

Laboratorio di Algoritmi e Strutture Dati

Vesti di Riferimento

F. Moavero

- [Str'13] B. Stroustrup, *The C++ Programming Language*, 2013, 4^a ed. A.A. 2024/2025
- [Str'14] B. Stroustrup, *Programming - Principles and Practice Using C++*, 2014, 2^a ed.
- [She'13] C.A. Sheffer, *Data Structures and Algorithm Analysis in C++*, 2013, 3.2^a ed.
- [CLRS'09] T.H. Cormen et al., *Introduction to Algorithms*, 2009/22, 3/4^a ed.

Argomenti del Corso

- Elementi di Linguaggio C++.
- Tipi di Dati Abstracti e Relative Implementazioni in C++.
- Progettazione di una Libreria Contenitore di Dati.
- Strutture Dati Elementari (Vetori, liste, Pile, code).
- Alberi Binari di Ricerca & Iteratori sui Dati.
- Tabella Hash & Grafici.

Lezioni ~ 8 Aprile - 10 Giugno

- Mercoledì, ore 16:00 - 18:00 - Aula A7
- Mercoledì, ore 14:00 - 16:00 - Aula A7
- Giovedì, ore 11:00 - 13:00 - Aula G3

Ricerca

- Lunedì, ore 16:00 - 18:00 (MS Teams)

Annesso all'Esame e agli Esercizi Intercorso

- Cognomi: A - G (Gruppo 1)

Laboratorio di Algoritmi e Strutture Dati

F. MAGARZO

Esercizi Intercorsa

- Esercizio 1 (Vettori, Liste, Pile e Code): 29 Aprile - 18 Maggio.
Str¹/13 (13.6,31) d She¹/13 (4.1,4.2,4.3,4.4).
- Esercizio 2 (Alberi: Binari e Iteratori): 16 Maggio - 1 Giugno.
She¹/13 (5.1,5.2,5.3,5.4) d Str¹/13 (4.5).

Modalità di Esame

- Esame semplificato: Discussione orale delle teorie e delle librerie implementate;
+ Precondizioni: Consegue esercizi intercorsa entro le sedute previste.
- Esame completo: Progettazione di una libreria da zero (Classi astratte + 1/2 Funzioni Concrete);
+ Precondizioni: Consegue esercizi entro il termine di presentazione dell'appello.

Modalità di consegnare esercizi

A.A. 2024/2025

- Invio di un file compresso (zip o tar.gz) tramite "Attività" MS Teams
- Formato file: Cognome - Nome - Matricola.zip/tar.gz.
- Selezioni improrogabili: Ore 23:59 dell'ultimo giorno previsto per l'esercizio.
- Profondità formale: Programmazione & Laboratorio di Programmazione
- Profondità sostanziale: Conoscenza dei concetti presentati ad AED

Elementi di Linguaggio C++

- 1) Struttura modulare di un programma: [Str¹/3(2.2, l.4, 15)].
- 2) Tipi fondamentali, puntatori e riferimenti: [Str¹/3(6, 7)] [Str¹/4(8)].
- 3) Allocazione dinamica delle memorie: [Str¹/3(11.2)] [Str¹/4(25.3, A.5.6)].
- 4) Tipi definiti dall'utente (structures & Enumerations): [Str¹/3(2.3, 8)].
- 5) Libreria iostream e funzioni di I/O: [Str¹/3(4.3, 38)] [Str¹/4(10)].
- 6) Libreria string: [Str¹/3(4.2, 36)] [Str¹/4(23.2, 27.5 B.8)].
- 7) Generazione pseudo-casuale di numeri: [Str¹/3(5.6.3, 40.7)] [Str¹/4(24.7, B.9.6)].
- 8) Eccezioni e relativa gestione: [Str¹/3(13)] [Str¹/4(5.6, 19.4, 19.5)].
- 9) Puntatori a funzione: [Str¹/3(12.5)].
- 10) Classi, oggetto, e template: [Str¹/3(3, PARTE III)].
- 11) Differenze e similitudini con i linguaggi C e Java: [Str¹/4(27)].

Struttura Modulare di un Programma

A simple/minimal program

- main.cpp (C++ program)

```

1) #include <iostream>           // Libreria standard C++ per l'I/O.
2) using namespace std;         // Diciturazione dello spazio dei tipi (accesso alle funzioni deve servire la specificazione "std::").
3) int main() {
4)     cout << "HelloWorld!" << endl;
5)     return 0;
6) }
```

↑
Istruzione di output

(Ottionale)

} Corpo principale del programma

Std:: fun

Esempio: "std::cout <<..."

- build.sh (Bash script)

#!/bin/bash output/file eseguibile

g++ [-O2] -o main main.cpp

Compilatore
GNU

parametri di ottimizzazione
optional

-O/-O1/-O2/-O3

ciclo forse

Estrutture Modulari di un Programma

A breve struttura del programma



- build.sh

#!/bin/bash

g++ [-O3] -o main test.cpp main.cpp

- main.cpp

- 1) #include <iostream>
- 2) #include "test.hpp"
- 3) using namespace std;
- 4) int main() {
- 5) cout << "Chiedere e funzione:";
- 6) test();
- 7) }

- test.hpp

- 1) #ifndef TEST_HPP
- 2) #define TEST_HPP
- 3) void test();
- 4) #endif

Risolve
dot progettore

- test.cpp

- 1) #include <iostream>
 - 2) #include "test.hpp"
 - 3) void test() {
 - 4) std::cout << "OK!" << endl;
 - 5) }
- Voir test();
test();
OK!

Tipi Fondamentali, Puntatori e Riferimenti

- Dichiarazione di variabili: type varname [= default value];
type varname [num elements] [= {val₀, ..., val_{numelements}}];
- Tipi fondamentali in C++: bool | [unsigned] (char | short | int | long int | long long int) | (uint/ulong)
true / false | float | double | long double

Esempi: unsigned int x = 3;

double vector[3] = {0.1, 2, 5.7}; $\rightarrow (0.1, 2, 5.7)$

long matrix[3][2] = {{0, 1}, {2, 3}, {4, 5}}; $\rightarrow \begin{pmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{pmatrix}$

- Tipi const: const type varname = value;

Esempi: const double pi = 3.141592;

Utili anche nel passaggio dei parametri nelle chiamate e funzioni.

Tipi Fondamentali, Puntatori e Riferimenti

• Puntatori: type* Variabile [= default address];

3 semp:

cout << p << endl;

char c = 'a';

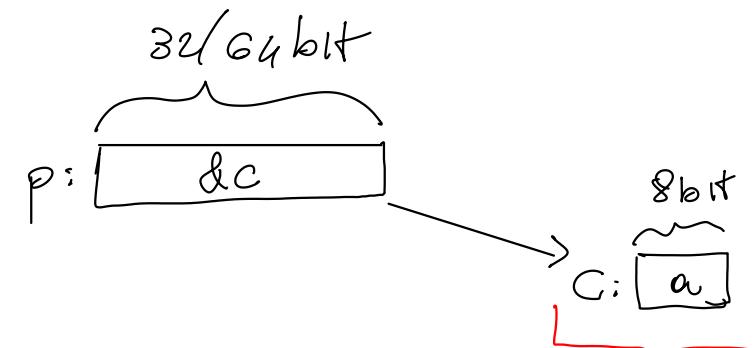
char* p = &c;

cout << *p << endl;

q = nullptr; // nullptr corrisponde alla mera NULL del linguaggio C.

cout << c << endl;

Cout << *p << endl;



• Void pointer: void* variabile [= ...];

3 semp:

char d = *p; // errore

char d = *(char*)p; // ok

Cout << *p << endl;

void* p = &c;

cout << *p << endl; // errore

{ cout << *((char*)p) << endl; // ok

{ cout << * (static_cast<char*>(p)) << endl; ok

const char c = 'b';
char* p = &c; // errore
const char* p = &c; ok

Tipi Fondamentali, Preziosi e Riferimenti

Dichiarazione dei Riferimenti: tipo & variazione = ... (L values)

\exists $\text{sempis} \rightarrow \left\{ \begin{array}{l} \text{int } d = 7; \\ \text{int } d - l = d; \\ \text{lt++; } (\neq d++) \\ \text{cout} << d; \end{array} \right\}$	\exists $\text{sempis} \rightarrow \left\{ \begin{array}{l} \text{int } d = 7 \\ \text{int } e = d; \\ \text{lt++; } \\ \text{cout} << d; \end{array} \right\}$
\equiv	8

} $\min d \geq z;$
 $\min p \geq dd;$
 $(\cancel{p})++;$
 count << d;

{ int & i = 1; // Error
{ const int & i = f; // OK

Var = Exp
Left-hand Right-hand
Value Value

Pointer & Reference

p: dd

→

d: 7

l →

Esempio:

Tipo Fondamentali, Puntatori e Riferimenti

{Object Type f();
Object Type var = f();
Copia del
valore

fusione w librie
(R value)
Object Type var = std::move(f());
Object Type && var = f();
Esempio

{Object Type* f(); ← Necessarie direzioni dinamiche
Object Type* var = f();

{string f() { return string("This is a long string---");}
string var = f(); // Copia delle stringhe
string && var = f(); // Riferisce temporaneo di valore e ritorno
string & var = f(); // Errore
string var1 = std::move(f()); // Spostamento del valore
string var2 = std::move(var); // Spostamento del valore

Allocazione Dinamica delle Fluenze

uint num = expression;
type & var = new type [num];
...
delete [] var;

{ Object type* var = new Object Type();

(delete var; . potential ↑
 default
 value)

- $\left\{ \begin{array}{l} \text{new} \rightarrow \text{malloc}() \\ \text{delete} \rightarrow \text{free}() \end{array} \right\}$ del linguaggio C
 - Realloc must be directe
Corrispondente in C++
 - Non solleva un'eccezione / non distrugge altri puntatori quando fallisce
"bad-alloc"
 - No mixed "new" "delete".
 - Al uso degli operatori new/delete non si consiglia mai chiamare a fuori malloc/realloc / free.
 - Non considerare gli smart-pointer.

Tipi Definiti dall'Utente

```
struct StructName {
    Campi
};
```

Member access:

StructName::Campi

Access via pointer:

PointerVarName → Campo

(*PointerVarName).Camps

```
Example
struct Studente {
    string Nome;
    string Cognome;
    string Matricola;
    int id;
};
```

```
Studente var;
Studente* var = new Studente;
var->Nome = ...;
```

```
Studente Var = { string("Alan"), string("Turing"),
                  string("N8600000"), 1 };
```

```
struct Struct1 {
    char c1; int i; char c2
};

struct Struct2 {
    char c1; char c2; int i;
};
```

Memory representation

Struct1 (Ideal):

char	int	char
32		

Struct1 (Real):

char	int	
char		

Struct2 (Real):

char	char	int	
------	------	-----	--

```
{ var.Nome = string("Alan");
  var.Cognome = string("Turing");
  var.Matricola = string("N8600000"); }
```

Le struct sono classi con accesso pubblico agli attributi e alle funzioni membri; Nessune altre restrizioni.

Tipi Definiti dall'Utente

Enumerazione

- enum class Nome { elemento₁, ..., elemento_n};
- enum Nome { ... }; // e lo C

Esempio

```

enum class Color{ white, gray, black };
Color colore = Color::White;

enum Color2 { red, yellow, green };
Color2 colore2 = red;

enum Color3 { red, brown }; // Errore.

enum Color4 { white, red };
Color4 colore4 = Color4::White; // OK
Color4 colore4 = Color::White; // Errore

```

- Gli enumerazioni possono essere confrontati: ==, !=, <, >, <=, >=
 e assegnati: =

- Le variabili di tipo enumerativo possono avviamente avere un valore iniziale o default.

Accesso ai valori di una enum class:
Nome :: elementi

#include <iostream>

Librerie costituenti & Funzioni d'I/O

Operazioni principali

"<<" : put-to // da applicare a oggetto di tipo "ostream"; e.g. cout
">>" : get-from // da applicare a oggetto di tipo "istream"; e.g. cin

```
[cout << "Hello:" << 3 << 'c';
```

```
String Name;
```

```
int i;
```

```
char c;
```

```
cin >> Name >> i >> c;
```

```
struct Studente {
```

```
};
```

```
Studente vor;
```

Standard Streams

cout	}	output stream : ostream
cerr		
clog		

cin	}	input stream : istream
-----	---	------------------------

cout << vor; } // è come scrivere nell'operatore "<<";
cin >> vor;

Libreria <iostream> & Funzioni d'I/O

```
{ ostream& operator<< ( ostream& os, const Studente& st) {  
    return os << st.Nome << " - " << st.Cognome << " - " << st.Matricola << " - " << endl;  
}  
?  
{ istream& operator>> ( istream& is, Studente& st) {  
    is >> st.Nome >> st.Cognome >> st.Matricola >> st.id;  
}?
```

- Classe istream

- good(); eof(); fail(); bad(); funzioni membri per l'analisi dello stato.
- get(c);
 ↑ char.
 getline(p, n);
 ↑ char*
 ↑ numero di caratteri da leggere.

- Classe ostream

- put(c); write(p, n);
 ↑ char.
 ↑ numero dei caratteri da scrivere.
 ↓ count char*.

Liberarie string

- string (#include <string>) \neq string letteral C e C++ (char*)
- string Name = "Alou"; // string Name ("Alou"); // string Name = {"Alou"};
- Gli operatori string sono confrontabili lessicograficamente $=, !=, <, \leq, \geq, \geq=$.
- Operatori di input/output <<, >> (via overloading).
- size(); empty(); [i]; front(); back().
- Costruzione + ; string Var = Var1 + Var2; string Var += Var2;
- Var. substr(i, n) // sottostringe da posizione i a posizione i+n-1.
- {
 } string ref = "Alou"
 cout << Var. substr(1,2); // "la"

Generazione Pseudo-Casuale di Numeri

// Linguaggio C

```
#include < stdlib.h> // srand, rand  
#include < time.h> // time  
  
srand( time( NULL ));  
for( uint i=0; i<15; ++i)  
{  
    printf("%d", rand());  
}
```

// Linguaggio C++

```
#include < random>  
  
default_random_engine gen( random_device{}());  
uniform_int_distribution< type > dist( i,j );  
for( uint i=0; i<15, ++i )  
{  
    cout << dist( gen );  
}
```

↑ e.g. int/unint/loop...

Eccellenza e Relativa Gestione

- Strutture di Gestione Eccezioni.

```
try {  
    ...  
    throw SomeException();  
}  
  
catch (SomeException exc)  
{  
    ... // gestione dell'eccezione  
}  
  
catch (...) {  
    ...  
    risolvo eccezione  
    ...  
    throw;  
}  
  
• type f (parameters) noexcept;
```

le funzioni non rilasciano eccezione

- Eccellenza Utili delle Standard Library exception
 - + logic_error
 - + length_error
 - + out_of_range
 - + runtime_error
 - + overflow_error
 - + underflow_error
 - + bad_alloc

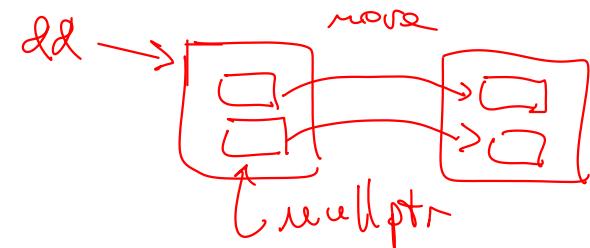
Puntatori e Funzione

- Possibili usi:
 - + Passaggio di funzione come parametro di altre funzioni;
 - + Funzioni di callback.
- Esempio:
 - + `void sort(int* A, int n);` // Ordinamento per default
 - + `bool (*Compare)(int x, int y);` // $x \leq y, x < y, \dots$
 - + `void sort(int* A, int n, Compare cmp)`
 - { ... if (cmp(x, y)) { ... }
 - else { ... }

uso rei confronto
- Definizione: `typedef type (*Funzione) (listi parametri); // Alc c.`
`typedef std::function<type(listi parametri)> Funzione; // Alc C++.`

richiede #include <functional>
- Un puntatore a funzione permette lo scambio di funzioni diverse con lo stesso prototipo conoscendone l'indirizzo.
- Chiamate: `*Funzione(...)` o `Funzione(...)`.
 - `bool lessThan(int x, int y) { return (x < y); }`
 - `bool GreaterThan(int x, int y) { return (x > y); }`
 - `bool CompareX(int x, int y) { ... }`
 - `Compare fun = [&]LessThan;`
 - `sort(A, 10, [&]LessThan);` `sort(A, 10, fun);`
 - `sort(A, 10, [&]GreaterThan);`

Classi, Oggetti e Tempore



- class ClassName {
 - [private:]
 - // Attributi e funzioni reservate (a.r.a metodi) privati, ossia accessibili solo all'interno stesse classe protected:
 - // Attributi e metodi visibili solo ad altre classi nello stesso gerarchia.
 - public:
 - // Attributi e metodi visibili a tutti.
 - Distruzione: ~ClassName(); // Destructor
 - Operatori, E.g.:
 - bool operator!= (const ClassName&) const noexcept; // Comparison
 - ClassName& operator= (const ClassName&); // Copy assignment
 - ClassName& operator= (ClassName& dd) noexcept; // Move assignment
- Costruttori:
- ClassName(); // Default constructor
 - ClassName(parameters); // Specific constructor
 - ClassName(const ClassName&); // Copy constructor
 - ClassName(ClassName& dd) noexcept; // Move constructor
- Diagram illustrating copy assignment. A box labeled "E" has two arrows pointing to another box. Inside the second box, there is a self-loop arrow labeled "copy". Below the boxes is the text "C++11ptr".

Classi, Oggetti e Tipi

- defn ClassName : [virtual] [private/protected/public] BaseClassName , ... {
 ...
};

• Metoli: [virtual] type NameFunc (parameterlist) [const] [noexcept] [override] [= assigned].

+ Pseudo assignment : =0 / = default / = delete .

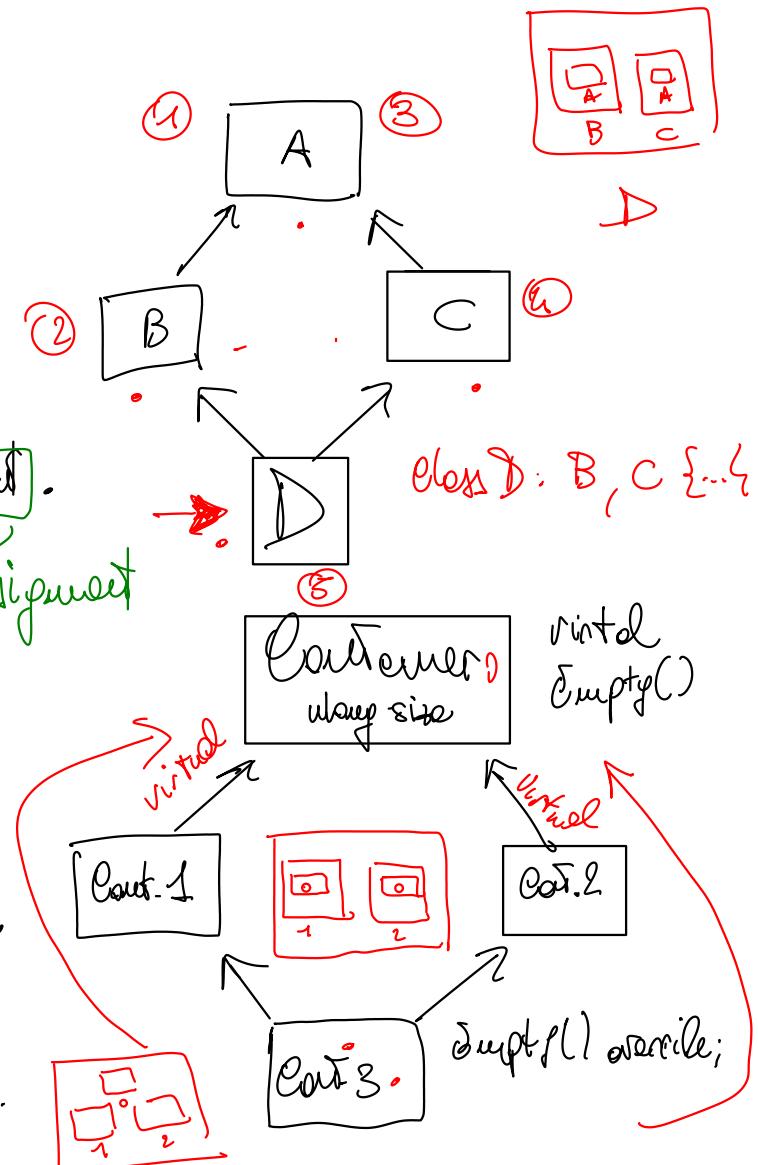
```

graph TD
    B["pure virtual"] --> D["default implementation  
(constructor & destructor)  
of default/copy/move"]
    D --> A["=0 / = default / = delete"]
    A -- "delete an existing method" --> C["= delete"]
  
```

The diagram illustrates the inheritance of a pure virtual assignment operator. It shows a base class node labeled "pure virtual" with an arrow pointing to a derived class node labeled "default implementation (constructor & destructor) of default/copy/move". From this derived class node, another arrow points to a final node labeled "=0 / = default / = delete". A third arrow originates from the "pure virtual" node and points directly to this final node. A green annotation "delete an existing method" is placed near the final node, with an arrow pointing upwards towards it.

- Définitione corrente di un metodo: type className:: FuncName(parameters) [specifiers] { Node }

- Interfazie Fair ≈ Close Distretto Poco (condiviso con uno o più virtuale pari).



Class, Objets e Tempate

Vector.hpp

- template <typename **Dof**>
 class Vector {
 private:
 int size = 0;
 Dof* Elements = nullptr;
 public:
 Vector() = default;
 Vector(int n);
 ~Vector();
 ...
 Dof& operator[](int);
 };
 #include "Vector.cpp"

- template <typename **Dof**>

```
Vector<Dof>::Vector(int n){
```

Elements = new **Dof**[n];

size = n;

- template <typename **Dof**>

```
Vector<Dof>::~Vector() {
```

delete [] Elements;

}

Vector.cpp

RAII

- Resource Acquisition Is Initialization

No "noted" new and delete in Vector.

Module.cpp

```
#include "Vector.hpp"
int main() {
```

```
Vector<int> vec(5);
```

```
vec[0] = 3;
```

```
cout << vec[0];
```

```
return 0;
```

Differenze e Somiglianze con i Linguaggi C e Java

- C++ è (quasi) un sovraccarico del linguaggio C
 - + C++ è C con le classi (e overloading, riferimenti, template, ecc.).
 - + Poche differenze a livello semantico (per i controlli validi in entrambi i linguaggi)
ad es. `int x = sizeof('a')` // x=k in C ma x=s in C++
*il tipo di un
carattere letterale è*
in C in C++

- Java è un linguaggio orientato agli oggetti che si ispira al C++.
Esistono due diverse sorgenti di differenze a livello progettuale:

C++	platform-dependent (native executable)	No Garbage Collector	Allocation on stack & heap	Operator overloading	Multiple Inheritance	Variety-Complete Object Programming via Template
Java	platform-independent (virtual machine)	Garbage Collector	Allocation (almost) only on heap	No operator overloading	Single Inheritance	Type-Personalization via Generics

Alcune differenze fondamentali: