alcuni*** esempi di algoritmi:
  - operazioni sequenziali senza modifiche: ***find***, for_each
  - operazioni di modifica degli elementi: ***fill***, replace, copy
  - algoritmi di ordinamento: ***sort***

find

- individua il ***primo*** elemento che corrisponde al valore cercato

```
vector<int> v;
fillVector(v,10);
vector<int>::iterator it;
int rval;
rval = rand()%100;
it = find(v.begin(),v.end(),rval);
if (it != v.end())
   cout << rval << " found at pos "
        << it-v.begin() << endl;
else
   cout << rval << " not found"
        << endl;
```

replace

- ***sostituisce*** con un ***nuovo*** valore il contenuto degli elementi il cui valore coincide con quello specificato

```
int oldval,newval;
cout << "old value: ";
cin >> oldval;
cout << "new value: ";
cin >> newval;
replace(v.begin(), v.end(),
          oldval, newval);
```

copy

- ***copia*** gli elementi di una sequenza in un'altra
- nell'esempio v2 ha 20 elementi tutti con valore 99
- la dimensione di v2 è maggiore di quella di v1
- il ***valore*** dei primi elementi di v2 viene ***sostituito*** con quello degli elementi di v1

```
vector<int> v2(20,99);
printVector(v2);
copy(v.begin(),v.end(),v2.begin());
printVector(v2);
```

```
99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
copy(v.begin(),v.end(),v2.begin());
1 | 6 | 10 | 16 | 30 | 44 | 75 | 81 | 88 | 98 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 | 99 |
```

sort

- la funzione template ***sort*** realizza l'***ordinamento*** dei dati di un contenitore *in place* (nello stesso container di partenza)
- i parametri formali sono due ***iteratori*** che si riferiscono al ***primo*** e al ***successivo all'ultimo*** elemento della parte del contenitore da ordinare
  - per ordinare tutto il container si utilizzano ***begin*** e ***end***

sort – esempio vector

```
vector<int> v;
fillVector(v,10); printVector(v);
sort(v.begin(),v.end()); printVector(v);

void printVector(vector<int> v) {
   for (auto elem : v)
      cout << elem << " | ";
   cout << endl;
}

void fillVector(vector<int> &v, int n) {
   srand(std::time(nullptr));
   for (int i=0; i<n; i++)
     v.push_back(rand()%(n*10));
}
```

- gli iteratori begin e end si riferiscono al primo elemento del vector e al successivo all'ultimo

```
vector
71 | 1 | 97 | 98 | 98 | 88 | 46 | 74 | 96 | 70 |
sort(v.begin(),v.end());
1 | 46 | 70 | 71 | 74 | 88 | 96 | 97 | 98 | 98 |
```

documentazione STL

- online
  - http://en.cppreference.com/w/
  - http://it.cppreference.com/w/
- offline
  - http://en.cppreference.com/w/Cppreference:Archives

C++

g2d

grafica 2d per codice c++

- https://github.com/albertoferrari/info_lab/tree/master/codice_lezioni_cpp/g2d
- le funzionalità e l'interfaccia verso i programmi sono le stesse presenti nella versione per Python
- in particolare vengono definite in **`basic.hpp`** strutture analoghe alle tuple Python:
  - `struct Point { int x, y; };`
  - `struct Rect { int x, y, w, h; };`
  - `struct Color { int r, g, b; };`
- per compilare codice cpp che utilizza g2d è necessario specificare parametri per i socket di interfacciamento con protocolli TCP/IP
  - Linux:
    - `g++ -pthread ...`
  - Windows:
    - `g++ -lws2_32 -lwsock32 ...`

esempio demo_g2d.cpp

```
#include "g2d/canvas.hpp"                //path relativo per g2d
int main() {
    int r,g,b;
    g2d::Point dim = {800,600};          //struct Point coppia interi
    g2d::init_canvas(dim);               //inizializzazione canvas
    r = g2d::randint(0, 255);            //random estremi inclusi
    g = g2d::randint(0, 255);
    b = g2d::randint(0, 255);
    g2d::Color c = {r,g,b};              //struct Color tripla interi
    g2d::set_color(c);                   //impostazione colore
    g2d::Point p = {200,300};
    g2d::fill_circle(p,50);              //cerchio (centro, raggio)
    g2d::Rect rett = {400,350,70,120};  //strict Rect quadrupla interi
    g2d::set_color({0,0,255});           //blu
    g2d::fill_rect(rett);                    //rettangolo x,y,w,h
    g2d::draw_line({0,0},{600,600});
    g2d::draw_text("hello",{400,300},30);   //testo,posizione,dimensione
    g2d::main_loop();                    //loop
}
```

```
g++ demo_g2d.cpp -o demo_g2d.exe -lws2_32 -lwsock32 -std=c++14
```

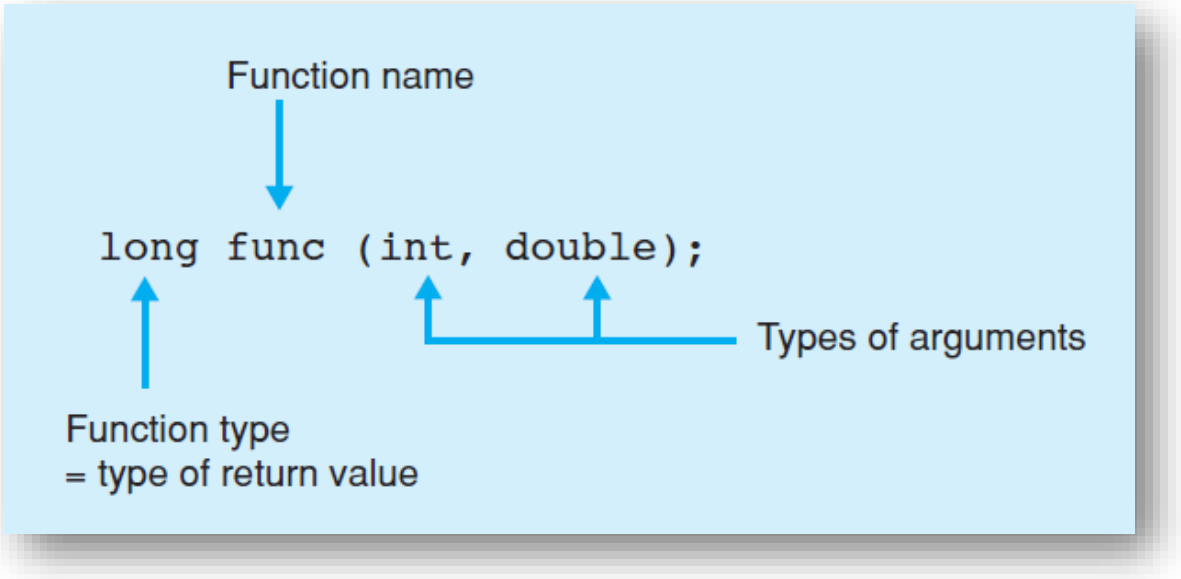
animazione.cpp

```
#include "g2d/canvas.hpp"

auto x = 50, y = 50, dx = 5, dy = 0;
auto ARENA_W = 480, ARENA_H = 360;
auto image = g2d::load_image("ball.png");

void tick() {
    if (g2d::key_pressed("LeftButton")) {
        dx = -dx;
    }
    g2d::clear_canvas();
    g2d::draw_image(image, {x, y});
    x = (x + dx) % ARENA_W;
}

int main() {
    g2d::init_canvas({ARENA_W, ARENA_H});
    g2d::main_loop(tick);
}
```

C++

funzioni

- caratterizzate da *nome*, *parametri* (numero, ordine e tipo) e *tipo* di ritorno
- le funzioni hanno un prototipo
- il prototipo non è necessario se la definizione della funzione appare prima del suo utilizzo
- nel prototipo i parametri possono non avere nome, ma per chiarezza in genere lo si mette



alcune funzioni matematiche

- la direttiva **`#include`** permette di importare i prototipi di funzioni delle librerie standard
- ogni libreria standard ha un file ***header*** contenente la definizione di funzioni, di tipo di dati e di costanti
- **`#include <cmath>`**
  - per utilizzare funzioni matematiche

```
double sin (double);           // Sine
double cos (double);           // Cosine
double tan (double);           // Tangent
double atan (double);          // Arc tangent
double cosh (double);          // Hyperbolic Cosine
double sqrt (double);          // Square Root
double pow (double, double);   // Power
double exp (double);           // Exponential Function
double log (double);           // Natural Logarithm
double log10 (double);         // Base-ten Logarithm
```

funzioni: prototipo e definizione

- ***prototipo*** (firma) di una funzione
  - **`<tipo-funzione> <nome-funzione>(<tipo-1>,<tipo-2>,…);`**
  - **`<tipo-funzione>`** è il tipo del valore restituito dalla funzione
    - ***void*** se la funzione non restituisce valori
  - **`<nome-funzione>`** è il nome (***identificatore***) della funzione
  - **`<tipo-1> <tipo-2>`** sono i tipi dei ***parametri***
    - possono mancare in funzioni senza parametri
    - oltre al tipo è possibile specificare il nome del parametro
- ***definizione*** di una funzione
  - analogo al prototipo
  - è obbligatorio specificare i ***nomi*** dei parametri (***formali***)
  - è obbligatorio specificare il ***corpo*** della funzione
  - è obbligatorio specificare il valore di ritorno mediante l'istruzione ***return*** *(non necessario per funzioni void)*

esempio: calcolo potenza

```
#include <iostream>
#include <cmath>

using namespace std;
int main() {
    double y, x = 5.22;
    // la funzione pow ha prototipo double pow( double, double)
    // y = pow("x", 3.0); // error: no matching function
                  // for call to 'pow(const char [2], double)'
    // y = pow(x + 3.0); // error: no matching function for call to 'pow(double)'
    y = pow(x, 3.0);     // ok!
    y = pow(x, 3);  // ok! Il compilatore converte l'intero 3 in double
    cout << x << " elevato al cubo vale: " << y << endl;
    return 0;
}
```

funzioni - esempio

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

double ipotenusa(double a, double b)
{
    auto c = sqrt(a * a + b * b);
    return c;
}

void stampa_ip(double ip)
{
    cout << "ipotenusa: " << ip << endl;
}

int main()
{
    auto l1 = 3.0, l2 = 4.0;
    auto l3 = ipotenusa(l1, l2);
    stampa_ip(l3);
}
```

esempio: prototipo

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

double media(int v1, int v2);    // prototipo

int main() {
    int val1, val2;
    cout << "Inserire due valori interi separati da spazio ";
    cin >> val1 >> val2;
    cout << "la media aritmetica fra " << val1 << " e " << val2
            << " = " << media(val1,val2) << endl;
    return 0;
}

double media(int v1, int v2) {  // definizione
    double med;
    med = (v1 + v2) / 2.0;
    return med;
}
```

passaggio dei parametri

- ***call-by-value***
  - il parametro attuale può essere una ***variabile o un'espressione***
  - il parametro formale viene inizializzato al ***valore*** del corrispondente parametro attuale
  - la funzione riceve una ***copia del valore*** del parametro
  - le azioni sui parametri formali ***non si ripercuotono*** sui parametri attuali
- ***call-by-reference*** *("**&**" precede il tipo del parametro formale)*
  - il parametro attuale deve essere una ***variabile***
  - il parametro formale viene associato al parametro attuale (si riferisco alla stessa zona di memoria)
  - le azioni sui parametri ***formali si ripercuotono*** sui parametri attuali
  - ***vantaggio***: migliori ***prestazioni***
  - ***svantaggio***: minore modularità, la funzione chiamata può corrompere i dati della chiamante (***side effects***)

call by value

```
scambio i valori x = 7 y = 5
passaggio per valore : x = 7 y = 5
```

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

void scambiaVal(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b= temp;
}

int main() {
    int x,y; x = 7; y = 5;
    cout << "scambio i valori x = " << x << " y = " << y << endl;
    cout << "passaggio per valore : ";
    scambiaVal(x,y);
    cout << "x = " << x << " y = " << y << endl;
    return 0;
}
```

call by reference

```
scambio i valori x = 7 y = 5
passaggio per riferimento : x = 5 y = 7
```

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

void scambiaRef(int &a, int &b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}
int main() {
    int x = 7;
    int y = 5;
    cout << "scambio i valori x = " << x << " y = " << y << endl;
    cout << "passaggio per riferimento : ";
    scambiaRef(x,y);
    cout << "x = " << x << " y = " << y << endl;
}
```

call by?

- suggerimenti:
  - parametri che devono essere ***modificati*** dalla funzione e ritornare modificati al chiamante: ***call by reference***
  - parametri di ***piccole*** dimensioni (memoria) che ***non*** devono tornare ***modificati*** al chiamante: ***call by value***
  - parametri di grandi dimensioni che ***non*** devono tornare ***modificati*** al chiamante: ***call by reference*** con specifica ***const***

overloading

- ***overloading*** (*sovraccarico*)
- all'interno di uno stesso programma è possibile definire più funzioni aventi lo ***stesso nome*** purché sia ***differente*** la ***lista*** dei tipi dei ***parametri***
- es
  - `double max(double, double);`
  - `int max(int, int);`

parametri con valori di default

- per i parametri ***call-by-value*** si può specificare un ***valore di default***
- se il corrispondente argomento ***manca***, il parametro assume il valore di default
- il valore di default va inserito nella prima tra dichiarazione e definizione
- i parametri con valore di default devono stare nelle posizioni ***più a destra***
- nella ***chiamata*** gli argomenti vanno ***omessi*** a partire da ***destra***

esempio: valori default

```
Volume of a box with Length = 4, Width = 6and Height = 2 is 48
Volume of a box with Length = 4, Width = 6and Height = 1 is 24
Volume of a box with Length = 4, Width = 1and Height = 1 is 4
```

```
#include <iostream>
using namespace std;
/*
 * Returns the volume of a box.
 * If no height is given, the height is assumed to be 1.
 * If neither height nor width are given, both are assumed to be 1.
 */
void showVolume(int length, int width = 1, int height = 1);

int main( ) {
    showVolume(4, 6, 2);  showVolume(4, 6);  showVolume(4);
    return 0;
}

void showVolume(int length, int width, int height) {
    cout << "Volume of a box with "
         << "Length = " << length << ", Width = " << width
         << "and Height = " << height
         << " is " << length*width*height << endl;
}
```

funzioni e vector (1)

```
#include <iostream>
#include <vector>

using namespace std;

/**
 * Restituisce lista (vector) di valori in input
 * */
vector<int> leggi()
{
    vector<int> v;
    int val;
    cout << "valore: (0 termina)"; cin >> val;
    while (val != 0)
    {
        v.push_back(val);
        cout << "valore: (0 termina)"; cin >> val;
    }
    return v;
}
```

funzioni e vector (2)

```
void raddoppia(vector<int> &v) {
    for (unsigned i=0; i<v.size(); ++i)
        v[i] = 2*v[i];
}

void stampa(vector<int> v) {
    for(auto e : v)
        cout << e << " ";
    cout << endl;
}

int main( ) {
    vector<int> valori;
    valori = leggi();
    stampa(valori);
    raddoppia(valori);
    stampa(valori);
    return 0;
}
```