Corso "Programmazione a oggetti in C++" (Giorgio De Nunzio, 2020)

Scaricare il materiale da

http://www.fisica.unisalento.it/~denunzio/allow_listing/CORSO_C++/

- books.zip (password: gdnCORSOC++2018)
- chapter01bgdn.ppt
- chapter02cgdn.ppt
- NOTE2020.docx (questo file)

[LEZIONE 1]

chapter01bgdn.ppt (fino alla slide numerata 14) (esclusa l'istruzione if)

Piattaforma scelta: GNU C++ (g++) in ambiente ubuntu.

https://itsfoss.com/c-plus-plus-ubuntu/

Installazione di GNU C++, e prima compilazione

Per verificare se GNU C++ è installato, usare il comando:

```
g++  --version
```

o

```
g++ -v
```

In caso di messaggio d'errore del tipo "command not found", occorre installare gli strumenti di compilazione, in particolare mediante il pacchetto `build-essential` (in realtà la maggior parte delle volte questo step è superfluo: può essere ad esempio necessario in caso di upgrade da una versione a un'altra del sistema operativo):

```
sudo apt install build-essential
```

Per editare i sorgenti, usare un editor come `gedit`, poi lavorare manualmente da riga di comando.

Se si vuole invece installare una IDE (Integrated Development Environment, https://www.binarytides.com/easy-to-user-c-ide-for-ubuntu-linux/), per esempio geany:

```
sudo apt install geany
```

Alternative:

- Nedit: `sudo apt install nedit` (editor, leggero)
- Kate: `sudo apt install kate` (leggero, editor con possibilita' di lavorare per progetti)
- Code::Blocks: `sudo apt-get install codeblocks` (leggero)
- Eclipse: `sudo apt-get install eclipse`, poi seguire la procedura di installazione di CDT, dal link dato all'inizio (pesante, complesso, versatile)

Ad esempio, per usare gedit:

```
gedit exer01.cpp
```

Chi usi il sottosistema linux di Windows, troverà i file linux in una directory Windows situata grosso modo a:

C:\Users\Giorgio De Nunzio\AppData\Local\Packages\CanonicalGroupLimited.UbuntuonWindows_79rhkp1fndgsc\LocalState\rootfs\home\giorgio\

Si consiglia di creare un link alla directory, ponendolo sul desktop. E' fondamentale non creare da Windows i file da usare in Linux, bensì crearli in linux (ad esempio con il comando `touch`) e (per comodità) una volta creati editarli da Windows, con notepad o notepad++ o altro editor.

Inseriamo il codice seguente (classico Hello World…)

```
#include <iostream>
using namespace std;
int main()
{
      cout << "Hello!\n";
      return 0;
}
```

Per compilare e generare un eseguibile:

```
g++  exer01.cpp
```

o meglio (per evitare il nome di default a.out per l'eseguibile):

```
g++  -Wall  exer01.cpp -o exer01
```

ed equivalentemente

```
g++ -Wall -g exer01.cpp -o exer01
```

(opzione `-Wall`: attiva i warning)

In caso di messaggio di errore perché g++ non è nel PATH, rimpiazzare il comando g++ con: /usr/bin/g++

Anche gcc può compilare il C++:

```
gcc    exer01.cpp    -lstdc++
```

Esecuzione del programma compilato:

```
./exer01
```

Un secondo esempio (exer02.cpp):

```
#include <iostream>
using namespace std;
int main()
{
      int x1, x2;
      cout << "Hello!\n";
      cout << "Insert a number: " << endl;
      cin >> x1;
      cout << "Insert another number: " << endl;
      cin >> x2;
      cout << "The sum is: " << x1 + x2 << endl;
      return 0;
}
```

Esercizi pag 51 e 52 del Rota.

The C++ compilation process

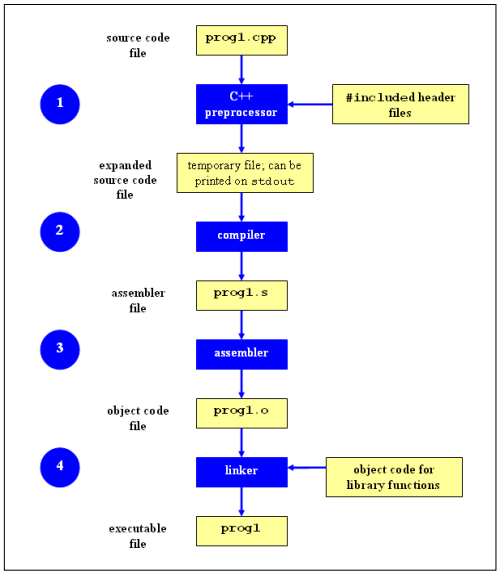
http://faculty.cs.niu.edu/~mcmahon/CS241/Notes/compile.html

Compiling a source code file in C++ is a four-step process. For example, if you have a C++ source code file named `prog1.cpp` and you execute the compile command

```
g++ -Wall -std=c++11 -o prog1 prog1.cpp
```

the compilation process looks like this:

a. The C++ preprocessor copies the contents of the included header files into the source code file, generates macro code, and replaces symbolic constants defined using `#define` with their values.
b. The expanded source code file produced by the C++ preprocessor is compiled into the assembly language for the platform.
c. The assembler code generated by the compiler is assembled into the object code for the platform.
d. The object code file generated by the assembler is linked together with the object code files for any library functions used to produce an executable file.



By using appropriate compiler options, we can stop this

process at any stage.

1. To stop the process after the preprocessor step, you can use the `-E` option:

```
2.    g++ -Wall -std=c++11 -E prog1.cpp
```

   The expanded source code file will be printed on standard output (the screen by default); you can redirect the output to a file if you wish, or add the –o option. Note that the expanded source code file is often incredibly large - a 20 line source code file can easily produce an expanded file of 20,000 lines or more, depending on which header files were included.

3. To stop the process after the compile step, you can use the `-S` option:

```
4.    g++ -Wall -std=c++11 -S prog1.cpp
```

   By default, the assembler code for a source file named *filename.cpp* will be placed in a file named *filename.s*.

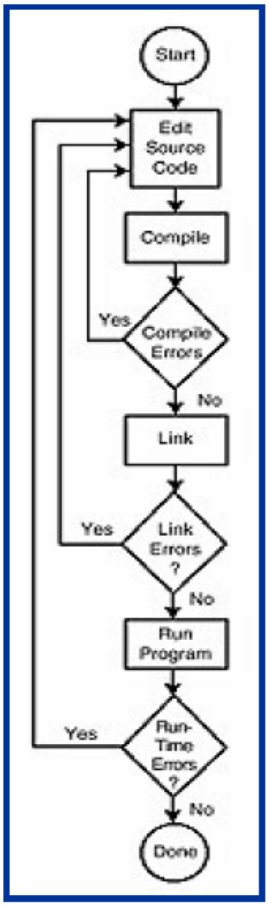
5. To stop the process after the assembly step, you can use the `-c` option:

```
6.    g++ -Wall -std=c++11 -c prog1.cpp
```

   By default, the assembler code for a source file named *filename.cpp* will be placed in a file named *filename.o*.

Ciclo di produzione di un software (The Development Cycle)

→ ***Sams,.Teach.Yourself.C++.For.Linux***. pag 8



Programmi composti da più sorgenti (in file diversi)

Vedere innanzitutto le opzioni di compilazione intermedia; oltre che la pagina Web vista in precedenza, un'altra fonte di informazioni è: → *Tom_Swans_GNU_C++_for_Linux* (pag 99)

Consideriamo i seguenti tre sorgenti:

exer03.cpp

```
#include <iostream>
#include "c3.h"  // <-- remark the difference from the preceding line
using namespace std;
int main()
{
      double x1, x2;    // <-- double instead of int
      cout << "Hello!\n";
      cout << "Insert a number: " << endl;
      cin >> x1;
      cout << "Insert another number: " << endl;
      cin >> x2;
      cout << "The sum is: " << x1 + x2 << endl;
      cout << "Now sum the squares\n";
      double x1q = calculateSquare(x1);
      double x2q = calculateSquare(x2);
      cout << "The sum of the squares is: " << x1q + x2q << endl;
      return 0;
}
```

c3.h

```
double calculateSquare(double x);
```

c3.cpp

```
#include <iostream>
#include "c3.h"
using namespace std;
double calculateSquare(double x)
{
      cout << "   hi! I calculate squares!" << endl;
      double xs = x * x;
      cout << xs << endl;
      return xs;
}
```

In sequenza, le operazioni da compiere per costruire l'eseguibile sono:

```
g++ -Wall  exer03.cpp -c -o exer03.o

g++ -Wall  c3.cpp -c -o c3.o

g++ -Wall  exer03.o    c3.o    -o exer03
```

oppure più semplicemente:

```
g++ -Wall  exer03.cpp    c3.cpp    -o exer03
```

Questo comando compila i due file .cpp, effettua il link, e salva l'eseguibile con il nome `eser03`.

Esecuzione del programma compilato:

```
./eser03
```

Che succede se si omette l'include di c3.h in exer03.cpp? Che succede se si omette la compilazione del file c3.cpp (eliminando cioè c3.cpp dalla riga di comando riassuntiva)?

Nota: formattare l'output:

```
#include <iomanip>
cout << setw(8);  // larghezza del campo di output
cout << setprecision(3); // num. cifre decimali compreso il ".” (se non “fixed”)
```

I dettagli sono su: http://www.cplusplus.com/reference/iomanip/setprecision/

Esempio completo:

```
// setprecision example
#include <iostream>     // std::cout, std::fixed
#include <iomanip>      // std::setprecision
```

```
int main () {
  double f =3.14159;
  std::cout << std::setprecision(5) << f << '\n';
  std::cout << std::setprecision(9) << f << '\n';
  std::cout << std::fixed;  // ora il numero indica solo le cifre decimali
  std::cout << std::setprecision(5) << f << '\n';
  std::cout << std::setprecision(9) << f << '\n';
  return 0;
}
```

In certi casi conviene tornare al buon vecchio printf….

```
#include <stdio.h>
int main()
{
  printf("%7.2f\n", 123.456);
}
```

SCRIPT per la compilazione dei file

Inserire le istruzioni di compilazione in un file chiamato (per esempio) `buildit`, da richiamare con `source buildit`

```
# call me with: source buildit
g++ -o exer03.o exer03.cpp -c
g++ -o c3.o c3.cpp -c
g++ -o exer03 exer03.o c3.o
```

Inserire ora le medesime istruzioni di compilazione in un file chiamato (per esempio) `buildit2`, da rendere eseguibile con `chmod` (vedi dopo), e da richiamare con `./buildit`

```
#!/bin/sh
# call me with: ./buildit
g++ -o exer03.o exer03.cpp -c
g++ -o c3.o c3.cpp -c
g++ -o exer03 exer03.o c3.o
```

chmod

```
chmod a+x scriptfilename
```

https://www.mrwebmaster.it/linux/gestire-permessi-chmod-chown-chgrp_10211.html

MAKE:

→ Pag 54 (capitolo 4) di "Linux Programming"

→ Pag 59 (capitolo 6) di "GNU Linux Application Programming"

A Simple Makefile Tutorial (da modificare per tener conto della sintassi C++)

http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/

Creare il seguente file, chiamato `Makefile`:

```
exer03: exer03.o c3.o
	g++ -o exer03 exer03.o c3.o

exer03.o: exer03.cpp c3.h
	g++ -o exer03.o exer03.cpp -c

c3.o: c3.cpp c3.h
	g++ -o c3.o c3.cpp -c

clean:
	rm *.o exer03
```

Nel makefile si riconoscono: target, dipendenze (dependencies) e regole (rules). La linea delle regole dev'essere indentata rigorosamente da un tab.

Compilare/linkare con make, e ripulire con make clean.

Alternativa con l'uso di variabili:

```
OBJS = exer03.o c3.o
HDRS = c3.h
exer03: $(OBJS)
	g++ -o exer03 $(OBJS)
exer03.o: exer03.cpp c3.h
	g++ -o exer03.o exer03.cpp -c
c3.o: c3.cpp c3.h
	g++ -o c3.o c3.cpp -c
clean:
	rm $(OBJS) exer03
```

Tipi di variabili del C/C++

→ ***Tom_Swans_GNU_C++_for_Linux pag 128***, programma sull'occupazione in memoria dei tipi di dati (modificato)

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

int main()
{
	char c;
	short s;
	int i;
	long l;
	float f;
	double d;
	long double ld;
```

```
    cout << "sizeof char == " << sizeof(c) << endl;
    cout << "sizeof short == " << sizeof(s) << endl;
    cout << "sizeof int == " << sizeof(i) << endl;
    cout << "sizeof long == " << sizeof(l) << endl;
    cout << "sizeof float == " << sizeof(f) << endl;
    cout << "sizeof double == " << sizeof(d) << endl;
    cout << "sizeof long double == " << sizeof(ld) << endl;
    return 0;
}
```

http://www.math.unipd.it/~sperduti/CORSO-C%2B%2B/Variabili_%20Tipi%20di%20dato_%20Costanti.htm

Nome	Byte*	Descrizione	Rango*
char	1	carattere o intero di 8 bit.	**signed:** -128 ... 127 **unsigned:** 0 ... 255
short	2	intero di 16 bit.	**signed:** -32768 ... 32767 **unsigned:** 0 ... 65535
long	4	intero di 32 bit.	**signed:**-2147483648 ... 2147483647 **unsigned:** 0 ... 4294967295
int	*	Intero. La sua lunghezza dipende dalla lunghezza del tipo **word** usato dal sistema operativo. Ad esempio, in MSDOS è di 16 bit mentre in sistemi a 32 bit (quali Windows 9x/2000/NT) è di 32 bit (4 bytes).	See **short**, **long**
float	4	numero in virgola mobile.	3.4e + / - 38 (7 cifre decimali)
double	8	numero in virgola mobile in doppia precisione.	1.7e + / - 308 (15 cifre decimali)
long double	10	numero in virgola mobile in doppia precisione estesa.	1.2e + / - 4932 (19 cifre decimali)
bool	1	Valori Booleani. Può assumere uno dei due valori: `true` o `false`. NOTA: è un tipo aggiunto recentemente allo standard ANSI-C++. Non tutti i compilatori lo accettano.	**`true`** or **`false`**
wchar_t	2	Carattere esteso. Viene usato per rappresentare tutti i caratteri internazionali. NOTA: è un tipo introdotto recentemente nello standard ANSI-C++. Non tutti i compilatori lo accettano.	caratteri estesi

* I valori nelle colonne Byte e Range possono variare a seconda del sistema. I valori quì riportati sono quelli più comunemente accettati ed usati da quasi tutti i compilatori.

Oltre a questi tipi di dato fondamentali vi sono anche i puntatori e il tipo **void**.

Compilatori recenti consentono dati specializzati.

[LEZIONE 2]

ARRAY / STD::VECTOR

Example with C-like allocation

```
#include <iostream>
using namespace std;
int main()
{
	int x[5];
	int j;
	for (j = 0; j < 5; j++)
		cin >> x[j];
	cout << "printing out!" << endl;
	for (j = 0; j < 5; j++)
		cout << x[j] << endl;
	return 0;
}
```

Now discuss the first part of the longer “Example” program reported later, up to (and without including) the STL part. Then come back here for a simple example with STL.

Example with STL allocation

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

using namespace std;
int main()
{
	// remark the () in the vector definition (no [] like in C arrays!)
	vector<int> x(5);   // show what happens with vector<int> x; Segmentation fault!

	int j;
	for (j = 0; j < 5; j++)
		cin >> x[j];   // x.at(j)    gives out of range

	cout << "printing out!" << endl;
	cout << "the x vector is " << x.size() << " positions long" << endl;
	for (j = 0; j < 5; j++)
		cout << x[j] << endl;
	return 0;
}
```

Now discuss the second part of the "Example" program: what is STL, with some commentaries).

C++ Standard Libraries and Standard Template Library (STL)

http://www.ntu.edu.sg/home/ehchua/programming/cpp/cp9_STL.html
http://www.cplusplus.com/reference

STL is not object-oriented, though it is object-based!

Difference between object-oriented and object-based:

https://www.quora.com/What-is-the-difference-between-object-based-object-oriented-and-pure-object-oriented-programming-concepts
https://www.slideshare.net/Kamrul23631/object-oriented-vs-object-based-programming

"Object Oriented: Does support all object oriented features like inheritance, polymorphism, encapsulation, data hiding and abstraction.
Object Based: Does not support inheritance and polymorphism."

Example

```
#include <iostream>
#include <vector>
#include <algorithm>     // find, reverse, count, max_element
#include <numeric>       // accumulate,

using namespace std;
void printIntVector1(vector<int> v);

// ARRAYS AND VECTORS
int main()
{
        // STACK (automatic variables, static allocation)
        // Fixed length

        int v1[10];

        // STACK (automatic variables, static allocation)
        // Variable Length Array, C99, inherited by some C++ compilers

        int n;

        n = 10;

        // or
        // cin >> n;

        int v2[n];

        // HEAP (dynamic allocation)

        int k = 10;
        int *v3 = new int[k];
        int *v3a = new int [4] {1, 2, 4, 5};  // C++0x, the vector size is compulsory
           // use the arrays vith v3[3], *v3, *(v3+2)...
        delete[] v3a;
        delete[] v3;

        // Various notes

        cout << sizeof(v2)/sizeof(int) << endl;  // to calculate the length of an array (statically allocated)

        int vv1[] = {1, 2, 3, 6, 5, 4, 1};  // Initialization
        int vv2[10] = {1, 2, 3, 6, 5, 4, 1};  // Partial initialization


        for (int j(0); j < n; j++)   // <-- direct initialization of j, C++
                v1[j] = j;
        for (int j = 0; j < n; j++)  // <-- copy initialization
                cout << *(v1+j) << endl;

        cout << endl;
```

```
// Use vectors from Standard Template Library!

// Standard Template Library (STL).  Iterators, Containers and Algorithms.

// STL It is a library of template-based container classes,
// including vectors, lists, queues, and stacks.
// The STL also includes a number of common algorithms, including sorting and searching.
// The goal of the STL is to give you an alternative to reinventing the wheel for these
// common requirements.
// Containers
// A container is an object that holds other objects.
// The standard C++ library provides a series of container classes that are powerful
// tools that help C++ developers handle common programming tasks. Two types of
// Standard Template Library (STL) container classes are sequence and associative.
// Sequence containers are designed to provide sequential and random access to their elements
// The Standard Template Library sequence containers provide efficient sequential
// access to a list of objects. The standard C++ library provides
// three sequence containers: vector, list, and deque (”dek”, double ended queue, nbgdn).
// (Sams, Teach Yourself C++ For Linux in 21 days)

// STL is not object-oriented. To achieve its versatility, the STL relies
// more heavily on templates than on encapsulation, inheritance, and virtual functions
// (polymorphism) - OOP's three main anchors - and you won't find any
// significant class hierarchies in the STL.
// (Tom Swans GNU_C++ for Linux)

vector<int> array_int;                  //array of type int, empty
vector<double> array_double;        //array of type double, empty (size is 0)
vector<double> array_double_2(10);  //array of 10 elements of type double, initialized with 0.0
vector<double> array_double_2a(10, 2.5);  // array of 10 elements of type double, initialized with 2.5

vector<int> v {1, 2, 3, 4, 5, 6};
vector<int> vv = {1, 2, 3, 4, 5, 6};  // used later in this code!

cout << "v was defined as { 1, 2, 3 , 4, 5, 6}" << endl;
cout << "v.size() is " << v.size() << endl;          // "dot" operator, Object Oriented syntax
cout << "v.capacity() is " << v.capacity() << endl;  // allocated storage in terms of elements
        // https://www.tutorialspoint.com/cpp_standard_library/cpp_vector_capacity.htm

v.push_back(100);
cout << "after v.push_back(100)" << endl;

cout << "v.size() is " << v.size() << endl;
cout << "v.capacity() is " << v.capacity() << endl;

v.push_back(100);
cout << "after v.push_back(100)" << endl;

cout << "v.size() is " << v.size() << endl;
cout << "v.capacity() is " << v.capacity() << endl;
cout << endl;

// empty()
cout << "is v empty?" << endl;
cout << v.empty() << endl;

// pop_back
v.pop_back();  // removes the last element

// Use indexing to access vector elements

cout << "Use indexing to access vector elements" << endl;
for (int j = 0; j < v.size(); j++)
        cout << v[j] << " ";
cout << endl << endl;

// Copying a vector to another
cout << "Copying a vector to another" << endl;

vector<int> vvv = vv;   //  !!!!! Great! Try and do this with C arrays....
cout << vvv[5] << endl;
cout << endl;

// Accessing past v limits
// The standard-library vector does not guarantee range checking while indexing!!!!!
cout << "Access past v limits, using indexing" << endl;
v[125] = 125;

// Use v.at()

// The at() operation is a vector subscript operation that throws an exception of type
// out_of_range if its argument is out of the vector's range

cout << "Access past v limits, using at()" << endl;
v.at(5) = 125;  // OK
// v.at(225) = 225;  // AHW!
```

```
// ITERATORS

// In a loop such as
// for (int j = 0; j < v.size(); j++) cout << v[j] << endl;
// j is used to access v's elements sequentially, but the compiler cannot know
//                  (so at each iteration &v[j] must be calculated).
//
// Iterators make this situation explicit, so optimizing it.
// They are similar to pointers.

// An iterator is a value that
// - Identifies a container and an element in the container
// - Lets us examine the value stored in that element
// - Provides operations for moving between elements in the container
// - Restricts the available operations in ways that correspond to what the container
// can handle efficiently
// (Accelerated C++ Practical Programming by Example (C++ in Depth))

// A few iterators referring
// to useful elements of a container can be obtained;
// begin() and end() are the best examples of this.
// In addition, many algorithms return iterators.

printIntVector1(v);

// Every standard container, such as vector, defines two associated iterator types:
//   container-type::const_iterator   ("::" is the Scope Resolution Operator)
//   container-type::iterator
// When we want to use an iterator to change the values stored
// in the container, we use the iterator type.
// If we need only read access, then we use the const_iterator type.

// v.begin() and v.end()  return iterators, which are automatically converted to const_iterator
// They return a value that denotes the beginning or (one past) the end of a container.

// Dereference operator, *


// Erasing a vector element
cout << "ERASING" << endl;
v.erase(v.begin() + 2);
cout << "After erasing element #2, v.size() is " << v.size() << endl;

printIntVector1(v);

// IMPORTANT! erase returns an iterator that is positioned on the element that follows the one
// that we just erased. The iterator used for deletion is invalidated: we know that iter can no
// longer refer to the same element because that element is gone. In fact, calling erase on a
// vector invalidates all iterators that refer to elements after the one that was just erased.

// Example: erase all the elements of a vector, greater than 100

// I MUST USE a while loop in this example, no for loop, because iter will not always be incremented....
cout << "Define a new vector" << endl;

vector<int> vx { 10, 200, 30, 400, 5, 600 };

printIntVector1(vx);

cout << "Now erase elements > 100" << endl;
vector<int>::const_iterator iter = vx.begin();
while (iter != vx.end())   // remark that vx.end() MUST be evaluated at each loop! Use no temp variable
{
        if (*iter > 100)  // Only erase elements > 100
                iter = vx.erase(iter);   // iter now points to the element AFTER the erased one
        else
                iter++;
}

printIntVector1(vx);

// Inserting vector elements http://www.cplusplus.com/reference/vector/vector/insert/
// The vector is extended by inserting new elements before the element at the specified position
cout << "INSERTING the v vector at the end of the vx vector" << endl;
vx.insert(vx.end(), v.begin(), v.end());          // position in vx were to copy the v vector, range
                                                  //       to be copied: start and end (end excluded)
printIntVector1(vx);

cout << "INSERTING a new value into vx" << endl;
vx.insert(vx.begin()+1, 999);  // position were to copy, value

printIntVector1(vx);

// Dereferencing as a means to address and change data
cout << "Dereferencing as a means to address and change data (pay attention to vector limits!!)\n";
vector<int>::iterator itx = vx.begin()+3;
*itx++ = 765;
*itx = 766;
```

```
        printIntVector1(vx);

        // GENERIC ALGORITHMS
        cout << "GENERIC ALGORITHMS" << endl;

        // Find an element by the find() algorithm (remember to #include <algorithm>)
        // Is there an element equal to 30?
        // find is an algorithm.
        // find(b, e, c) looks for a given value c in the sequence [b, e)
        // It looks for the specific value given as its third argument.
        // If the value that we want is present, the function returns an iterator denoting the first
        // occurrence of the value in the given sequence. If the value is not found, then find returns its
        // second argument e.

        cout << "FIND (number 5 in vx)\n";

        vector<int>::const_iterator it = find(vx.begin(), vx.end(), 5);  // find element 5
        if (it != vx.end())
                cout << "element found! It is at index " << it - vx.begin() << endl;

        // Now use find and insert, to insert a vector of three numbers {997 998 999}
        // after the first occurrence of value 125

        cout << "insert a vector of three numbers {997 998 999} after the first occurrence of value 125\n";
        vector<int> vi = {997, 998, 999};
        vector<int>::const_iterator it1 = find(vx.begin(), vx.end(), 125);  // find element 125
        if (it1 != vx.end()) {
                cout << "element found! It is at index " << it1 - vx.begin() << endl;
                cout << "Now insert the vi vector\n";
                vx.insert(it1+1, vi.begin(), vi.end());  // it1 + 1 points to the element after 125 in vx!
        }

        printIntVector1(vx);

        //
        // Other algorithms:

        // reverse the elements (in place!)
        cout << "REVERSE (in place)" << endl;
        reverse(vx.begin(), vx.end());
        printIntVector1(vx);

        // count the number of elements that have the value 5
        cout << "COUNT (how many numbers = 5?)" << endl;
        int num=count(vx.begin(), vx.end(), 5);
        cout << num << endl;

        // max vector value    (c++11/c++0x)
        cout << "MAX" << endl;
        vector<int>::const_iterator maxit = max_element(vx.begin(), vx.end());
        //auto maxit = max_element(vx.begin(), vx.end());
        cout << *maxit << endl;

        // accumulate vector values (summing them up)
        cout << "ACCUMULATE" << endl;
        int sum = accumulate(vx.begin(), vx.end(), 0); // 0 is the initial value
        cout << sum << endl;

        // Multidimensional vectors
        vector <vector <int> > v2d(3);

        return 0;
}


void printIntVector1(vector<int> v)
{
// remember that the end() iterator points PAST the end of the vector
// an iterator is defined as container-type::const_iterator or container-type::iterator
        // cout the vector
        //cout << sizeof(v) << endl;
        for (vector<int>::const_iterator iter = v.begin(); iter != v.end(); iter++)
                cout << *iter << " ";
        cout << endl;
}
```

Remarks: types of variables (from the 1st lesson).
Remarks: hexadecimal numbers.
https://www.tutorialspoint.com/computer_logical_organization/hexadecimal_arithmetic.htm
[sum of: coefficients times (2 raised to the various powers....)]
Remarks: **Pointers**. Teach yourself C++ in 24 h, hour 10 page 137; Teach yourself C++ in one hour a day, Lesson 8

(remember NULL vs nullptr)

Address operator &

```
#include <iostream>
using namespace std;
int main()
{
	unsigned short uShortVar = 5;
	int intVar;
	unsigned long uLongVar = 65535;
	long longVar = -65535;

	cout << &uShortVar << endl;
	cout << &intVar << endl;
	cout << &uLongVar << endl;
	cout << &longVar << endl;

	return 0;
}
```

Declaring a pointer and storing the address of a variable in a pointer, `int *ip = &n;`

Use of ++ and -- on a pointer (if memory is layed out in increasing addresses with the chronological order of variable definition)

```
	int n1 = 101, n2 = 102, n3 = 103;
	int *ip = &n1;
	cout << ip << " " << *ip << endl;
	ip++;
	cout << ip << " " << *ip << endl;
	ip++;
	cout << ip << " " << *ip << endl;
```

Use of pointers with arrays

```
	int nn[10];
	for (int j = 0; j < 10; j++)
		nn[j] = j;
	for (int j = 0; j < 10; j++)
		cout << nn[j] << endl;
	cout << endl;
	int *ip1 = &nn[0];
	int *ip2 = nn;
	for (int j = 0; j < 10; j++)
		cout << *ip2++ << endl;

cout << *(nn+3) << endl;
```

Remark: functions.

```
#include <iostream>

void changeAFunctionArgument1(int x);
void changeAFunctionArgument2(int *x);
void changeAFunctionArgument3(int &x);

using namespace std;

int main()
{
	int x = 133;
	cout << "MAIN: x is " << x << endl;
	changeAFunctionArgument1(x);
	cout << "MAIN: x is " << x << endl;
	changeAFunctionArgument2(&x);
	cout << "MAIN: x is " << x << endl;
	changeAFunctionArgument3(x);
	cout << "MAIN: x is " << x << endl;

	return 0;
}


void changeAFunctionArgument1(int x)  // passing by value!
{
	cout << "FUN: x is " << x << endl;
```

```
        x = 11111;
        cout << "FUN: and now x is " << x << endl;
}

void changeAFunctionArgument2(int *x)  // passing by address!
{
        cout << "FUN: x is " << *x << endl;
        *x = 22222;
        cout << "FUN: and now x is " << *x << endl;
}

void changeAFunctionArgument3(int &x)  // C++!! Passing by reference!
{
        cout << "FUN: x is " << x << endl;
        x = 33333;
        cout << "FUN: and now x is " << x << endl;
}
```

Remark: Calculation of max/min of a set of numbers.
First use of a pointer in C: arrays are just memory regions with a name!

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

double theMax1(double v[], int n);
double theMax2(double *v, int n);

int main()
{
        double w[] = {61, 55, 6, 3, 2, 5, 6};
        cout << "max is " << theMax1(w, 7) << endl;
        cout << "max is " << theMax2(w, 7) << endl;

        return 0;
}

double theMax1(double v[], int n)
{
        int maxVal = v[0];
        for (int j = 1; j < n; j++)
                if (v[j] > maxVal) maxVal = v[j];
        return maxVal;
}

double theMax2(double *v, int n)
{
        int maxVal = *v++;
        for (int j = 1; j < n; j++) {
                if (*v > maxVal) maxVal = *v;
                v++;
        }
        return maxVal;
}
```

EXERCISES:

2.1) Write a program that asks the user to enter some “double” numbers (the number of values entered is asked to the user with a first question: "how many numbers do you want to enter?"), Calculate the average, find the minimum and the maximum, write everything on screen.
Write three versions:

a. in the first version use the C arrays, statically allocated but with the number of elements contained in a variable entered by the user `n` (i.e.: `int n; cin >> n; double x[n];`) and write the functions calculating the average and max / min starting from the basic C/C++ functions (max/min is not trivial, the first time you do!)
b. the second version is like the first, but using C ++ vectors;
c. the third is like the second, but you use `accumulate` to make the sum, `max_element` and `min_element` for maximum and minimum;

If necessary (in case of compilation errors), use options `g++ -std=c++11`, `g++ -std=c++0x`, `g++ -std=gnu++11` or `g++ -std=gnu++0x`

2.2) Write a program that finds (and inserts into a vector<...>) the prime numbers lower than a maximum value '*maxv*' entered by the user; it should be structured as a main function that asks the user for the *maxv* value and then calls a findPrimes function that receives this value as input and returns a vector of integers; the main program, then, prints the vector of the prime numbers to the screen by using a printVector function. In a first version, put the three functions (`main`, `findPrimes`, and `printVector`) in a single file; in a second version, put the three functions in three different files, and write a Makefile to achieve compilation. Write findPrimes both using iterators and using indexing.
Remember that finding prime numbers may involve two approaches:

a. this one is simpler from the point of view of programming, but it is computationally less efficient: (after considering that 1, 2, and 3 are prime and are particular cases) for each number *n* from 4 to *maxv* (here is a first for loop!), consider the integer numbers *d* from 2 to *n* or (far better!) to `floor(sqrt(n))` (a second for loop!) and check if *n* may be divided by *d*. If you find a *d* for which it can, number *n* is not prime. If you want, you may use another level of function, say a `bool isNumberPrime(int d)` function, which contains the second for loop and returns true or false if the *d* number is prime or not. For the calculation of the integer part of the square root of a number you may use `#include <cmath>` and `floor(sqrt(x))`.
   Here is the `isNumberPrime` function:

```
bool isNumberPrime(int d)
{
            bool isPrime = true;
            for (int j = 2; j <= floor(sqrt(d)); j++) {
    //              cout << ".... " << j << "  ";
                    if (d%j == 0)
                            isPrime = false;
    //              cout << d%j << "  " << isPrime << endl;

            }
            return isPrime;
}
```

b. The Sieve of Eratosthenes (https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes)

Modify the code so that the `maxv` value is given through the command line parameters.

[LEZIONE 3]

C-style strings, pointers to pointers, arguments of the `main` function

```
#include <iostream>
#include <cstring>   // strlen, atoi, atof; sometimes atoi and atof are in cstdlib!
using namespace std;

// C-style strings
// A string is a char array terminated by a null character '\0' (or 0) (null-terminated char array).

int main(int argc, char **argv)   // char *argv[]
{
        // use of a string literal
        char name[] = "Marion";
        cout << sizeof(name) << endl;  // 7    size in bytes!!!!!! If other var types, divide by sizeof(_type_)
        cout << strlen(name) << endl;  // 6    #include <cstring>
        cout << endl;

        for (int j = 0; j < sizeof(name); j++)  // show a null-terminated C-style string
                cout << (int)name[j] << " " << name[j] << endl;

        cout << endl;
```

```
        // use a pointer to char to explore a C-style string
        char *p;
        p = name;   // the name of an array is a (const) pointer to the first element

        for (int j = 0; j < sizeof(name); j++)
                cout << *p++ << endl;

        // use a pointer to char to declare a C-style string
        const char *name2 = "Robin Hood";   // const!!!!!

        // what the hell is a pointer to (char) pointer???

        char **p1;  // a pointer to a variable which contains a pointer to char!

        // call this program this way: ./ex1 good morning

        cout << argc << endl;  // and argv is a char **

        // argv ->  p1  p2  p3
        //           |   |   |
        //           V   V   V
        //          'e' 'g' 'm'
        //          'x' 'o' 'o'
        //          '1' 'o' 'o'
        //           0  'd' 'r'
        //               0  'n'
        //                  'i'
        //                  'n'
        //                  'g'
        //                   0

        cout << argv[0] << endl;
        if (argc == 2)
                cout << argv[1] << endl;

        // following lines, calling the program this way: ./ex1 222
        if (1) {
                cout << endl;
                cout << argv[1] << endl;   // string
                cout << argv[1] + 5 << endl;   // no conversion to integer! Error!
                cout << atoi(argv[1]) + 5 << endl;   // conversion to integer (different from (int)x !!)
        }
        return 0;
}
```

You may also have e.g. `int **p`, which is a pointer to pointers to integers, and is used to define a multidimensional array (a vector of vectors of integers).

Dynamic allocation (in the heap memory).

C language:
https://www.geeksforgeeks.org/dynamic-memory-allocation-in-c-using-malloc-calloc-free-and-realloc/

example (code snippet)

```
// Ask for 100 integer variables. The ptr pointer will hold the
// base address of the block created.
// If space is insufficient, allocation fails and malloc
//   returns a NULL pointer

int *ptr = (int*) malloc(100 * sizeof(int));
// use them with the pointer
*ptr = 125; *(ptr+10) = 333; ptr[50] = 99;
int j;
for (j = 0; j < 100; j++)
       *ptr++ = 111; // 111 everywhere (but there are better ways)
// free the block of memory and give it back to the system
free(ptr);
```

C++ language:

```
int *ptr = new int[100];
       // exactly as before between malloc and free
```

```
delete[] ptr;
```

```
#include <iostream>

using namespace std;
int main()
{
	int j;
	int *ptr = new int[10];

	// set some values
	*ptr = 125; *(ptr+5) = 333; ptr[9] = 99;

	// print out
	for (j = 0; j < 10; j++)
		cout << ptr[j] << endl;
	cout << endl;

	// set all the values
	for (j = 0; j < 10; j++)
		ptr[j] = 111; // 111 everywhere (but there are better ways)

	it *ptr2 = ptr;
	//for (j = 0; j < 10; j++)
	//	*ptr2++ = 111; // ptr is updated!! munmap_chunk(): invalid pointer

	////////ptr -= 10;

	// print out
	///for (j = 0; j < 10; j++)
		cout << ptr[j] << endl;

	delete[] ptr;	// are we deleting from the right point? is it valid?

	return 0;
}
```

Note: besides std::vector, also std::array exists. For a discussion, see:

https://stackoverflow.com/questions/4424579/stdvector-versus-stdarray-in-c
https://stackoverflow.com/questions/6632971/what-is-the-difference-between-stdarray-and-stdvector-when-do-you-use-one-o

ARRAY MULTIDIMENSIONALI

https://www.programiz.com/cpp-programming/multidimensional-arrays

Static declaration of a 2D array:
int x[3][4];

table with 3 rows and each row has 4 columns as shown below.

	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Static declaration of a 3D array:

```
float x[2][4][3];
```

Declaration and initialisation of two two-dimensional arrays:

```
int test[2][3] = {2, 4, -5, 9, 0, 9};
int test[2][3] = { {2, 4, 5}, {9, 0, 0}};   // better
```

Initialisation of three-dimensional arrays:

```
int test[2][3][4] = {3, 4, 2, 3, 0, -3, 9, 11, 23, 12, 23,
                 2, 13, 4, 56, 3, 5, 9, 3, 5, 5, 1, 4, 9};


int test[2][3][4] = {         // better
                     { {3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2} },
                     { {13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9} }
                 };
```

Esample 1: display all elements of an initialised two-dimensional array

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

int main()
{
    int test[3][2] =
    {
        {2, -5},
        {4, 0},
        {9, 1}
    };

    // Accessing two dimensional array using
    // nested for loops
    for(int i = 0; i < 3; ++i)
    {
        for(int j = 0; j < 2; ++j)
        {
            cout<< "test[" << i << "][" << j << "] = " << test[i][j] << endl;
        }
    }

    return 0;
}
```

Dynamic allocation of multidimensional arrays

TO BE DONE

Dynamic allocation of multidimensional std::vector

TO BE DONE

STRINGS

Da `http://www.ntu.edu.sg/home/ehchua/programming/index.html`
`http://www.ntu.edu.sg/home/ehchua/programming/cpp/cp9_String.html`
Characters and Strings

C++ supports two types of strings:

1. The C-style string (or C-String) in header cstring (ported over from C's string.h), which represents a string as a char array terminated by a null character '\0' (or 0) (null-terminated char array).
2. The new C++ string class in header string. string is a regular class, with public interface defined in the constructors and public member functions.

C-String

```
int main() {
  // char * str1 = "hello";
          // warning/error: deprecated conversion from string constant to 'char*'
   const char * str2 = const_cast<char *>("hello");   // remove the "const"

   const char * str3 = "hello";  // ok const char * is constant
// *(str3 + 1) = 'a';  // error: assignment of read-only location '*(str3 + 1u)'

   char str4[] = "hello";    // ok array is a constant pointer
   str4[1] = 'a';

   const char str5[] = "hello";
//  str5[1] = 'a';   // error: assignment of read-only location 'str5[1]'


// all of these work!
   cout << str2 << endl;
   cout << str3 << endl;
   cout << str4 << endl;
   cout << str5 << endl;

}
```

C-string (null-terminated char array) can be declared as char* or char[]. This is because C treats an array name as a pointer to the first element of the array.
Unlike regular arrays, there is no need to pass the length of C-string into function, as the function can deduce the length from the terminating null character.

C-String Functions in `<cstring>` header

`strlen, strcpy, strcmp.....`

C-String Functions in `<cstdlib>` header

`atoi, atof, ...`

C-String Input/Output Functions in `<iostream>` header

`cin, cout, cin.getline...`

Useful for C-String in `<cctype>` header

`toupper, tolower`

The C++'s string class

```
string Class Constructors
string ();
   // (1) Default constructor: construct an empty string of length 0.
string (const string & str);
   // (2) Copy constructor: construct by copying str (by value)
string (const string & str, size_t pos, size_t len = npos);
   // (3) Substring constructor: copy the substring starting at pos, of the len.
   // size_t is usually typedef to unsigned int
   // npos is a static constant in string (i.e., string::npos),
   //   which holds the maximum value of size_t.
string (const char * cstr);
   // (4) C-string: construct by copying the C-string.
string (const char * cstr, size_t len);
   // (5) C-string buffer: construct by copying the cstr for len
string (size_t len, char c);
   // (6) Fill Constructor: fill len with char c
template <class Iterator>
string (Iterator first, Iterator last);
   // (7) Iterator: copy the char in [first, last)
string (initializer_list<char> initList);
   // (C++11)(8) Initializer list
string (string && str) noexcept;
   // (C++11)(9) Move Constructor
```

The following example defines and uses both C-strings and C++ strings

```
#include <iostream>
#include <string>   // C++ string class
#include <cstring>  // C-string
using namespace std;

int main() {
   char cstr[] = "hello world!";  // C-string literal

   string s1;                    // (1) Default constructor
   cout << s1.length() << endl; // 0
   cout << s1.size() << endl;   // 0 - length() and size() are synonyms

   string s4(cstr);    // (4) C-string
   s4[1] = 'E';        // [] does not perform index bound check
   cout << s4 << endl; // "hEllo world!"

   string s2(s4);       // (2) Copy constructor
   s2.at(0) = 'H';      // at() does index-bound check
                        // at() can be used on RHS
   cout << s2 << endl;  // "HEllo world!"
   cout << s4 << endl;  // no change - copy by value

   string s3a(s4, 2);   // (3) Substring
   cout << s3a << endl; // "llo world!"
   s3a += " again";     // Append
   cout << s3a << endl; // "llo world! again"

   string s3b(s3a, 4, 3); // (3) Substring
   cout << s3b << endl;   // "wor"

   string s5a(cstr, strlen(cstr));  // (5) C-string buffer
   cout << s5a << endl;  // "hello world!"
   string s5b(cstr, 15);             // (5) C-string buffer
   cout << s5b << endl;  // "hello world! ??"
        // If len > length of cstr, garbage copied

   string s6(5, '$');    // (6) Fill constructor
   cout << s6 << endl;   // "$$$$$"
```

```
   string s7a(cstr, cstr + 4);  // (7) Iterator
   cout << s7a << endl;  // "hell"
        // cstr1 is char*. Instantiate type parameter Iterator to char*

//   string s7b(s4, s4 + 2);
        // error: no match for 'operator+' in 's4 + 2'
        // s4 is a string object, not char*

   string s7c(&s4[0], &s4[2]);   // (7) Iterator
        // &s4[0] and &s4[2] are char*
   cout << s7c << endl;          // "hE"

   string s7d(s4.begin(), s4.end());
        // begin() returns an iterator pointing to the first character
        // end() returns an iterator pointing to past-the-end character
   cout << s7d << endl;   // "hEllo world!"


   cout << s1.empty() << endl;
   cout << s7d.empty() << endl;

   s7d.clear();
   cout << s7d.empty() << endl;

   cout << s5a.front() << endl;
   cout << s5a.back() << endl;

   string s99;
   s99 = s5a.substr(2, 4);
   cout << s99 << endl;

}
```

```
// Conversion of string to C-string

c_str;
```

Some functions, such as ofstream's `open()` which opens a file, accept only C-string. You can use c_str to get a C-string from a string object:

```
str99.c_str()
```

Here is an example:

```
// Conversion of C-string to string, use a constructor,
// creating on the fly a new string, then copying it to the desired one
// Eg, from argv[1]

string mystring;
mystring = string(argv[1]);

// or use a constructor from C-string:

string mystring(argv[1]);
```

Big example

```
/*
 * Guess a secret word (WordGuess.cpp)
 */
#include <iostream>
#include <string>
#include <cstdlib>   // srand, rand http://www.cplusplus.com/reference/cstdlib/rand/?kw=rand
#include <ctime>     // time
```

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

const int NUM_WORDS = 18;  // or read NUM_WORDS from sizeof(WORD_LIST)/sizeof(string)
const string WORD_LIST[NUM_WORDS] = {   // also []
   "apple", "orange", "banana", "watermelon", "pear",
   "pineapple", "papaya", "mango", "grape", "strawberry",
   "lemon", "peach", "cherry", "apricot", "coconut",
   "honeydew", "apricot", "blueberry"};

int main() {
   // Seed the pseudo-random number generator
   srand(time(0));    // http://www.cplusplus.com/reference/ctime/time/?kw=time

   bool over = false;   // gameover
   do {    // while not over
      string target = WORD_LIST[rand() % NUM_WORDS]; // choose a word between 0 to NUM_WORDS-1
      int target_length = target.length();
      string attempt(target_length, '-');            // init to all dashes
      string badChars;                               // contains bad chars used
      int trial = 1;                                 // number of trials

      cout << "Guess the secret word" << endl;

      while (attempt != target) {
         char letter;
         cout << "Your guess so far: " << attempt << endl;
         cout << "Trial " << trial << ": Guess a letter: ";
         cin >> letter;

         // Check if the letter has been used
         // (it might be in badChars or in attempt)
         if (badChars.find(letter) != string::npos    // string::npos is -1... it means "not found"
               || attempt.find(letter) != string::npos) {
            cout << "You already use this letter. Try again.\n";
            continue;    // get out of the while loop!!!
         }  // if badChars.find...

         // The letter had not been used before: Check for good or bad letter
         int pos = target.find(letter);
         if (pos == string::npos) {   // not found
            cout << "Oh, bad guess!\n";
            badChars += letter;     // add to badChars string: append!
         } else {
            cout << "Good guess!\n";
            attempt[pos] = letter;
            // Check if this letter appears again later
            do {
               pos = target.find(letter, pos + 1);
               if (pos != string::npos) attempt[pos] = letter;
            } while (pos != string::npos);
         }  // else

         ++trial;
      }  // while

      cout << "You got it in " << trial << " trials! The secret word is \""
           << target << "\"" << endl;

      char playAgain;
      cout << "Another game? <y/n> ";
      cin >> playAgain;
      if (playAgain != 'y' && playAgain != 'Y') over = true;
   } while (!over);

   cout << "Bye\n";
   return 0;
}
```

Altre funzioni:

http://www.ntu.edu.sg/home/ehchua/programming/cpp/cp9_String.html

append

```
insert
assign
erase
replace
swap
push_back
pop_back
compare
find
.....
```

FILE I/O

http://www.ntu.edu.sg/home/ehchua/programming/cpp/cp10_IO.html
http://www.bo.cnr.it/corsi-di-informatica/corsoCstandard/Lezioni/38IOCPP.html

C++ handles file IO similar to standard IO. You need to include both <iostream> and <fstream> headers in your program for file IO.

To write to a file, you construct a ofstream object connecting to the output file (file opening), and use the ostream functions such as stream insertion <<, put() and write(). Then the connection is removed by file closing.
Similarly, to read from an input file, construct an ifstream object connecting to the input file (file opening), and use the istream functions such as stream extraction >>, get(), getline() and read().Then the connection is removed by file closing.

File Output
The steps are:

- Construct an ostream object.
- Connect it to a file (i.e., file open) and set the mode of file operation (e.g, truncate, append).
- Perform output operation via insertion >> operator or write(), put() functions.
- Disconnect (close the file which flushes the output buffer) and free the ostream object.

```
#include <fstream>
.......
ofstream fout;
fout.open(filename, mode);
......
fout.close();

// OR combine declaration and open()
ofstream fout(filename, mode);
```

By default, opening an output file creates a new file if the filename does not exist; or truncates it (clear its content) and starts writing as an empty file.

File Input
The steps are:

- Construct an istream object.
- Connect it to a file (i.e., file open) and set the mode of file operation.
- Perform output operation via extraction << operator or read(), get(), getline() functions.
- Disconnect (close the file) and free the istream object.

```
#include <fstream>
.......
ifstream fin;
```

```
fin.open(filename, mode);
......
fin.close();

// OR combine declaration and open()
ifstream fin(filename, mode);
```

File Modes:

1. **ios::in** - open file for input operation
2. **ios::out** - open file for output operation
3. **ios::app** - output appends at the end of the file.
4. **ios::trunc** - truncate the file and discard old contents.
5. **ios::binary** - for binary (raw byte) IO operation, instead of character-based.
6. **ios::ate** - position the file pointer "at the end" for input/output.

You can set multiple flags via bit-or (|) operator, e.g., `ios::out | ios::app` to append output at the end of the file. For output, the default is `ios::out | ios::trunc`. For input, the default is `ios::in`.

Text I/O

```
/* Testing Simple File IO (TestSimpleFileIO.cpp) */
/* with changes by GDN                           */
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
using namespace std;

int main() {
   string filename = "test.txt";

   // Write to File
   ofstream fout(filename.c_str());  // default mode is ios::out | ios::trunc
   if (!fout) {
      cerr << "error: open file for output failed!" << endl;
      abort();  // in <cstdlib> header
   }
   fout << "apple" << " pineapple" << endl;
   fout << "orange" << endl;
   fout << "banana" << endl;
   fout.close();

   // Read from file, char by char
   ifstream fin(filename.c_str());  // default mode ios::in
   if (!fin) {
      cerr << "error: open file for input failed!" << endl;
      abort();
   }
   char ch;
   while (fin.get(ch)) {  // till end-of-file
      cout << ch;
   }
   fin.close();

   cout << endl;

   // Read from file, word by words
   ifstream fin2(filename.c_str());  // default mode ios::in
   if (!fin2) {
      cerr << "error: open file for input failed!" << endl;
      abort();
   }
   string aWord;
   while (fin2 >> aWord) {  // till end-of-file (>> returns null if nothing is read)
      cout << aWord << endl;
   }
   fin2.close();
```

```
   cout << endl;

  // Read from file, line by line
   ifstream fin3(filename.c_str());  // default mode ios::in
   if (!fin3) {
      cerr << "error: open file for input failed!" << endl;
      abort();
   }
   string aLine;
   while (getline(fin3, aLine)) {
    // do something with the line
      cout << aLine << endl;
   }
   fin2.close();

   return 0;
}
```

Binary I/O

TO BE DONE

EXERCISES:

3.1) Modify the file writing / reading example so that the code acquires the file name from the user, by means of a message to the user and the use of stdin (cin >>). Then, edit it again so that it acquires the file name from the command line (with argc / argv). Finally modify it so that it acquires the file name from the command line but, if the parameter is not passed by the user, then it explicitly asks for it with cin >>. Pay attention to the nature of the string (C-string or string) in the various cases.

3.2) Write a program composed of three different files: main.cpp, write.cpp, reread.cpp, containing respectively three homonymous functions. The main() program calls sequentially write() and reread(). write () asks the user for three values: *a*, *b*, *d*, and saves a text table to a file on disk, containing two columns: the first reports the values between *a* and *b* with step *d*, and the second showing the corresponding cosines. reread () reads the table and displays it on the screen. Do not use standard I/O and redirection, but I/O to a text file.

[LEZIONE 4]

Esercizio: il cifrario di Cesare

Fonte: http://www.crittologia.eu/critto/caesar.htm

Svetonio, nella “Vita dei dodici Cesari”, racconta che Giulio Cesare usava, per le sue corrispondenze riservate, una cifra (sistema di cifratura, ndGDN) monoalfabetica molto semplice, nella quale la lettera chiara viene sostituita dalla lettera che la segue di tre posti nell'alfabeto: la lettera A è sostituita dalla D, la B dalla E e così via fino alle ultime lettere che sono cifrate con le prime come nella tabella che segue (che fa riferimento all'odierno alfabeto internazionale). Svetonio non dice nulla su come cifrare le ultime lettere dell'alfabeto; di solito si intende che si proceda circolarmente ricominciando dalla A come nella lista seguente:

```
Chiaro    ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cifrato   DEFGHIJKLMNOPQRSTUVWXYZABC
```

Prendendo come esempio la frase "Auguri di buon compleanno" ed eliminando gli spazi, si otterrà il seguente messaggio cifrato:

```
Chiaro    AUGURIDIBUONCOMPLEANNO
Cifrato   DXJXULGLEXRQFRPSOHDQQR
```

Più in generale si dice cifrario di Cesare una cifra nella quale la lettera del messaggio chiaro viene spostata di un numero fisso di posti, non necessariamente tre; un esempio è la cifra che, sempre secondo Svetonio era usata da Augusto, dove la A era sostituita dalla B, la B dalla C fino all'ultima lettera dell'alfabeto latino, la X, che era sostituita da una doppia A.

Poiché l'alfabeto internazionale è composto da 26 caratteri sono possibili 26 cifrari di Cesare diversi dei quali uno (quello che comporta uno spostamento di zero posizioni) darà un cifrato uguale al messaggio chiaro iniziale. Dal punto di vista della crittanalisi il cifrario di Cesare è debolissimo essendoci solo 25 cifrari diversi.

Si scriva un programma C++ denominato `cifrarioCesare`, che:

- sia costituito da un `int main()` e da una funzione `char cifracarattere(char, int)`
- il `main()` acquisisca da riga di comando il nome di un file, creato dall'utente in maniera manuale, che contenga nella prima riga la chiave (valore numerico intero) e nelle righe successive il testo da criptare, che supponiamo per ora composto solo da lettere maiuscole ed eventualmente spazi, ad esempio:

  ```
  3
  AUGURI DI BUON COMPLEANNO
  ```

- dopo aver letto la chiave come int, allochi una stringa che conterrà il risultato della cifratura
- leggendo via via le "parole" contenute nel file, cicli sui caratteri di ciascuna parola, determinando, mediante la funzione `cifracarattere()`, il corrispondente carattere cifrato, e inserisca tale carattere cifrato nella stringa di output;
- leggendo il file di input per parole, automaticamente avverrà la non-criptazione degli spazi, che saranno ignorati;
- nell'esempio considerato, la stringa a questo punto sarà `DXJXULGLEXRQFRPSOHDQQR`
- finita la lettura del file di input, alla stringa di output sarà accodato un carattere il cui posto nell'alfabeto è il medesimo della chiave: essendo la chiave nel caso considerato pari a 3, la stringa di output a questo punto vale: `DXJXULGLEXRQFRPSOHDQQRC`
- una volta costruita la stringa di output, il programma apra in scrittura un file, il cui nome sarà dato dal nome del file di input preceduto dalla stringa "coded_": per esempio, se il file di input si chiama "text.txt", il nome del file di output deve essere "coded_text.txt"; si supponga, per semplicità, che il file di input sia situato nella medesima directory del programma eseguibile, per cui non vi saranno problemi di path più o meno complessi;
- nel file testé creato, il programma salvi la stringa cifrata
- La funzione `cifracarattere()` conterrà al suo interno una stringa costante, denominata alfabeto, definita come "ABCDEFGHIJKLMNOPQRSTUVWXYZ"; ricevuto un carattere da criptare, la funzione cercherà nella stringa la posizione di tale carattere, sommerà la chiave (che potrebbe anche essere negativa), e prenderà il resto della divisione

per 26 (lunghezza dell'alfabeto), ottenendo l'indice del carattere desiderato nell'alfabeto; prenderà quindi il carattere corrispondente e lo restituirà in output: in questo modo è realizzata la circolarità del codice (dopo la Z c'è la A, prima della A c'è la Z). Nota importante: il sistema funziona anche per numeri negativi! Supponiamo che nel calcolo otteniamo come carattere codificato il numero -1, ossia il carattere che precede la A: se calcoliamo -1%26 abbiamo 25, che è proprio la Z. Nota ulteriore: per ottenere il carattere corrispondente alla chiave, richiesto per "tappare" la stringa di output, è possibile adoperare la medesima funzione `cifracarattere()`: come?

- Ottenuto il software per la crittazione, si scriva il software per la decrittazione, per esempio `deCifrarioCesare()`: il main acquisisca da riga di comando il nome del file contenente il testo crittato, carichi il testo (un'unica stringa senza spazi, creata dal programma descritto in precedenza), ne prenda l'ultimo carattere per riconoscere la chiave, usi la chiave cambiata di segno per effettuare la decodifica con la medesima funzione `cifracarattere()`, e salvi un file con il nome preceduto da decoded_ contenente il testo, naturalmente privo degli spazi originali.
- Per riconoscere la chiave a partire dall'ultimo carattere del testo crittato, occorre accedere all'alfabeto usato dalla funzione `cifracarattere()`; diverse soluzioni sono possibili: nella più semplice l'alfabeto è una variabile (in realtà `const`) globale, oppure, meglio, una `#define` da inserire in un file a parte (`alfabeto.h`) da includere sia nella `cifracarattere()` che nella funzione `deCifrarioCesare()`.
- Se proprio vogliamo esagerare, realizziamo un terzo software che prenda i due file originale e decrittato, e scopra se il contenuto, a meno degli spazi, è uguale!

Note:

In questa versione supponiamo di non considerare le vocali accentate, le minuscole, i numeri e i segni di interpunzione. Essendo eventualmente interessati anche alla cifratura dei numeri, l'alfabeto dovrà avere al suo interno anche le cifre da 0 a 9; volendo codificare i segni necessari per esprimere numeri non interi, il segno "–" potrebbe essere sostituito dalla stringa "meno" e il punto decimale dalla stringa "punto".

In alternativa all'uso della stringa alfabeto, è possibile tenere conto del fatto che i caratteri alfanumerici hanno un corrispondente numerico nel codice ASCII (che è poi il valore contenuto nelle variabili char), per cui è possibile direttamente realizzare calcoli aritmetici sui char.

[LEZIONE 5]

Programmazione orientata agli oggetti (OOP)

https://www.html.it/guide/guida-c2/

Introduzione

La programmazione orientata agli oggetti consente di aggregare in un'unica entità i dati e le tecniche per la loro manipolazione. Ciò è reso possibile dalla definizione di classi che espandono l'insieme di tipi predefiniti previsti dallo standard C++.

Un modello di dati basato sul concetto di oggetto

- aggrega dati di vari tipi (predefiniti o composti) per rappresentare entità complesse.
- introduce per ogni oggetto anche un insieme di funzioni specializzate, e diversi livelli di accesso per ogni sua proprietà o membro.

Principio di **incapsulamento**: una classe definisce un'interfaccia verso il mondo esterno, che consente di manipolarne di dati e, allo stesso tempo, maschera le dinamiche del suo funzionamento interno.

Problematiche complesse possono essere decomposte e risolte mediante la definizione di oggetti e delle interazioni tra di essi. Ciò si traduce in molti casi in una metodologia di progettazione molto più concettuale o di alto livello di quella offerta dalla programmazione strutturata.

Nel linguaggio C++ tutto ciò è reso possibile dal costrutto class, la cui sintassi è la seguente:

```
class <nome>
{
public:
          // membri accessibili all'esterno della classe
protected:
          // membri accessibili dalla classe ed in tutte le sue derivate
private:
          // membri accessibili solo all'interno della classe
};
```

Nome di una classe: valgono le regole valide per gli identificatori

Le tre sezioni distinte (public etc) (non necessariamente tutte presenti) servono per la definizione del livello di accesso o visibilità di ogni membro: pubblico, protetto e privato. Come impostazione predefinita, se nessuna delle tre sezioni viene indicata esplicitamente, i membri della classe sono privati (differenza con le struct!).
I dettagli si comprenderanno quando si studieranno l'ereditarietà e le classi derivate.

Per ragioni storiche e tecniche in C++ è preferibile scomporre il codice relativo ad una classe in due file, uno contenente la definizione dell'interfaccia (detto header) e l'altro l'implementazione dei metodi. Per convenzione, il nome di entrambi tali file è dato dal nome della classe in esso contenuta, con estensione .h e .cpp rispettivamente. Alle volte il nome della classe è preceduto da "C".
Esempio in un unico file.

- Classe rettangolo
- Sovraccarico (overloading) di una funzione membro (un metodo) (in questo caso, il costruttore)

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


class rettangolo
{
        private:
                double a, b;
        public:
```

```
                rettangolo()
                {
                        cout << "Sono il costruttore di default\n";
                        a = b = 0;
                }

                rettangolo(double a, double b)  // compile with -Wshadow
                {
                        cout << "Sono il costruttore con parametri\n";
                        this->a = a;
                        this->b = b;
                }

                ~rettangolo()
                {
                        cout << "Sono il distruttore\n";

                }

                double getA()
                {
                        return a;
                }

} ;

int main()
{

        rettangolo r1;
        rettangolo r2(5,10);
        cout << r1.getA() << endl;
        cout << r2.getA() << endl;

        return 0;
}
```

Consideriamo la classe Point2D:

```
// Point2D.h
#ifndef POINT_2D_H
#define POINT_2D_H
namespace Geometry {
        class Point2D;
}
class Geometry::Point2D
{
public:
        double X();    // getter
        void setX(double value);
        double Y();    // getter
        void setY(double value);
        double distanceFrom(Point2D other);
private:
        double x;
        double y;
};
#endif // POINT_2D_H
```

La definizione della classe è compresa tra le cosiddette “include guards”: direttive al preprocessore che hanno lo scopo di impedire che il file header venga incluso più volte durante la compilazione dell’eseguibile, causando problemi di definizione multipla della stessa classe.

Per evitare di generare collisioni con altri nomi, è sempre saggio definire un namespace che contiene le definizioni delle nostre classi. In questo caso la classe appartiene al namespace Geometry, come dichiarato subito dopo le include guards. Usiamo una dichiarazione anticipata

all'interno del namespace per spostare la definizione della classe all'esterno di esso, evitando così un ulteriore livello di indentazione. Si noti che questa è una scelta stilistica che ci obbliga ad usare il prefisso Geometry:: ogni volta che il nome della classe ricorre nel listato. Nulla vieta, però, di definire la classe entro il blocco che delimita il namespace, se lo si preferisce.

La classe contiene due membri privati di tipo double, denominati x e y, e dei metodi pubblici per l'accesso a tali membri rispettivamente in lettura, X() e Y(), ed in scrittura, setX() e setY().

È inoltre presente un metodo pubblico per il calcolo della distanza euclidea rispetto alle coordinate di un altro oggetto di tipo Point2D. La definizione della classe deve essere seguita dal simbolo ; per ragioni di uniformità e retrocompatibilità con la definizione di struct mutuata dal linguaggio C, che in C++ è un costrutto equivalente a quello di classe, seppure con delle piccole differenze (essenzialmente, per default tutti i membri sono pubblici).

L'implementazione dei metodi di classe è contenuta nel file point2D.cpp, riportato nel listato seguente:

```
// Point2D.cpp
#include "Point2D.h"
#include <cmath>
double Geometry::Point2D::X() {
          return x;
}
void Geometry::Point2D::setX(double value) {
          if (!std::isnan(value) && !std::isinf(value))
                    x = value;
          else
                    x = 0;
}
double Geometry::Point2D::Y() {
          return y;
}
void Geometry::Point2D::setY(double value) {
          if (!std::isnan(value) && !std::isinf(value))
                    y = value;
          else
                    y = 0;
}
double Geometry::Point2D::distanceFrom(Point2D other)
{
          return std::sqrt((x - other.x) * (x - other.x) + (y - other.y) * (y - other.y));
}
```

Il file .cpp relativo all'implementazione di una classe deve includere il file header della classe, e da tutti gli altri header necessari, affinché il compilatore possa risolvere correttamente i nomi dei simboli in esso contenuti.

La definizione di membri interni privati e dei corrispettivi metodi getter e setter è una pratica comune nella definizione di classi che serve a garantire che, qualora sia necessario modificare i valori di x e y, ciò avvenga in maniera "controllata". Ad esempio, in questo caso, i valori da assegnare come coordinate del punto vengono prima validati.

Se x e y fossero stati membri pubblici, sarebbe stato necessario ripetere questo tipo di validazione ogni volta che ad essi fosse stato assegnato un nuovo valore, aumentando così le possibilità di errore. Come ulteriore beneficio della presenza di tali metodi e della validazione che essi operano, il risultato della funzione distanceFrom sarà sempre un valore numerico valido.

Inoltre, i metodi getter e setter generalmente aiutano a preservare l'espandibilità e la retrocompatibilità dell'interfaccia delle classi, permettendo di mutare la semantica di uno o più membri privati o protetti, pur mantenendo intatta l'interfaccia pubblica della nostra classe. Ad esempio, potremmo decidere di mutare la rappresentazione interna del punto in coordinate polari (ρ e θ), mantenendo esternamente la rappresentazione cartesiana (x e y) grazie ai metodi di accesso pubblici. Ciò comporterebbe la sola modifica della classe Point2D, ma non di altre parti del codice in cui essa era usata prima di queste modifiche.

Il listato seguente mostra un esempio dell'utilizzo di tale classe:

```
// testPoint2D.cpp
#include <iostream>
#include "Point2D.h"
using Geometry::Point2D;
//using namespace Geometry;
// or put Geometry:: before names coming from this namespace
using namespace std;

int main() {
        Point2D p;
        p.setX(1);
        p.setY(1);
        Point2D p2;
        p2.setX(2);
        p2.setY(2);

        cout << p.distanceFrom(p2) << endl;
        return 0;
}
```

Mostrare l'inizializzazione con `: X(valueX), Y(valueY)`

Compilare con

```
g++ -o testPoint2D testPoint2D.cpp Point2D.cpp
```

Sorgente completo in un singolo file (test03.cpp):

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


namespace Geometry {
        class Point2D;
}
class Geometry::Point2D
{
public:
        double X();
        void setX(double value);
        double Y();
        void setY(double value);
        double distanceFrom(Point2D other);
private:
        double x;
        double y;
};



#include <cmath>
```

```
double Geometry::Point2D::X() {
	return x;
}
void Geometry::Point2D::setX(double value) {
	if (!std::isnan(value) && !std::isinf(value))
		x = value;
	else
		x = 0;
}
double Geometry::Point2D::Y() {
	return y;
}
void Geometry::Point2D::setY(double value) {
	if (!std::isnan(value) && !std::isinf(value))
		y = value;
	else
		y = 0;
}
double Geometry::Point2D::distanceFrom(Point2D other)
{
	return std::sqrt((x - other.x) * (x - other.x) + (y - other.y) * (y - other.y));
}

using  Geometry::Point2D;

int main()
{
	Point2D p;

	p.setX(1); p.setY(1);

	Point2D p2;

	p2.setX(2); p2.setY(2);

	cout << p.distanceFrom(p2) << endl;
	return 0;
}
```

[LEZIONE 6]

Prerequisito

Funzioni: passaggio di variabili per valore, indirizzo, riferimento; valore restituito

Il passaggio di parametri a una funzione può avvenire in tre modi diversi:

per valore: il valore di una variabile o costante, parametro effettivo, è **copiato** nel parametro formale della funzione (e quindi nella corrispondente variabile locale); se il parametro effettivo è una variabile, il suo contenuto non può essere modificato dalla funzione.

per indirizzo: un puntatore alla variabile parametro effettivo, è passato nel parametro formale della funzione; il parametro effettivo (o, meglio, il valore da esso puntato) può in linea di principio essere modificato dalla funzione, salvo se si fa uso dell'attributo `const` (vantaggi: usa poca memoria, permette ad una funzione di restituire più di un valore; svantaggio: "pericoloso", sintassi meno immediata). In realtà è solo un passaggio per valore, in cui però il valore è quello del puntatore a una variabile (tipico del C!).

per riferimento: la funzionalità è analoga al passaggio per indirizzo, ma con una sintassi più semplice (C++).

I tipi dei dati (passati alla funzione) devono concordare con quelli previsti, a meno di possibili conversioni automatiche o di cast espliciti.

Cosa non possiamo fare con il passaggio per valore:

```
void scambia (int x, int y){
        int tmp ;
        tmp = x;
        x = y;
        y = tmp;
}

int main (){
        int a = 1, b = 2;
        scambia (a, b);
        cout << a << " " << b << endl;
        return(0);
}
```

... e invece possiamo con il passaggio per riferimento:

```
void scambia (int& x, int& y){
        int tmp;
        tmp = x;
        x = y;
        y = tmp;
}

int main (){
        int a = 1, b = 2;
        scambia (a, b);
        cout << a << " " << b << endl;
        return(0);
}
```

Con il passaggio mediante puntatore, *à la* C, avremmo:

```
void scambia (int *x, int *y){
        int tmp;
        tmp = *x;
        *x = *y;
        *y = tmp;
}

int main (){
        int a = 1, b = 2;
        scambia (&a, &b);
        cout << a << " " << b << endl;
        return(0);
}
```

ugualmente funzionante ma con sintassi meno agile.

Per passare parametri per riferimento (e analogamente per indirizzo) allo scopo di non muovere grosse zone di memoria (quindi, non desiderando l'alterazione del valore del parametro) è consigliabile usare una dichiarazione const.

```
void scambia (const int &x, const int &y){ ...
```

se si prova ad alterare il valore di x o y (assegnando a x un valore, per esempio), il compilatore dà errore.

Costruttori (continuazione)

https://www.html.it/pag/15519/costruttori-e-distruttori/

La classe Point2D, completata con un paio di costruttori:

```
// point2d.h
#ifndef POINT_2D_H
#define POINT_2D_H
namespace Geometry {
	class Point2D;
}
class Geometry::Point2D
{
public:
	Point2D();                                          // costruttore di default
	Point2D(double xValue, double yValue);              // costruttore sovraccaricato
	double X();
	void setX(double value);
	double Y();
	void setY(double value);
	double distanceFrom(Point2D other);
private:
	double x;
	double y;
};
#endif // POINT_2D_H
```

```
// point2d.cpp
#include "point2d.h"
#include <cmath>
Geometry::Point2D::Point2D()
{
	x = 0;
	y = 0;
}
Geometry::Point2D::Point2D(double xVal, double yVal)
{
	setX(xVal);
	setY(yVal);
}
void Geometry::Point2D::setX(double value) {
	if (!std::isnan(value) && !std::isinf(value))
		x = value;
	else
		x = 0;
}
void Geometry::Point2D::setY(double value) {
	if (!std::isnan(value) && !std::isinf(value))
		y = value;
	else
		y = 0;
}
```

```
// altro...
```

Costruttore/distruttore in caso di allocazione dinamica

La necessità di inserire esplicitamente dei costruttori (e soprattutto dei distruttori) si fa sentire soprattutto in caso di allocazione dinamica.

```
// test04a.cpp
class A {
public:
        A();
        ~A();
private:
        int m;
        int *ptr;
        int *ptr2;
};

A::A()
{
        m = 0;
        ptr = new int;
        ptr2 = new int[1000000];
}

A::~A()
{
        delete ptr;
        delete [] ptr2;
}

int main()
{
        A a;

        return 0;
}
```

E giacchè stiamo parlando di allocazione dinamica, vediamo se è proprio vero che la heap contiene più spazio allocabile…

Partiamo da:

```
// test04b1.cpp
int y[1000000000]{254};  // global!

int main()
{
        int x[2093000]{254};
        return 0;
}
```

e compiliamo variando la dimensione degli array per vedere dove subentrano problemi.
Poi inseriamo il seguente programma, che riserva un’area di memoria nella heap.

```
#include <iostream>
using namespace std;
// attention: new does not return 0 on fail!!!
// test04b.cpp

class A {
public:
	A(int sz);
	~A();
private:
	int size;
	int *ptr;
};


A::A(int sz)
{
	size = sz;
	cout << "Constructor: allocating memory\n";
	ptr = new int[size];
	cout << ptr << endl;
	cout << "Constructor: filling memory\n";
	for (int j = 0; j < size; j++) *ptr = 32323;
}

A::~A()
{
	cout << "Destructor\n";
	delete [] ptr;
}

// int y[1000000000]{254};  // global!

int main(int argc, char **argv)
{
	// int x[2093000]{254};

	int sz;
	if (argc > 1)
		sz = atoi(argv[1]);
	else
		sz = 100;
	cout << "Creating object with " << sz << " integers\n";  //1000000000000

	A a(sz);

	return 0;
}
```

Sperimentare con diversi valori di dimensione della memoria richiesta.

Costruttori per copia e operatore di assegnazione (o “assegnamento di copia”) (overloading)

https://www.html.it/pag/63405/copia-di-oggetti/

```
#include <iostream>
using namespace std;
// test05.cpp

class ctest {

	private:
		int x;
	public:
		ctest()
		{
			cout << "Default constructor\n";
			x = 0;
```

```
		}

		ctest(int x1)
		{
			cout << "Constructor by value\n";
			x = x1;
		}


		ctest(const ctest& c)   // <---- const!
		{
			cout << "Copy constructor\n";
	//		x = c.getX();
			x = c.x;  // !!
		}

		ctest& operator=(const ctest& c)
		{
			cout << "Assignment operator\n";
	//		x = c.getX();
			x = c.x;  // !! allowed because same class
			return *this;
		}



		int getX()
		{
			return x;
		}

		void print()
		{
			cout << x << endl;
		}


} ;


int main()
{
	ctest a;
	ctest b(12);

	a.print();
	b.print();

	ctest c = b;		// call Copy Constructor
	c.print();

	ctest d(b);		// call Copy Constructor
	d.print();

	ctest e(ctest(55));	// call Default constructor, then Copy Constructor,
				//works because of the "const" attribute!   ???
	e.print();			// It is not the copy constructor… clarify!

	a = b;			// call Assignment operator
	a.print();

	return 0;
}
```

Esercizio: modificare la classe in modo che sia conservato un campo in più, il nome della variabile, e tale campo sia scritto in output dalla print: “La variabile x vale 123”. Il nome sarà assegnato da un costruttore o tramite una funzione membro set.

Costruttore per copia in caso di allocazione dinamica

https://www.html.it/pag/63405/copia-di-oggetti/

Il costruttore per copia proposto per la classe ctest, non è differente da quello che il compilatore avrebbe generato per noi. Infatti, la semantica che viene applicata per il costruttore di copia generato automaticamente è quella della copia membro a membro.
Quando, però, una classe contiene dati membro che sono **puntatori o reference**, la copia membro a membro di un oggetto può non essere sufficiente.

In questo caso, si dice che la copia membro a membro è una copia "superficiale" (shallow copy). Un'istanza della classe ed una sua shallow copy, sono infatti legate, poiché **una modifica fatta ad un dato membro puntatore o reference** di una si riflette automaticamente nell'altra, con conseguenze che possono portare anche all'instabilità del programma.
In questi casi è quindi importante implementare il costruttore di copia correttamente, eliminando eventuali dipendenze tra un oggetto e le sue copie. Un esempio di questo approccio è mostrato nel listato seguente

```
class A {
public:
        A();
        ~A();
        A(const A& other);
private:
        int m;
        int *ptr;
};


A::A(const A& other)
{
        m = other.m;
        // shallow copy
        // ptr = other.ptr;
        // deep copy
        ptr = new int;
        if (other.ptr != nullptr)
                *ptr = *(other.ptr);
}

A a(b);
```

Esercizio: scrivere una classe A i cui member variable privati siano un int size e un int* ptr; ptr punti a una zona di memoria allocata con new, la cui lunghezza sia contenuta in size.
La classe preveda:

- un costruttore di default, che definisce ptr come null e size=0;
- un costruttore con un parametro sz, che allochi sz interi nella heap, e ponga ptr a puntare a tale zona di memoria, e size = sz; riempia poi la memoria di numeri casuali compresi tra 0 e 99;
- un distruttore, che liberi la memoria se era stata allocata (quindi, se ptr non è null);
- un costruttore per copia, `A::A(const A& other)`, che copi la size della sorgente nella size della destinazione, allochi altrettanta memoria, e copi il contenuto della memoria (se di lunghezza diversa da 0!) nella nuova zona; usare `memcpy`, presente in `<cstring>`;
- un operatore di assegnazione, `A& A::operator=(const A& other);` piuttosto complesso: deve infatti – prima di allocare una nuova zona di memoria da assegnare all'oggetto destinazione

– ricordare di deallocare la vecchia! Poi, se l'oggetto sorgente è non vuoto (quindi size != 0 oppure ptr != null), alloca ed effettua la copia;

- una funzione print() che stampa il contenuto della memoria a video (se non vuota); se vuota, stampa "Empty"
- aggiungere, eventualmente, una variabile che conservi il nome della variabile.

Esempio completo

```
// test06.cpp
#include <iostream>
#include <cstring>   // memcpy
#include <cstdlib>   // srand, rand http://www.cplusplus.com/reference/cstdlib/rand/?kw=rand
#include <ctime>     // time

using namespace std;

// attention: new does not return 0 on fail!!! catch exceptions (or use nothrow)!
// http://www.cplusplus.com/reference/new/operator%20new/

class A {
public:
        A();        // default constructor
        A(int sz);  // constructor
        ~A(); // destructor
        A(const A& other);   // copy constructor
        A& operator=(const A& other);   // copy constructor
        void print();
private:
        int size;
        int *ptr;
};

A::A()
{
        size = 0;
        ptr = NULL;
}

A::A(int sz)
{
        size = sz;
        cout << "Constructor: allocating memory\n";
        ptr = new int[size];
        cout << ptr << endl;
        cout << "Constructor: filling memory\n";
        for (int j = 0; j < size; j++) ptr[j] = rand()%100;
}

A::~A()
{
        cout << "Destructor\n";
        delete [] ptr;
}



A::A(const A& other)
{
        cout << "Copy Constructor\n";
        size = other.size;
        // shallow copy
        // ptr = other.ptr;
        // deep copy
        ptr = new int[size];
        // http://nadeausoftware.com/articles/2012/05/c_c_tip_how_copy_memory_quickly
        if (other.ptr != nullptr)
                memcpy( (void*)ptr, (void*) other.ptr, size * sizeof(int) );
}
```

```
A& A::operator=(const A& other)
{
	cout << "Assignment operator\n";

	size = other.size;
	// shallow copy (and memory leak):
	// ptr = other.ptr;

	// deep copy

	if (other.ptr) {
		if (ptr) delete [] ptr;
		ptr = new int[size];
		// http://nadeausoftware.com/articles/2012/05/c_c_tip_how_copy_memory_quickly
		if (other.ptr != nullptr)
			memcpy( (void*)ptr, (void*) other.ptr, size * sizeof(int) );
	}
	return *this;
}

void	A::print()
{
	if (size == 0)
		cout << "Empty\n";
	else
	{
		for (int j = 0; j < size; j++)
			cout << ptr[j] << ", ";
		cout << endl;
	}

}

int main()
{
	// Seed the pseudo-random number generator
//	srand(time(0));	// http://www.cplusplus.com/reference/ctime/time/?kw=time


	A a(10);
	a.print();

	A a1(a);
	a1.print();

	A a2 = a;
	a2.print();

	A a3;
	a3.print();
	a3 = a;

	a.print();

	A a5(20);
	a5.print();
	a5 = a;
	a5.print();

	return 0;
}
```

Default values of arguments in constructors

https://stackoverflow.com/questions/36108530/default-values-of-arguments-in-c-constructor

Lanciare un programma esterno da C++: gnuplot

Se necessario:

```
sudo apt install gnuplot
```

https://nccastaff.bournemouth.ac.uk/jmacey/ASE/labs/lab1.html

Primo test con gnuplot: creare un file, chiamato ad esempio line.dat, contenente le seguenti righe:

```
# is used for comments
# this indicates the format of the data
# X Y
-2.0 1.0
3.0 -2.0
```

Lanciare il programma gnuplot, poi dare il commando:

```
gnuplot> plot 'line.dat' with lines
```

Se ci sono messaggi di errore relative al modulo canberra-gtk-module,provare le soluzioni al seguente link:
https://askubuntu.com/questions/342202/failed-to-load-module-canberra-gtk-module-but-already-installed

(Ho risolto con sudo apt install libcanberra-gtk-module)

Altro esempio:
Generare i punti di una circonferenza con il seguente algoritmo (parametrico: variando il valore di un parametro t tra 0 e 2 π, in corrispondenza generiamo le coordinate del punto corrispondente della circonferenza):

- Chiedi all'utente di inserire il raggio (r) della circonferenza (di tipo double)
- Chiedi all'utente di inserire lo step (dt) di incremento del parametro t (di tipo double)
- Per t da 0 a 2π:
  - calcola i corrispondenti valore di x e y, come segue: x=r·cos(t), y=r·sin(t);
  - scrivi a video i valori x e y;
  - incrementa t della quantità dt: t += dt

I valori emessi a video possono essere rediretti in un file con ">", per poi essere usati con gnuplot: attenzione: così non avremo in output alcun messaggio del programma, nemmeno le richieste di inserire valori: andrà tutto nel file in cui avremo rediretto lo standard output. Per evitare problemi, possiamo:

- mandare in cout i messaggi per l'utente preceduti da #, in modo che poi gnuplot li consideri commenti.
- usare cerr (stderr) e cout (stdout) per mansioni diverse (uno per i messaggi e uno per le coordinate da redirigere), ma la soluzione non è pulitissima. Ricordare che > redirige (con troncamento) il cout, mentre 2> redirige il cerr.

In una seconda versione del programma, i dati x e y sono scritti in un file di testo.

Una volta salvato il file, usare gnuplot per disegnare la circonferenza.

In una terza versione, i dati sono scritti in un file di testo, e poi gnuplot è chiamato dall'interno del codice (vedere di seguito).

Nota: π è definito in <cmath> come M_PI; analogamente <cmath> contiene cos e sin.

Esercizio:
E' possibile tracciare un plot su più colonne; inserire nel file twocol.dat i seguenti dati:

```
0 0 1
0.01 0.0627905 0.998027
0.02 0.125333 0.992115
```

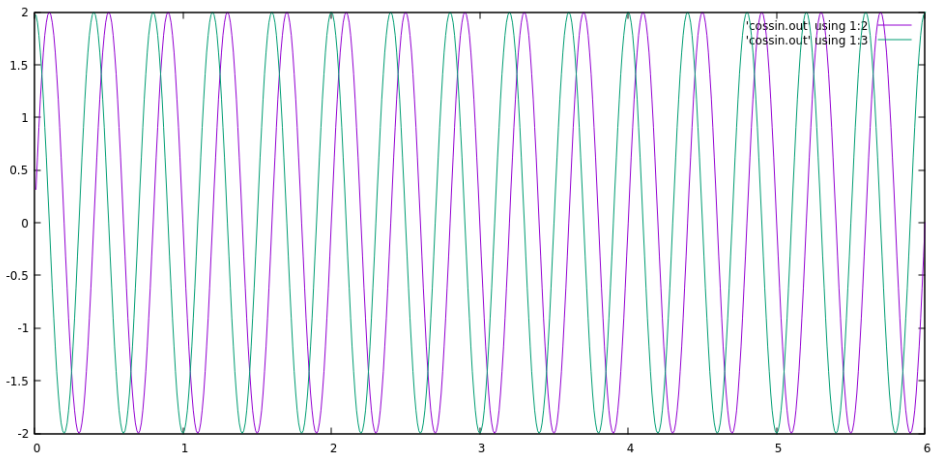
e usare il seguente comando di gnuplot:

```
gnuplot> plot 'twocol.dat' using 1:2 with lines , 'twocol.dat' using 1:3 with lines
```

Esercizio:

Write a program that requests two inputs from the user, one for frequency and one for amplitude, and then plots both the cos and sine waves for the given values using the algorithm

for x=0; x<max_value; x+=0.1
  ys = amplitude*sin(freq*M_PI*x)
  yc = amplitude*cos(freq*M_PI*x)



BASH Shell: How To Redirect stderr To stdout (redirect stderr to a File)
https://www.cyberciti.biz/faq/redirecting-stderr-to-stdout/
Bash and other modern shell provides I/O redirection facility. There are 3 default standard files (standard streams) open:
[a] **stdin** – Use to get input (keyboard) i.e. data going into a program.

[b] **stdout** – Use to write information (screen)
[c] **stderr** – Use to write error message (screen)

The Unix / Linux standard I/O streams with numbers:

Handle	Name	Description
0	stdin	Standard input
1	stdout	Standard output
2	stderr	Standard error

Redirecting the standard output stream to a file

```
$ command1 > output.log
```

Redirecting the standard error stream to a file

```
$ command1 2> error.log
```

Redirecting the standard error (stderr) and stdout to file

```
$ command-name &>file
```
OR
```
$ command > file-name 2>&1
```

Redirect stderr to stdout

```
$ command-name 2>&1
```

Chiamare gnuplot da C++ con system()

```
// 1
  std::string command1;
  std::string exe = "gnuplot -p ";  // "persistent"
  std::string plotfile = "cossin2.out";   // data file containing commands
  command1 = exe + plotfile;
  std::system(command1.c_str());

// 2
  std::string command2;
  command2 = "gnuplot -p ";
  command2 += "cossin2.out";
  std::system(command2.c_str());

// 3
// note the "'s in the command!
  std::string command3 = gnuplot -p -e \"plot 'cossin.out' using 1:2 with lines \";
  std::system("command3");
```

cossin2.dat contiene comandi e dati (questi ultimi inseriti in un blocco):

```
$data << EOD
0     0     20
0.01   6.18034    19.0211
....
6 -5.27271e-11 20
EOD

plot "$data" using 1:2 with lines, "$data" using 1:3 with lines
```

mentre cossin.dat contiene solo i dati.

Alternativa: linkare e chiamare direttamente gnuplot:
Gnuplot-iostream interface:
http://stahlke.org/dan/gnuplot-iostream/

Usare una libreria esterna, solo .h: CImg

http://cimg.eu scaricare la libreria da "download".

Inserire il codice (praticamente vuoto):

```
#include "CImg.h"
using namespace cimg_library;
int main() {

      return 0;
}
```

Compilare con
`g++ -o test01 test01.cpp -I ~/C++/CImg-2.4.2`

dove si suppone che il file CImg.h sia in `-I ~/C++/CImg-2.4.2` (modificare in base alla propria configurazione).

(lo spazio tra `-I` e `~/C++/CImg-2.4.2` è obbligatorio).

In caso di errore "CImg.h: File o directory non esistente", installare `libx11-dev`:

`sudo apt-get libx11-dev`

(magari preceduto da `sudo apt-get update`)

Implementare ora il codice in questo modo:

```
#include <iostream>


#include "CImg.h"
using namespace cimg_library;


int mandelbrot(double startReal, double startImag, int maximum);

int main(int argc, char **argv)
{
          // Read an image from file

          CImg<unsigned char> image1("PNG_transparency_demonstration_2.png");
          CImgDisplay disp1(image1,"Here is an image");

          // Create an empty colour image and populate it


          CImg<unsigned char> image2(500,400,1,3,0);      // 500 (x) by 500 (y) by 1 (z),
```

```
                                                  // three color planes, filled with 0

          for (int x = 0; x < image2.width(); x++)
                    for (int y = 0; y < image2.height(); y++)
                    {
                              image2(x, y, 0, 0) = 200;
                              image2(x, y, 0, 1) = 20;
                              image2(x, y, 0, 2) = 45;
                    }

          CImgDisplay disp2(image2,"Populated image");



          // Create the Mandelbrot set
          // derived from
// https://codereview.stackexchange.com/questions/124358/mandelbrot-image-generator-and-viewer


          const int IMAGE_WIDTH = 1000;
          const int IMAGE_HEIGHT = 600;
          double zoom = 0.004; // allow the user to zoom in and out...
          double offsetX = -0.7; // and move around
          double offsetY = 0.0;
          const int MAX = 255; // maximum number of iterations for mandelbrot()

          if (argc > 1) {
                    zoom = atof(argv[1]);
                    offsetX = atof(argv[2]);
                    offsetY = atof(argv[3]);
          }


          CImg<unsigned char> image3(IMAGE_WIDTH,IMAGE_HEIGHT,1,1,0); // 500 (x) by 500 (y) by 1(z),
                                                                      // one plane, filled with 0

        for (int x = 0; x < IMAGE_WIDTH; x++) {
            for (int y = 0; y < IMAGE_HEIGHT; y++) {
                // convert x and y to the appropriate complex number
                double real = (x - IMAGE_WIDTH / 2.0) * zoom + offsetX;
                double imag = (y - IMAGE_HEIGHT / 2.0) * zoom + offsetY;
                int value = mandelbrot(real, imag, MAX);
                image3(x, y, 0, 0) = value; // also image3(x, y) = value, see CImg doc
            }
        }


          image3.map(CImg<>::jet_LUT256());  // enable this Look Up Table as the color map

          CImgDisplay disp3(image3,"Mandelbrot");

          while (!disp1.is_closed() && !disp2.is_closed() && !disp3.is_closed()) {
          } // while

          return 0;

}


int mandelbrot(double startReal, double startImag, int maximum) {
          int counter = 0;
          double zReal = startReal;
          double zImag = startImag;
          double nextRe;

          while (pow(zReal, 2.0) + pow(zImag, 2.0) <= 4.0 && counter <= maximum) {
          nextRe = pow(zReal, 2.0) - pow(zImag, 2.0) + startReal;
          zImag = 2.0 * zReal * zImag + startImag;
          zReal = nextRe;
          if (zReal == startReal && zImag == startImag) { // a repetition indicates that the point
                                                                  //is in the Mandelbrot set
                              return -1; // points in the Mandelbrot set are represented by -1
                    }
          counter += 1;
          }
```

```
            if (counter >= maximum) {
            return -1; // -1 is used here to indicate that the point lies within the Mandelbrot set
            } else {
            return counter; // returning the number of iterations allows for colouring
            }
}
```

E compilare con

```
g++ -o test01 test01.cpp -I ~/C++/CImg-2.4.2 -O2 -L/usr/X11R6/lib -lm -lpthread -lX11
```

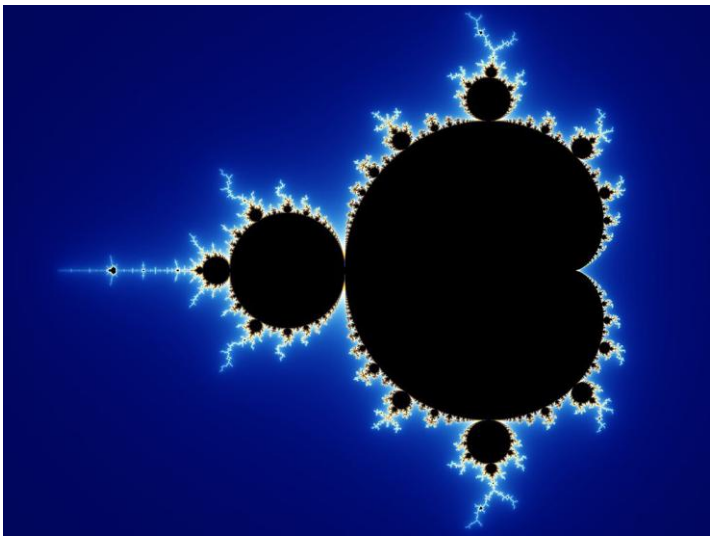
(pag 13 del documento CImg_reference).

Da wikipedia (https://it.wikipedia.org/wiki/Insieme_di_Mandelbrot):

L'insieme di Mandelbrot o frattale di Mandelbrot è uno dei frattali più popolari. È l'insieme dei numeri complessi *c* per i quali la successione definita da:

$$\begin{cases} z_0 = 0 \\ z_{n+1} = z_n^2 + c \end{cases}$$

è limitata. Si può dimostrare che se il modulo di z_n è maggiore di 2 allora la successione divergerà e quindi il punto *c* è esterno all'insieme di Mandelbrot. Le immagini multicolori dell'insieme di M. sono generate colorando i punti esterni all'insieme in dipendenza di "quanto velocemente" la sequenza z_n diverge. Il minimo valore di *n* per cui $|z_n| > 2$ è un indice di quanto "lontano dal contorno" si trova un punto ed è utilizzato per la rappresentazione "a colori". Nonostante la semplicità della definizione, l'insieme ha una forma complessa il cui contorno è un frattale. I punti colorati che conferiscono il fascino al frattale di Mandelbrot sono quelli che **non** appartengono all'insieme



Ricorsione

Soluzione dell'esercizio 3.2:

http://ictechnologies.altervista.org/matrice-trasposta-c/

sudo apt-get update

Library and Include Files

If you have library or include files in non-standard locations, the -L{DIRNAME} and -I{DIRNAME} options allow you to specify these locations and to insure that they are searched before the standard locations. For example, if you store custom include files in /usr/local/include/killerapp, then in order for gcc to find them, your gcc invocation would be something like
$ gcc someapp.c -I/usr/local/include/killerapp
Similarly, suppose you are testing a new programming library, libnew.so (.so is the normal extension for shared libraries— more on this subject in Chapter 24, "Using Libraries") currently stored in /home/fred/lib, before installing it as a standard system library. Suppose also that the header files are stored in /home/fred/include. Accordingly, to link against libnew.so and to help gcc find the header files, your gcc command line should resemble the following:
$gcc myapp.c -L/home/fred/lib -I/home/fred/include -lnew
The -l option tells the linker to pull in object code from the specified library. In this example, I wanted to link against libnew.so. A long-standing UNIX convention is that libraries are named lib{something}, and gcc, like most compilers, relies on this convention. If you fail to use the -l option when linking against libraries, the link step will fail and gcc will complain about undefined references to "function_name."

By default, gcc uses shared libraries, so if you must link against static libraries, you have to use the -static option. This means that only static libraries will be used. The following example creates an executable linked against the static ncurses. Chapter 27, "Screen Manipulation with ncurses," discusses user interface programming with ncurses:
$ gcc cursesapp.c -lncurses -static
When you link against static libraries, the resulting binary is much larger than using shared libraries. Why use a static library, then? One common reason is to guarantee that users can run your program—in the case of shared libraries, the code your program needs to run is linked dynamically at runtime, rather than statically at compile time. If the shared library your program requires is not installed on the user's system, she will get errors and not be able to run your program.

LIBRERIE OPENCV

http://parliamodi-ubuntu.blogspot.com/p/indice-del-corso-c.html

echo $SHELL

[LEZIONE 7]

File binari

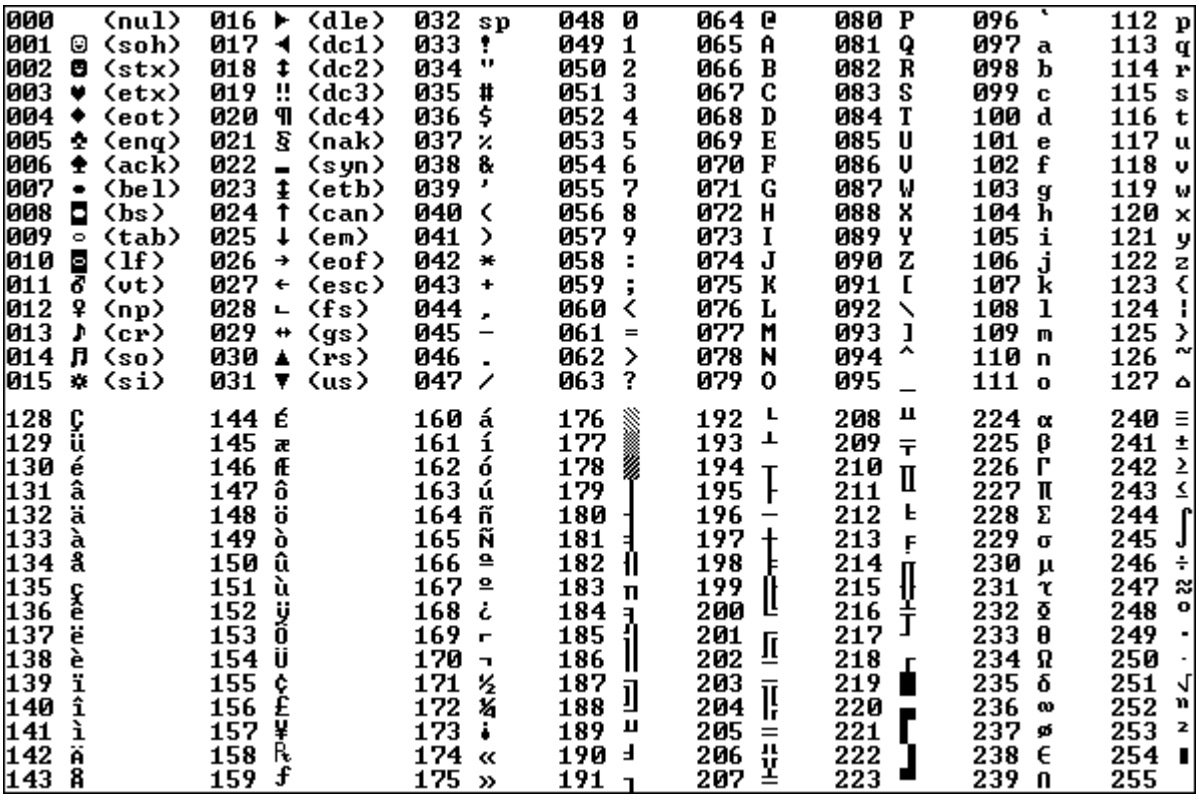
Sia i file binari che quelli testuali sono sequenze di byte (=char) salvati su disco.
In un file testuale, sia le stringhe sia i numeri sono sequenze di caratteri:

ciao 1234 è salvato come la sequenza di “c”, “i”, “a”, “o”, “ ”, “1”, “2”, “3”, “4”

In un file binario, le stringhe sono sequenze di caratteri, ma i numeri sono usualmente salvati nella codifica del tipo del numero stesso; per esempio il numero intero 1234 è conservato come 04D2 (esadecimale) che, in 4 byte (int) appare come D2, 04, 0, 0 (in codifica **little endian**, ossia byte meno significativi prima, in contrapposizione a big endian [usato nel formato pgm, vedere di seguito]), ovvero in decimale 210, 4, 0, 0. Sono sempre 4 byte… ma 1234567890 è codificato come 499602D2, ossia D2, 02, 96, 49, ancora 4 byte invece di 10 (nel caso di “una cifra = un byte”).

ciao 1234 è salvato come la sequenza di “c”, “i”, “a”, “o”, “ ”, 210, 4, 0, 0

I caratteri 210, 4, 0, 0 appaiono, se visti “come testo”, in base al loro corrispondente significato nel codice ASCII:



000	(nul)	016	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	(soh)	017	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	(stx)	018	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	(etx)	019	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	(eot)	020	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	(enq)	021	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	(ack)	022	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	(bel)	023	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	(bs)	024	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009	(tab)	025	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	(vt)	027	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	(np)	028	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013	(cr)	029	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	(so)	030	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	(si)	031	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂
128	Ç	144	É	160	á	176	▒	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▒	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	ø	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	∈	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

(da http://www.thealmightyguru.com/Pointless/ASCII.html)

Nei file testuali usualmente i fine-riga sono marcati da caratteri speciali : CR (dec 13) o LF (dec 10) o entrambi a seconda del sistema operativo (quindi i file non hanno la medesima lunghezza se creati in SO diversi).
Nei file binari il formato è rigido (o comunque può esserlo) perché i dati numerici occupano spazi fissi dipendenti dal tipo di dato. Nei file testuali la lunghezza di un numero dipende da quante cifre esso ha in decimale.

https://askubuntu.com/questions/19479/what-are-some-good-gui-binary-viewers-editors
sudo apt install ghex (editor binario)

```
#include <iostream>
#include <fstream>
#include <cstring>

int main()
{

	std::ofstream f("nothing.bin", std::ios::out | std::ios::binary);
	const char *cstr = "Pippo";
	f.write(cstr, strlen(cstr));
	char x = 255;
	f.write(&x, 1);
	f.close();
	return 0;
}
```

Esaminare con ghex.

Cambiare in:

```
        int x = 256;
        f.write((char *) &x, sizeof(int));
```

e vedere cosa è cambiato nel file.

Immagini in formato pgm

http://netpbm.sourceforge.net/doc/pgm.html

Each PGM image consists of the following:

1. A "magic number" for identifying the file type. A pgm image's magic number is the two characters "P5".
2. Whitespace (blanks, TABs, CRs, LFs).
3. A width, formatted as **ASCII characters in decimal**.
4. Whitespace.
5. A height, again in **ASCII decimal**.
6. Whitespace.
7. The maximum gray value (Maxval), again in **ASCII decimal**. Must be less than 65536, and more than zero.
8. A single whitespace character (usually a newline).
9. A raster of Height rows, in order from top to bottom. Each row consists of Width gray values, in order from left to right. Each gray value is a number from 0 through Maxval, with 0 being black and Maxval being white. Each gray value is represented in **pure binary** by **either 1 or 2 bytes**. If the Maxval is less than 256, it is 1 byte. Otherwise, it is 2 bytes. The **most significant byte** is first.

   A row of an image is horizontal. A column is vertical. The pixels in the image are square and contiguous.

All characters referred to herein are encoded in ASCII. "newline" refers to the character known in ASCII as Line Feed or LF. A "white space" character is space, CR, LF, TAB, VT, or FF (I.e. what the ANSI standard C isspace() function calls white space).

Plain PGM

There is actually another version of the PGM format that is fairly rare: "plain" PGM format. The format above, which generally considered the normal one, is known as the "raw" PGM format. See **pbm** for some commentary on how plain and raw formats relate to one another and how to use them.

……

Come salvare un file in format pgm

```
#include <iostream>
#include <fstream>
#include <cstring>

int main()
{
        // Write the pgm header
        //

        int width  = 20;
        int height = 10;
        const char *img  =    // 4 x 50 chars
                "qwertyuiopasdfghjklzxcvbnqwertyuiopasdfghjklzxcvbn"
                "qwertyuiopasdfghjklzxcvbnqwertyuiopasdfghjklzxcvbn"
                "qwertyuiopasdfghjklzxcvbnqwertyuiopasdfghjklzxcvbn"
                "qwertyuiopasdfghjklzxcvbnqwertyuiopasdfghjklzxcvbn";
        std::ofstream fimg("output.pgm", std::ios::out | std::ios::binary);
        fimg << "P5 " << width << " " << height << " " << 255 << std::endl; // Sintassi testuale!!
        fimg.write(img, width*height);
        fimg.close();
        return 0;
}
```

Visualizzare pgm in linux:

eog output.pgm (visualizzatore di immagini)

Come codificare una matrice bidimensionale in C++ senza usare librerie apposite (quale boost)

```
// 2D arrays (matrices)

#include <vector>

int main() {
  int rows = 100, cols = 200;
  int r = 20, c = 35;

  ////////////////////////////
  // DYNAMIC ALLOCATION
  ////////////////////////////
  // http://www.cplusplus.com/doc/tutorial/dynamic/     (new; exceptions)
  // https://stackoverflow.com/questions/1403150/how-do-you-dynamically-allocate-a-matrix

  int **matrixA = new int *[rows];
  for (int i = 0; i < rows; ++i)
    matrixA[i] = new int[cols];

  matrixA[r][c] = 135;

  // for(int r=0;r < rows;++r)
  //  for(int c=0;c < cols;++c)
  //     matrixA[r][c]= matrixA1[r][c];

  for (int i = 0; i < rows; ++i)
    delete[] matrixA[i];
  delete[] matrixA;

  ////////////////////////////

  int **matrixB = new int *[rows];
  matrixB[0] = new int[rows * cols];
  for (int i = 1; i < rows; ++i)
    matrixB[i] = matrixB[0] + i * cols;

  matrixA[r][c] = 135;

  delete[] matrixB[0];
  delete[] matrixB;

  ////////////////////////////

  int *matrixC = new int[cols * rows];
  int idx = r * cols + c;

  matrixC[idx] = 135;

  // Copy
  // for(int i=0;i < rows*cols;++i)
  //   matrixC[i]=matrixC1[i];

  delete[] matrixC;

  ////////////////////////////////////////
  // USE OF (STANDARD LIBRARY) VECTOR TYPE
  ////////////////////////////////////////
```

```
  std::vector<std::vector<int>> matrixD;

  // Grow rows by rows
  matrixD.resize(rows);
  for (int i = 0; i < rows; ++i) {
    // Grow Columns by cols
    matrixD[i].resize(cols);
  }

  matrixD[r][c] = 135;
}

// also:
// std::vector< std::vector<int> > matrixD (rows, std::vector<int>(cols,0));
// correct even if cols = 0, rows = 0;
```

Variable members "static" di una classe

(Dal testo di Cesare Rota)

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

class ctest
{
public:
        static int a;   // dichiarazione
        int b;
public:
        void impostaDati(int a, int b)
        {
                this -> a = a;
                this -> b = b;
        }
} ;

int ctest::a;  // definizione

int main()
{
        ctest t1;
        ctest t2;

        t1.impostaDati(10, 20);
        cout << t1.a << endl;
        cout << t1.b << endl;

        t2.impostaDati(100, 200);
        cout << t2.a << endl;
        cout << t2.b << endl;

        cout << t1.a << endl;
        cout << t1.b << endl;

        return 0;
}
```

Come sopra, con membri private

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

class ctest
{
private:
        static int a;   // dichiarazione
        int b;
public:
        void impostaDati(int a, int b)
        {
                this -> a = a;
                this -> b = b;
        }

        void stampaDati()
        {
                cout << a << endl;
                cout << b << endl;
        }

} ;


int ctest::a;  // definizione

int main()
{

        ctest t1;
        ctest t2;


        t1.impostaDati(10, 20);
        t1.stampaDati();

        t2.impostaDati(100, 200);
        t2.stampaDati();

        t1.stampaDati();

return 0;
}
```

Registrare variabili comuni a tutti gli oggetti di una classe, oppure uso come contatore!

[LEZIONE 8]

Ricapitolazione dei principali metodi “standard” di una classe

Notare che siamo in presenza dell’overloading di funzioni membro (i costruttori) e di un caso di operator overloading (l’assegnazione). L’overloading è ottenuto ridefinendo funzioni e operatori con differente “signature”, in particolare per ciò che concerne i parametri.

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

class A {
private:
  int x;
```

```
public:
  A() {
    x = 0;
    cout << "Default Constructor\n";
  }

  A(int x) {
    this->x = x;
    cout << "Constructor\n";
  }

  A(const A &acopied) {
    cout << "Copy Constructor\n";
    x = acopied.x;
  }

  A& operator=(const A &assigned) {
    cout << "Assignment operator\n";
    x = assigned.x;
    return *this;
  }

  void print() { cout << x << endl; }

  void setX(int x) { this->x = x; }

  int getX() { return x; }
};

int main() {
  A aa;
  A a(22), b(33);
  //    a.x = 22;

  a.print();
  b.print();
  aa.print();

  A aaa(a);
  aaa.print();

  aaa = b;
  aaa.print();

  aaa.setX(123);
  aaa.print();
  cout << aaa.getX() << endl;

  A aaaa(b);

  A aaaaa = aaaa;

  A d(A(55));  // <--
  return 0;
}
```

Overloading di funzioni

https://www.learncpp.com/cpp-tutorial/76-function-overloading/

Overloading di operatori

https://www.learncpp.com/cpp-tutorial/91-introduction-to-operator-overloading/

Dopo aver studiato le funzioni friend, trattate qui

https://www.learncpp.com/cpp-tutorial/813-friend-functions-and-classes/

è possibile leggere l'uso delle funzioni friend per l'overloading degli operatori:

https://www.learncpp.com/cpp-tutorial/92-overloading-the-arithmetic-operators-using-friend-functions/

Overloading con funzioni "normali":

https://www.learncpp.com/cpp-tutorial/9-2a-overloading-operators-using-normal-functions/

Overloading con member functions:

https://www.learncpp.com/cpp-tutorial/94-overloading-operators-using-member-functions/

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


vector<double> operator+(const vector<double> &v1, const vector<double> &v2)
{
	if (v1.size() != v2.size())
		// http://www.cplusplus.com/reference/cmath/nan-function/
		return vector<double>(1, nan(""));
	else
	{
		vector<double> v(v1.size());
		for (int j = 0; j < v1.size(); j++)
			v[j] = v1[j] + v2[j];
		return v;
	}
}

std::ostream& operator<< (std::ostream &out, const vector<double> v)
{
	for (int j = 0; j < v.size(); j++)
		out << v[j] << " ";
	return out; // return std::ostream so we can chain calls to operator<<
}

bool operator== (const vector<double> &v1, const vector<double> &v2)
{
	if (v1.size() != v2.size())
		// http://www.cplusplus.com/reference/cmath/nan-function/
		return false;
	bool b = true;
	for (int j = 0; j < v1.size(); j++)   //  && b
		if (v1[j] != v2[j])
			b = false;
	return b;
}


int main()
{
```

```
        vector<double> v1, v2;

        v1.push_back(10);
        v1.push_back(20);

        v2.push_back(100);
        v2.push_back(200);

        vector<double> v;
        v = v1 + v2;

        cout << v[0] << " " << v[1] << endl;
        cout << v << endl;

        cout << (v1 == v2) << endl;   //  <-- parentheses

        vector<double> v3;
        v3.push_back(10);
        v3.push_back(20);
        cout << (v1 == v3) << endl;

        return 0;
}
```

Esercizio

Scrivere un programma contenente una classe, denominate cimage, che modelli il tipo “immagine bidimensionale a 8 bit” (toni di grigio, unsigned int 8 bit, quindi unsigned char sui sistemi più diffusi), di dimensioni arbitrarie (per numero righe e numero colonne).

La classe abbia al proprio interno le seguenti variabili membro:

- “width” e “height”, che contengano rispettivamente il numero di colonne e di righe dell’immagine,
- “img”, variabile matriciale realizzata sulla falsariga di uno dei modi visti in precedenza
- “note”, di tipo string, che contenga una nota di testo.

Siano implementati:

- costruttore di default (creazione di un’immagine vuota di dimensioni 0x0) (azzera “width” e “height”, e crea una “img” come matrice vuota [puntatore nullo, oppure vector<vector<>> di dimensioni nulle], “note” vuota).
- costruttore, date le dimensioni “width” e “height” e una stringa per la “note”
- costruttore per copia
- operatore di assegnazione
- distruttore
- funzioni getter e setter per la variabile membro “note”, getter per “width” e “height”
- funzione plot, che accetta tre vector di interi della stessa lunghezza (rows, columns, grays) e accende con il livello di grigio gray[k] il pixel dell’immagine di coordinate (nel senso delle righe e delle colonne) rows[k] e columns[k], per k che va da 0 a gray.size()-1; sia fatto un check sull’effettiva appartenenza di tale pixel all’immagine (ossia le coordinate siano tra 0 e height-1, e tra 0 e width – 1)
- funzione printNote() che stampi a video la nota

- funzione saveAsPGM(string fname) che salvi l'immagine come pgm; i pixel hanno valori da 0 a 255, quindi si scriva opportunamente l'header; dopo l'header, le righe dell'immagine vanno salvate come sequenze di byte binari.

Il programma poi dichiari un'immagine, disegni dei pixel, salvi come pgm. Si visualizzi con eog (o altro strumento) per constatare il risultato.

[LEZIONE 9]

Richiamo più completo:

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

class A {
private:
  int x;

public:
  A() {
    x = 0;
    cout << "Default Constructor\n";
  }

  A(int x) {
    this->x = x;
    cout << "Constructor\n";
  }

  A(const A &acopied) {
    cout << "Copy Constructor\n";
    x = acopied.x;
  }

  A& operator=(const A &assigned) {
    cout << "Assignment operator\n";
    x = assigned.x;
    return *this;
  }


  void print() { cout << x << endl; }

  void setX(int x) { this->x = x; }

  int getX() const { return x; }    // <---- const!!! otherwise errors when getting from const a
};


// out of the class, "normal" functions

A operator+(const A &a1, const A &a2) {
  cout << "Operator+\n";
  A a;
  a.setX(a1.getX() + a2.getX());   // must use getters (no friend!)
  return a;
}
```

```
std::ostream& operator<< (std::ostream &out, const A a)
{
  out << a.getX() << " ";
  return out; // return std::ostream so we can chain calls to operator<<
}


bool operator== (const A &a1, const A &a2)
{
  return a1.getX() == a2.getX();
}


int main() {
  A aa;
  A a(22), b(33);
  //    a.x = 22;

  a.print();
  b.print();
  aa.print();

  A aaa(a);
  aaa.print();

  aaa = b;
  aaa.print();

  aaa.setX(123);
  aaa.print();
  cout << aaa.getX() << endl;

  A aaaa(b);

  A aaaaa = aaaa;

  A d(A(55));  // <--

//

  aa = a + b;
  aa.print();

  cout << aa << endl;

  cout << (aa == b) << endl;
  cout << (aa == A(aa)) << endl; // <---- on the fly...

  return 0;
}
```

Ereditarietà (inheritance)

https://www.learncpp.com/cpp-tutorial/111-introduction-to-inheritance/

```
#include <iostream>
#include <string>

class Person
```

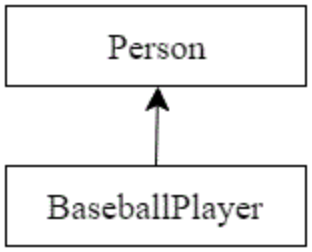
```
{
private:
    std::string m_name;
    int m_age;
public:
        // constructor with default parameters
    Person(std::string name = "", int age = 0) : m_name(name), m_age(age)
    {
        std::cout << "constructor for: Person\n";
    }

    std::string getName() const { return m_name; }
    int getAge() const { return m_age; }
};

class BaseballPlayer : public Person
{
private:
    double m_battingAverage;
    int m_homeRuns;
public:
    BaseballPlayer(double battingAverage = 0.0, int homeRuns = 0)
       : m_battingAverage(battingAverage), m_homeRuns(homeRuns)
    {
        std::cout << "constructor for: BaseballPlayer \n";
    }
    double getBattingAverage () const { return m_battingAverage; }
    int getHomeRuns () const { return m_homeRuns; }

};

int main()
{
    Person p1("Pippo", 35);
    std::cout << p1.getName() << std::endl;
    std::cout << p1.getAge() << std::endl;
    BaseballPlayer b1(5.5, 12);
    std::cout << b1.getName() << std::endl;
    std::cout << b1.getAge() << std::endl;
    std::cout << b1.getBattingAverage () << std::endl;
    std::cout << b1.getHomeRuns () << std::endl;
    return 0;
}
```



Versione 2: assegnazione dei variable members della classe base

```
#include <iostream>
#include <string>

class Person
```

```
{
private:
    std::string m_name;
    int m_age;
public:
        // constructor with default parameters
    Person(std::string name = "", int age = 0) : m_name(name), m_age(age)
    {
        std::cout << "constructor for: Person\n";
    }

    std::string getName() const { return m_name; }
    int getAge() const { return m_age; }
};

class BaseballPlayer : public Person
{
private:
    double m_battingAverage;
    int m_homeRuns;
public:
    BaseballPlayer(std::string name = "", int age = 0, double battingAverage = 0.0, int homeRuns =
0)
        : Person(name, age),
          m_battingAverage(battingAverage), m_homeRuns(homeRuns)
    {
        std::cout << "constructor for: BaseballPlayer \n";
    }
    double getBattingAverage () const { return m_battingAverage; }
    int getHomeRuns () const { return m_homeRuns; }

};

int main()
{
    Person p1("Pippo", 35);
    std::cout << p1.getName() << std::endl;
    std::cout << p1.getAge() << std::endl;
    BaseballPlayer b1("Topolino", 38, 5.5, 12);
    std::cout << b1.getName() << std::endl;
    std::cout << b1.getAge() << std::endl;
    std::cout << b1.getBattingAverage () << std::endl;
    std::cout << b1.getHomeRuns () << std::endl;
    return 0;
}
```

Private, protected, etc

https://www.learncpp.com/cpp-tutorial/115-inheritance-and-access-specifiers/

Overriding

https://www.learncpp.com/cpp-tutorial/11-6a-calling-inherited-functions-and-overriding-behavior/

Vedere tutto tranne l'ultimo esempio che usa le friend!

Esercizio no.1

Realizza una classe 'contatore' denominata C che contenga una variabile intera 'valore' come attributo e tre metodi, il primo sia un costruttore senza parametri che inizializzi 'valore' a 0; il secondo incrementi 'valore ' di 1 tutte le volte che è invocato; il terzo stampi a video il valore della variabile 'valore'.

Esegui il test del programma con un main che definisca un oggetto della classe contatore, e poi ne incrementi il valore in un ciclo for, via via stampandolo a video.

Fatto questo in un singolo file, si strutturi il programma su tre file:

- main.cpp (parte main che usa il counter)
- C.h (contenente le dichiarazioni relative alla classe C)
- C.cpp (contenente le definizioni dei metodi della classe C)

Aggiungere un costruttore che inizializzi a un valore diverso da 0.

Arricchire il programma derivando, dalla classe C, una sottoclasse "contatore colorato" denominata ColouredC, che abbia in piu' un colore che cicli tra quelli fondamentali, per esempio:

“red” corrispondente a 0, “orange” corrispondente a 1,…;

incrementare un ColouredC significa sia aumentare di 1 il valore, sia ciclare tra i colori (red->orange->…->violet->red…)

Scrivere un programma C++ avente le seguenti caratteristiche:

1) sia composto di tre file: main.cpp, myclasses.cpp, myclasses.h

2) i file myclasses.* contengano le definizione delle classi seguenti:

- una classe denominata "CProduct" che rappresenti un tipo di prodotto alimentare e abbia i seguenti data members privati:

- name (di tipo string; valori possibili "pasta", "patata", "olio", "passataDiPomodoro", "formaggioParmigiano")
- proteins (double; contenuto in termini di proteine, per 100 g)
- carbohydrates (double; contenuto in termini di carboidrati, per 100 g)

- lypids (double; contenuto in termini di grassi, per 100 g)

e i seguenti function members:

- un costruttore di default
- un costruttore con inizializzazione dei data members
- quattro funzioni di tipo get e di tipo set, rispettivamente per leggere e restituire i quattro data members, e per impostarne il valore
- una funzione di print che scriva a video le informazioni sul prodotto
- altri che si giudicasse necessario inserire.

- una classe CProductPortion derivata da CProduct che rappresenti una porzione del prodotto, e quindi abbia in piu' solo un data member privato:

- weight (tipo double, il peso in grammi della porzione).

e i function members necessari, in particolare un paio di costruttori (default e con inizializzazione) getter, setter, print...

Deve pero' avere anche un costruttore cosi' dichiarato:

- CProductPortion::CProductPortion(CProduct prod, double w);

che servira' a costruire una porzione di un particolare prodotto; per esempio, se si suppone di aver definito il prodotto pane di tipo CProduct, allora potremo scrivere

CProductPortion porzioneDiPane(pane, 75);

per definire appunto una porzione di pane di 75 g.

La classe CProductPortion abbia inoltre un function member che calcoli e restituisca in kcal il valore energetico della porzione, considerato che

- 1g di proteine dà 4 kcal / g
- 1g di grassi dà 9 kcal / g
- 1g di carboidrati dà 4 kcal / g

3) Nel main, si definiscano alcune variabili chiamate pane, pasta, etc di tipo CProduct contenenti i prodotti seguenti:

- nome dell'alimento: proteine lipidi e carboidrati
- pane: 8,1 0,5 63,5
- pasta: 4,7 0,5 30,3
- patata: 1,8 0,1 16,9
- passataDiPomodoro: 1,3 0,2 3,0
- olio: 0 99,9 0

- formaggioParmigiano: 33,5 28,1 0

Infine (ouff!) sempre nel main si dichiari un vettore di CProductPortion denominato spaghettata che abbia al suo interno, come elementi:

- un CProductPortion di 100 g di pasta
- un CProductPortion di 50 g di passataDiPomodoro
- un CProductPortion di 3 g di olio
- un CProductPortion di 10 g di formaggioParmigiano

Ciascuno di questi elementi sarà stato definito a partire dagli alimenti pane, pasta etc di cui si è parlato più sopra.

Infine il main calcoli, con un loop, il contenuto energetico in kcal di "spaghettata".

[Lezione 10]

Metodi virtuali

https://www.learncpp.com/cpp-tutorial/121-pointers-and-references-to-the-base-class-of-derived-objects/

https://www.learncpp.com/cpp-tutorial/122-virtual-functions/

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


class A {

protected:
	string name;
public:
	A(string n = "") : name(n)
	{
		cout << "A constructor\n";
	}
	virtual void whoAmI()
	{
		cout << "I am an A and my name is " << name << endl;
	}
} ;


class B : public A
{
public:
	B(string n = "") : A(n)
	{
```

```
                cout << "B constructor\n";
        }
        virtual void whoAmI()
        {
                cout << "I am an B and my name is " << name << endl;
        }
} ;


int main()
{
        A a("Pippo");
        a.whoAmI();
        cout << endl;
        B b("Minnie");
        b.whoAmI();
        cout << endl;

        A& rA = a;
        rA.whoAmI();
        cout << endl;

        B& rB = b;
        rB.whoAmI();
        cout << endl;

//// NOW THE PROBLEM!!! To fix it, "virtual void whoAmI()"
        A& rBA = b;  // <-- base pointer to a derived object, uses base member! Add virtual...
        rBA.whoAmI();
        cout << endl;



        return 0;
}
```

`sudo apt-get install clang-format` ("code beautifier")

`clang-format  -I  main.cc`

https://www.geeksforgeeks.org/const-member-functions-c/

Const member functions in C++

A function becomes const when const keyword is used in function's declaration. The idea of const functions is not allow them to modify the object on which they are called. It is recommended practice to make as many functions const as possible so that accidental changes to objects are avoided.

Following is a simple example of const function.

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

class Test {
    int value;
public:
    Test(int v = 0) {value = v;}

    // We get compiler error if we add a line like "value = 100;"
    // in this function.
    int getValue() const {return value;}
};

int main() {
    Test t(20);
    cout<<t.getValue();
    return 0;
}
```

Output:

```
20
```

When a function is declared as const, it can be called on any type of object. Non-const functions can only be called by non-const objects.

For example the following program has compiler errors.

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

class Test {
    int value;
public:
    Test(int v = 0) {value = v;}
    int getValue() {return value;}
};

int main() {
    const Test t;
    cout << t.getValue();
    return 0;
}
```

Output:

```
 passing 'const Test' as 'this' argument of 'int
Test::getValue()' discards qualifiers
```